Full-Whisker Tracking System: A new algorithm for accurate whisker tracking in untrimmed head-fixed mice

J.L.F. Betting

CE-MS-2018-09

Delft University of Technology

Abstract

The movement of whiskers in head-fixed mice is of high interest for neurological research, as it allows scientists to learn more about learning processes during active touch. However, manual tracking of whiskers in thousands of frames is not feasible, and reliable tracking of individual whiskers is not possible with the best current software available. In this work, a new system for the automatic tracking of whiskers in top-view videos of untrimmed, head-fixed mice is presented. The design of this system is based on a variety of image processing and computer vision algorithms. For each step, several alternatives have been considered and assessed. The best options have been combined and implemented in MATLAB. This new system, the Full-Whisker Tracking System (FWTS), detects the centerline of whiskers on each frame along their whole length with sub-pixel accuracy. The whiskers are defined by their angle, position, shape and length. FWTS uses these features to track and recognize whiskers over multiple frames in video segments with a frame rate of 1000 Hz. The system was tested on representative, challenging video segments from two experiments using different mice, and showed itself superior to the pre-existing system BWTT, by detecting on average 21.2% more whiskers, and with twice as much precision. In contrast to earlier systems, FWTS has been demonstrated reliably track individual whiskers in untrimmed mice. Furthermore, FWTS's processing time is on average 41.7% lower than that of BWTT. This thesis work shows that it is possible to track whiskers in untrimmed mice, even in the face of partial occlusions, and makes it possible to study the functioning of the intact whisker system of mice in unprecedented detail.

TU Delft Delft University of Technology

Quantum & Computer Engineering

Full-Whisker Tracking System: A new algorithm for accurate whisker tracking in untrimmed head-fixed mice

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Jan-Harm Betting
born in Winterswijk, The Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Full-Whisker Tracking System: A new algorithm for accurate whisker tracking in untrimmed head-fixed mice

by Jan-Harm Betting

Abstract

The movement of whiskers in head-fixed mice is of high interest for neurological research, as it allows scientists to learn more about learning processes during active touch. However, manual tracking of whiskers in thousands of frames is not feasible, and reliable tracking of individual whiskers is not possible with the best current software available. In this work, a new system for the automatic tracking of whiskers in top-view videos of untrimmed, head-fixed mice is presented. The design of this system is based on a variety of image processing and computer vision algorithms. For each step, several alternatives have been considered and assessed. The best options have been combined and implemented in MATLAB. This new system, the Full-Whisker Tracking System (FWTS), detects the centerline of whiskers on each frame along their whole length with sub-pixel accuracy. The whiskers are defined by their angle, position, shape and length. FWTS uses these features to track and recognize whiskers over multiple frames in video segments with a frame rate of 1000 Hz. The system was tested on representative, challenging video segments from two experiments using different mice, and showed itself superior to the pre-existing system BWTT, by detecting on average 21.2% more whiskers, and with twice as much precision. In contrast to earlier systems, FWTS has been demonstrated reliably track individual whiskers in untrimmed mice. Furthermore, FWTS's processing time is on average 41.7% lower than that of BWTT. This thesis work shows that it is possible to track whiskers in untrimmed mice, even in the face of partial occlusions, and makes it possible to study the functioning of the intact whisker system of mice in unprecedented detail.

Laboratory : Computer Engineering
Codenumber : CE-MS-2018-09

Committee Members :

Advisor:	Christos Strydis, Neuroscience, Erasmus MC
Chairperson:	Zaid Al-Ars, CE, TU Delft
Member:	Arjan van Genderen, CE, TU Delft
Member:	Laurens Bosman, Neuroscience, Erasmus MC

Dedicated to my family and friends

Contents

List of Figures

List of Tables

List of Acronyms

FWTS Full-Whisker Tracking System

BWTT BIOTACT Whisker Tracking Tool

GPU Graphics Processing Unit

CPU Central Processing Unit

SVM Support Vector Machines

CNN Convolutional Neural Network

k-NN k-nearest neighbors

ViSA Vibrissae and Snout Analysis

OMP Open Multi Processing

TSP Travelling Salesman Problem

TLD Tracking-Learning-Detection

DBSCAN Density-Based Spatial Clustering of Applications with Noise

Acknowledgements

First of all, I would like to thank my thesis advisor, Dr. Christos Strydis of the Neuroscience department of Erasmus MC, for his guidance and support over the course of this project. His feedback and comments on my work have been invaluable. I would also like to thank Dr. Zaid Al-Ars, my supervisor at Delft University of Technology, for helping me find the topic for this thesis, his support, and his comments on my work.

I would also like to thank Dr. Laurens Bosman of the Neuroscience department of Erasmus MC for his guidance and support, and Dr. Mario Negrello, Vincenzo Romano and Emily Middendorp for their comments and support over the course of this project.

Finally, I would like to express my gratitude to my partner Liudmila, my family, and my friends for their support throughout my years of study, and during the process of researching and writing. This accomplishment would not have been possible without them. Thank you.

Jan-Harm Betting
Delft, The Netherlands
April 16, 2018

1 Introduction

One of the most important ways in which animals explore their environment is by active touch: animals bring their receptors in contact with objects in their environment to obtain more information about its properties. Nature developed a wide range of systems for this purpose: primates use their fingertips, insects use their antennae, and rodents use their whiskers. As good assessment of the environment is vital for survival, animals have to adapt and control their active touching behavior over time, as they learn from previous input. Because of this, the study of active touch is important for research into the ways animals are able to learn and to adapt [15].

The whisker system of mice is of special interest to scientists: these animals usually live in dark environments and are heavily dependent on active touch. For this purpose, mice have intricate whisker systems, and the movement of the whiskers in response to stimuli can give a lot of information about the way these animals learn from external stimuli. However, tracking individual whiskers over time is a complex task: not only do these whiskers move rapidly, but they also spread, cross, align, and hide behind each other. Although a variety of whisker-tracking algorithms is already available in literature, these systems often fail at their task when the mouse moves its whiskers actively. This thesis presents the Full-Whisker Tracking System (FWTS), a new system for *whisker detection and tracking* in untrimmed, head-fixed mice, which can perform this task with good accuracy.

1.1 Motivation and problem statement

With the advent of high-speed cameras, it has become possible to record the movement of fast-moving mouse whiskers on video. The ability to track the angle and position of whiskers on video makes it possible to study the relation between brain activity and whisker movement in exploring rodents in a relatively non-invasive way. However, there are some challenges. The first challenge is that of data volume: high-speed cameras produce a large number of frames per seconds: a camera that records at a speed of 1 ms per frame produces 1000 frames per second. This means that a short segment of 50 seconds already contains 50.000 frames. Apart from making manual tracking virtually impossible, the large number of frames per segment also requires a lot of storage space.

These issues can be addressed with a system that detects and tracks whiskers in video segments automatically, and several attempts have already been made to implement such a system. One of the more extensive whisker tracking systems is the BIOTACT Whisker Tracking Tool (BWTT), which runs on MATLAB (MathWorks, Inc.) [16]. This system, of which the tracking algorithm will be described later in this thesis, is used by the Neuroscience department of the Erasmus MC in Rotterdam to track whisker movement in head-fixed mice.

This thesis is the latest installment in a long-standing and fruitful collaboration between the Erasmus MC and the Delft University of Technology ([17], [18]). The previous collaboration involved porting BWTT to two hardware accelerators (a NVIDIA GPU and a Maxeler DFE) and parallelizing the design (CPU with OMP) in an attempt to reduce BWTT's overall execution times and tackle the inherent throughput and latency challenges of the experimental setup. [9]

However, this work only tackled part of the problem. Although BWTT detects individual whiskers in video frames, it is unable to track individual whiskers. The system was originally designed to generate a 'naive average' of the angle of all whiskers per frame, over the course of the video (although its output per frame does give information about individual whiskers). A post-processing script, developed at the Erasmus MC Neuroscience department, exists that can compare the properties of individual whisker shafts in different frames in an attempt to track individual whiskers [19]. But in the video segments where the whiskers move fast (which, from the neurological point of view, are the most interesting ones), the script loses track of many whiskers. This is problematic, since individual whisker movement provides much more useful information for neurological research than a 'naive average'. Even though it is often not possible to track every single whisker in a video segment due to the fact that whiskers often cross and hide, it should usually be possible for most of the whiskers, especially the longer and hence more visible ones. The problem at hand, therefore, is **enabling the tracking of individual whiskers over time in the video segments of head-fixed mice, extracting their positions and angles to the snout, to the extent that this is possible regarding the quality of the video segment.**

1.2 Thesis scope and contributions

This thesis addresses the whisker-tracking problem by exploring a wide range of techniques that can be used for whisker tracking in top-view two-dimensional video images of head-fixed mice. The advantages and disadvantages of these techniques are assessed, and where possible, the algorithms were implemented and tested using MATLAB. The most suitable techniques were combined into a new whisker tracking system: the FWTS that can track a predefined set of whiskers over time in an accurate way. Whereas previous efforts were directed at acceleration, this thesis project was aimed at designing a new system that can detect and track whiskers with a much higher accuracy.

1.2.1 Thesis goal

The design goal of this work can be summarized as follows:

"The exploration of the possibilities to improve individual whisker tracking in untrimmed head-fixed rodents, by selecting existing algorithms and inventing new ones, in order to implement a working, quality-optimized whisker angle and position tracker in MATLAB."

At this point, BWTT is used by the researchers at Erasmus MC. The Vibrissae and Snout Analysis (ViSA) algorithm, which is at the core of BWTT, has been optimized

for speed by Ma et al. [20]. It would therefore be advantageous if it were still possible to use the output from BWTT to track individual whiskers. The first subgoal of his project can, thus, be formulated as follows:

1. Assess BWTT and the existing post-processing script. More specifically:
   (a) Assess the possibilities for post-processing the BWTT output;
   (b) Determine where data reduction leads to loss of accuracy;
   (c) Determine how whisker tracking can be improved, which parts of BWTT are useful, and where alternative algorithms are required.

The different algorithms that were assessed are divided into three categories: low-level processing, intermediate-level processing, and application-level processing. This categorization is derived from Davies (1997) [21], and will be described in more detail in Section 2.2. The second subgoal can be formulated as follows:

2. Assess low-, intermediate- and application-level processing techniques that are useful for whisker tracking, by
   (a) Reviewing relevant scientific literature on these topics, selecting relevant algorithms;
   (b) Inventing new algorithms;
   (c) Implementing these algorithms in MATLAB, applying them to the available video data.

After the assessment of processing algorithms, it becomes possible to combine a selection of them into a single MATLAB program that can detect whiskers in videos, describe them in terms of angle and position, and track them over different frames. This system takes video segments as its input, and produces data about the angle and position of single whiskers as its output. This leads us to the third subgoal:

3. Create a new system for whisker tracking in MATLAB:
   (a) Combine the algorithms and techniques that produce the best results in terms of quality into a single program.
   (b) Test the system for different parameter settings and video segments and compare the results.

Although the whisker-tracking system which is developed as part of this project is optimized for accuracy, rather than processing speed, it is important that the processing speed of a new whisker-tracking system be reasonable. In practical terms, this means that the processing time per frame of the MATLAB implementation of the system should be either comparable to that of BWTT, or lower. According to Ma, the processing time per frame of BWTT is about 5.7 seconds [9]. Although MATLAB is an interpreted language, which is inherently slower than compiled languages such as C and C++, there are different options to speed up MATLAB scripts by implementing parallelism (a concept that will be explained further in Section 2.4). A fourth, additional subgoal can therefore be added:

4. Where possible, optimize computation in the MATLAB implementation of the new whisker-tracking system by implementing parallelism.

1.2.2 Thesis contributions

In this thesis project, the following contributions were made:

- A detailed exploration of the existing whisker-tracking techniques was performed.
- A large number of algorithms from the field of image processing have been studied and were applied to the specific problem of whisker tracking in top-view videos of untrimmed, head-fixed mice, and the results were assessed.
- A new way to parametrize mouse whiskers was proposed, and several new algorithms to achieve this parametrization were invented and implemented.
- Algorithms were compared and combined to create a new whisker-tracking system, the FWTS, in MATLAB that takes video segments as its input and produces parametrized whisker traces as its output.
- Parallelism was implemented in the aforementioned whisker tracking system, making the processing time per frame lower than was tested by Ma et al. for BWTT on the same hardware.
- The tracking system was tested on segments from two different, representative videos, in which the mice move their whiskers numerous times. FWTS's performance was evaluated on different criteria.

The block diagram in Figure 1.1 shows the different algorithms that were analyzed. The algorithm steps that have become part of FWTS are marked, as well as the algorithms that were part of BWTT and the post-processing script.

1.3 Thesis organization

This thesis work is organized in nine chapters. Chapter 2 will describe the whisker-tracking problem and give a theoretical background on some of the basic concepts that are used in this work. Chapter 3 analyzes the possibilities to post-process the output of BWTT and describes the decision to create a new tracking system. Chapter 4 gives an overview of existing works regarding whisker tracking, including BWTT.

The process of developing FWTS is described in Chapters 5, 6 and 7. Chapter 5 describes various low-level processing algorithms and accounts for the decisions that were made in selecting algorithms for the implementation. Chapter 6 describes the intermediate-level processing steps, and Chapter 7 assesses the application-level processing steps. These three chapters form the core of this work.

Chapter 8 presents evaluation results of running FWTS on several video segments, with different parameters. The results are compared with each other and with the output of BWTT and the post-processing algorithm. Chapter 9 concludes this work and gives recommendations for further research on this topic.

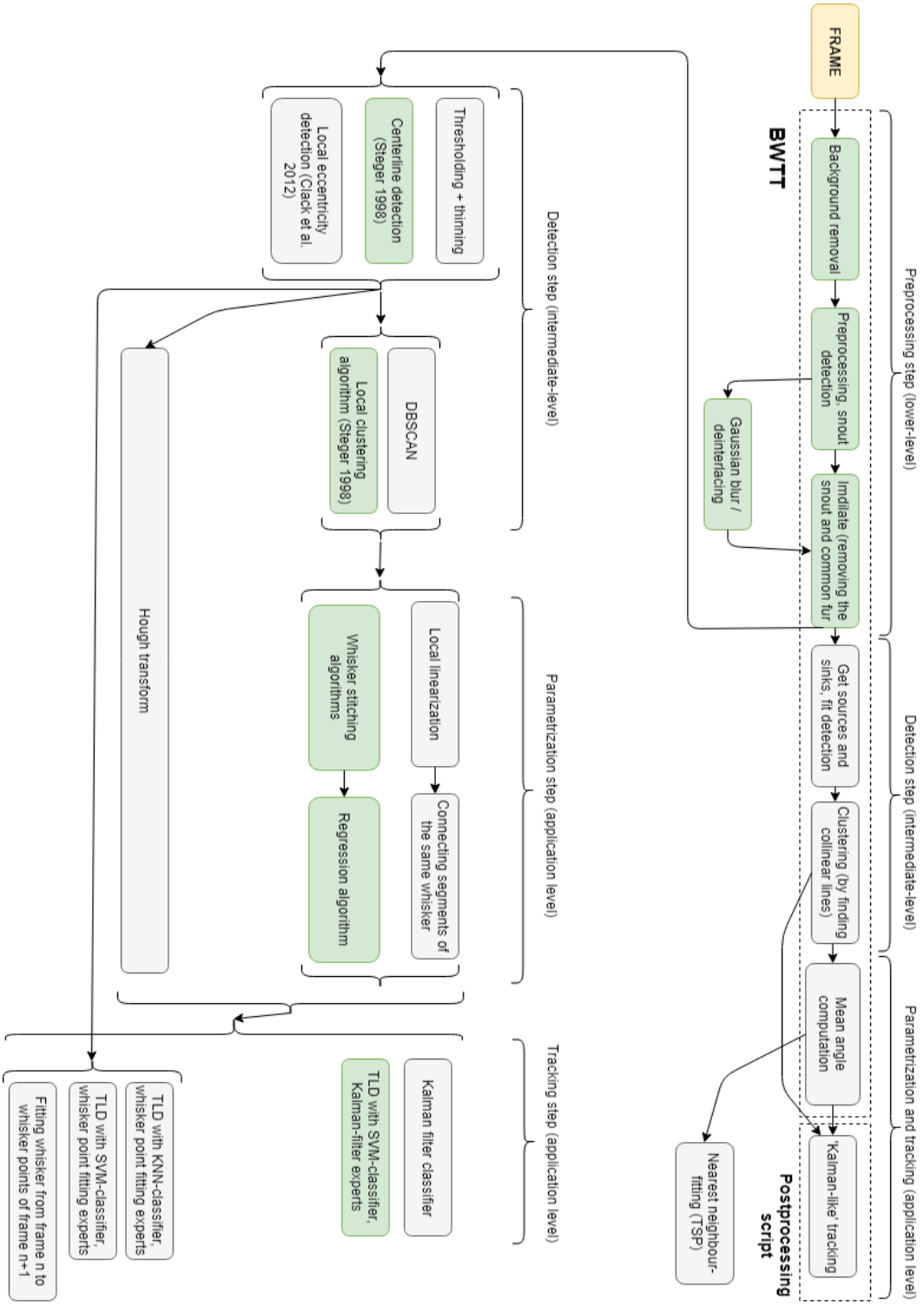


Figure 1.1: Different algorithms that were assessed. The steps that were part of BWTT and the existing post-processing script are shown in dashed boxes. The algorithm steps in grey boxes were assessed, but have not become part of FWTS. The algorithm steps in the green boxes were assessed and have become part of the whisker tracking system developed in this work.

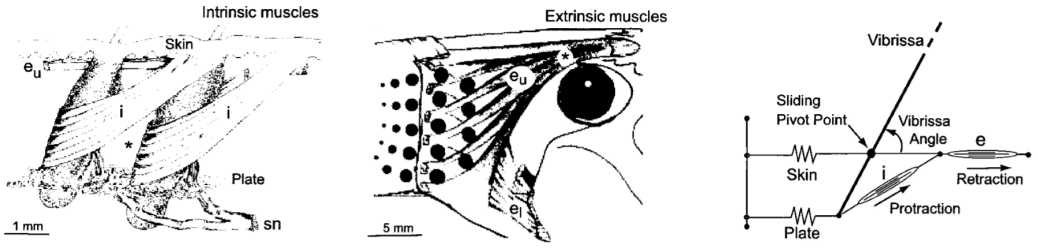
2 Background

This chapter serves as an introduction to the topic of whisker tracking and the different scientific fields upon which this work is based. Section 2.1 will describe the problem of whisker tracking in head-fixed mice. Section 2.2 will describe some of the fundamentals of computer-vision algorithms. Section 2.3 will give an introduction into how machine learning algorithms can be used for classification, and will introduce the principle of Support Vector Machines (SVM). Lastly, Section 2.4 will focus on the advantages of parallel processing.

2.1 Whisker movement and tracking in head-fixed rodents

The long whiskers (macrovibrissae) of rats and mice are located on the region of the snout known as the mystacial pad. Rats and mice have muscle control over their whiskers, which allows them to sweep them with frequencies up to 25 Hz. The whiskers are driven by two types of muscles: the intrinsic muscles, which are attached to the follicle-sinus complexes and thereby regulate the angle of the whiskers, and the extrinsic muscles, which allow the animal to move the pivot point of the whiskers by shifting the surface of the mystacial pad.

The mechanics of whisker movements are illustrated in Figure 2.1. The state of a whisker is a function of its pivot point, which can be shifted by the extrinsic muscles, and the angle, which is changed by the interplay of extrinsic and intrinsic muscles. The shift in the pivot point is limited by the elasticity of the skin; the change in angle is limited by the elasticity of the pad relative to the skin [1].

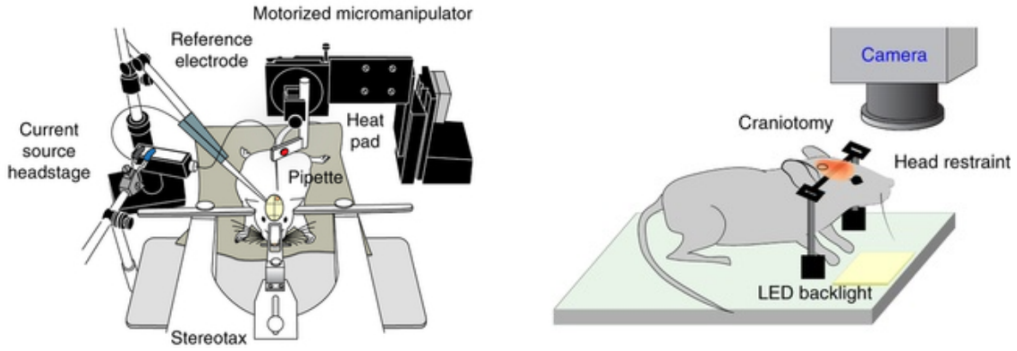


(a) The intrinsic muscles around whiskers

(b) The extrinsic muscles near the eye. The smaller black dots represent the whisker follicles.

(c) Mechanical model of whisker movement

Figure 2.1: Intrinsic and extrinsic muscles. Image taken from [1].



(a) Operating view (b) Head constraint and camera.

Figure 2.2: Illustration of the experimental setup.

2.1.1 Whisking experiments

At the Neuroscience department of the Erasmus MC in Rotterdam (the Netherlands), high-speed cameras operating at a frequency of about 1000 Hz are used to record whisking experiments. The test subjects are placed in the setup as illustrated in Figure 2.2. The mouse is prevented from moving its head, which is necessary to reliably track the whiskers, as these can move at speeds up to 5° per millisecond, and to collect signals from the brain. The infrared LED backlight makes sure that the thin whiskers are clearly visible to the camera. Over the course of the experiment, the animal is provided with stimuli in the form of air puffs to one of its eyes.

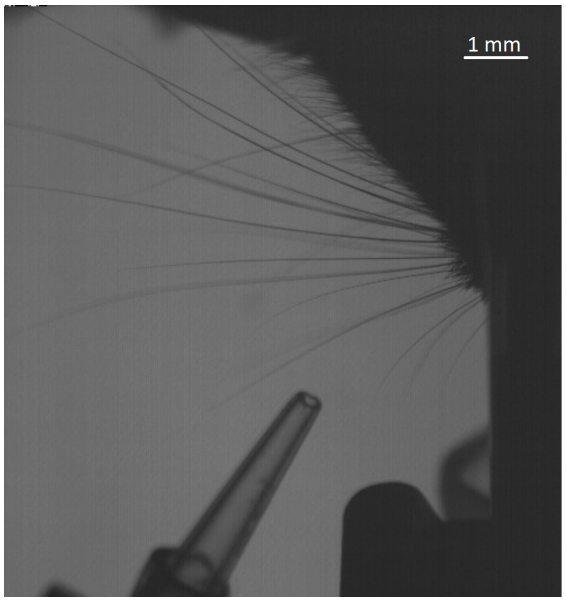


Figure 2.3: Frame from a video segment with added scale bar.

In order to make the whiskers stand out as much as possible, mice of the strain C57BL/6 ('C57 Black 6') are used. These mice have dark brown to black fur, which decreases the amount of noise in the infrared video. The whole animal appears as a black silhouette on the recording and can be filtered out afterwards. [9] The system of vibrissae is kept intact: no trimming is done. As a result, whiskers cross over the course of the experiment. From the top-down point of view of the camera, whiskers are often partially or even fully hidden behind other whiskers.

The experimental setup yields grayscale video fragments of which an example frame is shown in Figure 2.3. The whiskers appear dark against a light background, and the upper row of whiskers, the fur, the snout, and parts of the setup are also clearly visible. A time

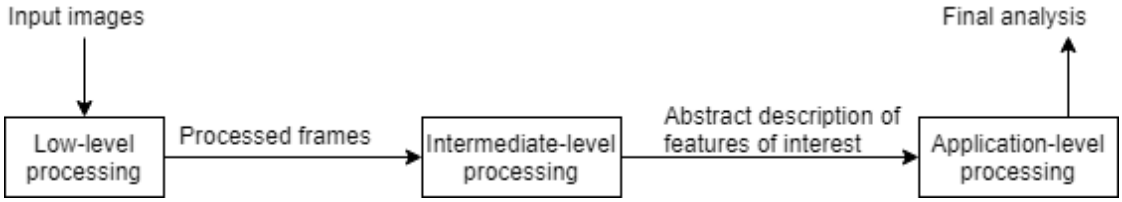


Figure 2.4: Block diagram of a simple computer vision pipeline

stamp is hard-coded in the upper left corner of each frame. The time stamp is used to synchronize the video with the electrophysiological recording of brain activity.

To the researchers at Erasmus MC, the most useful information that could be extracted from the segments is the angle of the whiskers relative to the snout over time. However, there is often some change in position along the snout. As the mystacial pad is not visible on the frames, this position change of the pivot point cannot be seen directly on the video.

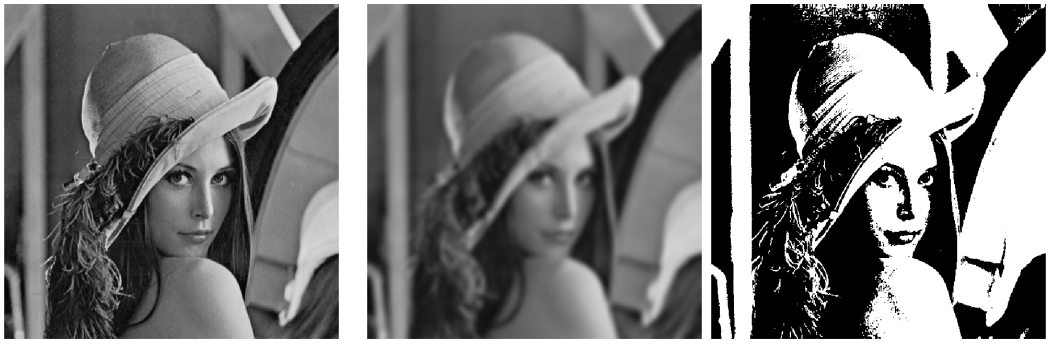
2.2 Computer vision

Computer vision can be described as technology that allows computers to interpret images; that is, to extract useful information from what, from the computer's point of view, is simply a matrix filled with values ranging from 0 (black) to 255 (white).[1] Computer vision, as opposed to computer graphics (the generation of images by a computer), is not a deterministic process: images can be interpreted in a myriad of ways, given the fact that a two-dimensional image is in itself a many-to-one projection of the three-dimensional reality. However, quoting Davies (1997), as a guiding principle, it can be stated that *"if the eye can do it, so can the machine."* [21] Moreover, there are many tasks that modern computers can perform even better than humans, such as face recognition and object classification. Classical computer-vision systems are usually set up as a processing pipeline. This means that the process of interpretation consists of consecutive steps, in which the output of each step serves as input for the next one. Davies (1997) categorizes the many steps of this pipeline into three levels: low-level processing, intermediate-level processing, and application-level processing. A simple pipeline can be seen in Figure 2.4; details for each stage will be discussed in the next sections. In more complex cases, the pipeline is often less straightforward; for instance, it would be possible to create feedback loops between steps of different levels.

2.2.1 Low-level processing

Low-level processing is aimed at making the features of interest more visible, and filtering out components that do not contain useful information (such as noise). Examples of such operations are Gaussian smoothing and thresholding, two of which are shown in Figure

[1] This is the case for gray-scale, two-dimensional images. In the case of two-dimensional color images, there are usually three matrices for each of the three primary colors of light: red, green and blue. In the remainder of this work, gray-scale will be assumed.



(a) Original test image. (b) Test image after Gaussian blur ($\sigma = 4$ px). (c) Test image after thresholding at a value of 100.

Figure 2.5: Examples of low-level image processing operations

2.5. These operations can be performed in parallel, as the gray values of pixels after Gaussian smoothing can be calculated independently from each other. The output of the low-level processing steps is a matrix with values, usually one value per pixel.

2.2.2 Intermediate-level processing

In the intermediate-level processing steps, abstract information is acquired from the images. There are several methods that are widely used to detect basic shapes such as lines and circles and define them in a few parameters: lines can be described as a function of slope and offset, and circles can be defined as a function of the coordinates of the center and the radius.

The Hough transform, invented by Paul Hough in 1962, has become the main algorithm for the detection and parametrization of straight lines in images. The main principle behind the Hough transform is the idea that any point in a two-dimensional image can be defined by the collection of all straight lines that pass though it. The different lines that all intersect at the same point can never be parallel to each other. If we define a line in terms of the parameters ρ and θ, as shown in Figure 2.6, this observation means that, for all these lines, θ maps uniquely to ρ when $-\pi \leq \theta < \pi$. Since a line can pass a point $P_n(x_n, y_n)$ at any angle, θ can take any real value, which means that all the lines that pass through any point $P_n(x_n, y_n)$ can be represented by a curve. It is easy to show that this curve has the equation:

$$\rho = x_n \cos\theta + y_n \sin\theta \tag{2.1}$$

In other words, any point in the (x, y) domain can be represented by a curve in the (θ, ρ) domain (the Hough domain). By mapping every detected point in the Cartesian domain to the Hough domain, it is possible to find lines in the original image by finding places where multiple curves intersect in the Hough domain. The process is shown in Figure 2.7: the presence of two lines in the original image leads to two main points of intersection in the Hough domain, which can be used for the detection and parametrization of the two lines. The Hough transform was later generalized to detect circles and even arbitrary shapes. [21]

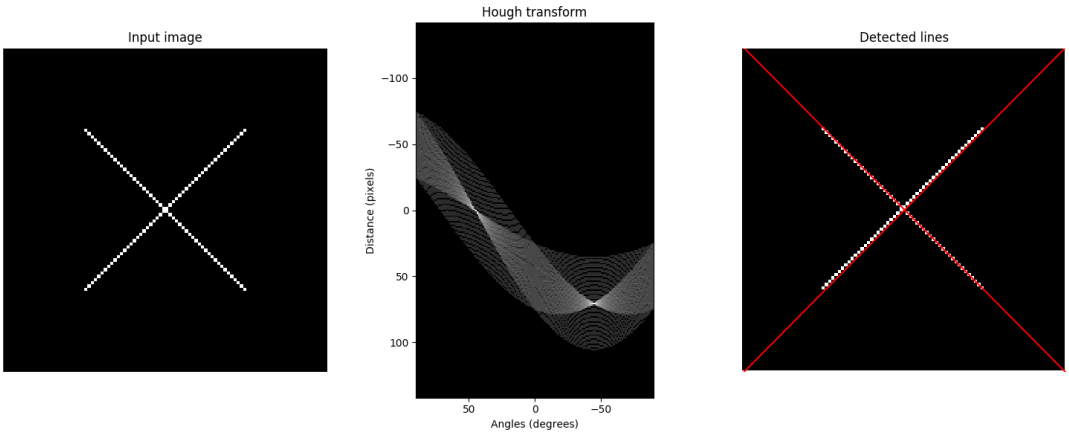


Figure 2.7: Line detection via Hough transform. Image taken from [2].

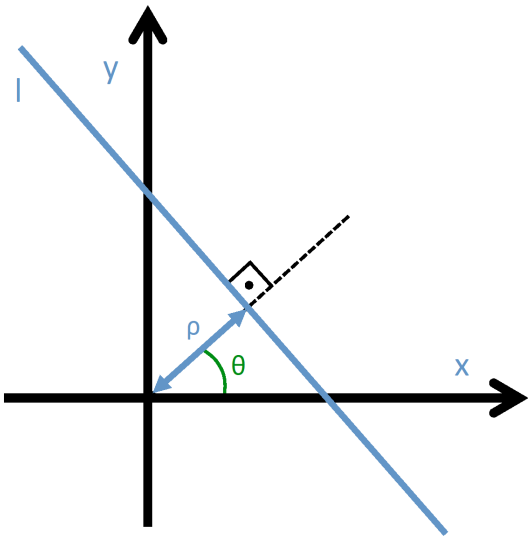


Figure 2.6: Normal parametrization (θ, ρ) of a straight line. Image adapted from [3].

Although the Hough transform is an elegant way to detect lines which is also robust to occlusions, it is not without its drawbacks. One of the drawbacks of the Hough transform is the fact that the algorithm is computationally expensive if the image has a high resolution or simply contains a lot of line points, or if the angle of the line needs to be determined very precisely. Furthermore, if lines in an image are not completely straight, the algorithm might not detect them; on the other hand, when pixels from different parts of the image are 'accidentally' aligned, they might appear as peaks in the Hough domain (after all, the Euclidean distance between points in the (x, y) domain is not visible in the Hough domain). Furthermore, the begin- and endpoints of line segments in the original image cannot be derived from the Hough transform: to determine those points, one has to go back to the original image to obtain the location of the points that were identified as parts of a line [21].

Oftentimes, the objects that need to be detected have less well-defined shapes. In such cases, it is necessary to represent those objects "by a small number of good features" [22]. It is important to choose the right features: for instance, when it comes to whiskers, 'length' is a better choice than 'color'. After all, whiskers can be distinguished from fur by their length, and there is also some difference in length among whiskers. In contrast,

on the videos, all whiskers have approximately the same gray-scale value, as does the common fur.

To successfully recognize, describe and classify the objects of interest, it is important that objects that are similar in their appearance, are also similar in their abstraction. This requirement is known as the *compactness hypothesis* [22]. Following this requirement allows more accurate detection and classification of objects in images, even if the appearance of an object is not the same on all the frames.

2.2.3 Application-level processing

Application-level processing refers to a wide range of additional processing steps that are specific to the application, such as composing an image from multiple frames (in the case of multiple camera's) or tracking objects over multiple frames (in the case of video data). For this work, the application-level step will consist of techniques for distinguishing and tracking whiskers over time.

With the development of machine-learning technologies over the last years, the classical pipeline of processing steps is often replaced by techniques such as that of a Convolutional Neural Network (CNN). With this technique, a deep neural network is trained to interpret information in an image with great accuracy. However, this technique requires the availability of a large amount of labelled images that can serve as training data, and in many cases, this data is simply not available. Therefore, the development of computer-vision systems based on a pipeline of processing steps remain important.

2.3 Machine learning for classification

Machine learning refers to techniques in which computers are able to learn to perform a specific task, without every step being specifically programmed. In some applications, the computer is supplied with a large amount of example inputs and correct outputs ('labelled data'), from which the computer is supposed to learn to give a correct output for an unknown input. This type of machine learning is called *supervised learning*. In other applications, the computer is supplied with large amounts of unlabeled data, from which it is supposed to deduct structures by itself: this type of machine learning is called *unsupervised learning*. An third approach is *semi-supervised learning*, in which a computer is supplied with a small amount of training data, and it uses that to generate more training data by looking at similarities between the labeled data and the unlabeled data.

Classification is a task in which an object or data point is assigned to a category, based on its features. Classification is a task that can be performed using machine learning in either of its three varieties. It is important to choose the right features for classification: as objects are classified based on their similarity to other objects, it is important that the set of features follow the compactness hypothesis.

In unsupervised classification, the algorithm first needs to find the clusters based on the existing data, make a decision about the number of categories based on the clustering, and then assign the data points to those categories. The number of clusters is

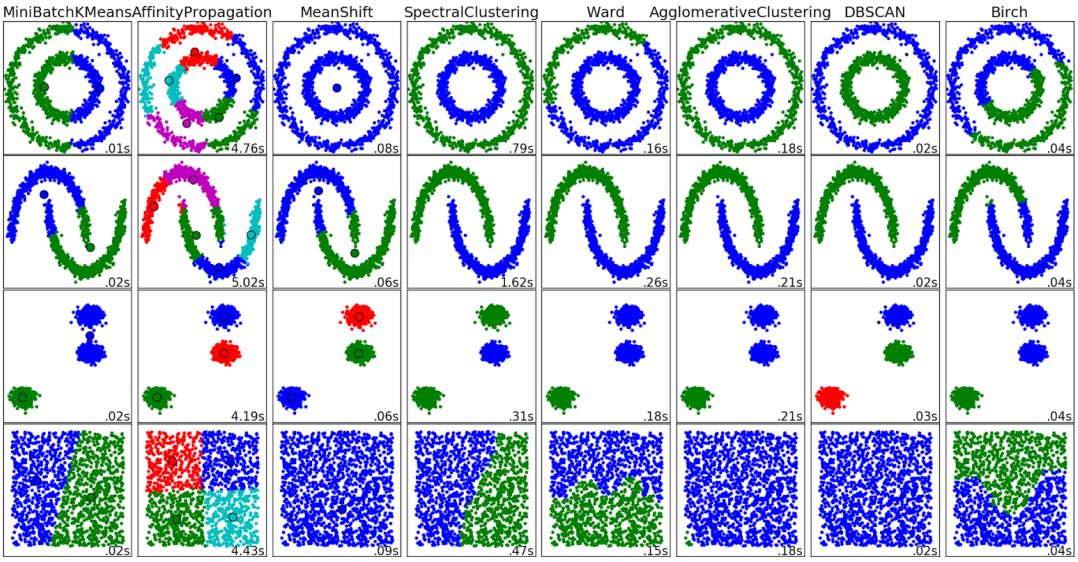
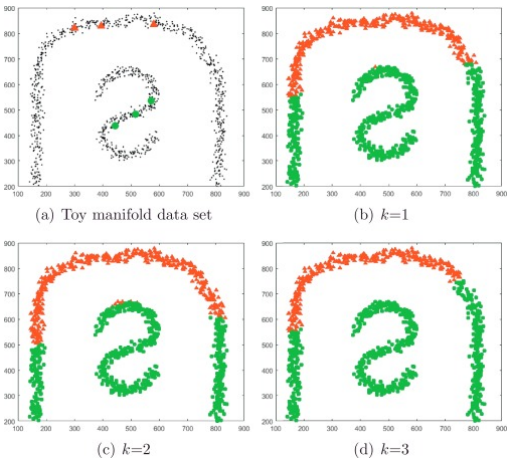


Figure 2.8: Eight widely used clustering algorithms, applied on four different two-dimensional data sets. Different algorithms produce different clusters, but the correct way of clustering is not always obvious. Image taken from [4].

often ambiguous: it depends on the algorithm and its parameters, and there is often no single correct way of clustering. In Figure 2.9, an analysis of four two-dimensional data sets by eight different algorithms. Data points that are assigned to one cluster have the same color in the figure. It can be seen that, if there is no further information about the nature of the data sets, it is difficult to say which algorithm produces the best results for each data set.

In semi-supervised learning, the number of categories is often known from the small set of labeled data, but the way unlabeled data points are assigned to the categories is not predefined by the labeled data. One of the simplest and most widely-used semi-supervised learning algorithms is k-nearest neighbors (k-NN). This algorithm takes the k nearest labeled neighbours of an unlabeled data point (with k a predefined natural number). The unlabeled data point is then assigned to the category that is most common among these neighboring points. The newly labeled data point then becomes part of the training data and is used to label other data points. Hence, the outcome of the algorithm depends not only on the parameters and the data set, but also on the distribution of the labeled data points.

In supervised learning, available training data is used to train a machine that uses a specific learning algorithm, after which the system becomes capable of classifying unlabeled data. Given enough good training data, classification systems trained by a supervised learning algorithm are often able to perform their task with very high accuracy, especially after the development of deep learning systems. In algorithms such as CNN, the machine is often even able to decide by itself which features or a data point or image are of importance for classification, which is a great advantage.



(a) Toy manifold data set (b) k=1

(c) k=2 (d) k=3

Figure 2.9: Semi-supervised k-NN clustering on a data set with different values of k. (a) shows the original data set, (b)-(d) show the results of the algorithm for different values of k. [5]

2.3.1 Support Vector Machines

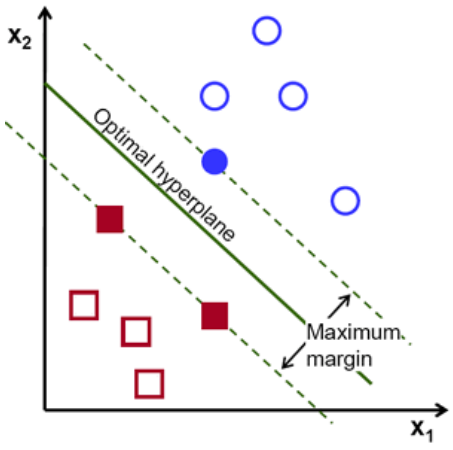


Figure 2.10: Optimal hyperplane in a situation with $k = 2$. Image taken from [6].

The concept of SVMs refers to a type of supervised machine learning that can classify data points based on k parameters, by constructing a $k-1$-dimensional hyperplane in k-dimensional space that serves as a boundary that separates data points that belong to different sets. This hyperplane should be chosen in such a way that points of the same category are on one side of the hyperplane, and points of the other category on the other side. Furthermore, the margin between points of two categories should be as wide as possible. This is achieved by making the hyperplane depend on those data points in the training set that are most difficult to classify. Those data points are called 'support vectors'. The best position of the hyperplane is found when the minimum distance between the plane and the support vectors is maximized. This gives an optimal margin on both sides of the plane. An example of a situation with a set of data points with $k = 2$ is shown in Figure 2.10 [6].

There are different strategies that can be followed to classify data points to more than two categories using SVMs. One of the strategies is the 'one-vs-all' strategy. In this strategy, the number of SVMs that is trained is equal to the number of categories. For each SVM, the training points that belong to the corresponding category are marked as 'members'; the other data points are marked as 'the rest'. The SVMs are then trained

to distinguish between this one category and all the other categories. This technique is useful if it is possible make a linear separation between members and non-members. Another strategy is 'one-vs-one', in which an SVM is trained for every pair of categories in the training set. Training data that does not belong to either of the two categories is not taken into account for that specific SVM. Here, if the number of categories is N, the number of SVMs is $\frac{(N)(N-1)}{2}$. The advantage of this technique is that it is not necessary to have a linear separation between members and non-members, only between every combination of two categories. The disadvantage is that, if the number of categories is higher than 3, this strategy requires more SVMs to be trained than the 'one-vs-all' strategy [23].

2.4 Parallelism and high-speed processing

Around the year 2002, the clock rate of microprocessors stopped increasing, which forced designers to find other ways to increase the performance of processors. They looked at parallelism: rather than trying to make a single core faster, they decided to place multiple cores on a Central Processing Unit (CPU) that could work in parallel. This, however, came with its own challenges: making cores work together requires additional scheduling, communication, synchronization and load balancing. It also forced programmers to design their software in a different way: it became necessary to think about which parts of their algorithms could be run in parallel, and which parts should remain sequential [24].

2.4.1 Amdahl's law, parallelism and pipelining

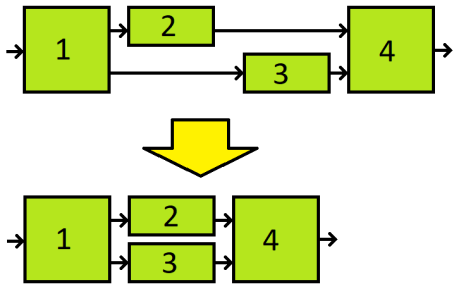
The maximum theoretical speedup that can be achieved by parallelism is limited by Amdahl's law, which can be formulated as

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

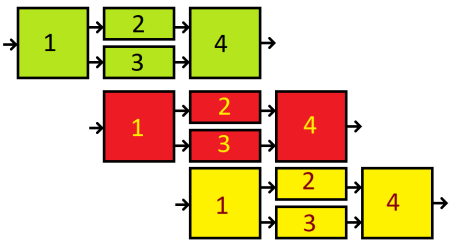
Here, $S_{\text{latency}}(s)$ is the possible speedup of the entire task, p is the fraction of the program that benefits from the increased resources, and s is the possible speedup of that fraction. If s and p are independent, the highest speedup is achieved if both s and p are as high as possible.

There are several ways in which the increase of resources that is brought by the availability of multiple cores can be exploited. One of them is having operations performed in parallel. This, however, is only possible if these operations are independent from each others output. In the field of image processing, many low-level processing operations (including all of the operations shown in Figure 2.5) can be run in parallel on a per-pixel basis, and are therefore suitable for parallel processing.

Another way in which one can benefit from the availability of multiple cores is by making use of pipelining. If a task, consisting of several sequential steps, needs to be performed on every piece of input data, and the maximum duration of every step is predictable, it is possible to increase the throughput of the system, even if the processing time a single piece of input data remains the same. The theoretical increase in throughput



(a) Schematic example of parallelism. If enough resources are available, step 2 and 3 can be performed in parallel, which leads to a speedup



(b) Schematic example of pipelining. By running multiple processes at the same time, input can be processed at a higher frequency.

Figure 2.11: Schematic illustration of parallelism and pipelining.

is then dependent on the duration of the slowest step of the task and the available resources, instead of the duration of the whole task. This idea is especially useful when processing live video streams. A disadvantage of this approach might be the fact that it would require extra data transfers from the memory of one core to that of another, which could contribute to a longer processing time.

2.4.2 Multicore CPUs and GPUs

Whereas a modern multicore CPU contains a relatively small number of cores that are optimized for serial processing, a Graphics Processing Unit (GPU) contains thousands of smaller and more effient cores. GPUs are widely used for tasks in which a high degree of parallelism can be achieved, such as image processing and deep learning [25]. However, GPUs have their drawbacks: GPU cores have a much simpler architecture than CPU cores, which makes them less efficient at sequential tasks. Furthermore, in order to use the GPU, it is necessary to transfer data to the GPU memory before processing, and fetch back the processed data afterwards. This leads to an overhead in processing time. Therefore, the choice between the GPU or the CPU for a certain task usually depends on the nature of the task regarding parallelism and the amount of data that needs to be transferred.

3 Related work

This chapter will give an overview of the existing work regarding whisker tracking. Section 3.1 will give an assessment of the original BWTT, which is the tool that is used at the Erasmus MC. Section 3.2 will describe the adapted version of BWTT which was developed by Yang Ma. Apart from the algorithms used in BWTT, there are several other approaches to whisker tracking. Section 3.3 will assess two works that use spline fitting for whisker tracking. Lastly, Section 3.4 will assess two works in which adaptations were made to the experimental setup.

3.1 The BIOTACT Whisker Tracking Tool (BWTT)

BWTT is a tracking system largely based on the ViSA algorithm, which was published in 2011 by Perkon et al. [7]. The algorithm was built to track whiskers in free-moving rodents, and Perkon et al. tested it on adult male Wistar rats. The algorithm consists of two different modules. The first module was built to track the position and orientation of the head contour. The second module uses that information to track the position of the whiskers.

To detect the snout, the user can initialize the position of the snout by giving the positions of the tip and the middle of the snout on the first frame. Then, the image is converted to a binary image by a thresholding operation, and the information about the position of the snout is used to extract the snout contour. In the remainder of the video segment, the position of the snout is determined by the algorithm by fitting the position of the snout in frame $n-1$ to the binary image of frame n, with the help of a number of 'control points', as can be seen in Figure 3.1. The position of the snout can also be initialized by the ViSA algorithm itself: it picks every 20^{th} frame of the video, scanning the video for moving objects. According to the authors, this works well for free-roaming rats. However, for their experiments with head-fixed mice, the researchers at Erasmus MC still have to specify the position of the snout manually.

To highlight the whiskers and remove background noise, the background is extracted by comparing every pixel of the video for a predefined number of frames, and choosing the pixel with the highest value (i.e. the brightest pixel). In this way, objects that move over the course of the segment are removed, whereas static elements (such as most of the equipment) are removed. The background is then subtracted from every frame, and the whiskers appear bright on a dark background.

ViSA does not extract a whole whisker, but only a short segment that can usually be estimated by a line. Therefore, a narrow band around the snout is defined for whisker detection; all pixels outside that band are discarded. The distance from the snout is chosen in such a way that common fur is not detected, and the width of the band is chosen is such a way that the whisker segments are still approximately linear. These

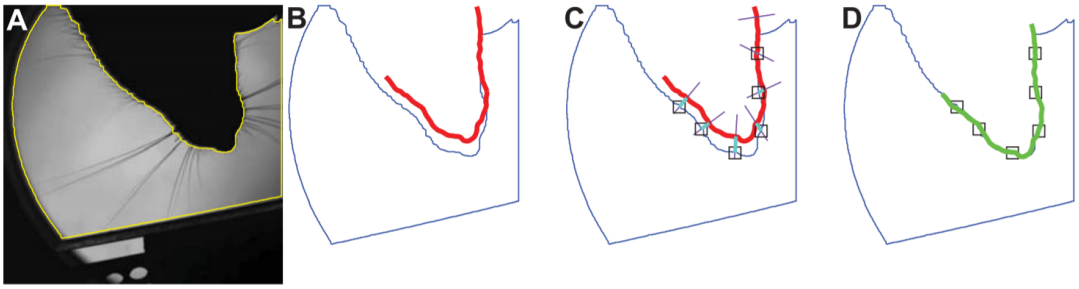


Figure 3.1: Determining the head orientation and position of a rat by fitting a template in ViSA. A: the image is converted to a binary image and the whiskers are removed. B: the contours of the binary image (blue) are compared to the earlier head template (red). C: the perpendicular distances between the contour and the template are detected. D: The template is fitted to the snout contour. Image taken from [7].

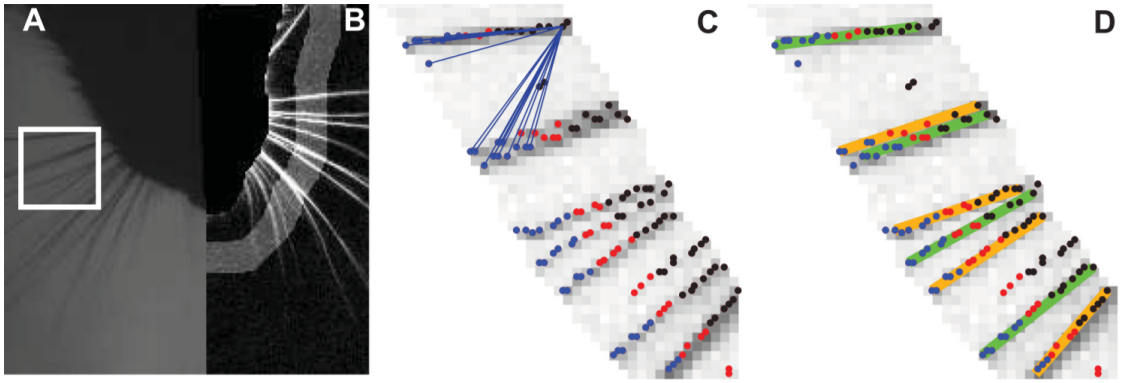


Figure 3.2: Low- and intermediate-level processing steps of ViSA. (A) shows the situation before processing, (B) shows the highlighted whiskers and the narrow band from which the nodes will be extracted. (C) shows the extracted end, center and start nodes, and a constructed 'whisker tree'. (D) shows the extracted whisker shafts. Figure taken from [7].

steps can be seen in (A) and (B) Figure 3.2.

A local-maximum detection algorithm decides which pixels are likely to be centers of whisker shafts, and classifies them as 'start nodes', 'end nodes' and 'central nodes', according to their position in the detection band. Because whiskers often partly overlap, a simple clustering algorithm is not enough to detect all the whiskers. Therefore, the method of 'whisker trees' was developed: every single start node is connected to all the end nodes within a certain distance. Lines are then drawn between the connected nodes. An example of a whisker tree for a single start node can be seen in (C) of Figure 3.2.

Where there is a whisker, the connecting lines between the nodes that belong to that whisker will be collinear, whereas lines between nodes of different whiskers are not collinear. When, for a certain set of line parameters, the number of collinear lines reaches a certain threshold, the algorithms decides that these lines describe a whisker and saves

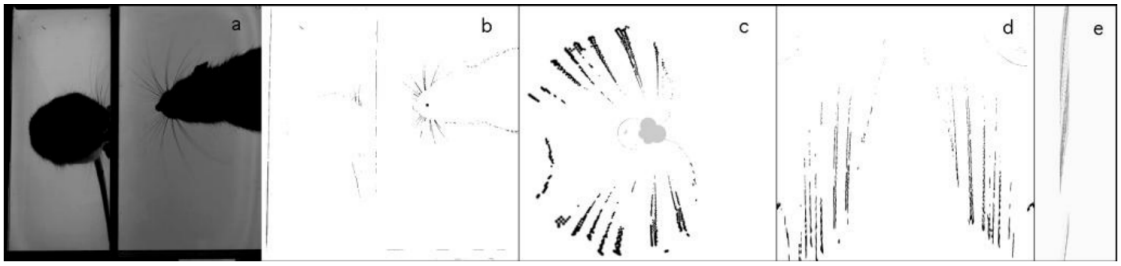


Figure 3.3: Alternative whisker tracking algorithm. (a) Input image. (b) Pre-processed image with the inversion center. (c) Inverted image. (d)Polar-rectangular transform. (e) Part of the Hough accumulator, in which peak detection can be used to detect whiskers. Image taken from [8].

the parameters of these lines; it also assigns all nodes within a certain distance to this whisker. The procedure is then repeated, until all the nodes are either analyzed or assigned to a whisker. This can be seen in (D) of Figure 3.2.

As soon as all the whiskers segments are parametrized, their angle to the snout is calculated. The authors of ViSA are most interested in the average angle of all the whiskers on either side of the rats. The algorithm does not track whiskers from frame to frame, but processes all the frames as if they were independent images. Tracking, therefore, has to happen at a post-processing step. For BWTT, an attempt was made to track single whiskers using a shortest-path algorithm, but according to the author, it was not possible to track more than ten whiskers. Furthermore, he stated that tracking individual whiskers is only possible in trimmed animals: in untrimmed mice, whiskers often overlap and are difficult to detect even for the human eye [26]. The authors of BWTT also implemented a version of an algorithm that makes use of a polar-rectangular transformation, as shown in Figure 3.3 [8]. This algorithm was also cited in the original ViSA paper, but the authors regarded it inferior to ViSA, as applying a Hough transform on a binary image can often lead to inaccurate results [7].

3.2 Yang Ma implementation of ViSA

At the Erasmus MC, the researchers have been routinely using the BWTT algorithm for the analysis of a large number of video segments, featuring head-fixed mice that receive stimuli in the form of air puffs. On the available hardware, the analysis of a test segment of 50.000 frames (i.e. 50 seconds) takes about 80 hours: an average 5.76 seconds per frame, which means that one second of video data took more than 1.5 hour to process. The long processing time greatly restricts scientific research, as every experiment required a computer to run the analysis for several days. In a joint effort with the Delft University of Technology, Yang Ma et al. profiled the algorithm and removed some of the features that were not necessary for whisking experiments in head-fixed mice [20].

First of all, Ma determined that head and snout detection could be taken out of the loop, and only had to be performed on one of the frames, since the position of the head barely changes over the course of the experiment. Furthermore, Ma replaced some of the

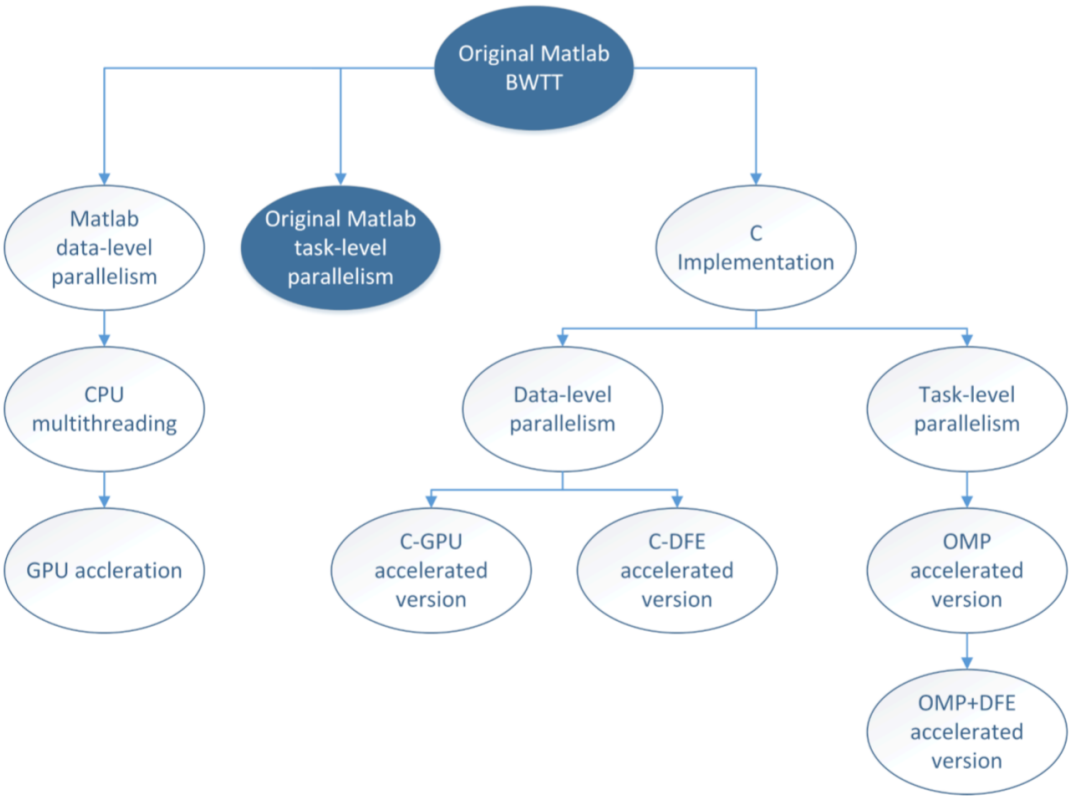


Figure 3.4: Road map of Ma's work. The dark blue circles are supported in BWTT, the other circles are implementations by Ma. Image adapted from [9].

computationally expensive calculation steps of BWTT by more efficient equivalents. For clustering, k-means clustering was explored, but the accuracy was unsatisfactory: only 71.6% of the whiskers that was recognized using the original algorithm was recognized when k-means was used. Ma shortly described k-NN as a promising alternative, but this was not implemented due to time limitations [9, section 3.4.2]

Ma's main contribution is his analysis of possibilities for parallelism in BWTT, ported to a variety of platforms, as summarized in the road map in Figure 3.4. He differentiates between task-level parallelism (which he defines as the simultaneous processing of independent frames or segments), and data-level parallelism (which he defines as the accelerations in bottleneck functions within the system). After introducing data-level parallelism into the original MATLAB version of the ViSA algorithm, Ma ported the whole algorithm to C, and then to a GPU (using CUDA), a multicore processor (using Open Multi Processing (OMP)) and a Maxeler DFE device. He compared the performance of all the versions and his fastest version (the OMP-accelerated one) had the best performance, at an average processing time of 1.21 ms/frame on a device that could handle 16 parallel tasks [9, chapter 6].

Although Ma did not fundamentally change the ViSA algorithm, he showed that it is possible to implement a whisker-tracking algorithm at a high speed. He predicted that

in a device supporting 32 threads (not available at the time), the processing time would be reduced to 0.72 ms/frame, which means that real-time tracking, i.e. the tracking of whiskers during the experiment, becomes possible. Real-time tracking would allow the researchers to create a feedback loop, which would open up a whole new field in neurological research [9, section 1.2].

3.3 Spline-fitting methods

Apart from the algorithms used in BWTT, several other whisker-tracking methods were developed. This section will describe the work by Knutsen et al. (2005) and Clack et al. (2012). These algorithms have in common that they use spline fitting as an intermediate-level processing method to obtain a description of a whisker. The algorithm by Knutsen et al. was tested on trimmed and untrimmed rats, whereas Clack et al. validated their algorithm on videos of trimmed rats and trimmed mice. [11] [10].

Both algorithms perform some low-level processing before fitting splines to the detected whiskers. Knutsen et al. first remove the background noise by subtracting an average of the frames in which the rat was absent (Figure 3.5 shows the original frame with the rat and different objects, Figure 3.5 (B) shows the frame after background subtraction). The algorithm then makes an estimation of the next position, creates a mask (C) and performs a convolution on the image, filtering out some of the whiskers (D). Then, a convolution with spline templates that are based on the predicted position, angle, and bending of the whisker is done, so that only the whiskers of interest are kept (E). Then, the line is localized by looking at a region of interest, which is again determined by the predicted position of the whisker. A cubic polynomial is then fitted to the whisker to achieve parametrization. [10].

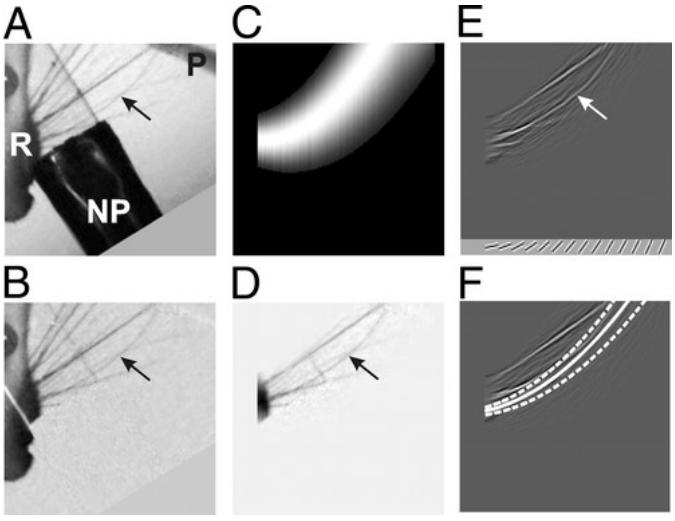
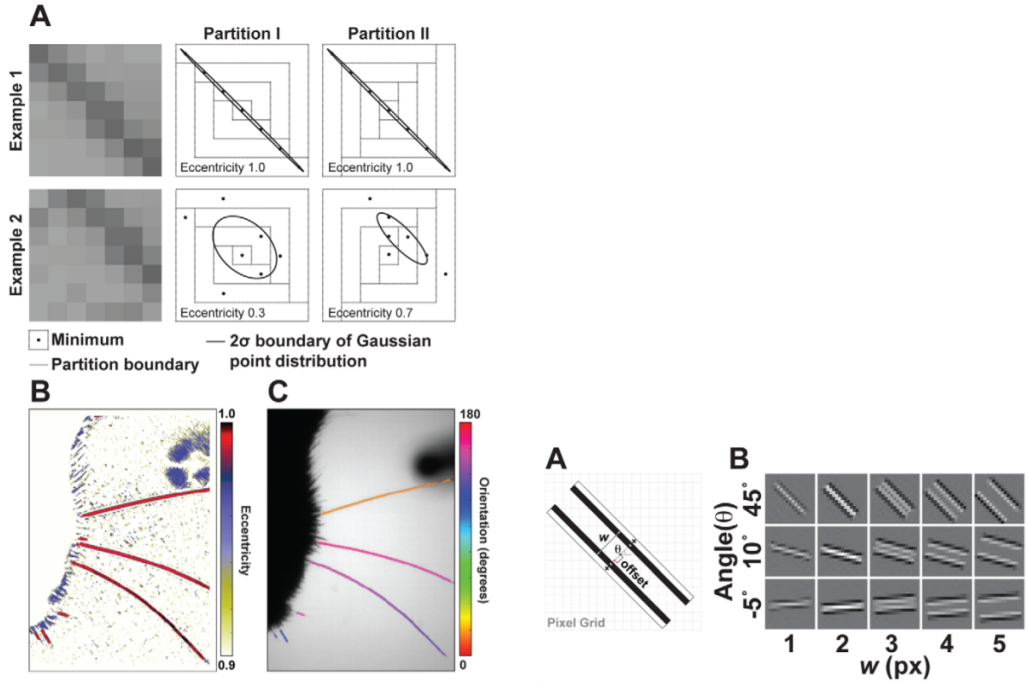


Figure 3.5: Different stages of algorithm by Knutsen et al. Image taken from [10].

Clack et al. (2012) take a different approach to filtering whisker points. They use only limited background removal, and detected whiskers with a local intensity-based algorithm. The algorithm places a 7x7 grid around a pixel of interest, for two different partitions, divided into seven subregions. For every subregion, the darkest pixel is marked with a 1 (i.e. is a potential whisker point), the other pixels are marked as 0. For each partition, the covariance matrix is calculated. Using the two eigenvalues of this covariance matrix, it is possible to calculate the eccentricity (de-



(a) Whisker point detection algorithm. (b) Parametrized line detector.

Figure 3.6: Images taken from [11].

fined as $e = \sqrt{1 - \frac{b^2}{a^2}}$, with a and b being the highest and the lowest eigenvalue) of the collection of marked pixels. If the eccentricity exceeds a certain threshold, the pixel is marked as part of the whisker. The eigenvectors of the covariance matrix, then, can be used to calculate the local angle of the whisker. The process is illustrated in Figure 3.6a [11].

The whisker is traced by fitting predefined templates for a locally linearized whisker over the detected whisker points. These templates are defined at sub-pixel precision and fit over the whisker point in such a way that the Laplacian of the correlation between the model and the actual whisker is minimized (see Figure 3.6b). In this way, Clack et al. claim to achieve sub-pixel precision in tracing the whisker. The whisker is traced from a more or less randomly chosen starting point on the whisker, in the direction that was detected earlier. If the template cannot be fitted over the whisker point in a satisfactory way, the algorithm assumes that there is an occlusion and estimates the position of the whisker, until the whisker becomes visible again. A statistical approach is taken to distinguish the whiskers from fur. No polynomial is fitted to the whiskers. [11]

3.4 Other whisker-tracking methods

The previous sections described a variety of purely visual whisker tracking methods: apart from trimming, no other adaptations are made. However, tracking can be improved by making further adjustments to the whiskers. Venkatraman et al. (2008) were able

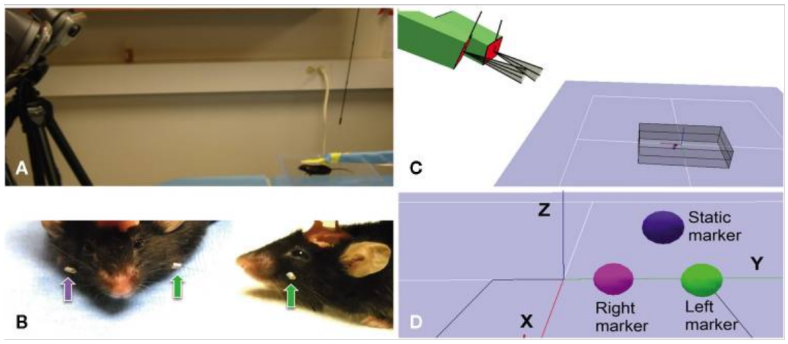


Figure 3.7: Setup for three-dimensional whisker tracking, using markers attached to the whiskers. Image taken from [12].

to track rat whiskers by attaching light, self-adhesive markers to the whiskers. The markers can be distinguished by color and expected position [27]. This allows more precise whisker detection and tracking. However, attaching something to the whiskers changes their dynamics, which is problematic if one wants to study the sensory feedback that the animal receives from an intact whisker system.

Roy et al. (2011) used two cameras for tracking mouse whiskers, which allowed the construction of a three-dimensional description of the state of the whiskers. The researchers used an existing 3D tracking system that uses retro-reflective tape to track objects over time in three-dimensional space. This tape was attached to the head and the whiskers of interest, as show in Figure 3.7 [12].

Although the use of multiple cameras certainly decreases occlusion and could increase the quality of whisker tracking, it complicates the experimental setup and increases the amount of data that needs to be processed. Furthermore, the researchers at Erasmus MC would like to be able to analyze video segments of the experiments that were done in the past. The development of a system that can track individual whiskers in videos generated with the existing setup will be the aim of this work. Nevertheless, one should keep in mind that in two-dimensional videos of untrimmed mice without markers on the whiskers, occlusions are inevitable.

4 Towards a new tracking system

At Erasmus MC, BWTT is used for the detection of whiskers in videos of head-fixed mice. However, no tracking algorithm was implemented in BWTT. Igor Perkon noted that he tried to implement one, but did not finish it [26]. This chapter will give an account of the steps that were made before the decision was made to create a new tracking system, instead of post-processing the output of BWTT. Section 4.1 will describe an existing post-processing algorithm. Section 4.2 will describe some of the attempts that were done to improve the algorithm. In Section 4.3, the decision to design a new tracking system, rather than post-processing the output of BWTT, is described.

4.1 Jochen Spanke's post-processing algorithm

Since the researchers at Erasmus MC want to track individual whiskers instead of an average, a post-processing algorithm was developed and implemented in MATLAB by Jochen Spanke [19].

Spanke's algorithm consists of two steps. The first step takes the output of BWTT as its input. BWTT describes every whisker shaft by two coordinate pairs, i.e. the x- and y-coordinates of the start and end points of the shaft. It also defines the position of the head by giving the coordinates of the tip and the middle of the snout. Spanke's algorithm calculates the angle between every whisker shaft and the snout, and retains an x-coordinate from BWTT. The other coordinates are discarded.

The second step of the algorithm consists of, as Spanke called it, a 'Kalman-like tracking' algorithm. The script assesses all the frames one by one in chronological order, and searches for 'tracks', i.e. collections of detected whisker shafts on different frames that represent the same whisker. The algorithm differentiates between 'valid', 'temporary', 'missed' and 'gathered' tracks. 'Temporary' tracks contain less than five data points. 'Valid' tracks contain five or more data points, including one whisker shaft from the frame that was last analyzed. 'Missed' tracks were not detected in the last frame, but were detected in at least one of the previous T_{missed} frames (in Spanke's original script, T_{missed} is set to 3). If a 'missed' track remains undetected for more than n_{missed} subsequent frames, it is moved to the 'gathered' category. The tracks in this category will not be updated anymore, but are given as output in the end.

Tracks also have a variable named `fit_error`, which can be set in different ways, depending on the type of the track. For temporary tracks, this value is always set to 999, whereas for other tracks, the value is calculated as the sum of squared differences between, on the one hand, the angle of the six last detected whisker shafts of a track, and on the other hand, the second-degree polynomial that best fits those whisker shafts.

At the first frame, naturally, there are no tracks yet. The algorithm then takes all the points from the frame and creates new 'temporary tracks' out of all of them.

For the rest of the video segment, the algorithm iterates through the segment three times: first for the 'valid' tracks, then for the 'missed' tracks, and then for the 'temporary' tracks. The tracks are sorted in ascending order by their value for `fit_error`. For each track, the values for the x-coordinate and the angles are predicted. The whisker shaft that fits the prediction most closely is added to the track, provided that the difference in coordinate and angle between the prediction and the whisker shaft remain within certain tolerance bounds.

After all the tracks are assessed, the values for `fit_error` and `missed_frames` are updated. Some tracks will change categories: temporary tracks that now contain five data points become valid; valid tracks that had no whisker shaft assigned for this frame become 'missed' tracks; missed tracks that had a whisker shaft assigned become valid again, and missed tracks that were missed for more than T_{missed} times become gathered tracks. Any whisker shafts that were not assigned to a track become new temporary tracks. This whole procedure is then repeated for the next frames, until the last frame of the segment is reached. In Algorithm 1, the algorithm is summarized in pseudo-code, and in Figure 4.1, the way tracks change categories is described in a state diagram.

Algorithm 1: Pseudo-code for Jochen Spanke's algorithm. The way the categories are updated, is shown in the state diagram in Figure 4.1.

```
foreach frame do
    for tracks ← [valid tracks, missed tracks, temporary tracks] do
        while tracks available do
            track ← track with the lowest value for fit_error;
            make prediction for x-coordinate and angle;
            choose best-fitting whisker shaft;
            if fit error between prediction and whisker shaft ≤ tolerance bounds then
                add whisker shaft to track;
                update track.fit_error;
                track.missed_frames ← 0;
            end
        end
    end
    foreach track that was not updated do
        update fit_error;
        missed_frames ← missed_frames+1;
        update category;
    end
    if whisker shafts left unassigned then
        foreach unassigned whisker shaft do
            create new temporary track;
        end
    end
end
```

4.2 The need for new post-processing algorithm

As a first possibility for improvement, the replacement of the Kalman-like prediction-and-fitting algorithm was explored. Although a Kalman filter is often used for tracking

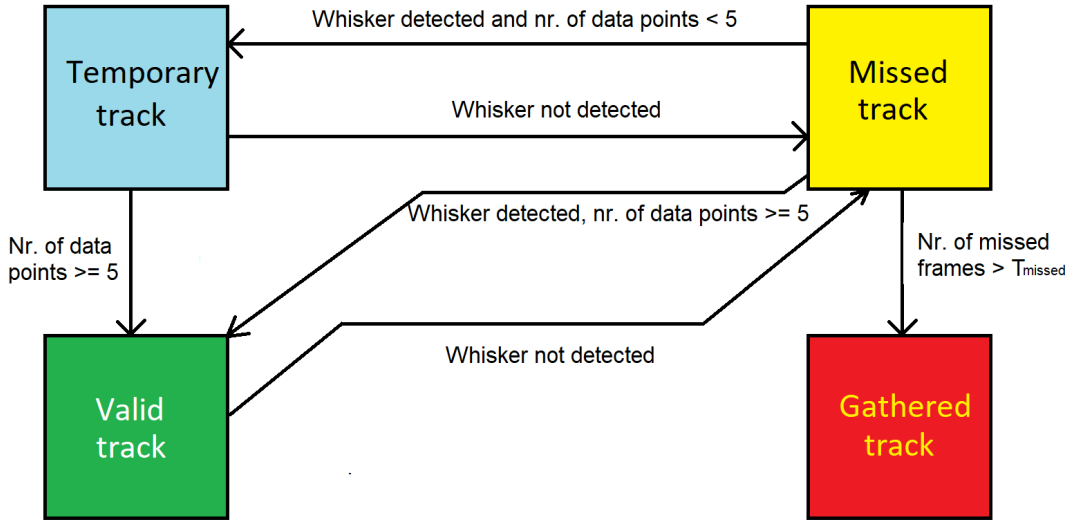


Figure 4.1: State diagram, in which the state represents the categorization of the tracks in Spanke's post-processing algorithm.

and classification (e.g. [28] and [29]), its implementation in post-processing the results of ViSA is not without problems. The algorithm predicts the position of a whisker based on a second-degree polynomial fit. However, since whiskers move in ways that are difficult to predict accurately even by humans, a polynomial fit is not guaranteed to give good results. Furthermore, by retaining only one x-coordinate and discarding the three other coordinates (the y-coordinate and the x- and y-coordinates of the other point), Spanke discarded some of the information about the position of the whisker. If, apart from the x-coordinate, also the corresponding y-coordinate were retained, the position of the whisker shaft would be fully defined, but with only the x-coordinate, this is not the case (even though it is likely that the detected position of the shaft can usually be estimated well, due to the fact that BWTT only looks at a narrow band around the snout).

The problem of matching whiskers in frame n to those in frame $n+1$ was approached by treating it as a variety of the Travelling Salesman Problem (TSP). The original TSP concerns finding the shortest route that visits all cities and returns to the city the route started from (with the distance between all cities known). For whisker tracking, the problem consists of matching each whisker in n with a whisker in $n+1$ (which, apart from a temporal distance of 1 ms, are separated by a spatial distance defined by a difference in angle and a difference in position) in such a way that the sum of all the distances between each member of a pair of whiskers is minimized. For this approach, we mainly relied on the angle of the whisker (as calculated in step 1 of Spanke's algorithm), but we also incorporated as parameters the coordinates that we originally discarded into the algorithm, giving each parameter a certain weight. The primary assumption behind this approach is that the movement of whiskers in 1 ms is small enough to identify whiskers from frame n in frame $n+1$.

In reality, this turned out to be more complicated. Whiskers often move close to each other, cross, or hide behind each other, which makes distance-based identification unreliable. During different parts of the whisking process, ViSA might detect different whiskers and miss others. This does not only affect individual whisker detection, but even makes the average of the angle of the whiskers unreliable. Apart from these limitations in accuracy, the practical implementation of a TSP-based approach is difficult. The TSP is an NP-complete problem, and assuming a maximum of 20 whiskers per frame, there are 20! (about 2.43 quintillion, or 2.43×10^{18}) different ways of connecting the whiskers, and even though there is a possibility for parallelism, it is impossible to analyze all the possibilities. Even after filtering out unfeasible connections on beforehand, the computational costs remain huge. Another possibility would be supervised machine learning. As stated in Chapter 2, neural networks are widely used for classification. However, this approach would require us to produce a training set, and since the existing post-processing algorithm makes mistakes, it would be necessary to manually create a training set. A GUI was created in MATLAB, which would allow users to manually create tracks by connecting data points in a plot of the frame numbers against the angle in degrees (as shown in Figure 4.2). However, the creation of a training set would require manual annotation of a number of video segments for different mice. This would take more time than was available for this project, so it was decided to take a different approach.

4.3 From post-processing to redesigning

While exploring the options for post-processing the output, the question arose whether the creation of a reliable training set was even possible. Even when the whiskers move slowly, inter-frame whisker identity is sometimes difficult to derive. In Figure 4.2 there are different clusters of whiskers which seem to move in the same direction, but it is impossible to determine where the whiskers are crossing. Even when incorporating other coordinates, we could not always reliably identify whiskers over time. Another problem with BWTT is that it linearizes a small section of a whisker at a certain distance from the snout, and uses that to estimate the angle to the snout. However, many of the whiskers are bent, and is is questionable whether the chosen section of the whisker leads to a good estimation of the angle of the whisker to the snout.

We contacted the designers of ViSA. Mr. I. Perkon let us know that tracking whiskers reliably over time in untrimmed mouse is, in fact, impossible with ViSA: he had attempted to implement a tracking algorithm, which was able to track 10 whiskers in trimmed mice, but was unsuccessful in untrimmed mice [26].

Therefore, we decided that, instead of trying to post-process the results of ViSA, it was necessary to redesign parts of the ViSA algorithm, so that the data that is required to identify whiskers over time is retained, and design a system that can reliably track whiskers over time. This became, then, the main design goal of this project.

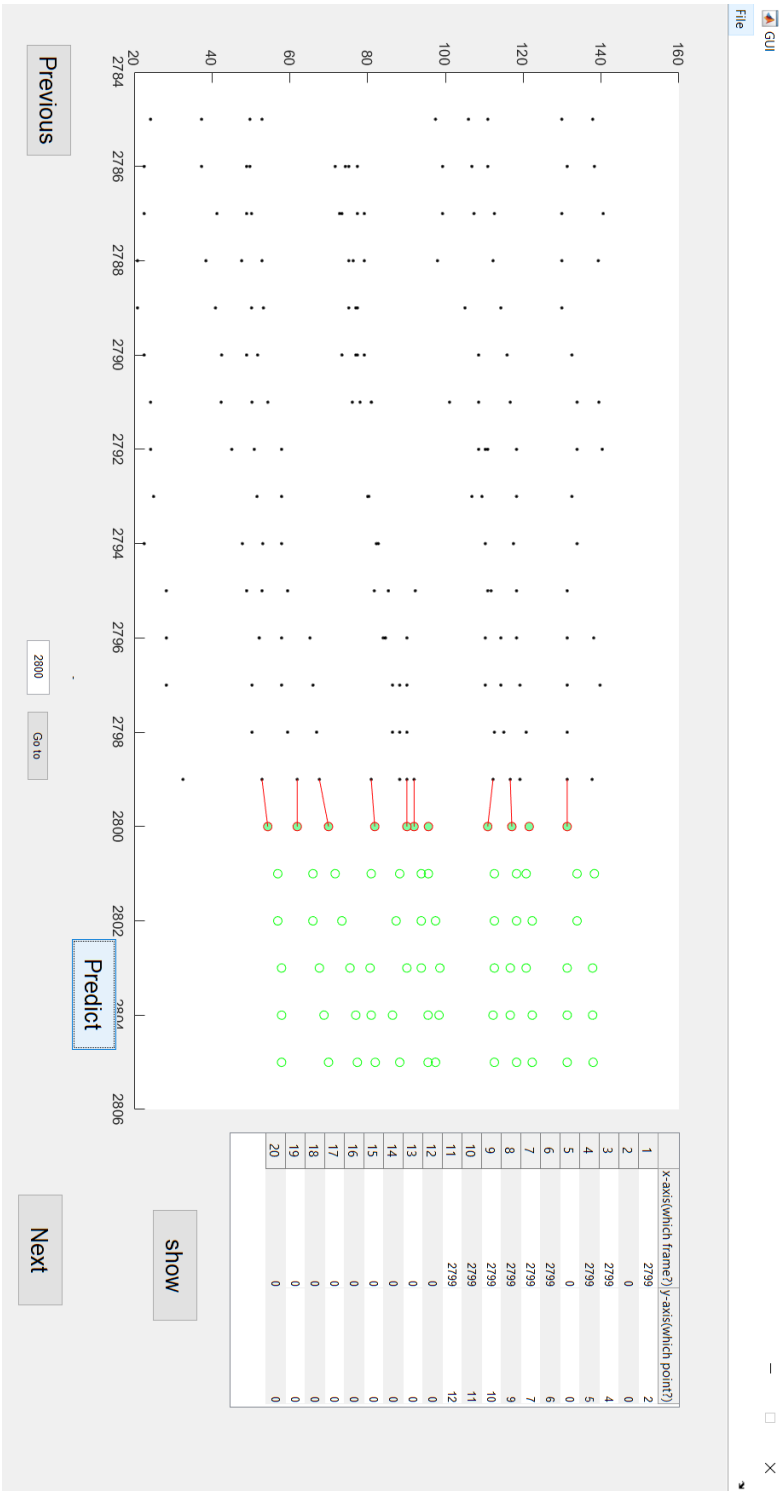


Figure 4.2: GUI for manual creation of tracks. The system is able to make a prediction based on a closest-neighbor approach, but the user is able to change the connections. In this image, one nearest-neighbour prediction is made at frame 2799.

5 Low-level processing

This chapter describes the different low-level processing steps that were explored for the construction of a new whisker-tracking system. The purpose of low-level processing in the context of whisker detection is twofold: in the first place, it is necessary to determine for each pixel in the frame whether it belongs to a whisker or not. Then, if it is decided that a pixel belongs to a whisker, it is desirable to extract as much useful information from it as possible: for example, information about the sub-pixel position of the centerline, information about the local direction of the whisker, or information that tells which whisker points are likely to belong to one and the same whisker.

For our purposes, it is sensible to distinguish two types of low-level processing:

- The actual whisker-point detection and analysis. This step decides which pixels contain data about the whiskers and extract as much useful information from those pixels as possible.

- low-level processing steps in which no whiskers are detected yet, but are aimed at making the actual whisker detection easier and more accurate. These steps include the removal of the fur, the animal, and the background, so that only the whiskers remain, but also Gaussian blurring and gamma correction. These steps can be called *preprocessing*.

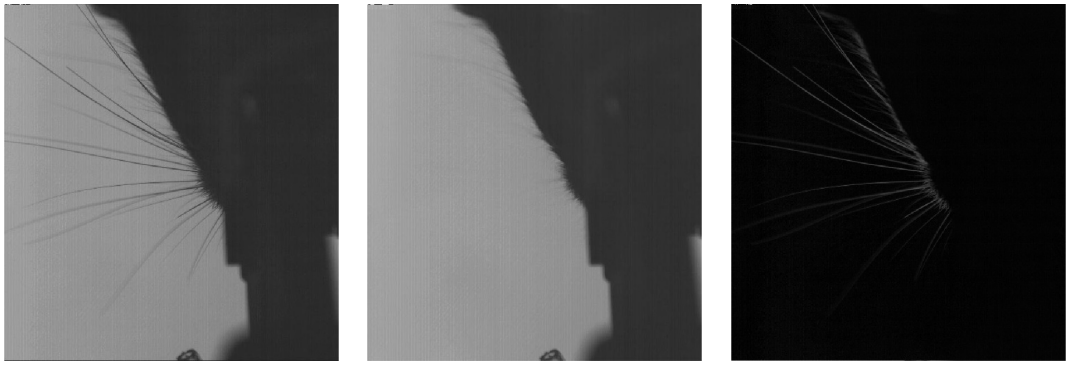
Section 5.1 describes the preprocessing algorithms, some of which were adopted from BWTT. Section 5.2 describes several algorithms that were considered for whisker-point detection: skeletonization, maximum-intensity-local-eccentricity-based whisker detection, and curvilinear structure detection. Curvilinear structure detection was chosen for implementation in FWTS.

5.1 Preprocessing

This section will describe several preprocessing algorithms. Some of them were inherited from BWTT, others were added later. Section 5.1.1 describes the preprocessing steps that BWTT uses. Section 5.1.2 describes some preprocessing steps that were not present in BWTT, but were added to the new whisker-detection system.

5.1.1 BWTT preprocessing steps

The background-removal step of BWTT takes a number of frames from the video, and compares them pixel by pixel. In BWTT, a number of 60 frames is hard-coded for 'normal' mode. There also the option to run the program in 'developer mode'; then only 5 frames are taken. The authors assume that none of the whiskers remains at its



(a) Frame 1 of a test segment.

(b) Background, extracted from the test segment. The image is constructed by taking the maximum value for each pixel over 60 frames.

(c) Background with frame subtracted. The whiskers appear bright against a dark background.

Figure 5.1: Frame of a test segment, its extracted background, and the frame with the background subtracted

place over the course of the video. Therefore, each pixel that is occasionally covered by a whisker, will show the background in at least one frame. Because the background is lighter than the mouse and its whiskers, the frame in which the pixel has the highest value, is likely to be the frame in which this pixel shows the background. Figure 5.1a shows the first frame of a test segment; Figure 5.1b shows the background that has been extracted from the test fragment. The long whiskers have disappeared in the extracted background. The mouse, the equipment and most of the fur are still visible. After the background is detected, the frame is subtracted from the background, which leads to an image in which the whiskers appear bright on a dark background, as shown in Figure 5.1c.

The frames BWTT takes for background extraction are spread linearly over the whole video. These frames are grabbed from the video in the very beginning of the process of analysis, after which the frames are analyzed independently. This works well in an off-line situation (where the experiments are done first, after which the videos are analyzed), but could pose difficulties in an on-line situation (where each frame is analyzed as soon as it is available). This could be solved by requiring a short 'calibration time' in an on-line implementation, in which the algorithm gathers a number of frames, extract the background, and only then starts detecting and tracking the whiskers. However, since this project focuses on off-line tracking only, the implementation of this solution lies beyond the scope of this work.

The BWTT snout-detection-and-removal algorithm was also mostly retained from BWTT. This algorithm consists of a series of low-level image-processing operations: dilation (implemented with the MATLAB function `imdilate`) and erosion (implemented with the MATLAB function `imerode`. For dilation, first a neighborhood is defined; in our case, this is defined by a binary matrix that is very small compared to the image (e.g. a 7x7 matrix). The matrix is centered around every pixel of the image. The value of

the output for that pixel is equal to the value of the brightest pixel of the neighborhood in the original image, i.e. the highest value in the neighborhood. Erosion is similar to dilation, with the difference that not the value of the brightest, but the value of the darkest pixel (i.e. the lowest value) of the neighborhood is taken. This operation can be performed on black-and-white, as well as on gray-scale images.

BWTT first thresholds the image with a predefined threshold `bwThreshold`, then dilates the image with a disk-shaped 5x5 neighborhood, then erodes the output with a 7x7 disk-shaped neighborhood, and then dilates the output again with a 15x15 disk-shaped neighborhood. This step was only mentioned briefly in [7], so it is not entirely clear why these particular steps and parameters were chosen. However, the procedure they implemented works well on our test segments: the dilation steps remove the whiskers, whereas the erosion steps keep the snout intact and remove some of the fur. It can be seen that the whiskers have disappeared, whereas the snout contour is largely intact. The output of these four steps can be seen in Figure 5.2. By performing another erosion operaton with a disk-shaped neighborhood of a certain diameter (which can be set up by the user) and multiplying each pixel of the original frame by the inverse of the dilated snout contour, it is possible to remove much of the mouse's fur, while retaining the longer whiskers. The part of this algorithm that was not retained, was the use of `imdilate` to remove the whole image apart from a small band around the snout: after all, we concluded that detecting only a small part of the whisker was not enough to reliably track whiskers over time.

(a) Frame 1 after thresholding

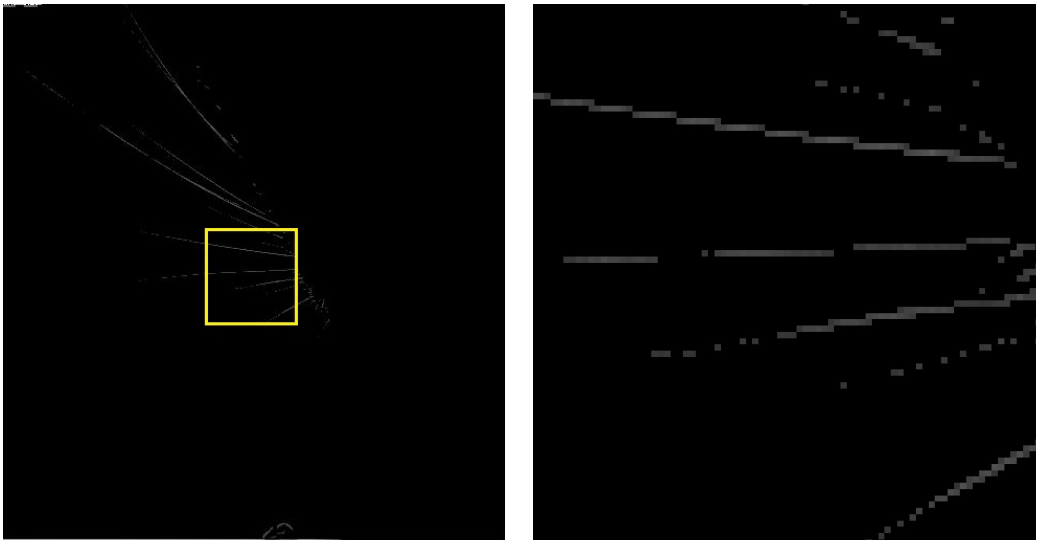
(b) Figure 5.2a after dilation

(c) Figure 5.2b after erosion

(d) Figure 5.2c after dilation

Figure 5.2: First four steps of the snout detection algorithm

Another pre-processing operation that was retained, is contrast adjustment, implemented using the MATLAB function `imadjust`. This function takes as an input (apart from the image itself) an input range `[low_in high_in]`, an output range `[low_out high_out]` and a 'gamma' value. Then, the gray values of the image are changed in such a way, that the input range is mapped to the output range. BWTT takes the output range `[0 1]` (which represents the full range from black to white). The input range can be decided by the user. The gamma value determines the way the input range is mapped to the output range. A gamma value of 1 represents a linear mapping; a value of less than



(a) Frame 1, noise filtering step of the whisker pixel detection algorithm of BWTT

(b) Detail from Figure 5.3a. It can be seen that some of the whisker pixels are lost, and some parts of the same whisker are no longer connected.

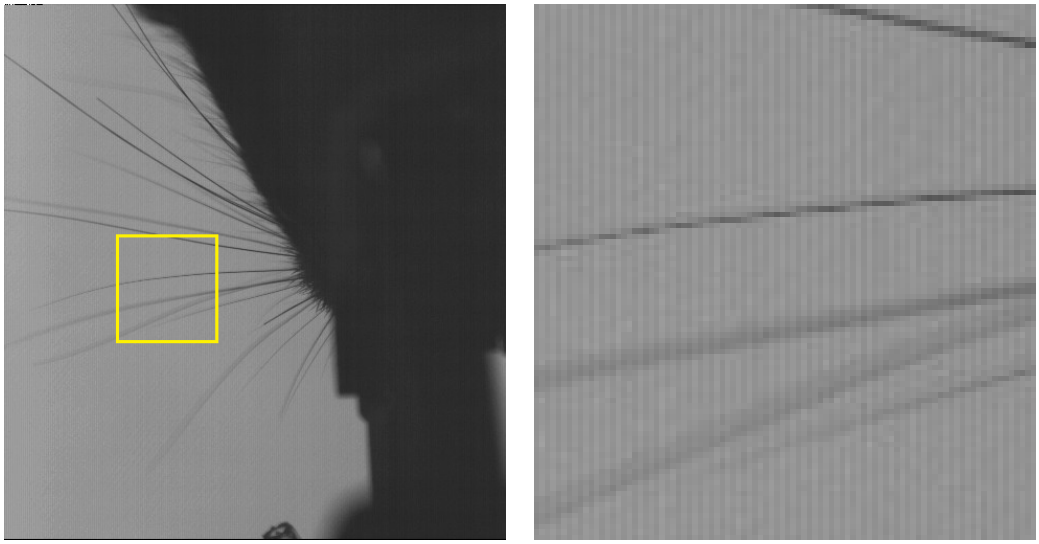
Figure 5.3: Frame of a test segment, after noise filtering in BWTT. It was decided to remove this step.

1 gives more weight to lower (darker) values, and a value of more than 1 gives more weight to higher (lighter) values) [30]. The value of gamma, too, can be decided by the user.

The whisker-pixel detection algorithm from BWTT, which consists of a noise-filtering step and a detection step, was not retained. The noise-filtering step, which is a prerequisite for the accurate detection of whisker shafts in BWTT, removes part of the whiskers as well, as can be seen in Figure 5.4. In the context of BWTT, this is an advantage rather than a disadvantage. After all, whiskers only need to be detected locally, and the BWTT parametrization algorithm is not dependent on whether or not pixels that belong to the same whisker are connected in the original frame: it only detects the center points and connects them later, and if there would be more center points than strictly necessary, this would increase the computational costs of the algorithm. However, if we want to detect and describe the whole whisker accurately, it is better, at this point, to retain as much information about the whisker as possible.

5.1.2 Additional pre-processing steps

Due to some peculiarities of the input data, it was necessary to add some additional steps to prepare the image for the whisker-detection step. These additional steps are all optional: if desired, the user is able to adjust the parameters in such a way that they have no effect on the video data. However, it was found that, for the available video data, these steps can make the whisker detection and parametrization significantly more



(a) Frame 1 (original)

(b) Detail of Figure 5.4a. The vertical, pixel-wide lines are clearly visible.

Figure 5.4: Frame of a test segment, which shows the interlacing effect that is visible on all frames

accurate.

The first two steps are deinterlacing and Gaussian blur. When analyzing the video data at Erasmus MC, it was found that all the videos show an effect that is similar to interlacing. We later found out that this phenomenon is caused by the video compression algorithm that is used; the algorithm combines four pixels into one, which produces the lines as an artifact. This phenomenon could make whisker detection less accurate (e.g. the vertical scan lines could be mistaken for pixel-wide whiskers, or the detection of the local direction of the whiskers could become less accurate). This issue can most likely be avoided in future experiments, but for the existing videos, the only possible option is to diminish the effect.

Two ways to diminish the effect of interlacing were determined: the MATLAB `vision.Deinterlacer` system object, and Gaussian blur. `vision.Deinterlacer` is the MATLAB built-in deinterlacing tool, and it is able to perform deinterlacing using three algorithms. As can be seen in Figure 5.5, the three algorithms all do a decent job in reducing the effect: there is not much difference in the results. However, an argument against the use of deinterlacing algorithms in this case is the fact that all three algorithms work by replacing half of pixels in the video by a function of the neighboring pixels; in other words, half of the pixels are discarded. Since we want to minimize the loss of information in the frame, it might be better to implement a solution that does not discard half of the pixels.

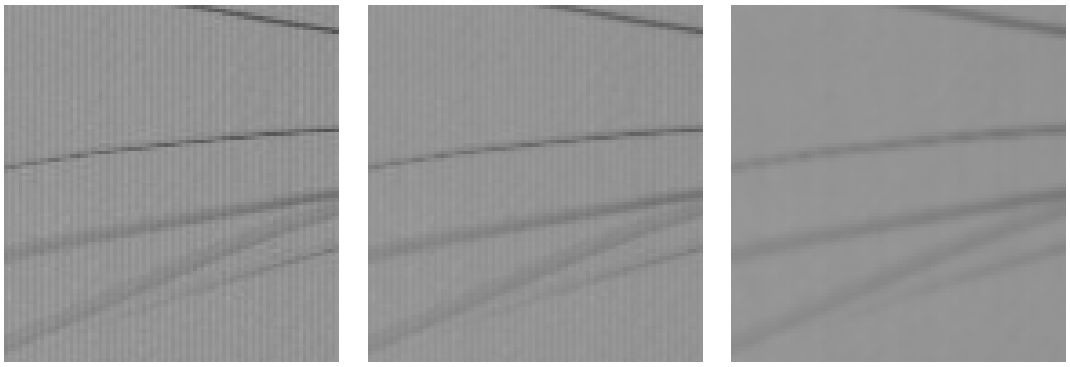
Another solution that could mitigate the interlacing effect is a simple Gaussian blur. A Gaussian blur is obtained by convolving the image with the two-dimensional Gaussian function for a certain value of the standard deviation σ. This can be implemented with the MATLAB function `imgaussfilt`. Figure 5.6 shows the effect of such a blur on

(a) After deinterlacing using the 'line repetition' algorithm.

(b) After deinterlacing using the 'linear interpolation' algorithm.

(c) After deinterlacing using the 'vertical temporal median filtering' algorithm.

Figure 5.5: Detail from Figure 5.8b after deinterlacing



(a) $\sigma = 0.25$ (b) $\sigma = 0.5$ (c) $\sigma = 1$

Figure 5.6: Detail from Figure 5.8b after Gaussian blur operations

the image for three different values of σ. It can be seen that the interlacing effect is diminished by the Gaussian blur. However, when σ is increased, the quality of the image noticeably deteriorates due to the blurring.

The choice between a Gaussian blur and a deinterlacing algorithm eventually depends on the robustness of the rest of the system against this effect, and the degree to which the effect is present in the video segment. If the detection system can handle the effect relatively well or the effect is relatively weak, a Gaussian blur with a small value for σ might be the best choice. However, if it is necessary to remove this effect completely, it could be better to use a deinterlacing algorithm. Eventually, it was decided to give the user the option to choose whether or not to use these preprocessing steps.

Since it was decided to detect whiskers in the whole image instead of a small band around the snout, there is a higher risk that certain non-whisker elements are detected as whiskers. Most of the fur consists of short hairs, but at times, there are long hairs that are not removed by the preprocessing steps of BWTT. Another example of 'noise' is the timestamp that is hardcoded into the upper left corner of each frame. Because of this,

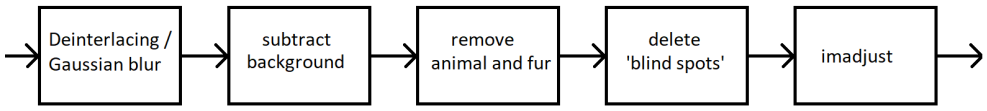
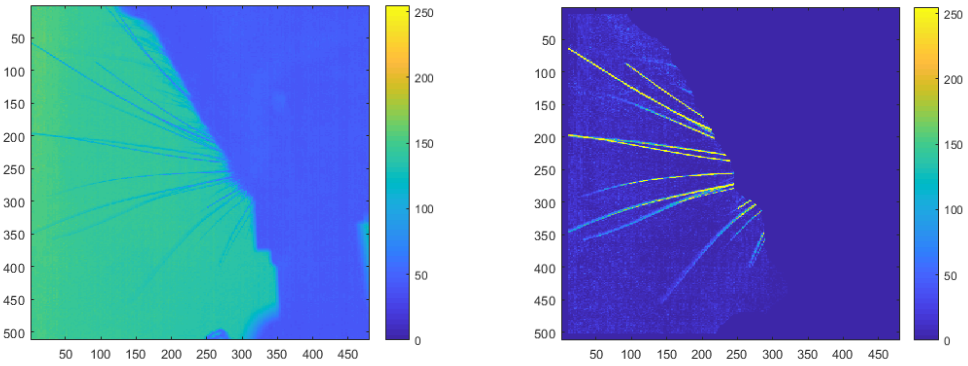


Figure 5.7: The preprocessing pipeline



(a) Frame from the test video, original in scaled colors.

(b) Frame from the test video after preprocessing, in scaled colors.

Figure 5.8: The effect of the preprocessing steps. Instead of the gray-scale values, scaled colors are used.

a step was added to remove the outer n_{edge} pixels from the outer edge of all the frames; the parameter can be chosen by the user. Furthermore, it is possible to define *blind spots* on the frame, by defining a number of rectangular regions in which all the pixels are set to 0. The disadvantage of defining such blind spots is that any part of the whisker moves into that region, it will also remain undetected. Nevertheless, for situations where there is a small area with constant noise that is not removed by the other preprocessing steps, this option can be useful.

Figure 5.7 shows the preprocessing steps in the order they are applied. Figure 5.8 shows the effect of preprocessing in scaled colors (i.e. using the full range of colors in the colormap, as indicated in [31]). The image shows that the whiskers have become much more prominent, whereas the fur, the mouse and the background have been removed. After the preprocessing, the system will detect the whisker points. This will be the topic of the next section.

5.2 Whisker point detection and analysis

The ViSA algorithm selects the locally brightest pixels from the pre-processed images in a narrow band around the snout, classifies them and connects them with straight lines. However, it is difficult to use the original ViSA algorithm for full-whisker tracking, which we would like to do in FWTS: not only has the number of nodes become much higher,

but a straight line is no longer sufficient to estimate the position of the whisker.

In his thesis, Ma shortly described two possible alternatives to the original whisker detecting algorithm: K-means and k-NN. He tested different implementations of K-means, but concluded that this was inaccurate [9]. Ma estimated that k-NN could be a solution that is less computationally expensive than the algorithm in ViSA, but this was not implemented due to time limitations. Ma also expected problems regarding the accuracy of k-NN, since the whiskers move together so closely, that it is often difficult to determine to which whisker a node belongs purely based on the neighboring clusters. Another difficulty with k-NN is the fact that it requires the number of clusters (in this case, whisker shafts) to be determined beforehand: this requires either additional manual setup or another algorithm that can determine the number of shafts. When tracking complete whiskers, k-NN is likely to make errors due to the fact that whiskers cross: after all, at a crossing point, it is difficult to determine to which of the two crossing whiskers an individual node belongs. Therefore, k-NN was not considered as a possible solution in this work.

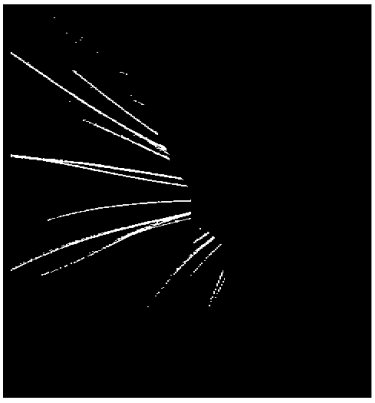
For the detection of whisker points, three algorithms were implemented and tested in MATLAB. The goal of these algorithms is the transformation of a frame from a grayscale image into a binary representation that distinguishes pixels that show a whisker point from those that do not, and also the extraction of as much useful information as possible from those whisker pixels, for example local direction and sub-pixel location.

Section 5.2.1 describes the 'thresholding and skeletonization' algorithm, which was implemented and tested, but not incorporated in FWTS. Section 5.2.2 will describe the detection algorithm developed by Clack et al. [11]. This algorithm was also tested, but not incorporated in the tracking system. Section 5.2.3 describes a curvilinear structure detection algorithm, developed by Steger [13]. This algorithm became part of FWTS.

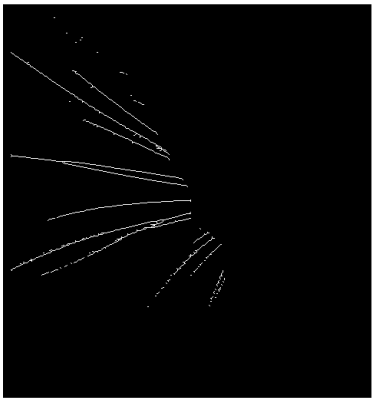
5.2.1 Skeletonization

The first algorithm that was implemented and explored, is the skeletonization algorithm. This algorithm consists of two steps. The first one is binarization based on simple thresholding: pixels with an intensity higher than the threshold are assigned a one, the other pixels are assigned a zero. The threshold should be chosen in such a way that separate whiskers can still be distinguished and pixels of the same whisker are connected. Figure 5.9a shows the outcome of such an operation on a frame from a test video when a threshold of 70 is chosen; this means that all pixels with a grayscale value of 70 or higher will be assigned a 1, and all pixels with a grayscale value of lower than 70 will be assigned a 0. The optimal threshold depends on the parameters chosen in the preprocessing stage. For a threshold of 70, most whiskers are represented well, although some of them appear shorter, and some short whiskers have disappeared. The situation where three whiskers are crossing, which is difficult to see on the original picture even by the human eye, leads to some cluttering, and it is unclear to which whisker some pixels belong.

The second step is thinning. Most whiskers are thicker than one pixel, and to be able to trace them properly, it is necessary to reduce the whiskers to a single array of eight-connected whisker pixels, i.e. pixels that are horizontally, vertically or diagonally connected. The thinning algorithm is a widely used low-level image processing operation,



(a) Preprocessed image after thesholding at $t = 70$



(b) Frame after skeletonization.

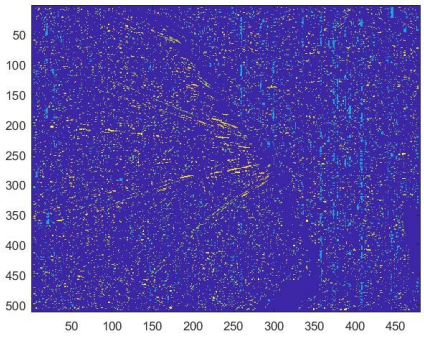
Figure 5.9: The effects of thresholding and thinning on the frame shown in Figure 5.8a.

and is described in detail in Davies (1997) [21]. Figure 5.9b shows the result of the thinning algorithm. It can be seen that in general, it is easy to distinguish individual whiskers, but pixels of crossing whiskers are connected, which means that it is not possible to distinguish whiskers purely by clustering based on connectivity.

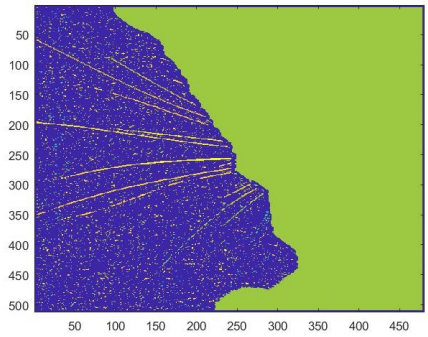
Another difficulty that becomes clear after thinning, is the fact that, when whiskers come close together, the algorithm tends to create circles of 8-connected pixels. A possible solution is to remove those pixels that are connected to 3 or more pixels, but this will also separate pixels that belong to the same whisker. Furthermore, it should be noted that, although whisker pixels may be marked for deletion in parallel, it is necessary to pass over the image several times to complete the thinning process; as Davies stated, thinning is "essentially sequential", and it is difficult to estimate the number of passes that is necessary, even though, for thin structures such as whiskers, the number of passes is expected to be relatively low.

5.2.2 Maximum-intensity local eccentricity-based whisker detection

The second whisker point detection algorithm that was implemented and tested, was the local eccentricity-based algorithm by Clack et al. [11]. This algorithm fits a 7x7 grid around each pixel, divides the grid is subregions and selects the darkest pixel in each of these subregions. Then, it calculates the covariance matrix of the local coordinates of these pixels. Then, the two eigenvalues of the covariance matrix are used to calculate the eccentricity (defined as $e = \sqrt{1 - \frac{b^2}{a^2}}$, with a and b being the highest and the lowest eigenvalue), which is used to determine whether a whisker point was detected or not (the algorithm was described in detail in Section 3.3). A main advantage of this algorithm compared to the skeletonization algorithm is the fact that the Clack detection algorithm treats all the pixels independently, which means that the algorithm is fit for execution in parallel, e.g. on a GPU. Furthermore, the algorithm does not contain any loops for



(a) Output of Clack algorithm, applied on an original input frame.



(b) Output of Clack algorithm on a preprocessed frame.

Figure 5.10: Output of the Clack algorithm, shown in scaled colors, of the frame in Figure 5.8a

which the number of passes is unknown. This means that the processing time is more predictable, which makes pipelining easier.

The authors claim that their algorithm is so robust that it does not require much preprocessing. Since it would be a big advantage if preprocessing could be left out, the algorithm was also applied to a original, non-processed frame. The output is shown in Figure 5.10. It was noticed that, for the Erasmus MC videos, preprocessing is still needed to obtain a decent result. Without preprocessing, one can barely see the whiskers because of the noise. With preprocessing, the whiskers are well visible and their angle is estimated adequately, but even with a very high threshold for eccentricity ($e > 0.992$), the image contains a lot of noise. At places where the whiskers do no cross, they are clearly visible. At places where whiskers cross, nothing is detected, but most probably, the shape can be estimated using the extracted local directions. The detection algorithm is done independently per pixel and can be run in parallel. However, for our case, further processing would be necessary to remove the noise and trace the whiskers.

5.2.3 Curvilinear structure detection

The last algorithm that was tested for low-level feature detection is the algorithm developed by Carsten Steger [13]. This algorithm does not detect which pixels belong to a curve, but extracts the points of the centerline, i.e. the line that can be drawn through the middle of a whisker. Steger used the responses of the two-dimensional convolution of a line profile with derivatives of two-dimensional Gaussian kernels as a means to detect the curve. For the case of a one-dimensional line, Steger gives the formula for a Gaussian curve as:

$$g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{5.1}$$

The value for σ is determined by the width w of the line. Steger advises that $\sigma \geq \frac{w}{\sqrt{3}}$. Steger also added an implementation to reduce a bias that appears if there is an intensity

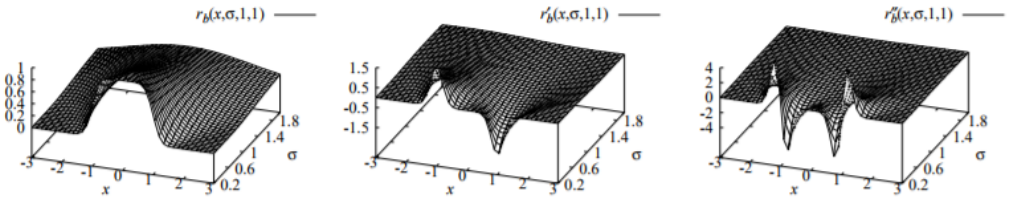


Figure 5.11: Response of a one-dimensional curvilinear structure to convolution with the Gaussian curve and its first and second derivatives for varying σ. Image taken from [13].

difference between the backgrounds on either side of the curvilinear structure, but since in our case, the background is either removed or fairly even, this bias reduction is not needed for our application. For a well-chosen value of σ, the profiles of a structure convolved with the Gaussian curve and its first and second derivatives (denoted as r, $\dot{r}$ and $\ddot{r}$) show a particular response, as shown in Figure 5.11. In principle, it is possible to detect the position of the line by detecting where the first derivative changes its sign. However, Steger correctly states that this would give the position of the line with only pixel accuracy. To achieve sub-pixel accuracy, Steger constructs the second order Taylor polynomial of the section of the one-dimensional line, which is defined by

$$p(x) = r + \dot{r}x + \frac{1}{2}\ddot{r}x^2 \tag{5.2}$$

The position of the centerline is given as the point where $\dot{p}(x) = 0$, i.e. $x = -\frac{\dot{r}}{\ddot{r}}$. This analytical approach gives the position of the line with sub-pixel accuracy. Steger expands the one-dimensional case to a situation with two dimensions, constructing five two-dimensional kernels, based on the derivatives of the Gaussian curve:

$$g_{x,\sigma}(x,y) = g_\sigma(y)\dot{g}_\sigma(x) \tag{5.3a}$$
$$g_{y,\sigma}(x,y) = \dot{g}_\sigma(y)g_\sigma(x) \tag{5.3b}$$
$$g_{xx,\sigma}(x,y) = g_\sigma(y)\ddot{g}_\sigma(x) \tag{5.3c}$$
$$g_{xy,\sigma}(x,y) = \dot{g}_\sigma(y)\dot{g}_\sigma(x) \tag{5.3d}$$
$$g_{yy,\sigma}(x,y) = \ddot{g}_\sigma(y)g_\sigma(x) \tag{5.3e}$$

Convolving the image with these kernels will give a good estimate of the partial derivatives r_x, r_y, r_{xx}, r_{xy} and r_{yy} of the image. The direction of the whisker can be determined by the eigenvectors of the Hessian matrix:

$$\mathbf{H} = \begin{bmatrix} r_{xx} & r_{xy} \\ r_{xy} & r_{yy} \end{bmatrix} \tag{5.4}$$

The eigenvector that corresponds to the eigenvalue with the largest absolute value λ shows the direction perpendicular to the line, and is denoted by $\mathbf{n} = \begin{bmatrix} n_x \\ n_y \end{bmatrix}$. The position

of the line is again determined by a Taylor polynomial, this time the position of the line is determined by

$$(p_x, p_y) = (tn_x, tn_y) \tag{5.5}$$

with

$$t = -\frac{r_x n_x + r_y n_y}{r_{xx} n_x^2 + 2 r_{xy} n_x n_y + r_{yy} n_y^2} \tag{5.6}$$

To determine whether the line point falls within a pixel, it is required that

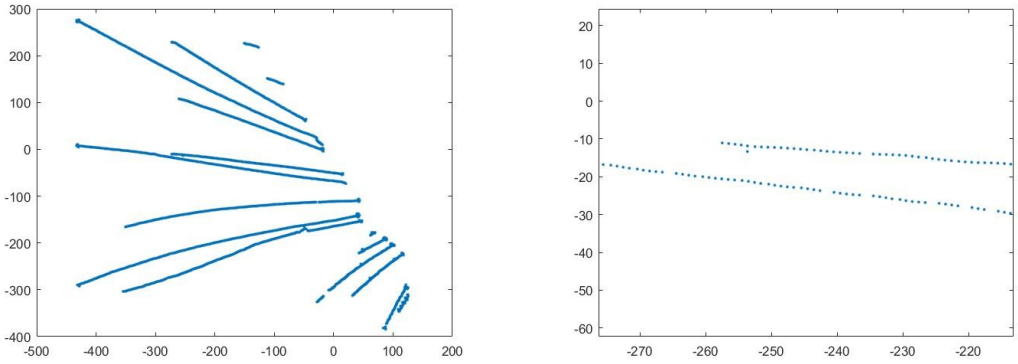
$$(p_x, p_y) \in \left[-\frac{1}{2}, \frac{1}{2} \right] \times \left[-\frac{1}{2}, \frac{1}{2} \right] \tag{5.7}$$

Furthermore, a threshold $\lambda_{\text{threshold}}$ for the absolute value of the largest eigenvalue is set to select 'salient' lines, i.e. lines of a certain width that can be distinguished from noise. In the case of light lines on a dark surface, as is the case after preprocessing, the absolute largest eigenvalue will be negative, so a negative upper limit to the value of the eigenvalue will detect the lines well. If this condition holds, the pixel is considered to contain a line point, and (p_x, p_y) will be used to determine the sub-pixel position of the line point. Steger describes the case that the sides of the curvilinear structure have different gray values; in that case, an additional correction is needed. However, this is not relevant for our purpose, since the background is extracted for every frame.

For whisker tracking, extracting the centerline has several benefits. First, no thinning algorithm is needed, and linking points of a centerline that is determined with sub-pixel accuracy is not too difficult. Steger developed a linking algorithm based on the local (pixel-level) direction of the line, which will be described in the next chapter, but even simpler forms of clustering can be used. Secondly, if whiskers are thicker than one pixel, crossing whiskers will be easy to distinguish: the upper whisker will have a continuous centerline, whereas the centerline of the lower whisker is interrupted. Furthermore, the algorithm consists of a convolution step to find five partial derivatives of the frame, and a pixel-based calculation step in which the position of the line is determined. Both operations are highly parallelizable.

The Steger detection algorithm was implemented in MATLAB. The image was moved to the GPU using `gpuArray`, as were the Gaussian kernels (which do not change over the course of the program and are therefore calculated only once). The convolution and calculation steps were performed fully on the GPU. Because the original video is compressed and some whiskers have a width of about a pixel, the image had to be upscaled by approximately two times to give a good result. Although this increases the data that has to be processed, it is acceptable as the operation can be performed in parallel.

The results of the detection process on a frame are shown in Figure 5.12. The detection process was done using linear interpolation to upscale the image, a Gaussian kernel size of 30x30 pixels, $\sigma = 2.3$ and the lower limit of $\lambda_{\text{threshold}} = 2.7$ for the absolute largest eigenvalue. On Figure 5.12a, it can be seen that the centerlines are nicely detected, there is barely any noise, and the different whiskers are clearly distinguishable. Figure 5.12b shows a detail of the image, where one whisker starts hiding behind another whisker. It can be seen that the centerline is indeed extracted with sub-pixel accuracy, and the distance between the centerlines makes them easy to distinguish.



(a) Extraction of centerlines via Steger's algorithm.

(b) Detail from Figure 5.12a.

Figure 5.12: Output of the Steger algorithm.

The implementation in MATLAB is shown in Algorithm 2. The iterations of the `foreach` blocks can be performed in parallel. The largest eigenvalues and the corresponding eigenvectors of each $\mathbf{H}$ can be performed according to the method described in [32]; as for a two-dimensional image, $\mathbf{H}$ is always a 2x2 matrix. Since each frame consists of a matrix of values, the Gaussian kernels were also constructed as matrices. The convolution that is performed is therefore a per-pixel discrete convolution. The Gaussian curves are centered on the kernel matrix, and the matrix dimensions are determined by σ (after all, for small values of σ, the curve approximates 0 faster than for larger values of σ).

Steger's curvilinear structure-detection algorithm was chosen as an additional low-level processing step: it is highly parallelizable and more accurate than the other two algorithms that were distinguished. With the right parameters, the amount of noise is negligible. Furthermore, as will become clear in the next chapter, Steger's algorithm can be used to obtain information about the local direction of the whisker. The main disadvantage of applying the algorithm to the available videos is that, for the compressed videos, some upscaling is needed to acquire accurate results, but since the algorithm can process every pixel of the frame in parallel, the advantages outweigh this disadvantage.

After all the low-level processing steps, the algorithm detects points that are part of centerlines of the whiskers with sub-pixel precision. However, what is still unknown, is the way these points are connected: which points are part of the same whisker? This question becomes even harder to answer if the whiskers are partially hidden behind other whiskers. There is no information about the number of whiskers yet, and there is no abstract description of the whiskers that follows the compactness hypothesis as defined in Section 2.2.2. These challenges belong to the domain of intermediate-level processing, which will be the subject of the next chapter.

Algorithm 2: Pseudo-code for the MATLAB implementation of Steger's curvilinear structure point detection algorithm

```
foreach pixel do
    foreach kernel in equation 5.3 do
        Center Gaussian kernel on pixel;
        Perform convolution to calculate r_x, r_y, r_xx, r_xy and r_yy;
    end
    Construct H according to eq. 5.4 and calculate its absolute largest eigenvalue λ and
     corresponding eigenvector n;
    Calculate t according to eq. 5.6;
    Calculate p_x and p_x according to eq. 5.5;
    Condition 1: λ ≤ −λ_threshold;
    Condition 2: equation 5.7;
    if both conditions are met then
        Whisker point was detected. Determine intrapixel position of the whisker point
         according to eq. 5.5;
    else
        No whisker point was detected;
    end
end
```

6 Intermediate-level processing

In the previous chapter, a system was proposed to convert gray-scale frames into a representation of points of the centerlines of the whiskers. In this representation, it is determined for each pixel whether it is part of a centerline, and if yes, what the intrapixel position this point of the centerline is. In this chapter, a system is presented which takes the output from the previous step as its input, and converts it into an abstract representation of the whiskers. This abstract representation should be compact, but also give an accurate description of the whisker so that it is possible to track over time. This chapter consists of three sections. Section 6.1 will explore whether the Hough transform could be used for the purpose of whisker tracking and explore the possibilities for whisker parametrization. Section 6.2 describes the idea of clustering: assigning the centerline points that belong to the same whisker to the same cluster, and points from different whiskers to different clusters. Several algorithms are proposed. Section 6.3 assesses the issue of parametrization of whiskers. Different ways to accurately describe a whisker using a limited number of parameters are assessed.

6.1 The Hough transform and whisker parametrization

The Hough transform, invented by Paul Hough in 1962, has become the main algorithm for the detection and parametrization of straight lines in images, and has been extended to detect circles, curves and arbitrary objects. The algorithm, which maps points in the Cartesian domain as curves to the (ρ,θ) Hough domain and then searches for intersections find lines, was described in Section 2.2.2. Although it was decided not to use the Hough transform in FWTS, it was explored extensively over the course of the project, which lead to valuable insights regarding whisker parametrization.

The Hough transform was recognized as a method that could easily be adapted for the detection of other objects, such as circular objects: if a point P is assumed to be part of a circle, ρ can be defined as the radius of the circle and θ as the angle of the line that passes though the center of the circle and P. However, this requires a good approximation of the size of the circle: after all, if the size is unknown, θ does not uniquely map to ρ anymore. Other researchers have shown that the Hough transform can be used to detect any object of which the shape and size are more or less predefined [21]. Even though whiskers are often not straight, they can be described by a relatively small number of parameters, so it is worth exploring the option of using a Hough transform for whisker detection.

Because of the fact that whiskers are all connected to the snout, we can state that it is possible to draw a line along the snout, which surely intersects with all the whiskers, at a point ρ of the line at an angle θ with the line. To make sure that for every whisker point, θ maps uniquely to ρ, it is necessary to assume that whiskers are more or less straight,

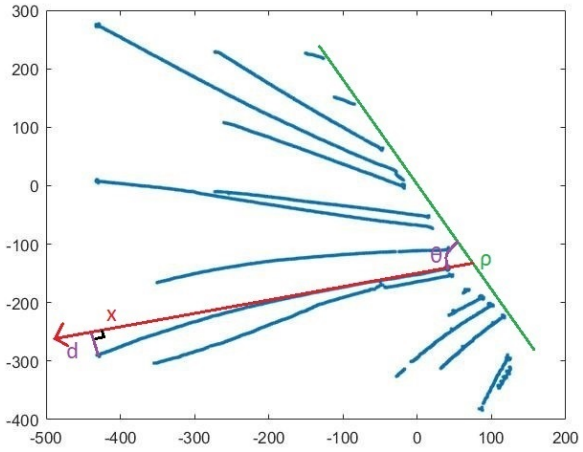


Figure 6.1: Definition of deviation due to bending d, defined as $d = bx^2$, annotated on Figure 5.12b. The red line, which functions as x-axis, represents the whisker in an unbent state at position ρ and angle θ.

which is often not the case. Even if the bending is very slight, its effect increases, the further away the whisker point is located from the snout.

For rat whiskers, it is usually possible to approximate the shape of the whisker by a second-order polynomial. Although mouse whiskers are thinner and hence might have different dynamics, we found empirically that a cubic fit was a good approximation of the shape of the whiskers in our videos, too. The deviation of the shape of the whisker from a straight line, was defined as $d = bx^2$, where b is a predefined parameter for bending, and x the position along the line defined by θ and ρ, as show in Figure 6.1. If the variance of b is small, it might be possible to detect lines using the Hough transform.

However, it turns out that the computational costs of this approach are high, whereas it accuracy turns out to be limited. The number of detected whisker points in the low-level detection steps of the algorithm is approximately 4000. If we would like to detect whiskers with 0.5° accuracy, this would mean that at least $4000 \times 360 = 1.440.000$ options would need to be evaluated per frame; not taking into account the effects of bending. Furthermore, it turns out that it is necessary to determine b rather precisely, which would not only lead to more computational costs, but would also require the detection of peaks in three-dimensional space. Therefore, it was decided to explore different options for parametrization.

6.2 Clustering

In the previous section, it became clear that directly fitting curves to the detected whisker points with the Hough transform is too complex and computationally expensive. To reduce the complexity of the intermediate-level processing of the frame to make parametrization possible, we decided to split up the problem in two steps: the first step

is to decide which whisker points belong to the same whisker. This makes the second step, the actual parametrization step, less complicated and less computationally expensive, since each cluster can be treated in isolation. An additional advantage is that, after clustering, the parametrizations of all the whiskers based on the clusters can be performed in parallel.

This section will describe two clustering algorithms that were considered. The first algorithm is Density-Based Spatial Clustering of Applications with Noise (DBSCAN), which was invented by Ester et al. in 1996 [33]. This algorithm will be described in Section 6.2.1. It was considered and implemented, but is not used in FWTS. The second algorithm, which is described in Section 6.2.2, is the local clustering algorithm by Steger (1998), which uses information about the local direction of a whisker to cluster points that belong to the same line. This algorithm was implemented in our system, albeit with small adaptations. Section 6.2.3 will describe the cluster stitching algorithms, which are performed after the general DBSCAN or Steger algorithms. These algorithms are needed to make sure that segments of whiskers that are partially hidden are still seen as part of the same whisker, so that parametrization can be performed successfully later.

6.2.1 DBSCAN

DBSCAN is a density-based clustering algorithm, which means that the decision whether two points are part of the same cluster is not taken based on the distance between these points or the distance between the points and a third point (which is the case with the k-means algorithm), but by the density of points in the region between the two points: if the two points are separated by a region with a high density, the points are likely to be part of the same cluster, whereas the presence of a region with a sparse point density suggests that the points belong to different clusters. This makes density-based clustering very suitable for the detection of long or irregularly shaped clusters, such as whiskers.

The DBSCAN algorithm accepts two parameters: ε and $n_{\min}$. The algorithm looks at all the points in sequence. It starts with an unlabeled point, and scans the neighborhood of this point, i.e. the circle with radius ε around the point. If the number of neighboring points is lower than $n_{\min}$, the point is labeled 'noise' and another random, unlabeled point is assessed. Otherwise, the point is used to start a new cluster.

Each point and all its neighbors are added to a set S of 'seeds', from which the cluster can grow. All the seeds are assessed; if a seed had already been labeled 'noise' or it turns out that its own number of neighbors is lower than $n_{\min}$, it is considered a 'border point' and added to the cluster. If a seed has $n_{\min}$ or more neighbors, it is considered a 'core point' and added to the cluster; all its neighbors are added to S as new seeds. When all the seeds in S are assessed, no more points are added to the cluster and the algorithm visits an unlabeled point, until all the points are labeled. The method is summarized in pseudocode in Algorithm 3.

To detect whisker centerlines as clusters, $n_{\min}$ should have a value of 2: after all, each point that is not at one of the ends of the centerline is connected to two other points. The optimal value for ε needs to be determined empirically: a low value for ε might cause whisker points that belong to the same whisker to end up in different clusters. A high value of ε might have the opposite effect: whisker points that belong to different

whiskers might end up in the same cluster. Because of the fact that computer vision is not a deterministic process, it is difficult to assess whether there is a value for ε for which no clustering errors are occur. However, it is usually better to opt for a low value of ε: after all, DBSCAN will not be able to assign whisker points of the same whisker to one cluster if the whisker is partially hidden; therefore, additional clustering is still needed (the algorithm for this will be described in Section 6.2.3). Any clusters that contain whisker points of one whisker can then be merged.

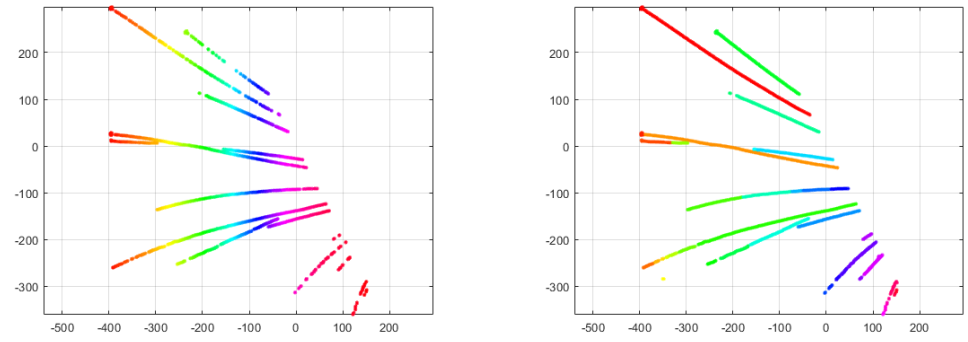
Algorithm 3: Summary of DBSCAN in pseudocode [33].

```
C = 0; // cluster label
foreach point p do
    if p.label then
        continue;
    end
    N = set of points in radius of ε around p;
    if size(N) < n_min then
        p.label = 'noise';
        continue;
    end
    C = C + 1;
    p.label = C;
    S = N; // set of seeds
    foreach point q ∈ S do
        if q.label == 'noise' then
            q.label = C;
        end
        if q.label then
            continue;
        end
        N = set of points in radius of ε around q;
        if size(N) ≥ n_min then
            S = S ∪ N;
        end
    end
end
```

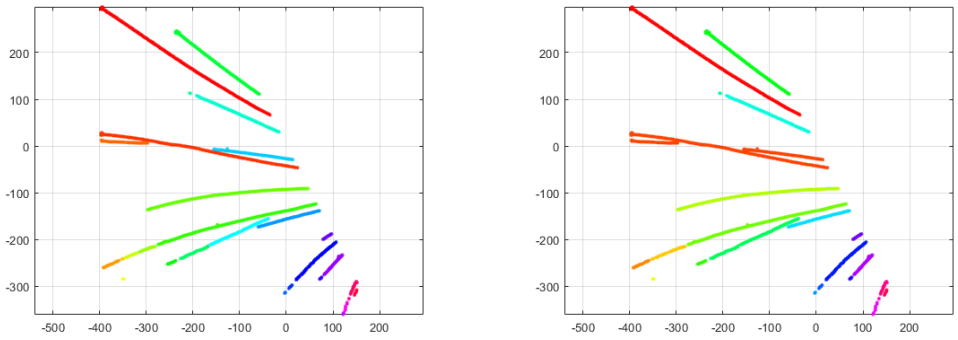
Figure 6.2 shows the result of DBSCAN clustering for several values of ε. For all shown values of ε, there are whiskers that consist of only one cluster, as well as whiskers that are scattered over two or more clusters. For $\varepsilon = 7$, the two long crossing whiskers ended up in the same cluster, whereas another long whisker is still spread over several clusters. For this frame, therefore, there is no value of ε for which the clustering immediately leads to a situation where every whisker is represented by one cluster, but a good choice would be a value between 3 and 5: for these values, we have a limited number of large clusters that can be stitched together in a later step, but there are no clusters that consist of more than one whisker.

6.2.2 Steger's local clustering algorithm

The second algorithm that was implemented, was Steger's local clustering algorithm, which was presented in the same work as the algorithm described in Section 5.2.3 [13].



(a) $\varepsilon = 1$ (b) $\varepsilon = 3$



(c) $\varepsilon = 5$ (d) $\varepsilon = 7$

Figure 6.2: Clustering of whisker points after performing DBSCAN with different values for ε, with $n_{\min} = 2$. Different colors denote different clusters. It can be seen that with a low value of ε, no significant clustering happens, whereas with a high value for ε, crossing whiskers end up in the same cluster.

The algorithm makes use of the fact that the same information that is used to detect the sub-pixel position of the whisker point, can be used to detect the local direction of the line, since:

$$(n_x, n_y) = (\cos\alpha, \sin\alpha) \tag{6.1}$$

This information can be used in the clustering process in two different ways. Firstly, if the local direction of the whisker is known, we can search in that direction to find the next whisker point of the whisker. Secondly, there are now two parameters that can be used as criteria for linking up whiskers: the Euclidean distance d and the difference in angle $\beta = |\alpha_2 - \alpha_1|$, with $\beta \in [0, \pi/2]$ (remember that for DBSCAN, there is only the neighborhood parameter ε). Even if the local (Euclidean) direction is not entirely accurate, it gives a good indication. Having more parameters means that we have more information to base the clustering decisions on, which leads to more accurate clustering.

Steger implemented the 'searching' algorithm in the way illustrated in Figure 6.3. The light gray pixels are scanned for possible continuations of the line. If there are multiple whisker point that could be continuations, the point for which the value for

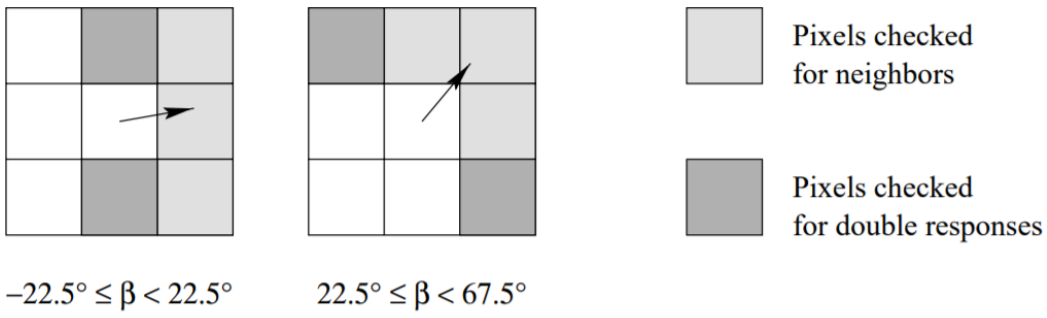


Figure 6.3: Neighborhoods examined during the linking process, based on the local direction of the line. The middle square contains the whisker point for which the neighbors are examined. Image taken from [13].

$d + c\beta$ is minimized is chosen (with c a parameter that can be configured manually). The dark gray pixels, perpendicular to the local direction of the line, are scanned for double responses; i.e. places where a point of the centerline of whisker leads to a detection in two pixels. Steger implemented this because he noticed that for thicker curvilinear structures, it sometimes happens that multiple parallel points are detected that have the same direction. Since these points are not part of the centerline, they should be ignored.

Since every pixel of the image contains at most one centerline point, and the result of the analysis of one pixel is not dependent on the result of the analysis of other pixels, all pixels can be processed in parallel. This will allow the construction of a connectivity matrix: a sparse, symmetric, square matrix with dimensions equal to the number of whisker points. If the p^{th} line point is connected to the q^{th} line point, the matrix will have a 1 at position (p, q); if they are not connected, this position contains a 0. This connectivity matrix can then be converted to a categorization into different clusters (in MATLAB, there is the function `graphconncomp` for this purpose).

The implementation for the whisker-tracking system is changed in a few ways. First, because whiskers are very thin, long structures, the phenomenon of parallel lines that Steger described did not occur. Therefore, it is not necessary to check the pixels perpendicular to the local whisker direction for double response. However, it occurs that there is an 'empty' pixel between two parts of a centerline. This phenomenon can be seen, for example, in Figure 5.3b. To make sure that the points on either side of the 'empty' pixel are still recognized as part of the same whisker, the clustering algorithm was adapted, so that it is possible to 'peek' one pixel further than the immediate neighbors.

The last adaptation that was made, was to peek not only in one direction, but both in the local direction and the opposite direction. The vector that shows the local direction of the line can be directed either upwards (towards the tip of the whisker) or downwards (towards the snout). Steger opted to force the vector in one direction, flipping it in cases where it was pointing in the 'wrong' direction. In our implementation, it turned out this approach was sometimes prone to mistakes: at times, vectors of two adjacent whisker points still pointed in different directions, which led to clustering errors. Therefore, it

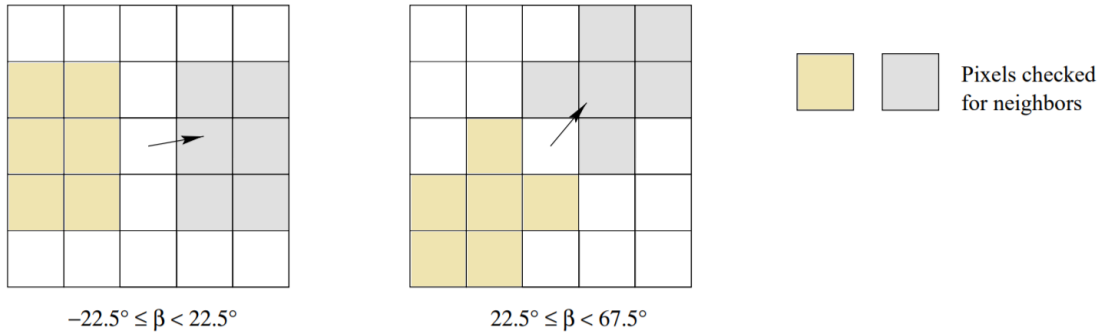


Figure 6.4: Adapted local clustering algorithm, as implemented in the whisker-tracking system. Image adapted from [13].

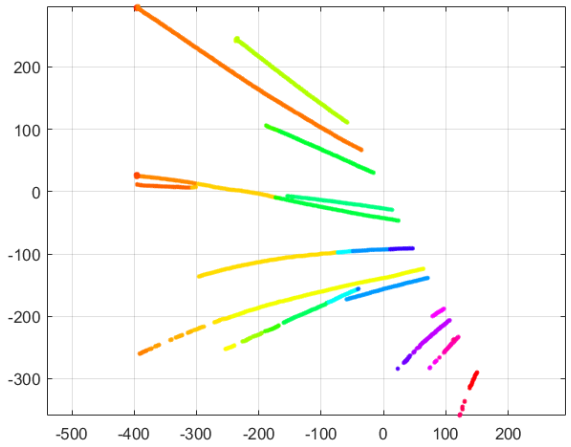


Figure 6.5: Clustering of whisker points after performing Steger's local clustering algorithm, with $c = 1$. Different colors denote different clusters.

was decided to apply this adaptation. The fact that vectors do not need to be flipped makes the implementation easier and more robust. Since it is possible to perform the search in both directions in parallel, the extra computation time is limited, whereas the fact that edges between two whisker points are now created twice (once in either direction) serves as an extra check: the edges that are only created once are removed, since these are likely to be incorrect.

Figure 6.4 shows the adapted procedure, as it is implemented in the final whisker-tracking system. Clustering results are shown in Figure 6.5. As can be seen by comparing with Figure 6.2, the differences with the DBSCAN algorithm are minimal. We prefer the Steger algorithm over DBSCAN because the Steger algorithm takes the local direction into account and ensures that every whisker point is connected to at most two other

whisker points. Since it makes use of more information, the algorithm is less likely to make mistakes.

6.2.3 Cluster stitching algorithms

As becomes clear from figures 6.2 and 6.5, after the first clustering step, the number of clusters is still larger than the number of whiskers. Therefore, it is necessary to reduce the number of clusters by merging those that are part of the same whisker together, in such a way that every cluster represents one whisker.

In general, it is possible to distinguish three causes of the fact that whisker points that belong to the same cluster may still end up in different clusters after performing the DBSCAN or Steger clustering algorithm:

1. **Missed whisker points**: There are cases in which the algorithm described in Section 5.2.3 misses two or more whisker points of a whisker in a row. In this case, the adapted Steger detection algorithm described in the previous section will split the whisker into two clusters. The DBSCAN algorithm splits whiskers into two different clusters if the distance between two whisker points is smaller than ε. This problem can sometimes be resolved by (in the case of DBSCAN) choosing a larger value for ε (which increases the risk that whisker points from different whiskers are clustered together), or changing the parameters of the whisker point detection algorithm to make it more sensitive (which increases the risk of noise being detected as whisker points).

2. **Crossing whiskers**: When whiskers cross, the bottom whisker partially hidden, which means that the centerline is interrupted. Therefore, the parts of the whisker on either side of the crossing point end up in different clusters. This issue is particular to the problem of whisker tracking and independent of the clustering algorithm used in the previous section.

3. **Overlapping whiskers**: Near the snout, the whiskers are often close together, whereas they fan out at their tips. At times, this leads to the phenomenon that the bottom parts of whiskers overlap, which means that of one whisker, only the top part is visible. In this case, the centerline of the bottom whisker is not just interrupted, but simply disappears behind the other whisker.

The first two issues can be addressed by merging clusters of whisker points that belong tot the same whiskers. Clusters that contain a certain number of points form line segments, and clusters that are part of the same whisker are usually more of less collinear. This collinearity can be used to decide which clusters belong to the same whisker. Therefore, we designed a cluster stitching algorithm, which can be described as follows: a section of a certain length from the tip of a cluster a is approximated by a line l. This line is extended upwards, and all the clusters of which the bottom points come close to this line are marked. The cluster b of which the bottom point comes closest to the tip of the cluster a, can be merged with cluster a. This procedure can be repeated for every cluster, as long as it is long enough to be linearized. In this way, clusters that are part of the same whisker are stitched together. The procedure is summarized

in Algorithm 4. The values for $L_{\min}$, $p_{\max}$ and $q_{\max}$ can be configured manually. To obtain a good result for the case where whiskers cross, it makes sense to define $q_{\max}$ as a function of p: after all, if whiskers cross, the interruption in the centerline is likely to be more than a few pixels, which makes it more likely that the actual shape of the whisker deviated from the linearization. Therefore, it is a good idea to let $q_{\max}$ increase as p increases. The result of applying the cluster stitching algorithm on a frame is shown in Figure 6.6. As can be derived from the colors, every cluster now corresponds to one whisker. The algorithm also created a correct clustering of the crossing long whiskers.

Algorithm 4: Pseudo-code for cluster stitching algorithm

```
define p_max; // parameter for max. distance collinear with the whisker tip.
define q_max; // parameter for max. distance perpendicular to the whisker tip.
foreach cluster c of length L_min or longer. do
    Linearize tip of c with length L_min to define axis p;
    define axis q perpendicular to p, with origin in the whisker point at the very tip of c
    b_chosen = 0; // cluster chosen to merge with c
    b_distance = ∞;
    foreach cluster b that is not c do
        Determine location of tip of b in (p, q)-coordinates;
        if p_tip of b < p_max AND |q_tip of b| < q_max AND p_tip of b ≥ 0 then
            v = sqrt(p²_tip of b + q²_tip of b);
            if v < b_distance then
                b_chosen = b;
                b_distance = v;
            end
        end
    end
    if b_chosen ≠ 0 then
        Merge cluster c and b_chosen;
    end
end
Discard clusters with L < L_min. // filter out small clusters as noise
```

Although, as illustrated in Figure 6.6, cluster scattering due to missed whisker points and crossing whisker points are addressed by the cluster stitching algorithm, the case were the bottom part of a whisker is hidden due to an overlap with another whisker remains. In Figure 6.6, this happens near $(-50, -150)$. Since the bottom part of the whisker is hidden behind another whisker, the best way to estimate the shape of the bottom part of this whisker is to follow the shape of the whisker that covers it. To do this, we followed the following procedure: we linearize the bottom of all the clusters that appear to be 'floating' (not connected to the snout) and draw a line towards the snout. At the first point where this line intersects with another whisker, all the whisker points below the intersection become part of both whiskers. The effect of this is illustrated in Figure 6.7: the whisker points between $(-10, -140)$ and $(65, -125)$ are considered part of two whiskers, as the whiskers overlap there.

Instead of linearizing the tips of the clusters, it would also be possible to approximate the shape of the whole cluster. This, however, would require at least a second-degree polynomial fit, since longer whisker segments cannot be approximated by a line due to their bending. However, this would a bit more prone to errors, as it requires two

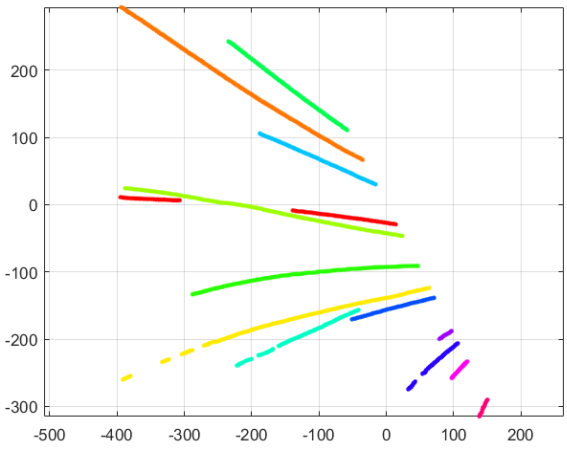


Figure 6.6: Situation as in Figure 6.5, after the cluster stitching algorithm. Different colors denote different clusters.

parameters to be estimated instead of one, especially for short segments which give little information about the shape of the whisker. Since the Euclidean distance between tips and bottoms of different whiskers is usually small, a linearization of the tip usually gives correct results. However, it could be valuable to implement both approaches and compare their results. This, however, falls beyond the scope of this thesis.

When the clustering is completed, whisker points of the same whisker belong to the same cluster, and whisker points that belong to different whiskers belong to different clusters. With this problem solved, we can now proceed to the next step, which is the creation of a parametrization of the whiskers. This will be the subject of the next section.

6.3 Parametrization of whiskers

This section will address parametrization step, in which the final clusters of whisker points are used describe the whiskers they represent by "a small number of good features" [22]; in other words, by a small number of parameters. To allow tracking over time, it is important that this description follow the compactness hypothesis as defined in Chapter 2: objects that are similar in their appearance, are also similar in their abstraction. After all, a whisker in frame n is usually similar to the same whisker in frame $n+1$, and therefore, their parametrizations should be similar too, to be able to recognize them as the same whisker.

Which parameters can be used to describe a whisker? Spanke's algorithm used angle and position to describe whisker shafts. These are important parameters, but they were not sufficient to track whiskers over time, since whiskers sometimes move close to each other and overlap, which makes them indistinguishable if only these two parameters are used. Detecting the full length of the whisker, as was done in this work, can give information about two more characteristics: the length of the whisker, and its shape. In

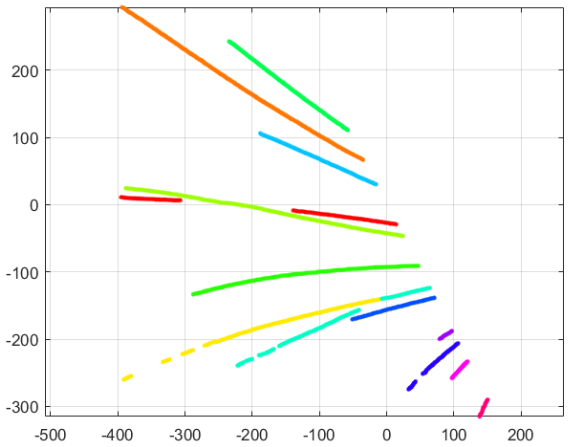


Figure 6.7: Situation as in Figure 6.6, after the issue of overlapping whiskers was addressed.

this section, different ways for parametrization are explored. Section 6.3.1 describes a local-linearization algorithm, which describes a whisker in terms of a collection of line segments. Section 6.3.2 describes a regression algorithm where the system determines the parameters of a line by means of a regression algorithm.

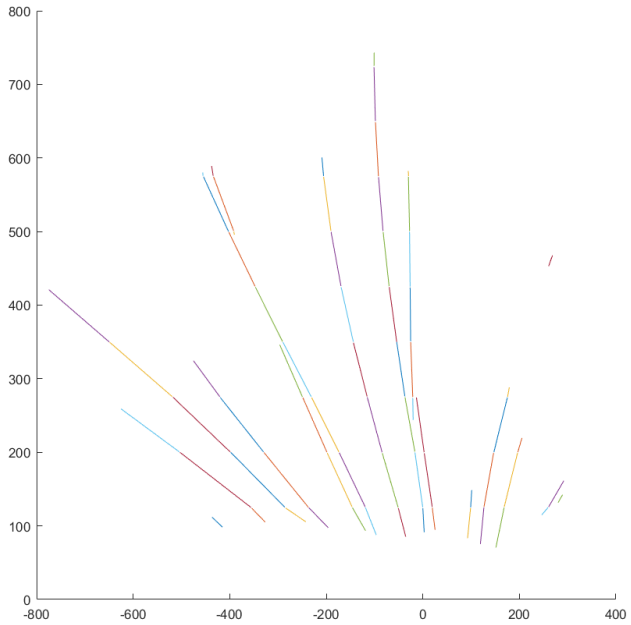
6.3.1 Local linearization

Linearization was attempted by splitting up the clusters into small parts and fitting lines lines on each of them (this can be done by using the covariance matrix (as shown in [11]), polyfit, or detecting the two whisker points at the top and bonttom of each cluster), and connecting line segments that are collinear. A result of the splitting and reconnecting algorithm is shown in Figure 6.8.

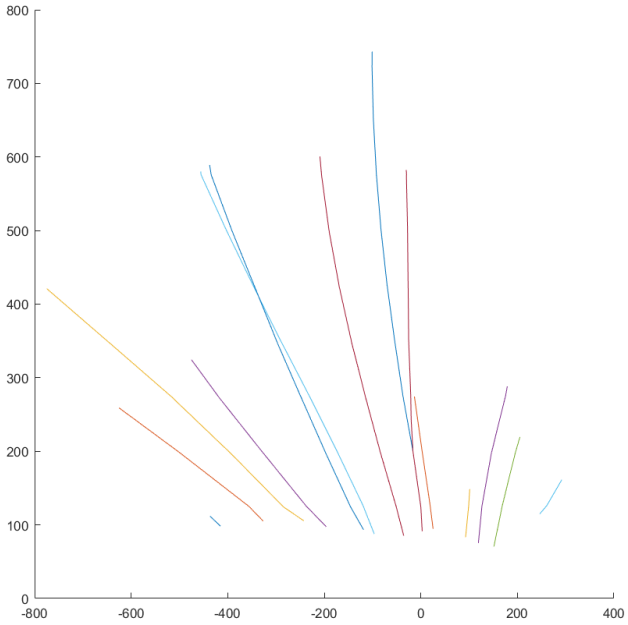
Although the procedure is simple and is a great abstraction of the whiskers compared to the situation after clustering, the parametrization does not fully comply with the compactness hypothesis: a whisker is still described by a series of coordinates, situated at predefined positions on the whisker, not by a small number of parameters. Furthermore, it undoes part of the work of the clustering that happened in the previous step: whiskers are split up in clusters again and need to be reconnected afterwards. Because of these disadvantages, it was decided to explore another solution.

6.3.2 Iterative curve fitting

There are many algorithms for curve fitting, e.g. the one implemented in the MATLAB-function `polyfit`. Most of these algorithms, though, reduce the whisker to a polynomial function of the form $y = ax^2 + bx + c$ (or a higher degree). Though this might fit the points well, it, too, does not comply with the compactness hypothesis: similar whiskers might contain completely different values for a, b and c, and the relationship between



(a) Linear segments after whiskers are cut up and linearized



(b) Reconnected whisker segments.

Figure 6.8: Frame of a test segment. The image was turned so that the x-axis coincides with the line along the snout.

these parameters and physical reality remains unclear. This will make tracking whiskers over time (in which each whisker changes its position and shape a little bit over two frames) difficult. Therefore, it is better to define a compact set of parameters, and then decide for a method that can determine these parameters for a given cluster of whisker points.

A better way to describe the whiskers, is by using parameters for characteristics that are similar for similar whiskers, and dissimilar for dissimilar whiskers. These characteristics are the angle to the snout, position, shape, and length. In Section 6.1, a way to quantify these parameters was proposed (and shown in Figure 6.1), by drawing a line along the snout and measuring the angle θ relative to the line and position ρ on the line. The shape of the line can be approximated by one parameter b. The length of the whisker can be approximated by measuring the distance between the bottom and the tip of the whisker, and defining this as length L. In this way it can be said that the whisker is a function of the set of parameters $\{\rho, \theta, b, L\}$.

To determine whether a set of parameters fits a cluster, it is possible to calculate the square distance of all the points of that cluster to the whisker as represented by its parametrization. The set of parameters for which the mean square distance (msd) is minimized, is the best fit; in other words, we are looking for a least-squares fit. Given a certain precision, it is theoretically possible to calculate the msd for all possible whiskers parametrizations at the same time. This would be highly parallelizable, but is also very computationally expensive: after all, the number of possibilities is equal to the product of the range of three of the four parameters (the length of the whisker can be determined directly after the best fit is found).

The problem of finding the right parameters for a whisker based on a least-squares fit has been researched extensively, and there are multiple ways to resolve this problem. For this work, the following regression algorithm was used: first, a rough approximation of the whisker is made by fitting a straight line through the data. Then, a set W of parametrizations for which the parameters are slightly different than in the initial approximation is created, and for these parametrizations, the msd is calculated again. The parametrization with the smallest msd will be chosen as the new approximation of the whisker, the other options will be discarded. Then the process is repeated, as long as the improvement per step, $\frac{\text{msd}_{\text{old}} - \text{msd}_{\text{new}}}{\text{msd}_{\text{old}}}$ exceeds a certain threshold (e.g. 0.01, or 1%). After that, the procedure is terminated.

This process is only partially parallelizable: the number of necessary iterations depends on the number of parametrizations that is tested on every step, but the process will need at least two iterations (even if the first iteration immediately gives a good fit, a second iteration is needed to determine that further improvement is not possible). Although the exact number of necessary iterations is not known beforehand, it is possible to set an upper limit to the number of iterations, after which the process is ended. Since every cluster is treated in isolation, it is possible to process multiple whiskers in parallel.

The number of parameter tuples in set W is determined by parameters that can be set up by the user. Let $\Delta_{\text{max},\theta}$, $\Delta_{\text{max},\rho}$ and $\Delta_{\text{max},b}$ be the maximum amounts by which the members of set W are allowed to vary in either direction, and n_{var} be the number of variants that will be created per parameter. Then, W will contain $n_{\text{var}}^3 + 1$ parametrizations, as three parameters need to be estimated (the length can be determined directly),

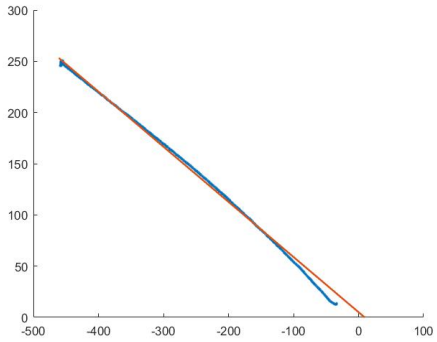
and the original parametrization should be included in W (as it is possible that the optimal fit had been reached already). Furthermore, the accuracy for all the parameters will be $2 \times \frac{\Delta_{\max}}{n_{\text{var}}}$ (i.e. the range, divided by the number of different values). Configuring a high value for n leads to higher precision, but makes every iteration more computationally expensive. Configuring high values for $\Delta_{\max}$ decreases the number of iterations, but makes the final approximation less accurate. Therefore, setting up these parameters requires a trade-off, and the optimal value depends on the machine that is used. Another option is to decrease $\Delta_{\max}$ gradually over the iterations: this will allow the algorithm to take big, less precise steps during the early iterations, and 'fine-tune' the parametrization later. It is also a good idea to limit the maximum number of iterations to prevent an endless loop. The procedure is summarized in Algorithm 5. Although the algorithm proved effective, it must be emphasized that, apart from the searching algorithm, other methods to find the best-fitting set of parameters exist (e.g. genetic algorithms, such as in [34]) which might be more efficient. For time reasons, different algorithms were not implemented, but it is worth assessing and comparing them in future research.

Algorithm 5: Pseudo-code for iterative regression algorithm

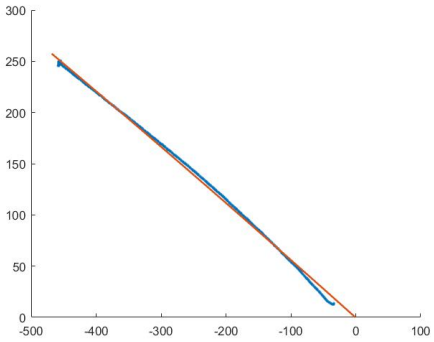
```
foreach whisker w do
    Approximate w.θ, w.ρ, w.b, w.L;
    c_old = msd(w); // determine msd for approximation
    q = ∞; // improvement per iteration
    for i = 0, i < i_max, i = i + 1 do
        if q < threshold then
            break;
        end
        Generate set W of whiskers that are similar to w;
        C_w = msd(W); // Calculate msd for every member of W
        c_new = min(C_w);
        w = member of W that corresponds to min(C_w);
        q = (c_old − c_new) / c_old;
        If necessary, adjust Δ_max for each parameter.
    end
end
```

The regression algorithm, as described above, was implemented in MATLAB. At the last moment, while experimenting with possibilities for speedups, we also tried MATLAB function `nlinfit` for non-linear regression. We found that it gives results similar to those produced by the algorithm we implemented ourselves, but performs the regression much faster. However, more research is needed to test its reliability vis-à-vis our own implementation. The testing in Chapter 8 was done using our own implementation of the regression algorithm.

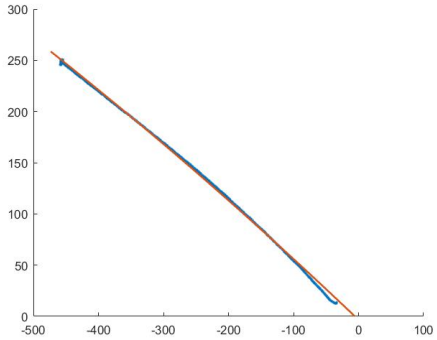
Figure 6.9 shows an example of a curve being fit to a set of whisker points in our implementation. Figure 6.9a shows the initial approximation, after that, the approximations after 5, 10, 15 and 20 iterations are shown. Figure 6.9f shows the frame from Figure 6.7 after the fitting procedure has been performed on all the whisker clusters. It is clear that in this frame, the parametrization is a good approximation of the whisker situation, and therefore, this is the algorithm that was implemented in the new whisker-tracking system. There are frames where this is less easy, for example when more than



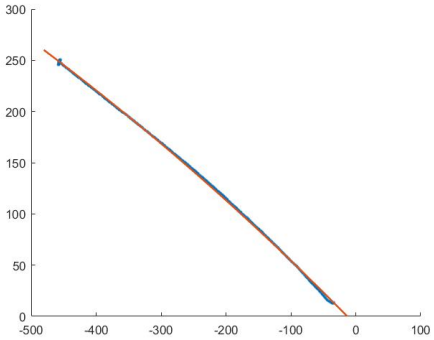
(a) Initial approximation.



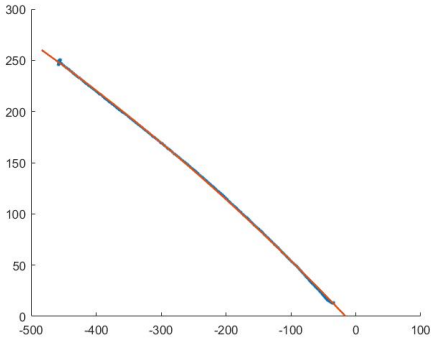
(b) Fit after 5 iterations.



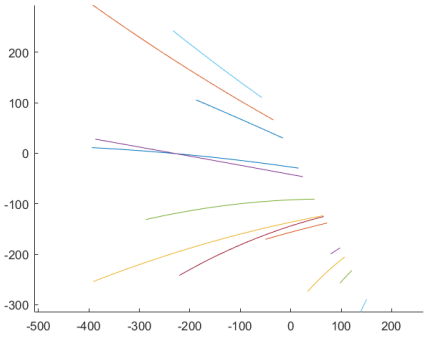
(c) Fit after 10 iterations.



(d) Fit after 15 iterations.



(e) Fit after 20 iterations.



(f) Visualization of parametrized whiskers.

Figure 6.9: Steps of iteratively fitting a curve to points using a least-squares fit

two whiskers cross. For some frames, it is not possible to obtain a clear view of the whisker situation, even by human inspection. Therefore, to track whiskers over time, it is necessary to take information from multiple frames into account. This, and other application-level processing steps, will be the topic of the next chapter.

7 Application-level processing

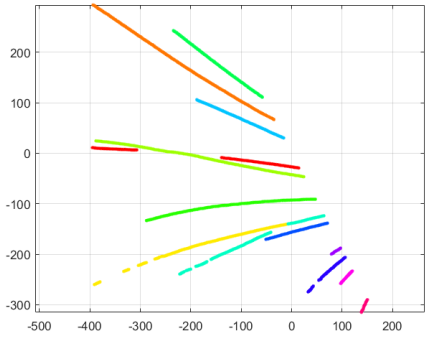
In the previous chapters, components of FWTS were developed to detect, identify and parametrize whiskers on individual frames. So far, the system treats all frames in isolation, without identifying whiskers over time. However, to determine the frequency, speed, and changes in angle of individual whiskers, it is necessary to follow them over time. This process is called 'tracking'. To successfully track a whisker, it is necessary that whiskers in different frames, in different positions, are recognized as the same whisker. Due to the nature of the video segments, not all whiskers will be detectable on all frames. Also, the detection algorithm may detect whiskers incorrectly. Therefore, the tracking algorithm needs to be robust: if there is one or a few frames in a row in which a whisker is not visible, this should not influence the tracking of the other whiskers. Furthermore, when the 'lost' whisker reappears, the system needs to recognize it and continue tracking.

In Section 7.1, the challenges of tracking whiskers over time will be discussed in further detail. Section 7.2 describes a tracking algorithm in which parametrizations of whiskers in frame $n-1$ are matched with whisker points in frame n. Section 7.3 describes an algorithm in which whisker parametrizations from successive frames are matched. Section 7.4 assesses the possibility to use a machine-learning technique, such as SVMs, to recognize whiskers over time. Section 7.5 describes a technique in which some of these algorithms can be combined, so that we can take advantage of the strong points of each of them.

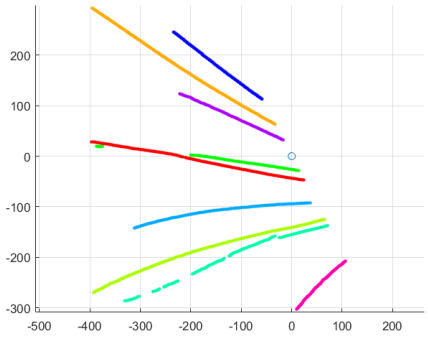
7.1 Challenges in whisker tracking

Because of the fact that whiskers cross, it is inevitable that the detection and parametrization algorithms make mistakes. An example of a detection and parametrization mistake is shown in Figure 7.1. The error occurs near the point $(-40, -160)$. In frame 7, it is clear that there are three crossing whiskers, because the tip of the short whisker lies beyond the places where the long whiskers start. In frame 14, it is also clear that there are three whiskers, as part of the short whisker appears next to the long whiskers. However, in frame 10, the short whisker becomes invisible, just at the point that one of the long whiskers appears. The algorithm concluded that both clusters are part of the same whiskers, thus detecting a whisker that does not exist, composed of parts of two whiskers.

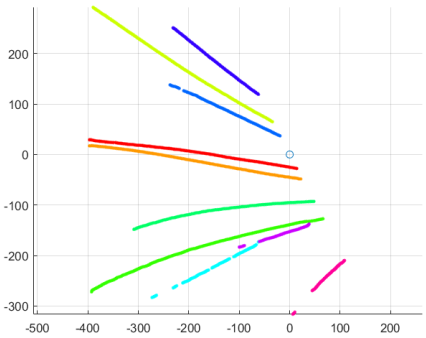
Furthermore, whereas in frame 7, some short whiskers are visible in the lower part of the image, these remained undetected in frame 10 and frame 14. In frame 7, the clusters that represent these whiskers were linearized, whereas in frame 10 and 14, the number of whisker points fell just below this threshold of $L_{\min}$ (see Algorithm 4 in Section 6.2.3). This could be resolved by choosing a smaller value for `imdilate` in the preprocessing step, or draw the baseline closer to the snout. This will, however, increase the risk that



(a) Frame 7



(b) Frame 10



(c) Frame 14

Figure 7.1: Three frames from the same video fragment. Figure 7.1a and 7.1c correctly most of the whiskers, whereas there is an error in Figure 7.1b. Furthermore, a few whiskers that were detected in Figure 7.1a, remain undetected in the other frames.

hairs from the fur are incorrectly detected as whiskers. The fact that a whisker remains undetected in a frame is not necessarily a problem: a whisker can still be tracked if it remains undetected or is detected incorrectly in one or a few frames, but it requires the tracking algorithm to be robust: a misdetection should not cause the rest of the tracking to be inaccurate.

As opposed to recognition algorithms, which treat frames or images in isolation, tracking algorithms detect relationships in the time domain, between different frames of the same video. This makes them vulnerable to errors. For instance, it is possible to track whiskers over time by performing whisker detection on all frames, and then processing the frames 2 to $n_{\max}$, matching whiskers on frame n to whiskers on frame $n-1$. If the matching process fails at some point (for example, when a whisker becomes hidden, or two whiskers are matched incorrectly), the error is likely to be repeated in all the frames that come after that. This phenomenon is known as drift [35].

One could counter drift by having the tracking result in frame n not only depend on the result of frame $n-1$, but also on other earlier frames (for instance, $n-2, n-3, \ldots, n-k,$). This makes the algorithm more robust to errors, but also makes it

more complicated: after all, when the whisker changes its position considerably in two consecutive frames, it can be difficult to match whiskers on frame $n-2$ to those on frame n, simply because the position of the whiskers has changed too much.

7.1.1 Using length for classification

To solve this problem, one could choose a feature of the whisker that does not change over time, and use this as a parameter to classify different appearances of the whisker ('classification' here refers to the process of putting all detections of a certain whisker in different frames into the same group, i.e. recognizing them as the same whisker). This would allow the use of a recognition algorithm instead of a tracking algorithm. One feature for recognition can be the length L: after all, the length of a whisker does not change over time (at least not during the experiment). However, there are a few caveats.

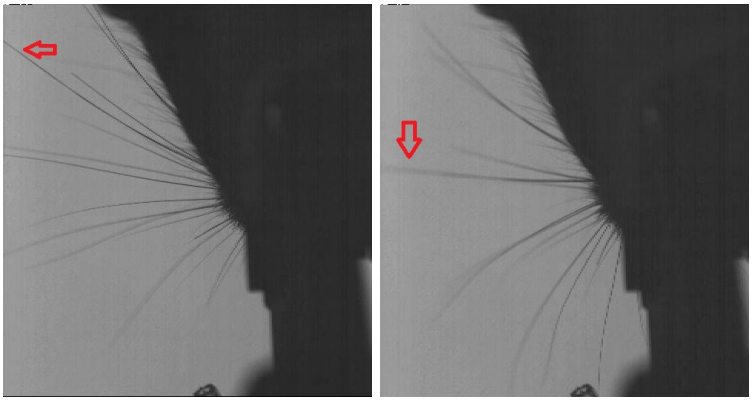
The tips of long whiskers sometimes fall outside the frame, and the part that is visible. For example, in the frame in Figure 7.2, the whisker that is denoted by the red arrow appears slightly shorter in frame 4300 than in frame 1, as a little bit more of its tip is not visible on the video frame, and much shorter in frame 3740. Frame 3740 shows another issue: when whiskers move very fast, there is motion blur at the tips of the whiskers, which makes these whisker points difficult to detect. As a result, the whisker is detected as shorter than it is in reality. Length is still a useful parameter to distinguish whiskers that have very clear differences in length. However, many whiskers are almost the same length, and for those whiskers, length is not enough of a distinguishing factor.

7.1.2 Using the pivot point for classification

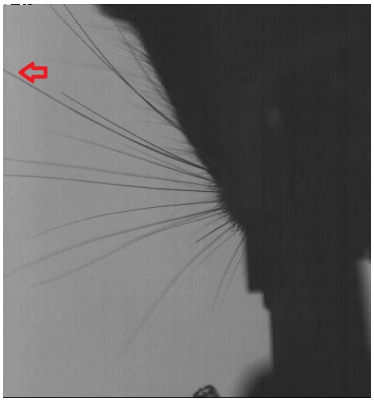
Another parameter that can serve as a classifier is the point where the whisker is placed in the snout: the pivot point, as described in Section 2.1. As was shown in that section, a mouse can change both the angle of the whisker, and the position along the snout. However, if we assume that the movement along the snout is small and is not as fast as the change in angle of the whisker, the position where the whisker is fixed in the snout can be used to detect the same whiskers in different frames and in different positions.

On the videos, the pivot point of the whisker is not visible, as it is hidden by the silhouette of the mouse. Nevertheless, if the line along the snout, which is used to determine ρ and θ, is chosen well, and the whisker changes its angle over multiple frames, it is possible to determine the pivot point with good accuracy, in the following way:

Let ρ be the position on the line along the snout, and θ the angle of the whisker relative to this line (as defined before). This is visualized in Figure 7.3. Everything below the line along the snout will not be detected, but it is known that the whiskers we see are connected to the snout. It will be assumed that the part of the whisker close to the snout is more or less linear (which is often the case; whiskers are more bent at the top). We can construct a line perpendicular to the snout that crosses the pivot point R. Let ρ_f be the point where this line intersects with the line along the snout, and L_f be the distance between R and ρ_f. This will create a right triangle $\rho_f \rho R$. Because we assume a linear whisker situation near and below the line along the snout, the angle $\angle \rho_f \rho R = \theta$.



(a) Frame 1 (b) Frame 3740



(c) Frame 4300

Figure 7.2: Three frames from the same video fragment. The red arrows point at the same whisker in all three frames.

According to trigonometry, $\cot\theta = \frac{\rho - \rho_f}{L_f}$. This leads to the equation:

$$\rho = L_f \cot\theta + \rho_f \tag{7.1}$$

We assumed R to be a fixed point, therefore L_f and ρ_f are constants. Equation 7.1 therefore represents a straight line Q of the form $y = ax + b$. If we detect ρ and θ for the same whisker in different positions, it is possible to estimate Q by plotting $\cot\theta$ against ρ. Nevertheless, it should be noted that the assumption that the whisker is completely linear near the snout does not always hold: if a whisker is strongly bent, some of the bending also occurs near the snout. Furthermore, the mouse is able to move its whisker laterally along the snout. Therefore, in reality, a plot of $\cot\theta$ against ρ will rather show a distribution of which the main principal component will be approximately collinear with Q, rather than a straight line, as can be seen in Figure 7.4, where one whisker was tracked over 1200 frames. Whether it is possible to distinguish whiskers based on this technique depends on the deviations and the variation among Q for different whiskers.

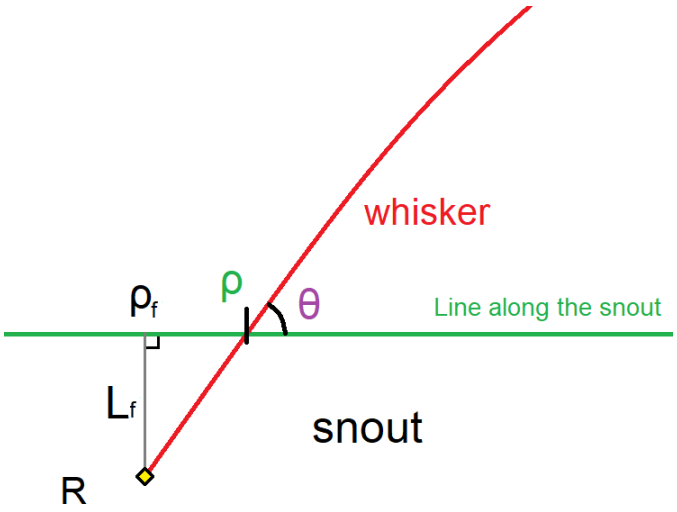
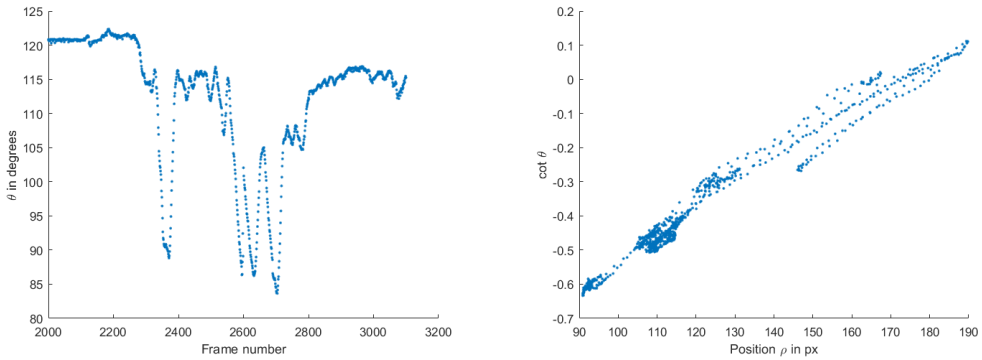


Figure 7.3: Constructing the pivot point R of the whisker, based on measured ρ and θ.



(a) Measured θ of the whisker per frame

(b) Position ρ of the whisker, plotted against $\cot\theta$

Figure 7.4: Whisker, tracked over 1200 frames. As can be seen in Figure 7.4a, the whisker sweeps several times during the fragment. Figure 7.4b shows the linear relation between ρ and $\cot\theta$.

7.1.3 Using bending for classification

A third parameter that can be used to recognize whiskers over different frames is the bending. There is a reasonable variation in bending parameter b among different whiskers, which slowly changes over time. However, since different whiskers often have similar bending, bending on itself is not enough to reliably classify whiskers: in other words: if bending is the only classifier, this could lead to incorrect tracking and recognition.

To reliably track and recognize whiskers over time, it is best to make use of a combination of parameters. The optimal combination might vary: if two whiskers have

approximately the same length but stand far away from each other, then θ and ρ can be used to distinguish these two whiskers. If there are two whiskers with a large difference in length that are situated close to each other, length might be the best parameter to distinguish between the two whiskers. Therefore, the classification process should be flexible and able to classify whiskers on different frames based on the optimal combination of parameters.

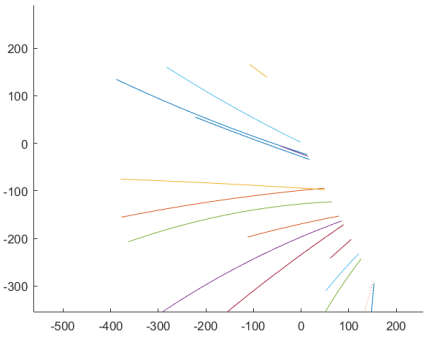
7.2 Fitting parametrizations to whisker points

This section describes an algorithm that tracks whiskers by fitting the parametrizations of whiskers in frame $n-1$ to the detected clusters of whisker points in frame n. The absolute distance of every whisker point in frame n to every parametrization in frame $n-1$ is calculated. For every cluster, the average of all the distances to every parametrization is calculated. The parametrization that has the smallest distance to a cluster is identified as the same whisker. Then, the parametrization of the whisker in frame $n-1$ is used as a first estimation of the whisker position in frame n, after which the whisker is fit to the cluster using the searching described in Section 6.3.2. The full procedure is summarized in Figure 7.5.

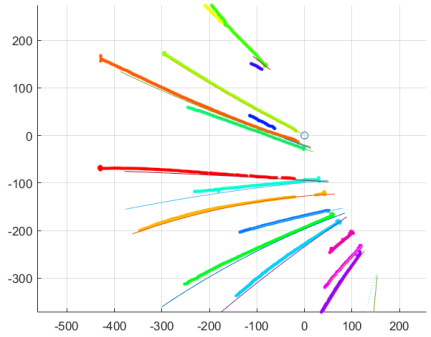
This procedure has different strong and weak points. One of the main strong points is the fact that angle, position and shape are all taken into account, by calculating the distance of every whisker point to every parametrization, and only then calculating the average distance for every cluster. The parameter 'length' is partially taken into account: the whisker points in the tips of the clusters in n have a longer distance to short whiskers in frame $n-1$ because of their length; therefore, they contribute to a higher average absolute distance, which makes long clusters less likely to be identified with short whiskers. Because of this, it is not necessary to manually choose the parameters that are used to distinguish whiskers to take both characteristics into account.

Another strong point of this procedure is the fact that the parametrization step will take few iterations from frame 2 on. After all, since the movement of whiskers is limited, the parametrization in frame $n-1$ is usually a good estimation of the position of the whisker in frame n.

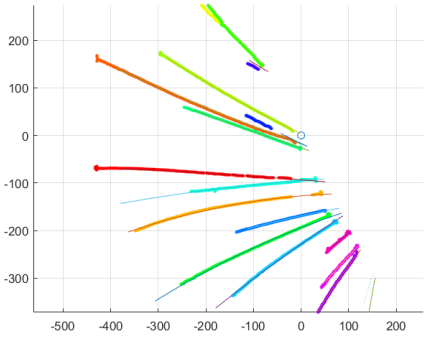
The main weak point of this technique is that information from only one frame is used to determine the new position of the whiskers, which makes the algorithm vulnerable to errors: if at any frame, two whiskers are mixed up, which could happen for a variety of reasons, this error is likely to be repeated in all the frames that follow. Furthermore, it is necessary to decide how to deal with whiskers that hide and reappear. If a whisker disappears, it might reappear on a position that is relatively far away from the position it had when it was last detected (for example, when it hides behind a moving whisker). With this algorithm, there is no way to detect whiskers again when they reappear in a different position. It is possible to make an estimation (for example, by taking the average of the change in position of all the other whiskers, since whiskers often move in the same direction), but there is no guarantee that this estimation is correct; i.e. the algorithm does not have recovery properties.



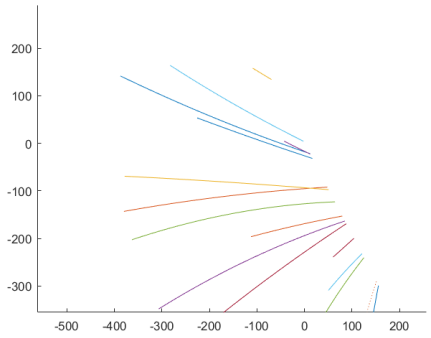
(a) Visualization of parametrizations of detected whiskers on frame 208.



(b) Detected whiskers on frame 208, with overlay of whisker points of frame 209



(c) Frame 209, after fitting whiskers to whisker point clusters



(d) Detected whiskers on frame 209

Figure 7.5: Procedure of fitting whisker parametrizations to whisker points. The length of the whiskers was determined in frame 1, and considered constant.

7.3 Extrapolating parametrizations

The algorithm described in this section is similar to Spanke's algorithm, which was described in Section 4.1. In this algorithm, the full detection and parametrization procedure, as described in the previous chapters, is performed on all the frames. To classify whiskers in frame n, the parametrizations from the frames $n-k$ to $n-1$ are analyzed. Each parameter is plotted on a time axis, and a polynomial fit is applied; this could be a second-degree polynomial as in Spanke's algorithm, or a polynomial of another degree. The polynomial fit is then used to predict the new values in frame n. The values for the different parameters are weighed and added up. The same is done for each detected whisker in frame n. Then, each detected whisker in frame n is matched with the whisker in the previous frames, according to the similarities between the prediction and the detection.

The advantage of this algorithm compared to the one described in Section 7.2, is the fact that the matching is based on more than one previous frame. If k is sufficiently large, it is still possible to find back whiskers that remained undetected in a few frames.

Disadvantages include the fact that the complete detection process has to be performed for every frame individually, which was not the case in the algorithm described in Section 7.2. Therefore, this algorithm is more computationally expensive. Furthermore, as was already stated in Section 7.2, there is no guarantee that extrapolation using a polynomial fit will lead to a good prediction, as whisker movement is often unpredictable, especially when the position of a whisker changes considerably between two consecutive frames.

7.4 Whisker detection using SVMs

The technique of SVMs, which was described in Section 2.3.1, can be used to classify whiskers. Each class can be defined as a whisker that we are tracking. In this representation, all the data points that belong to that class are instances of the same whisker in different frames. For this to be successful, it is necessary to choose a set of parameters that leads to a linearly separable set that remains constant over a relatively large number of frames.

In Section 7.1, length was identified as such a parameter. Bending is another parameter that possibly remains relatively stable over time. Furthermore, equation 7.1 showed that there is a linear relation between ρ and $\cot\theta$. If the values for either L_f or ρ_f are different for every whisker, the combination of ρ and $\cot\theta$ can be used by the SVM to define a linear boundary. Figure 7.6 shows the relation between ρ and $\cot\theta$ for all whiskers in one short video fragment. The data points for many different whiskers can be distinguished by using the 'one-vs-one' strategy based on these two parameters alone, but there are some overlapping clusters for which this is not possible. In those cases, it is a good idea to include either bending parameter b or whisker length L as parameters for the SVMs.

In MATLAB, the function `fitcsvm` can be used to train support vector machines. A labeled training set is used as input. The output is a `ClassificationSVM` model object that can be used in the `predict` function to classify an unlabeled data point using the trained SVM. `fitcsvm` can also take different settings as input, such as the fraction of outliers in the training data, i.e. the fraction of data points in the training data that are allowed to end up on the 'wrong' side of the hyperplane. There is also the option to normalize the training parameters, i.e. making sure that the average and standard deviation of all the parameters are approximately equal.

One of the challenges for the implementation of SVMs for whisker tracking is the fact that it is a method that works best for supervised learning: i.e. when there is a training set available. In the case of whisker tracking, this training set is initially unavailable. But by combining tracking algorithms with the SVM, it is possible to generate training data that can be used by the SVM. This will be the topic of the last section of this chapter.

7.5 Combining tracking and recognition

In the previous sections, several algorithms were introduced to identify whiskers in different frames. It is useful to distinguish between tracking and recognition. A tracking

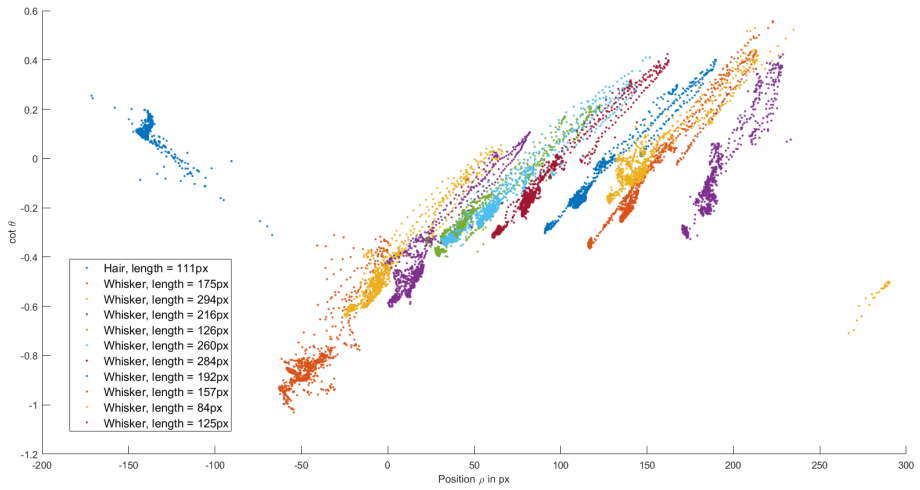


Figure 7.6: Position ρ plotted against $\cot\theta$ for all whiskers in a 1200-frame segment (compare with Figure 7.4b, in which only one whisker is plotted for the same fragment). All whiskers (excluding the hair, shown in blue in the upper-left corner) show the same type of linear relation between these two variables.

algorithm does not need training data: after an object is detected, it can be followed accurately from frame to frame if the movement between frames is not too large, the maximum change in position is defined, and the object remains visible. However, tracking algorithms suffer from drift, and as soon as the tracker loses the object, it becomes difficult to find it back.

Recognition algorithms, on the other hand, do not attempt to track whiskers over time, but focus on recognition: the previous position or angle of a whisker does not play a role. The advantage of a recognition algorithm, as opposed to a tracking algorithm, is the fact that a recognition algorithm can recover: it can find back a whisker it had lost previously. It treats frames in isolation, therefore drift does not play a role. A disadvantage of a recognition algorithm is the fact that it need to have detailed information of what defines an object and what distinguishes it from other objects that might be visible, in order to classify it correctly. It is, therefore, necessary to provide the algorithm with enough information, or train it based on existing, labeled data. A tracking algorithm, on the other hand, just needs to know the position of a whisker in one or a few previous frames in order to identify a whisker in the current one.

Kalal, Mikolajczyk and Matas (2012) invented the principle of Tracking-Learning-Detection (TLD), which decomposes the problem of tracking unknown objects over time into tracking, learning and detection [14], combining the advantages of detection and tracking. To allow the detector to learn, the authors propose the concept of P-N learning. The block diagram of this concept is shown in Figure 7.7a. The classifier is trained using training data, produced by the tracker and itself, making it a form of semi-supervised learning. Especially in the beginning when there is little training data available, the

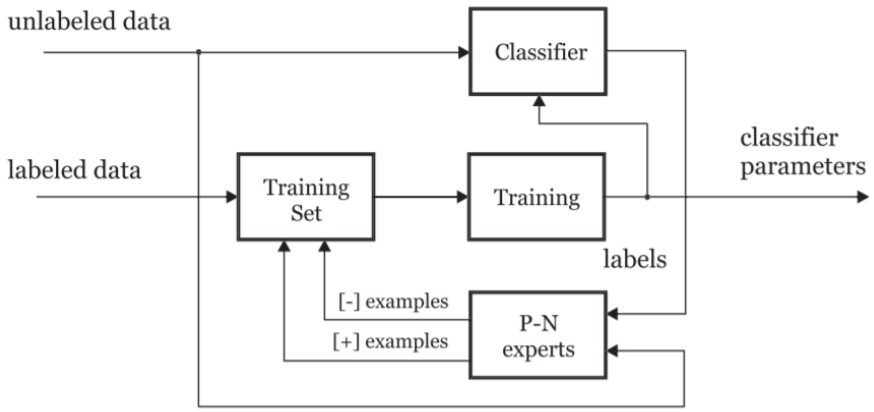
classifier is bound to make mistakes. To prevent the training set from deteriorating because of this, there is a set of 'experts': a N-expert that identifies false negatives (the classifier fails to recognize the object correctly, even though it is present), and an P-expert that identifies false positives (the classifier claims to have detected an object when this is not the case). These experts make mistakes themselves and also can correct each other. Kala, Mikolajczyk and Matas showed that, though the training set contains errors, the presence of experts can prevent progressive deterioration of the classifier, thus changing a divergent case (the classifier makes mistakes, those mistakes end up in the training data, which makes the classifier deteriorate, which leads to more mistakes, etcetera) to a convergent case, in which the classifier improves over time [14].

The Tracking-Learning-Detection, as described in the original article, is summarized in the block diagram in Figure 7.7b. The tracker tracks one object in a video using a bounding box using a Median-Flow tracker, which works directly on the pixels. The detector only gets activated when the tracker needs to be reinitialized. The case of whisker tracking, though, is a bit different: we are able to detect and parametrize full whiskers in frames with good accuracy. The problem at hand is most of all a classification problem: given a detected set of whiskers per frame, the system needs to determine the identity of the whiskers in different frames. In the original TLD system, the tracker constitutes the primary algorithm; the detector is only used in re-initialization. In the whisker-tracking system, the primary algorithm should be the classifier; the tracking system is used in the cases that the classifier fails.

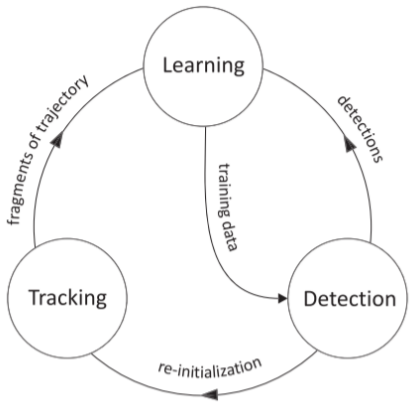
The implementation in the whisker-tracking system is a follows. A 'bootstrapping' period with a length of k_b frames can be predefined by the user. In the first k_b frames, the classification is done by a tracker, which can be either the tracker in Section 7.2 or the one described in Section 7.3. In the case that the tracker loses the whisker, an estimation of its position is made, based on the information from the other whiskers (as whiskers often move in the same direction). This works best if the whiskers do not yet move too fast. This will create an initial training set for the classifier.

For the classifier, an SVM, 'one-vs-one' approach is chosen to recognize whiskers. Length L of a whisker and bending parameter b are two of the parameters. For ρ and $\cot\theta$, there are two options: adding them directly as the third and fourth parameter, or after correcting for the average position and angle of all the whiskers. Since whiskers usually move together, subtracting the average position ρ and the average of the cotangents of angle θ changes the stretched clusters, as shown in Figure 7.6, into the denser clusters shown in Figure 7.8. The advantage of this is that the denser clusters are easier to distinguish. The disadvantage is that one incorrectly detected whisker now potentially influences the tracking results for all other whiskers in that frame. Although the latter option indeed leads to denser clusters, the former option (of adding ρ and $\cot\theta$ directly as parameters) is more suitable for the SVM algorithm: the less dense, but thinner clusters are easier to separate by a straight line. Therefore, this option was chosen for implementation in FWTS.

After k_b frames, the classifier is used first to recognize whiskers in a frame. The N-expert determines the difference in θ and ρ between frame n and $n-1$ for each whisker, and determines whether it is plausible that a whisker changed its position by this amount in 1 ms. If a whisker position in frame n is too dissimilar from its counterpart in frame



(a) Block diagram of P-N learning



(b) Block diagram of the original TLD framework

Figure 7.7: Block diagrams of P-N learning and the original TLD framework. Both images were taken from [14].

$n-1$ (i.e. its position (angle, position) changed more than is possible between two frames, based on limitations predefined by the user), the N-expert marks the detection in n as a false positive and removes it from the training data and the output.

The tracker now takes the role of P-expert: it tracks whiskers from frame $n-1$ to frame n. If it assigns an identity that was not used by the classifier to a whisker that was not classified by the classifier, this is marked as a false negative. The whisker is added to the training data and the output. If a whisker remains undetected by both the classifier and the P-expert, its position is estimated. The P-expert can use this estimation to redetect the whisker in later frames, but the estimation will not be part of the output.

The classifier consists of $\frac{(N)(N-1)}{2}$ SVMs (with N the number of whiskers that is being tracked) that are retrained in parallel every s frames (s a parameter that can be configured by the user). The training data consists of all classified whiskers from frames

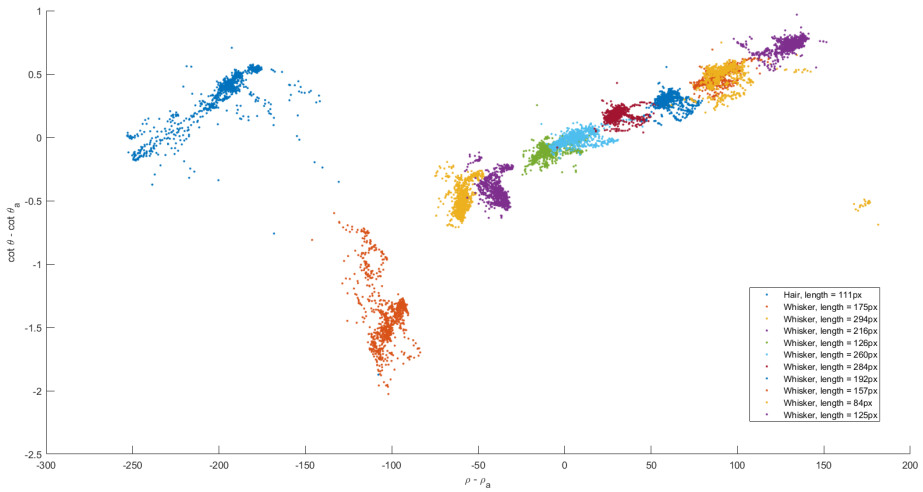


Figure 7.8: $\rho - \rho_{\text{avg}}$ plotted against $\cot\theta - \cot\theta_{\text{avg}}$ for all whiskers in a 1200-frame segment. Most clusters are denser than those in Figure 7.6, making them potentially easier to separate.

$n - W$ to n (with W being the window size, which can be configured by the user). For this, the MATLAB function `fitcsvm` is used. The parameters are standardized and the outlier fraction can be configured manually.

This concludes the description of the whisker detection-and-tracking algorithm. In the next chapter, the algorithm will be tested on segments from two videos of actual experiments.

The Full-Whisker Tracking System in practice

8

In the previous chapters, the FWTS, a new algorithm for whisker tracking, was described, and it was implemented in MATLAB. For this chapter, FWTS will be used to analyze segments from two videos of two different mice. Each of the videos shows a head-fixed mouse. Over the course of the experiments recorded in the videos, a large number of air puffs is given, to which the mouse reacts by moving its whiskers. Assessing the quality of the system is not a trivial task, since there is no baseline 'perfect' output available for comparison: even if a human would manually analyze the video data (which, given the amount of frames, would be a huge task), their measurements are not guaranteed to be accurate on sub-pixel level. We can, however, make a comparison between FWTS and BWTT.

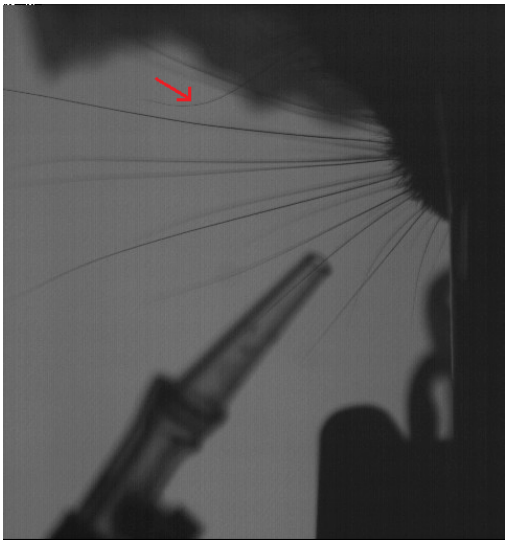
FWTS will be assessed on three aspects: the detection rate, the precision of the angle measurement, and the quality of whisker tracking over time. Where possible, a comparison between FWTS and BWTT will be made. There are situations in which comparison is not possible, because FWTS does more than BWTT: it tracks individual whiskers over the whole segments, whereas the tracking script that Spanke designed for BWTT could only detect short whisker tracks. For aspects that cannot be assessed for BWTT, such as the detection rate of individual whiskers over time, only the results for FWTS are given.

In Section 8.1, the two videos that were used for evaluation purposes will be described. Section 8.2 will compare BWTT and FWTS on the whisker detection rate. In Section 8.3, the precision of the measured angles in FWTS will be compared to the precision in BWTT. Section 8.4 describes the quality of tracking over time, using parts of the videos in which the whiskers move fast, i.e. when they are most difficult to track. Section 8.5 analyzes the processing time per frame, and assesses how long every part of the FWTS algorithm takes.

8.1 Description of chosen segments and the analysis

For the analysis, segments from two videos of two different mice were chosen. Videos of experiments are identified by numbers; the chosen videos are 170606_031 and 170607_027. For clarity, in this chapter, video 170606_031 will be referred to as video A, and video 170607_027 will be referred to as video B. From both videos, segments with a length of 60.000 frames (about 60 seconds) are taken for analysis. From video A, the frames 26000-86000 were taken; from video B, the frames 60000-120000. Segments were intentionally chosen to start after the first air puff and end before the last air puff. This ensures that the ratio between movement and non-movement is similar for both segments. Examples of frames from from both videos are shown in Figure 8.1. In each of the recorded experiments, air puffs are provided every 2 seconds; the mouse usually reacts to these

(a) Frame 2000 of video A. The red arrow marks a long hair.



(b) Frame 53102 of video B. The red arrow marks a long hair. The shadow is also visible at the top.

Figure 8.1: Frames from the two videos that were used to test FWTS.

air puffs by sweeping its whiskers. Each video poses its own challenges to the tracking algorithms. Video A barely contains any shadows, which is an advantage for the tracking algorithm. However, apart from the whiskers, at least one long, straight hair is clearly visible (marked with a red arrow in Figure 8.1a). Because the hair is so long, the algorithms cannot distinguish it from the whiskers. The fact that it crosses with several whiskers possibly poses a challenge for the tracking algorithms.

Video B also contains a long hair (marked with a red arrow in Figure 8.1b). This one is more curved than the one in video A. Furthermore, the upper part of video B is covered by a large shadow, which makes several whiskers and part of the long hair almost invisible. The piece of equipment that is used for providing air puffs is placed rather close to the snout, crossing with the whiskers. Although the shadow and the equipment do not move over the course of the video and are thus filtered out in the background removal step, the fact that they hide parts of the whiskers possibly poses a challenge for whisker tracking, especially in combination with the long, curved hair of which only the tip is visible.

For the analysis of either video, the parameters were determined on an empirical basis, by analyzing a short sequence of frames. The parameters were configured, the frames were processed by FWTS, and the results were assessed. Based on the assessment, the parameters were adjusted, the frames were processed again by the FWTS system, and the new results were compared to the earlier ones. These steps were repeated several times, until a satisfactory result was obtained. The hair in video A was detected and tracked as if it were a whisker, and its track can easily be removed manually after the analysis. The visible tip of the hair in video B turned out to complicate the analysis of the other whiskers; therefore, the problem was addressed by creating small 'blind spots'

in the video at the places where the hair is visible. This also hides parts of the actual whiskers when they cross with the hair, but if the blind spots are kept sufficiently small, FWTS is still able to reconstruct the whisker. The SVMs were not retrained at every single frame, but with an increasing interval: in the beginning, the interval was set at 3: the SVMs were retrained every 3rd frame. Then, every time 100 frames had been processed, the interval was increased by 3 frames, up to a maximum of 10. The idea behind this setup is that the importance of the new training data, which is generated every frame, is higher in the beginning, since the amount of training data we have is lower at that moment. Training data older than 3000 frames is discarded; in other words, we use a sliding window. This allows us to keep the set of training data small, in order to make sure that retraining the SVMs can be done in a reasonable amount of time. 3000 frames corresponds to about 3 seconds, and since air puffs (to which the mice react) are given every 2 seconds, the training data always contains data points both from episodes of fast whisker movement, and from periods of slower movement. Because quality, not processing time, was our main focus, the parametrization step was made as precise as possible: the process stops if the reduction in the mean square distance becomes smaller than 10^{-4}%.

The analysis was performed on a desktop computer in the Erasmus MC, running Microsoft Windows 10 Enterprise (64 bit) and MATLAB R2017b. The computer has 32.0 GB of RAM at its disposal, and runs on an AMD Ryzen 7 1800X Eight-Core processor @ 2.60 GHz. Furthermore, the computer contains two NVIDIA GeForce GTX 1080 Ti GPUs. The version of MATLAB that was used was R2017B.

8.2 Detection rate

Detection rate is defined as the number of whiskers detected per frame. The average number of whiskers that the system detects per frame is an indicator if the quality of the system. If consistently more whiskers per frame are detected, this is an indication that the system works better. In this section, FWTS and BWTT are compared on this metric. It should be noted that the tracking system for BWTT (i.e. Spanke's script) and FWTS work in slightly different ways. Spanke's script identifies tracks, i.e. detected whiskers in consecutive frames, that are identified as the same whisker. If a whisker remains undetected for a certain number of frames, the track will be terminated, and no further detections will be added to that track. However, the algorithm constantly searches for new tracks. It is therefore possible that a whisker is tracked for some time, then lost (with the track being terminated) and then re-detected, which leads to the creation of a new track for the same whisker. This means that the number of tracks vastly exceeds the actual number of whiskers. In the video B, the number of detected tracks is 11625, with the shortest track consisting of just 11 frames and the longest track of 3038 frames. No whisker is tracked throughout the whole video. In contrast, FWTS identifies a number of whiskers in one specific frame, and keeps tracking those same whiskers throughout the rest of the video. Even if a whisker remains undetected for some time, the system assumes that it still exists and attempts to find it back. The number of tracks does not change over the course of the analysis.

To be able to compare the detection rate of the two systems, despite their differences,

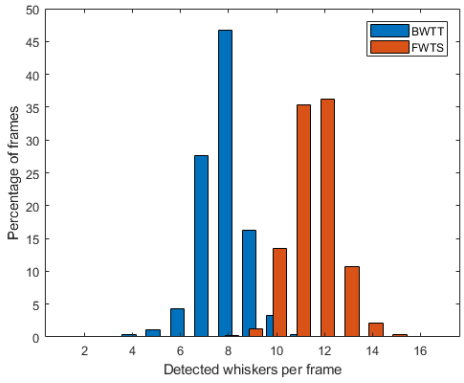
it is good to take into account the number of detected whiskers per frame both before tracking and after tracking. When it comes to detection, BWTT simply detects all the whiskers in a narrow band around the snout and gives them as output. FWTS attempts to detects all the whiskers in a frame, and then filters out the whiskers shorter than a certain length (as those are likely to be part of the fur). When it comes to tracking, Spanke's algorithm identifies tracks of whiskers in consecutive frames. The number of whisker tracks per frame is equal to or smaller than the number of detected whiskers in that frame. In FWTS, the number of recognized whiskers in a frame is equal to or smaller than the number of whisker tracks, which is determined by one specific frame.

For both BWTT and FWTS, the number of detected whiskers per frame was determined for the selected segments. The results are shown in the histograms of Figure 8.2. For both video segments FTWS detected more whiskers per frame than BWTT. For the segment from video B, the difference is rather small (9.7 vs. 8.9); for video A, the difference is considerable (11.5 vs. 7.8). On average for our segments, FWTS detects 21.2 % more whiskers per frame than BWTT. Performing the same operation on the number of tracks per frame leads to the results shown in Figure 8.3. For both BWTT and FWTS, the average number of whiskers per frame after tracking is lower than before tracking. This was expected, since not all detected whiskers can be assigned to tracks. Furthermore, it can be seen than the results for BWTT follow a Gaussian distribution, whereas the results for FWTS do not. This can be explained by the fact that BWTT generates new tracks during the analysis, whereas FWTS follows a fixed set of whiskers throughout the segment: 11 whiskers for video A and 9 whiskers for video B.

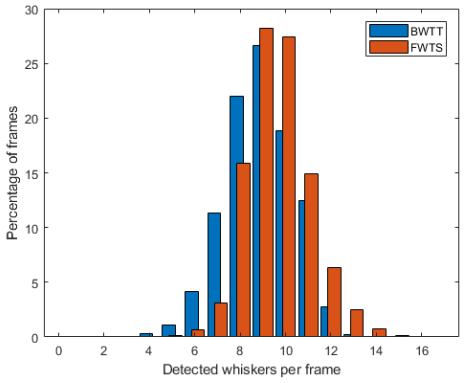
For FWTS, we can determine the detection rate by calculating the percentage of frames in which each specific whisker was detected. This cannot be done for BWTT, as none of the whiskers are tracked from beginning to end of a video: BWTT in itself does not have this function, and Spanke's script only detects short tracks. The detection rate for each whisker is shown in Table 8.1. One needs to keep in mind that the detection rate is highly dependent on the dynamics of the whiskers: if whiskers move more frequently and faster, the detection rates are expected to be lower. Nevertheless, the two video segments are comparable in this respect, as the frequency of the air puffs is the same for both videos. The detection rates are similar for most whiskers: roughly between 94% and 100%. A few whiskers (whisker 10 in Table 8.1a and whiskers 1 and 4 in Table 8.1b) have markedly lower scores; these are the whiskers that most likely were lost at some point and were not found back by the algorithm. The median detection rate per track for both fragments are 98.7% and 96.0%; on average 97.4%.

8.3 Precision of angle detection

The precision of the parametrization of whiskers depends on multiple factors, such as the quality of the video segment and the quality of the different detection and parametrization algorithms. Noise and measurement imprecisions are inevitable, but a good tracking system minimizes these. A good way to estimate the precision of the parametrization, is *taking a segment in which the movement of whiskers is so small and predictable that the changes in angle over time can be modeled by a lower-degree polynomial.* This polynomial can be estimated with the available measurements. Then, the differences between this

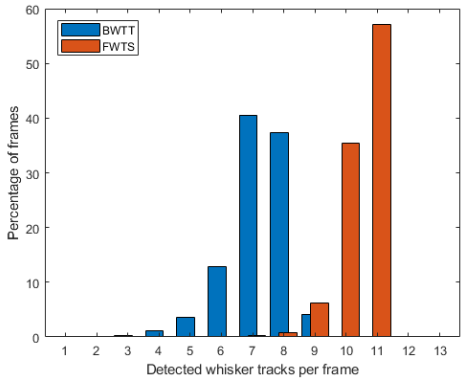


(a) Frames 26000-86000 of video A. $\mu_{\mathrm{BWTT}} = 7.8$ and $\sigma_{\mathrm{BWTT}} = 0.96$; $\mu_{\mathrm{FWTS}} = 11.5$ and $\sigma_{\mathrm{FWTS}} = 1.04$.

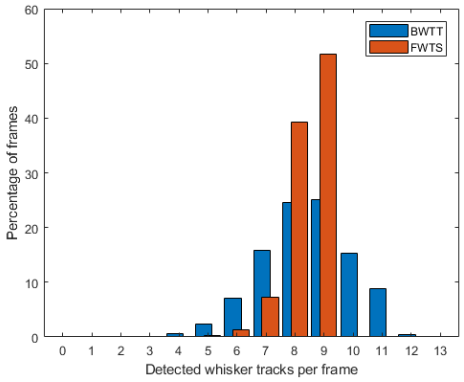


(b) Frames 60000-120000 of video B. $\mu_{\mathrm{BWTT}} = 8.9$ and $\sigma_{\mathrm{BWTT}} = 1.52$; $\mu_{\mathrm{FWTS}} = 9.7$ and $\sigma_{\mathrm{FWTS}} = 1.41$.

Figure 8.2: Histograms of the number of whiskers per frame for both videos.



(a) Frames 26000-86000 of video A. $\mu_{\mathrm{BWTT}} = 8.4$ and $\sigma_{\mathrm{BWTT}} = 1.52$; $\mu_{\mathrm{FWTS}} = 10.5$.



(b) Frames 60000-120000 of video B. $\mu_{\mathrm{BWTT}} = 7.2$ and $\sigma_{\mathrm{BWTT}} = 0.97$; $\mu_{\mathrm{FWTS}} = 8.4$.

Figure 8.3: Histograms of the number of tracks per frame for both videos. σ_{FWTS} is not defined, as the results for FWTS do not follow a Gaussian distribution.

(a) Video A

Whisker no.	Detection rate
1	99.4 %
2	98.9 %
3	94.4 %
4	96.2 %
5	97.4 %
6	97.8 %
7	99.0 %
8	99.3 %
9	98.8 %
10	68.0 %
11	98.7 %

(b) Video B

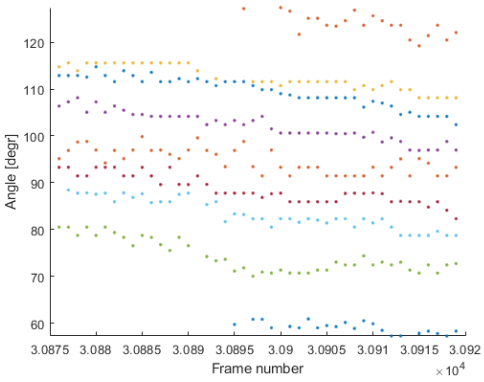
Whisker no.	Detection rate
1	83.8 %
2	96.0 %
3	95.7 %
4	84.7 %
5	92.2 %
6	97.8 %
7	97.8 %
8	97.9 %
9	94.8 %

Table 8.1: Detection rates of whiskers in FWTS for both video segments. The numbering of the whiskers is based on their arrangement on the snout, as detected by FWTS.

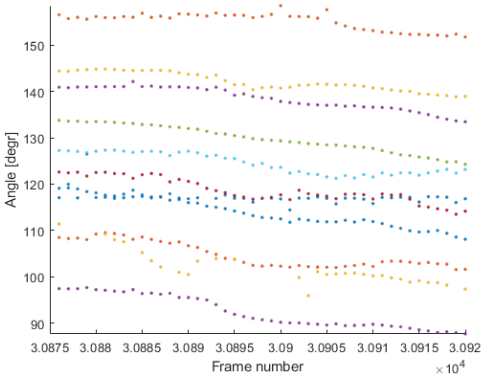
polynomial and the measurements can be assessed: if the whiskers move in a predictable way, the polynomial is a good estimation of the 'true' whisker position. The differences are expected to have a Gaussian distribution for which the standard deviation σ can be determined, and the less the measurements deviate from the polynomial (which is the best estimation of the 'true' whisker position we have), the more precise the system is. If the polynomial is a good estimation of the 'true' whisker position, a relatively large value for σ suggests that the measurements are relatively imprecise; a small value for σ means that the measurements are relatively precise. For the whisker videos that were analyzed, the segments in which the whisker movement can be accurately modelled by a lower-degree polynomial are short; sometimes 200 frames or more, but typically no longer than 60 frames.

From either video, three segments were chosen in which the whiskers move slowly and predictably. These segments have lengths between 25 and 100 frames. Plots of the angle over time, as measured by BWTT and FWTS, are shown in figures 8.4 and 8.5. For both BWTT and FWTS, the whisker angle over time per track was estimated by a fourth-degree polynomial, constructed using MATLAB's `polyfit` function: this gives a smooth curve, consistent with what we would expect from the 'true' whisker movement in the chosen segments. Then, for every whisker point of the track, the difference between the measured value and the polynomial was calculated. For every track, the standard deviation of these measurements was determined. Then, for every segment, the average standard deviation over all the tracks was calculated. Table 8.2 show the average standard deviation over all the tracks per segment for both systems. Tables containing the standard deviation per track for every segment can be found in Appendix A.

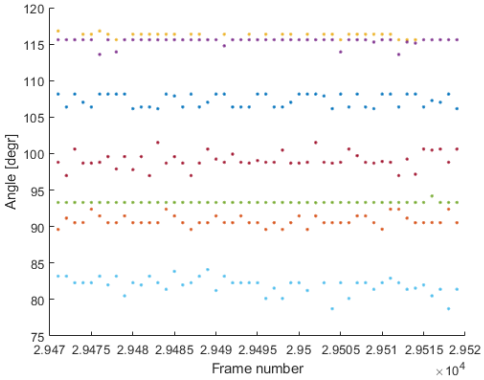
Figures 8.4 and 8.5 show the angles detected by BWTT and FWTS side by side for each fragment. Note that the systems detect different angles for the same whisker: this is partially due to the fact that the algorithms measure the angles at different places of



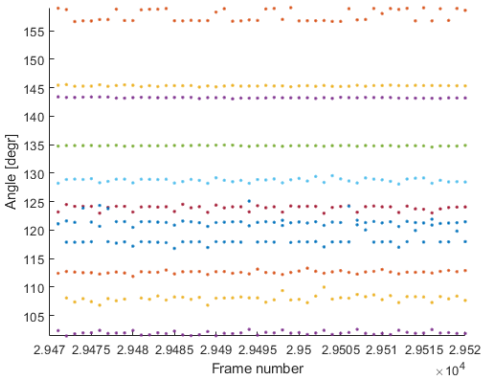
(a) Tracks in segment A1 (BWTT)



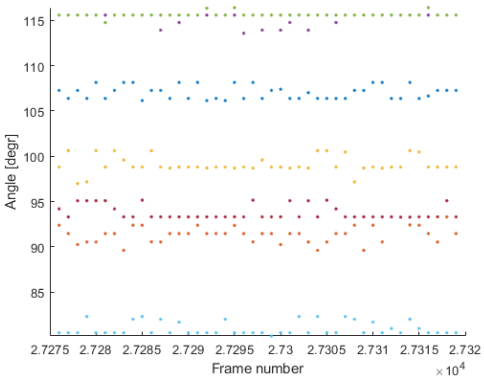
(b) Tracks in segment A1 (FWTS)



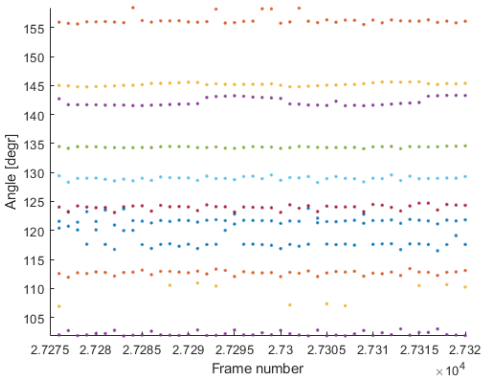
(c) Tracks in segment A2 (BWTT)



(d) Tracks in segment A2 (FWTS)

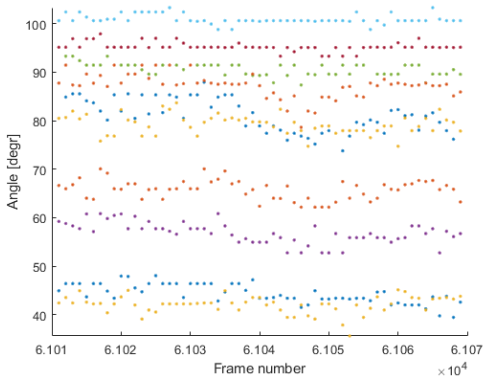


(e) Tracks in segment A3 (BWTT)

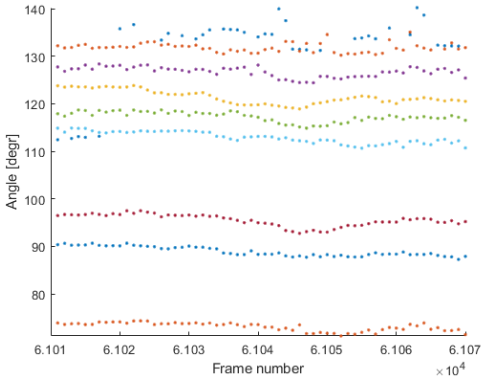


(f) Tracks in segment A3 (FWTS)

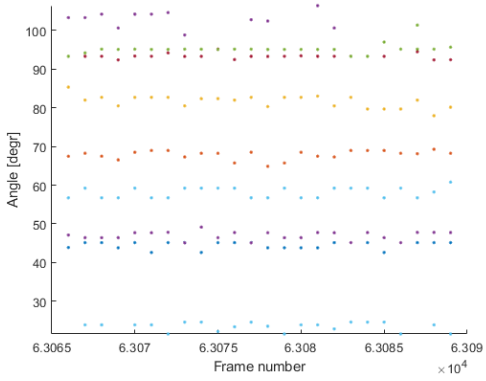
Figure 8.4: Segments from video A. Segment A1 contains frames 30875-30920. Segment A2 contains frames 29470-29520. Segment A3 contains frames 27275-27320.



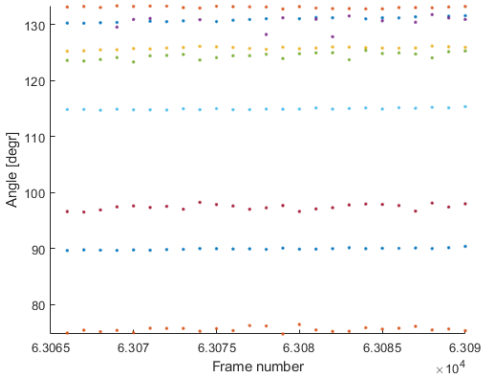
(a) Tracks in segment B1 (BWTT)



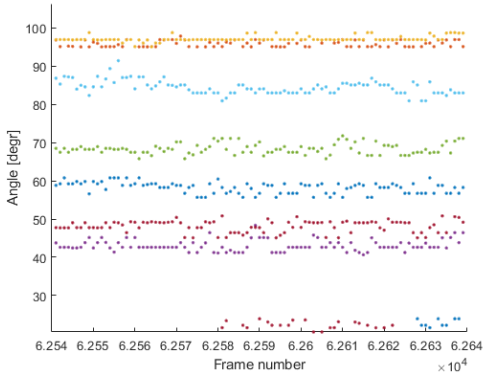
(b) Tracks in segment B1 (FWTS)



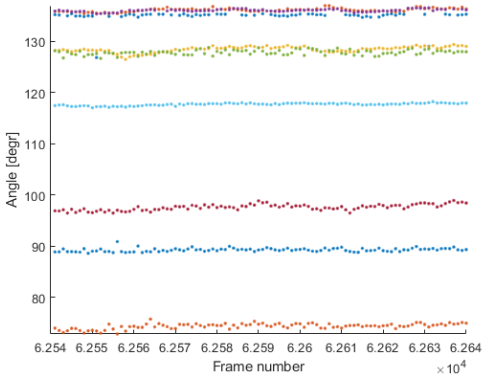
(c) Tracks in segment B2 (BWTT)



(d) Tracks in segment B2 (FWTS)



(e) Tracks in segment B3 (BWTT)



(f) Tracks in segment B3 (FWTS)

Figure 8.5: Segments from video B. Segment B1 contains frames 61010-61070. Segment B2 contains frames 63065-63090. Segment B3 contains frames 62540-62640.

Table 8.2: Average standard deviations over all the tracks per segment, for BWTT and FWTS. The names of the segments correspond to those given in figures 8.4 and 8.5.

Segment	Length (frames)	avg. σ BWTT	avg. σ FWTS
A1	46	1.10	0.60
A2	51	0.67	0.49
A3	46	0.67	0.47
B1	61	1.51	0.80
B2	36	1.08	0.29
B3	101	1.15	0.39
avg.	56.8	1.03	0.51

the whisker, but mainly because BWTT takes the center line of the snout as its base for angle calculation, whereas FWTS takes a line along the snout as its base. Furthermore, some whiskers that are detected by one algorithm remain undetected by the other one.

In the figures, it is visible that FWTS typically gives smoother whisker tracks than BWTT, which suggests that FWTS is more precise than BWTT. This is due to the fact that BWTT operates with only pixel precision on a small part of the whisker, whereas FWTS traces the full-length whisker with sub-pixel accuracy. Table 8.2 shows that, for every segment, the standard deviation of the difference between the polynomial and the whisker points is lower with FWTS than with BWTT. This means that for every segment, on average, FWTS determines the angle of the whiskers with a higher precision than BWTT. If we take the average standard deviation as a indicator for precision, we can state that the FWTS is $\frac{1.03}{0.51} = 2.02$ times as precise as BWTT.

8.4 Tracking quality

The quality of tracking (i.e. the identification whiskers over time) is partially related to the precision of the parameters and the detection rate. After all, if whiskers are detected more often and their parameters are measured with a higher precision, tracking becomes considerably easier. However, tracking is more than that: good tracking requires that the algorithm make the right classification decisions based on the characteristics of the detected whiskers. Furthermore, when whiskers move fast, align, cross, or hide, it is important that the tracking algorithm be robust, so that even if an error is made in one frame or whisker track, this does not make the rest of the tracking unreliable. For successful tracking, the whisker-tracking algorithm needs to succeed at two tasks:

- Distinguishing between different whiskers over time, even when whiskers cross or their appearance becomes similar.
- Redetecting whiskers that reappear after having hidden or remained undetected for some time.

Spanke's algorithm performs the first task by making a prediction for the position and angle of a whisker for every new frame, and then matching the detected whiskers to

those predictions, which it does pretty well given the limited precision of BWTT and the absence of information about the length and shape of whiskers. However, it does not perform the second task: whenever a whisker remains undetected for a certain number of frames, the algorithm terminates the track; the best one can hope for is that the whisker is rediscovered in future frames and a new track is created. The algorithm is not aware of the fact that both tracks describe the same whisker.

FWTS uses SVMs to performs both tasks, as it learns the characteristics of the whiskers over time, using data from previous frames as training data to identify whiskers in new frames. The pair of experts performs only the first task: it compares the position of whiskers in subsequent frames. It uses this information to correct the SVMs when they make mistakes, protecting the training data against corruption.

Figures 8.6 and 8.7 show a typical episode of whisker movement, in which the mouse sweeps its whisker several times. Figure 8.6 shows the analysis by BWTT and Spanke's algorithm. This particular segment contains 98 tracks: far more than the actual number of whiskers. Although Spanke's algorithm makes an attempt to distinguish different whiskers, it is impossible to assess the quality of the tracking of the data from BWTT, because the data is not precise and accurate enough. Figure 8.7 shows the same episode as analyzed by FWTS. Even though this plot only shows one parameter (the angle), it is possible to distinguish several whisker tracks. Furthermore, FWTS tracked all the whiskers from the beginning of the segment until the end: the segment contains 9 tracks, which is equal to the number of whiskers that were tracked.

Figures 8.8 and 8.9 show a selection from the same video fragment. Figure 8.8 shows the segment as analyzed by BWTT/Spanke's algorithm. The algorithm identified 44 tracks in this short segment. The actual number of whiskers visible per frame is much smaller, which means that the algorithm lost track of the whiskers several times. Figure 8.9 shows the segment as analyzed by FWTS. The quality of tracking is considerably higher in FWTS than in BWTT. Nevertheless, especially during retraction, FWTS's tracking method makes identification decisions that are different from what one would expect: whisker points that appear to be part of the same curve are sometimes assigned to different whisker tracks. These might be cases where the N-expert failed to identify the choice made by the SVM as an error. The frequency of these errors differs from segment to segment and from video to video. Appendix B shows plots of two more fragments per video, analyzed by both BWTT/Spanke's algorithm and FWTS, in which the mouse moves its whiskers fast.

In general, FWTS tracks whiskers with accuracy, and the fact that whiskers are tracked over the whole segment makes it possible to assess the movement of individual whiskers. This was also confirmed by our experts. Figure 8.10 individually shows two of the whiskers from the segment of Figure 8.7. The density of the data points during retraction is sometimes low, but in all cases, the algorithm successfully redetects the whiskers: before and after the period of fast whisker movement, the whisker tracks have not switched places (in Figure 8.7, the whisker tracks keep the same colors). In this way, FWTS makes it possible to reliably track individual whiskers in situations where this was impossible with BWTT.

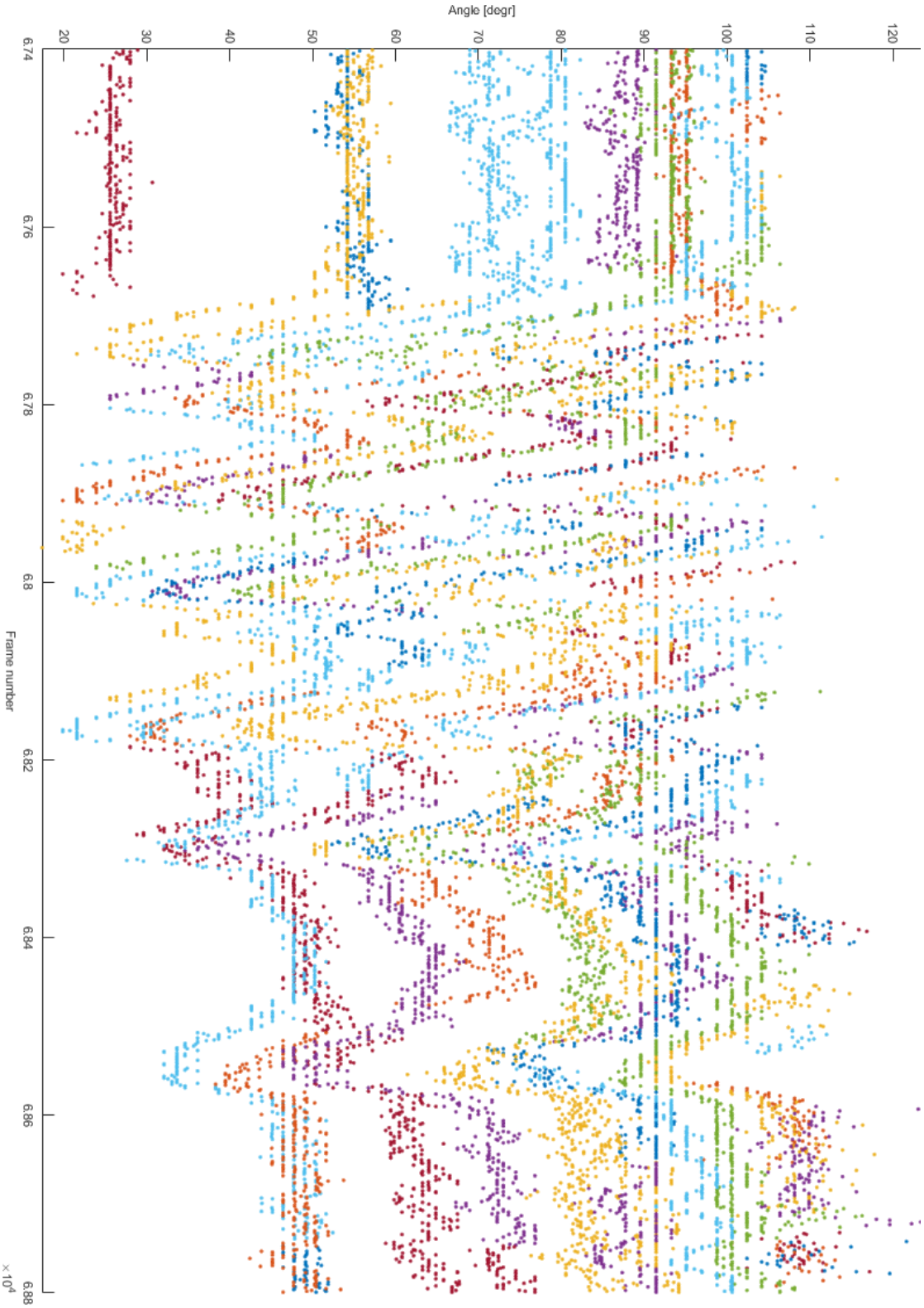


Figure 8.6: Detected whiskers and tracks on frames 67400-68800 of video B, as detected by BWTT and Spanke's algorithm. Different colors denote different tracks, but the number of tracks is higher than the number of colors available in MATLAB.

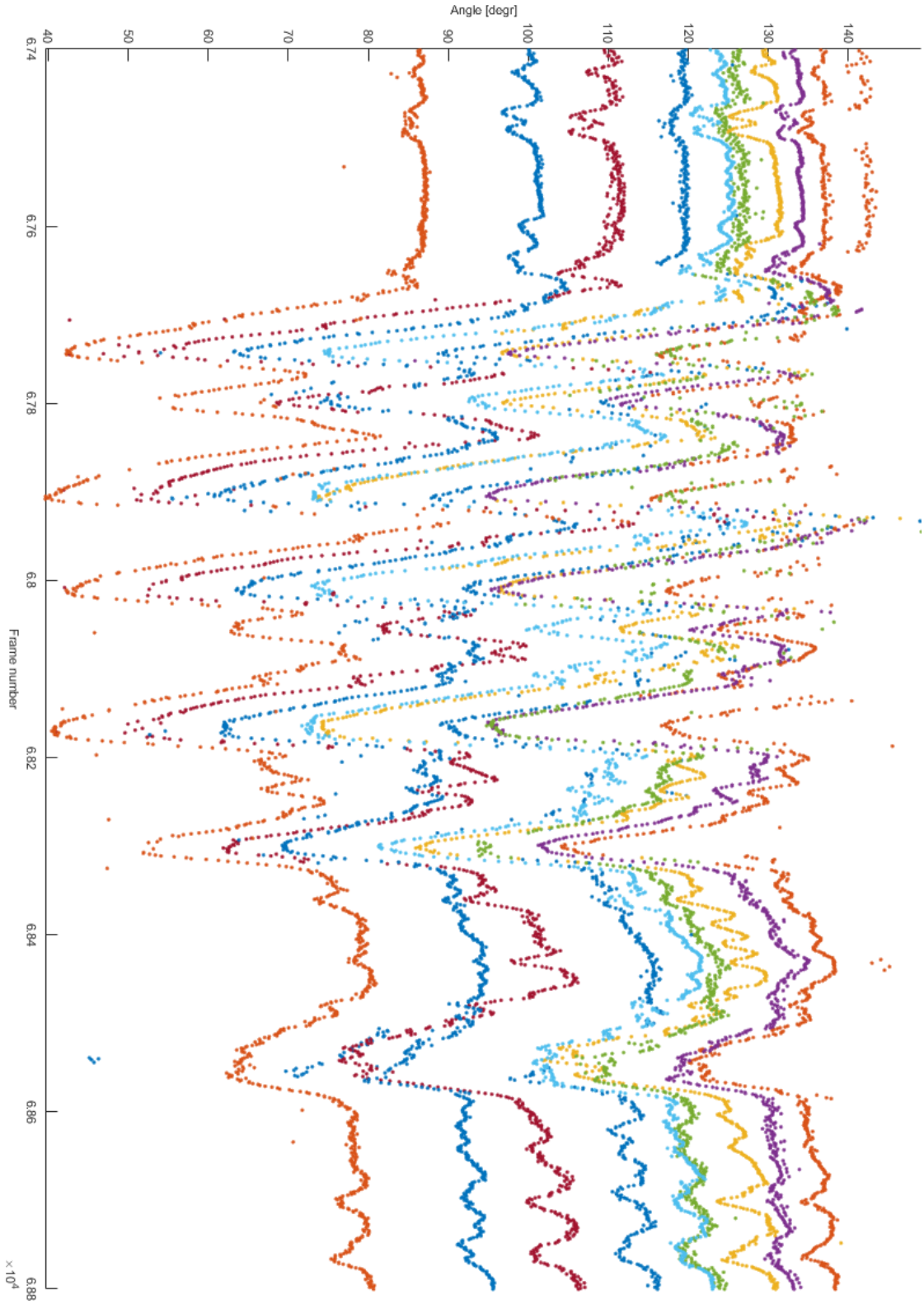


Figure 8.7: Detected whiskers on frames 67400-68800 of video B, as detected by FWTS. Different colors denote different tracks.

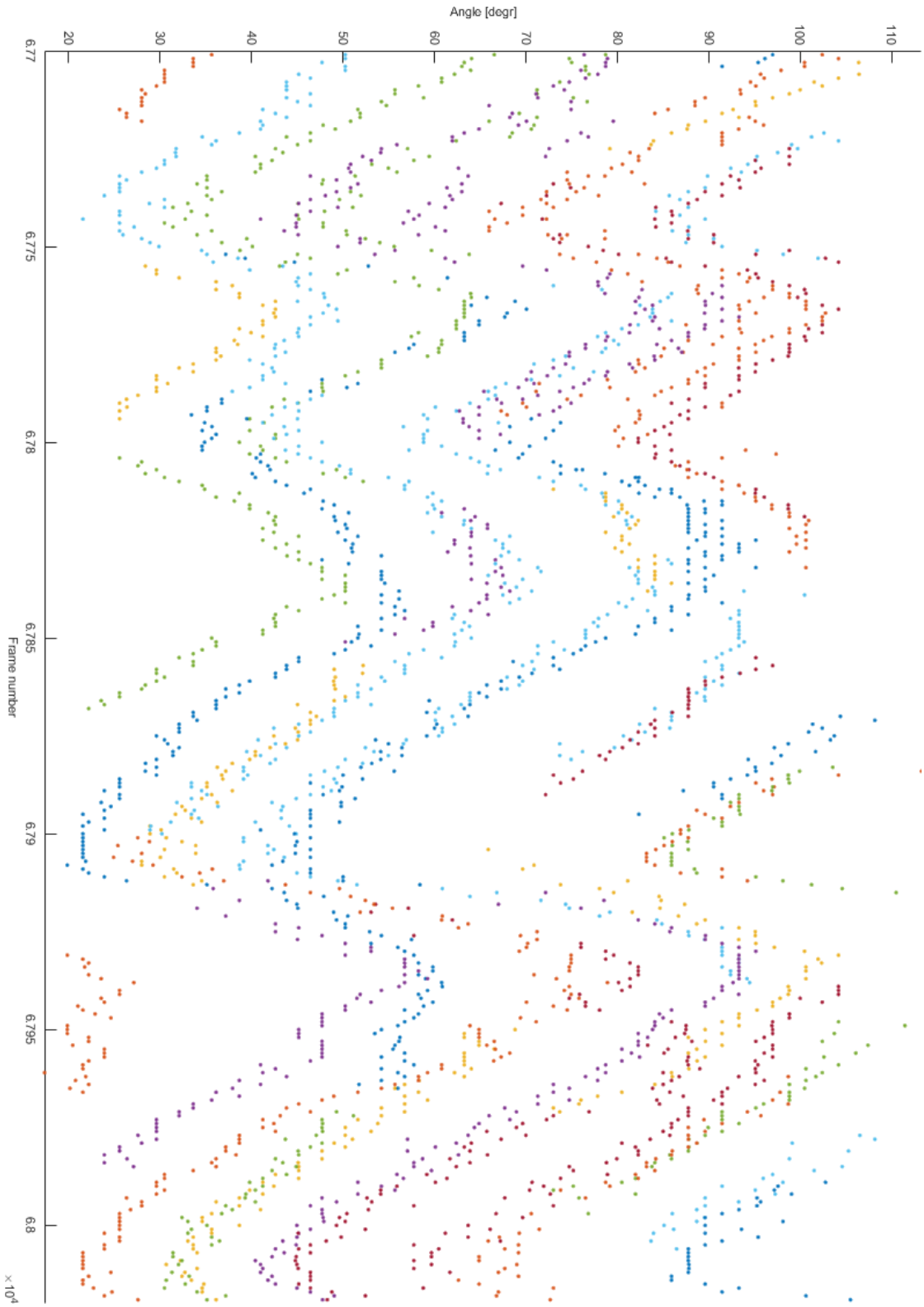


Figure 8.8: Detected whiskers on frames 67700-68020 of video B, as detected by BWTT. Different colors denote different tracks, but the number of tracks is higher than the number of colors available in MATLAB.

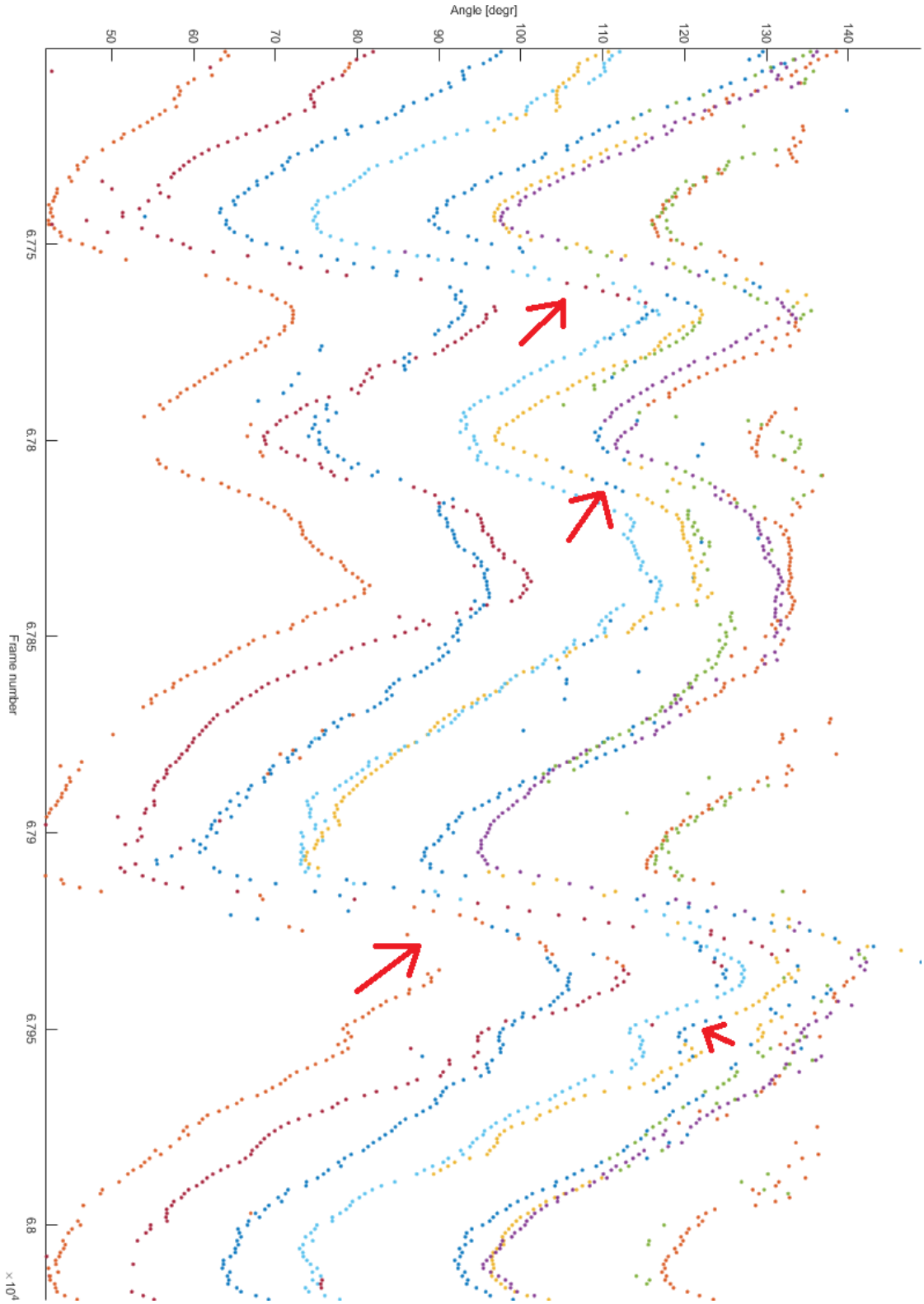
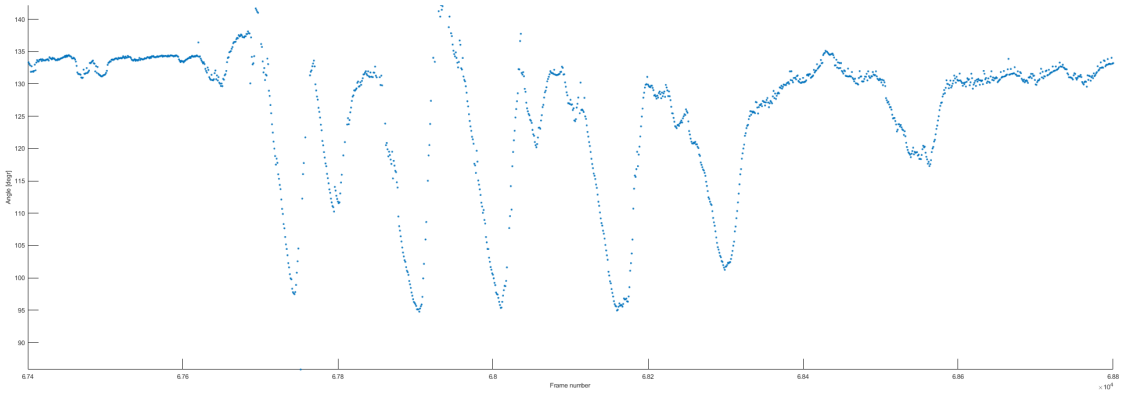
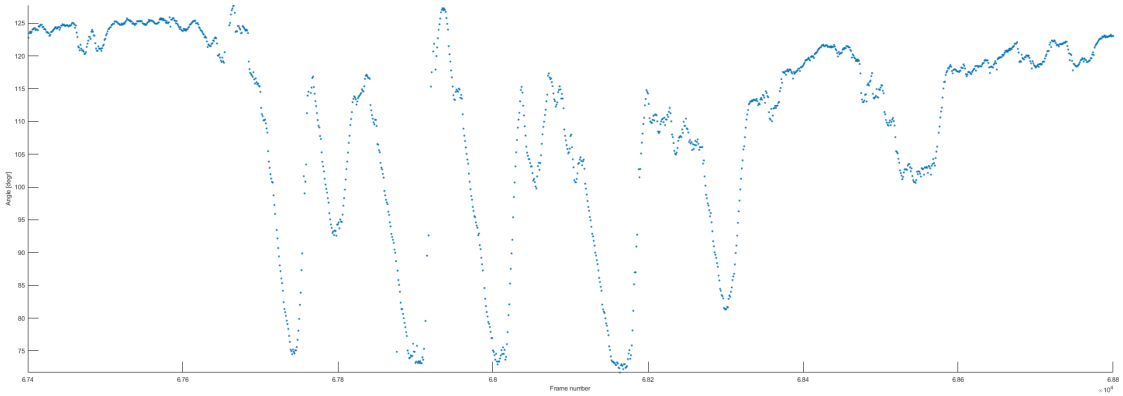


Figure 8.9: Detected whiskers on frames 67700-68020 of video B, as detected by FWTS. The arrows show places where the tracking result is different from what one would expect.



(a) Whisker 4



(b) Whisker 6

Figure 8.10: Two of the whisker tracks on frames 67700-68020, as detected by FWTS.

8.5 Processing speed

We were were able to use parallelism to decrease the processing time of FWTS per frame. The whisker-point detection step and the local-clustering algorithm were ported to the GPU with MATLAB's `gpuarray` function. The regression algorithm was parallelized on a per-whisker basis using MATLAB's `parfor` function. To compute the mean-square distance between the parametrization and the whisker points, the GPU, too, is used for parallelism.

To determine the processing speed of the algorithm, the processing times for the different steps of the program were measured using MATLAB's built-in stopwatch. The following steps were distinguished.

1. Lower-level processing
   (a) Preprocessing (such as background removal and blurring).
   (b) Whisker point detection (Steger's algorithm)
2. Intermediate-level processing
   (a) Clustering (Steger's algorithm)
   (b) Parametrization (Regression, as implemented by ourselves)
3. Application-level processing
   (a) Identification of whiskers (TLD)
   (b) Retraining SVMs

After the code profiling was finished, the average processing time per frame was calculated for each of the steps. The results are shown in Tables 8.3a and 8.3b. A remarkable difference between the two analyses is the fact that the parametrization step, on average, lasted almost 25% longer in video A than in video B. This is due to the fact that the duration of this step depends on the number of whiskers that are detected, and the number of iterations that is necessary to perform the parametrization. Parallelism was implemented using MATLAB's `parfor` function. On our machine, MATLAB was able to create a 'parallel pool' of 12 workers. The average number of detected whiskers per frame is higher in video A than in video B (as shown in Figure 8.2), which explains the difference in processing time. This observation, as well as the fact that the parametrization step is by far the slowest step of the algorithm, suggest that it is a good idea to explore alternatives to the implemented regression algorithm. As stated in Section 6.3.2, there are a plethora of different algorithms to obtain a least-square fit, and a more efficient algorithm might speed up a future implementation of the FWTS considerably.

For their analysis of BWTT, Ma et al. used a device with an Intel Xeon E5-2690 processor, running at a frequency of 2.9 GHz; the GPU was a Maxwell-based GeForce GTX Titan X GM200, on which he measured about 5.7 s per frame [9]. The version of MATLAB installed on that machine is R2015a. We were able to run a segment of 1000 frames from video A on this machine and determine the processing time per frame. The results are shown in Table 8.3c. Here, the parametrization took an even larger portion of

the processing time; nevertheless, the processing time per frame remained significantly lower than was reported for BWTT: 3.321 s per frame: a reduction in processing time of 41.7%. Ma et al. managed to reduce the processing time per frame for BWTT to 1.21 ms/frame [9]. The fact that the processing time per frame of the MATLAB version of FWTS is lower than in the MATLAB version of BWTT, suggests that real-time full-length whisker tracking is attainable.

The processing time per frame in the current MATLAB-implementation could be reduced further by replacing our own implementation of the regression algorithm by MATLAB's `nlinfit` function, which was tried at the last moment. It appears to give similar results when it comes to quality, but processes frames at a much higher speed. For videos A and B, the duration of the parametrization step on our hardware was reduced from 1614 ms and 1228 ms to 44.77 ms and 49.27 ms. This reduces the processing times per frame to less than 0.5 s per frame. Further efforts, such as porting the algorithm to a compiled language, would reduce to processing time even further. That, however, lies beyond the scope of this work.

Level	Algorithm step	time (ms)	% of time
	Preprocessing	35.60	1.79
	Whisker point detection	44.95	2.26
	Clustering	123.90	6.22
	Parametrization	1614.43	81.08
	Identification	44.39	2.23
	Retraining	124.82	6.27
Total		1991.09	100

(a) Results for video A

Level	Algorithm step	time (ms)	% of time
	Preprocessing	30.59	1.82
	Whisker point detection	43.61	2.60
	Clustering	145.56	8.67
	Parametrization	1228.02	73.10
	Identification	32.14	1.91
	Retraining	199.39	11.87
Total		1679.31	100

(b) Results for video B

Level	Algorithm step	time (ms)	% of time
	Preprocessing	13.43	0.40
	Whisker point detection	69.67	2.10
	Clustering	116.0	3.49
	Parametrization	3001	90.40
	Identification	71.18	2.14
	Retraining	49.32	1.49
Total		3321	100

(c) Results frames 2100-3100 from video A on the hardware Ma et al. used.

Table 8.3: Mean duration per frame for different steps of the algorithm, for both fragments on our own machine, and for a small segment of video A on the hardware Ma et al. used.

9 Conclusions

In this work, the Full-Whisker Tracking System (FWTS) was presented, which can be used to track individual whiskers in untrimmed, head-fixed rodents. In contrast to the existing BWTT system, FWTS, detects full-length whiskers and tracks them over time. This thesis has described the full algorithm. When designing the system, the structure as outlined in [21] was used as a guideline, dividing the system into low-, intermediate- and application-level techniques. The same structure was used in this work: every level was described in a separate chapter.

This concluding chapter consists of three sections. Section 9.1 gives an overview of the thesis. Section 9.2 summarizes the contributions made in this work. Section 9.3 concludes with a critical reflection on some of the steps taken and offers recommendations for future work on this topic.

9.1 Thesis overview

The goal of this thesis was the exploration of the possibilities to improve individual whisker tracking in untrimmed head-fixed rodents, by selecting existing algorithms and inventing new ones, in order to implement a working, quality-optimized whisker angle and position tracker in MATLAB.

As a first step, BWTT's output was analyzed, as well as the output from Spanke's algorithm (which was designed to process BWTT's output). It was found that the fact that BWTT only analyzes a thin band around the snout leads to the loss of valuable information about the whiskers, such as their shape and length. Our conclusion that BWTT is not very suitable for individual whisker tracking in untrimmed mice was confirmed by one of the designers of ViSA and BWTT, Igor Perkon. Spanke's algorithm does a good job in trying to analyze the output from BWTT in order to track single whiskers, but in order to track whiskers reliably, a redesign of BWTT was needed. Nevertheless, the preprocessing steps from BWTT, such as the background-removal algorithm, were still considered valuable and could be reused in a new whisker-tracking system.

In order to structure the difficult design process for a new whisker-tracking system, the new-algorithm components were divided into three steps: low-, intermediate-, and application-level processing, in accordance with Davies et al. [21]. Each of the steps contains several algorithms, and for each of the steps, several alternatives were assessed.

The low-level processing step was divided into preprocessing and whisker-point detection (Chapter 5). The preprocessing step contains several preprocessing algorithms from BWTT: background removal, gamma correction, and snout removal. For optimization, the options of Gaussian blur and deinterlacing were added. For whisker-point detection, three algorithms were considered: skeletonization, centerline detection, and local eccentricity detection. Centerline detection was chosen for implementation in FWTS, as it

detects whisker points with sub-pixel accuracy, it is able to distinguish between crossing whiskers, it gives information about the local direction of the whiskers, and it is highly parallelizable.

For intermediate-level processing (Chapter 6), the Hough transform, which is the most widely used intermediate-level processing algorithm, was assessed. However, it was concluded that this algorithm is not suitable for full length whisker tracking: the large number of whisker points would make a Hough-based algorithm computationally expensive; the fact that whiskers bend in different ways made employing the Hough transform even more problematic. Therefore, a two-step approach was taken: whisker points were clustered in such a way that each cluster represents an individual whisker, and then each cluster was parametrized. For clustering, DBSCAN and local clustering were assessed. Local clustering was chosen, as this takes the local direction of a whisker into account, which leads to better clustering decisions. To finish the clustering process for overlapping and partially hidden whiskers, which is a highly application-specific problem, an algorithm was designed in which collinear clusters of whisker points are merged, so that each cluster represents exactly one whisker. For parametrization, local linearization and regression were considered. The regression algorithm was chosen, as it allows a whisker to be described accurately in a smaller number of parameters, and complies with the compactness hypothesis, which makes tracking over time easier.

For application-level processing (Chapter 7), algorithms for recognition and for tracking were assessed. For tracking, a Kalman-filter and a whisker-point-fitting algorithm were assessed. For recognition, a SVM-based algorithm and a k-NN-based algorithm were assessed. Apart from that, an algorithm that makes it possible to combine recognition and tracking, TLD, was studied. Eventually, it was decided to implement TLD, with the SVM-based algorithm at its core, and a Kalman filter as an auxiliary, as this makes it possible to combine the strong points of recognition and tracking.

A block diagram, summarizing the design process was given in Figure 1.1. The algorithms we chose were combined into one single tracking system in MATLAB, which we called FWTS. Segments from two videos were analyzed with the system. The quality of the results was assessed by comparing it to a BWTT analysis of the same fragments, on three criteria: detection rate, precision of angle detection, and tracking quality. On the criteria where a comparison was possible, FWTS performed considerably better than BWTT. Furthermore, FWTS is able to track whiskers over time, which is something that BWTT cannot.

We were able to use parallelism to decrease the processing time of FWTS per frame. The whisker-point detection step and the local-clustering algorithm were ported to the GPU. The regression algorithm was parallelized on a per-whisker basis. To compute the mean-square distance between the parametrization and the whisker points, the GPU, too, is used for parallelism. The processing time per frame in FWTS is, on average for our segments, 41.7% lower than Yang Ma reported for BWTT on the same hardware.

9.2 Contributions

- A detailed exploration of the existing whisker-tracking techniques, BWTT in particular, was performed.

- A broad variety of algorithms from the fields of image processing and machine learning were applied to the specific problem of whisker tracking in top-view videos of untrimmed, head-fixed mice, and the results were assessed.
- A new way to parametrize mouse whiskers was proposed, and several new algorithms to achieve this parametrization were invented and implemented.
- Algorithms were compared and combined to create a new whisker-tracking system, the FWTS, in MATLAB that takes video segments as its input and produces parametrized whisker traces as its output.
- Parallelism was implemented in FWTS.
- The tracking system was tested on segments of 60 seconds from two different, challenging videos, in which air puffs, to which the mice react, are given every two seconds. FWTS's performance was evaluated on different criteria: detection rate, precision, and tracking quality, with the following results:
  - On average for our segments, FWTS's detection rate is 21.2% higher than BWTT
  - Furthermore, the measured precision of FWTS is 2.02x higher than it is in BWTT
  - In contrast to earlier whisker trackers, FWTS is able to reliably track individual whiskers over time, and has the ability to find back whiskers that reappear.
  - The processing time per frame for FWTS is 41.7% lower than for BWTT on the same hardware. Furthermore, average processing times of less than 0.5 s per frame in MATLAB are within reach.

9.3 Discussion and future work

The goal of this thesis was to design a quality-optimized whisker tracking system that can track individual whiskers accurately. As tracking whiskers in top-view videos of untrimmed mice without available training data is a complicated and specialized procedure, a system for automated whisker tracking consists of different algorithms. Some of these algorithms might have been used for this purpose before (such as the steps adopted from BWTT); some of these algorithms were invented by others, but had not been applied to this specific problem before (such as the centerline detection algorithm and DBSCAN); and other ones were invented for this specific purpose (such as the cluster-stitching algorithm).

Such a design process inevitably has a heuristic component: although for every step, available literature is studied, alternatives are compared and strong and weak points are taken into consideration in order to account for their choices, different researchers may take different approaches. Even if for every step, the best algorithms are chosen, this does not guarantee a globally optimal result. The fact that researchers have taken radically different approaches to whisker tracking in the past (as became clear in Chapter

3 of this work) shows that this applies for this thesis, too. In our system, we decided for quality optimization: algorithms that were considered best for the criteria we defined, were chosen for implementation in FWTS. For us, quality was the main criterion, as we wanted to create a robust base for future work. However, for different criteria, different design choices could be made. This applies especially for the regression algorithm, which is used for parametrization: although the algorithm is able to find optimal values for the given parameters with high precision, its long processing time and unpredictability when it comes to the number of iterations could make it less suitable for a speed-optimized version of FWTS. The use of MATLAB's `nlinfit` function would resolve this issue to a great extent, but further testing is needed to see if this function gives results that are of the same quality as our own implementation of the regression algorithm.

MATLAB was chosen to implement this first working version of FWTS. Although MATLAB is well-suited for designing algorithms because of its wide range of built-in functions and possibilities to visualize data, it is by no means the best environment for speed-optimization. We achieved a per-frame processing time that is lower than that of BWTT (which, too, was built in MATLAB), but if one really wants to optimize for speed, a compiled language (such as C or C++), rather than an interpreted language, should be used. Ma et al. [9] did such an optimization for BWTT; a comparable optimization is possible for FWTS in the future. In addition, the possibilities for implementing more parallelism could be researched. It is possible that some of the algorithms can be streamlined or replaced by other algorithms of which the quality is comparable to the ones implemented in this work, but are faster or more parallelizable.

FWTS was tested on segments from two videos of experiments. Two different mice were used in either experiment, and since an air puff is given every two seconds, many different episodes of whisker movement were analyzed. The videos are challenging and sufficiently different to give an indication of the quality of the tracking system. However, it is not enough to draw definitive conclusions about all top-view videos of head-fixed mice. Computer vision in itself is a non-deterministic process: it is impossible to predict the response of the system to every single possible input frame, let alone every single possible segment, but more testing will give more guarantees about the quality of tracking. One part for which this is necessary, is the tracking algorithm. In the original TLD, the authors stated that the training set does not deteriorate over time. In our tests in Chapter 8, we saw that deterioration of the training data occurred with some whiskers, but most whiskers were tracked successfully over a segment in which multiple air puffs were given.

There might be an opportunity for improvement when it comes to the tracking/recognition algorithm. The algorithm that was implemented (based on the principle of TLD, using a Kalman-filter and an SVM-based system) recognizes and tracks whiskers in isolation, without looking at the whiskers around it. However, when a human observer tries to track or recognize objects, they do look at the environment. For example, one might recognize a particular short whisker easily if it is constantly flanked by two longer whiskers. In other words, there is information in the environment that can be used to improve tracking, that is not used by the current system. An algorithm that takes a more holistic approach, instead of viewing whiskers in isolation, is likely to make fewer mistakes.

A last point that needs to be taken into consideration is the fact that FWTS requires quite a large number of user-configured parameters. In BWTT, the parameters were mostly limited to the preprocessing stage; the fact that FWTS consists of more algorithms and does not only detect, but also describe and track whiskers led to the number of of parameters to be higher. In the two videos we used for testing, this did not lead to many problems: for the intermediate- and application-level steps, the parameters were not changed. Only for the parameters for lower-level processing, some adjustments were needed. However, since we only tested on two videos, it is not yet clear whether this applies to all whisker videos. Testing the system on a large number of video segments with different mice will make clear whether the large number of user-defined parameters is a practical problem, and whether it is possible to have those parameters set up automatically, instead of manually.

The system presented in this thesis shows that tracking of individual whiskers in untrimmed, head-fixed mice is possible by detecting and parametrizing the full-length whiskers. Moreover, it is possible to do this with reasonable speed, even faster than BWTT, which only detects a small part of the whisker. The thesis presented a system, constructed by combining different algorithms and methods. The system, FWTS, performed better than BWTT on three criteria in segments from two videos with two different mice. Furthermore, it was able to track individual whiskers over time, which is not possible with BWTT. Speeding up individual whisker tracking to the point that the processing time per frame is less than 1 ms would allow real-time processing of frames during the experiment, and would allow the researchers to close the feedback loop; i.e. adjust their experiments immediately based on the processed data. This would open up many new possibilities. Ma et al. almost achieved such a speedup for BWTT. The fact that the processing time per frame of the MATLAB version of FWTS is already lower than in the MATLAB version of BWTT, suggests that real-time full-length whisker tracking is attainable.

Bibliography

[1] R. W. Berg and D. Kleinfeld, "Rhythmic whisking by rat: Retraction as well as protraction of the vibrissae is under active muscular control," *Journal of Neurophysiology*, vol. 89, no. 1, pp. 104–117, 2003, pMID: 12522163. [Online]. Available: https://doi.org/10.1152/jn.00600.2002

[2] scikit image. (2018) Straight line hough transform. [Online]. Available: http://scikit-image.org/docs/dev/auto_examples/edges/plot_line_hough_transform.html

[3] Y. Berenguer, L. Pay, M. Ballesta, and O. Reinoso, "Position estimation and local mapping using omnidirectional images and global appearance descriptors," *Sensors*, vol. 15, no. 10, pp. 26 368–26 395, 2015. [Online]. Available: http://www.mdpi.com/1424-8220/15/10/26368

[4] K. Salton. (2017) How dbscan works and why should we use it? [Online]. Available: https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80

[5] E. Tu, Y. Zhang, L. Zhu, J. Yang, and N. Kasabov, "A graph-based semi-supervised k nearest-neighbor method for nonlinear manifold distributed data classification," *Information Sciences*, vol. 367-368, pp. 673 – 688, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025516305023

[6] OpenCV. (2018) Introduction to support vector machines. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

[7] I. Perkon, A. Koir, P. M. Itskov, J. Tasi, and M. E. Diamond, "Unsupervised quantification of whisking and head movement in freely moving rodents," *Journal of Neurophysiology*, vol. 105, no. 4, pp. 1950–1962, 2011, pMID: 21307326. [Online]. Available: https://doi.org/10.1152/jn.00764.2010

[8] G. Gyory, B. Mitchinson, G. Gordon, I. Perkon, V. Rankov, T. Prescott, and R. Grant, *An algorithm for automatic tracking of rat whiskers*, 2010, p. 14. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/VAIB10PAPERS/gyoryvaib.pdf

[9] Y. Ma, "Towards Real-Time Detection and Tracking of Spatio-Temporal Features: Rodent Whisker Tracking," Master's thesis, Delft Univerity of Technology, The Netherlands, 2017.

[10] P. M. Knutsen, D. Derdikman, and E. Ahissar, "Tracking whisker and head movements in unrestrained behaving rodents," *Journal of Neurophysiology*, vol. 93, no. 4, pp. 2294–2301, 2005, pMID: 15563552. [Online]. Available: https://doi.org/10.1152/jn.00718.2004

[11] N. G. Clack, D. H. O'Connor, D. Huber, L. Petreanu, A. Hires, S. Peron, K. Svoboda, and E. W. Myers, "Automated tracking of whiskers in videos of head fixed rodents," *PLOS Computational Biology*, vol. 8, no. 7, pp. 1–8, 07 2012. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1002591

[12] S. Roy, J. Bryant, Y. Cao, and D. Heck, "High-precision, three-dimensional tracking of mouse whisker movements with optical motion capture technology," *Frontiers in Behavioral Neuroscience*, vol. 5, p. 27, 2011. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnbeh.2011.00027

[13] C. Steger, "An unbiased detector of curvilinear structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 2, pp. 113–125, Feb 1998.

[14] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.

[15] V. Romano *et al.*, "Adaptation of whisker movements requires cerebellar potentiation," submitted.

[16] B. Mitchinson *et al.* (2015) Biotact whisker tracking tool. [Online]. Available: http://bwtt.sourceforge.net

[17] G. Smaragdos, R. Kukreja, I. Sourdis, Z. Al-Ars, C. Kachris, D. Soudris, and C. Strydis, "Brainframe: A node-level heterogeneous accelerator platform for neuron simulations," *Journal of Neural Engineering*, vol. 14, November 2017.

[18] H. D. Nguyen, Z. Al-Ars, G. Smaragdos, and C. Strydis, "Accelerating complex brain-model simulations on gpu platforms," in *Proc. 18th Design, Automation Test in Europe conference*, Grenoble, France, March 2015.

[19] N. Rahmati, C. B. Owens, L. W. J. Bosman, J. K. Spanke, S. Lindeman, W. Gong, J.-W. Potters, V. Romano, K. Voges, L. Moscato, S. K. E. Koekkoek, M. Negrello, and C. I. De Zeeuw, "Cerebellar potentiation and learning a whisker-based object localization task with a time response window," *Journal of Neuroscience*, vol. 34, no. 5, pp. 1949–1962, 2014. [Online]. Available: http://www.jneurosci.org/content/34/5/1949

[20] Y. Ma, P. R. Geethakumari, G. Smaragdos, S. Lindeman, V. Romano, M. Negrello, I. Sourdis, L. W. Bosman, C. I. De Zeeuw, Z. Al-Ars *et al.*, "Towards real-time whisker tracking in rodents for studying sensorimotor disorders."

[21] E. Davies, *Machine Vision: Theory, Algorithms, Practicalities*, 2nd ed. Academic Press, 1997.

[22] R. Duin and D. Tax, "Statistical pattern recognition," in *Handbook of Pattern Recognition and Computer Vision*, 3rd ed., C. Chen and P. Wang, Eds. World Scientific, 2005, ch. 1, pp. 3–24.

[23] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[24] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.

[25] NVIDIA. (2018) What is gpu-accelerated computing. [Online]. Available: http://www.nvidia.com/object/what-is-gpu-computing.html

[26] I. Perkon, private communication, Jul. 2017.

[27] S. Venkatraman, K. Elkabany, J. D. Long, Y. Yao, and J. M. C. *, "A system for neural recording and closed-loop intracortical microstimulation in awake rodents," *IEEE Transactions on Biomedical Engineering*, vol. 56, no. 1, pp. 15–22, Jan 2009.

[28] B. Heisele, "Motion-based object detection and tracking in color image sequences," 01 2000.

[29] P. Smets and B. Ristic, "Kalman filter and joint tracking and classification based on belief functions in the tbm framework," *Inf. Fusion*, vol. 8, no. 1, pp. 16–27, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.inffus.2005.06.004

[30] MathWorks. (2018) Documentation: imadjust. [Online]. Available: https://nl.mathworks.com/help/images/ref/imadjust.html

[31] I. MathWorks. (2018) Documentation: imagesc. [Online]. Available: https://nl.mathworks.com/help/matlab/ref/imagesc.html

[32] E. Cheever. (2015) Eigenvalues and eigenvectors. [Online]. Available: http://lpsa.swarthmore.edu/MtrxVibe/EigMat/MatrixEigen.html

[33] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." AAAI Press, 1996, pp. 226–231.

[34] D. Dave, C. Strydis, and G. N. Gaydadjiev, "Impede: A multidimensional design-space exploration framework for biomedical-implant processors," in *ASAP 2010 - 21st IEEE International Conference on Application-specific Systems, Architectures and Processors*, July 2010, pp. 39–46.

[35] B. Babenko, M. hsuan Yang, and S. Belongie, "Visual tracking with online multiple instance learning," 2009.

List of definitions

Cluster A collection of coordinate pairs.

Clustering The process of assigning coordinate pairs to clusters.

Detection Discovering the presence of an object.

Frame An image from a video. Multiple frames form a video or a video segment.

Parametrization (1) A parametric representation of an object. (2) The process of creating a parametric representation of an object.

Recognition Discovering the fact that a detected object is the same as an object that was seen earlier.

Tracking The process of following an object over time in subsequent frames.

Whisker point A coordinate pair in Euclidean space that was detected as part of a whisker in a frame.

Standard deviations per track

A

This appendix contains, for every segment defined in Section 8.3, information per track on the standard deviation of the difference between the fitted polynomial over the segment and the measured angle, for BWTT and FWTS. A more detailed description of how these values are calculated can be found in Section 8.3. Apart from the standard deviations, the detection rate per track, which is defined as the fraction of frames of the segment in which the track was present, was added. In Tables A.1 and A.2, the whisker tracks are places in descending order of detection rate; since the systems have different ways of defining tracks, a one-on-one comparison between individual whiskers is not possible.

Table A.1: Standard deviation σ per whisker track for BWTT. Between parentheses, the detection rate.

	A1	A2	A3	B1	B2	B3
1	1.69 (1.00)	1.07 (1.00)	0.91 (1.00)	1.83 (1.00)	1.39 (1.00)	1.41 (1.00)
2	0.97 (1.00)	0.87 (1.00)	0.90 (1.00)	1.83 (1.00)	1.30 (1.00)	1.39 (1.00)
3	0.86 (1.00)	0.71 (1.00)	0.74 (1.00)	1.61 (1.00)	1.06 (1.00)	1.35 (1.00)
4	0.76 (1.00)	0.50 (1.00)	0.72 (1.00)	1.47 (1.00)	0.98 (1.00)	1.26 (1.00)
5	0.71 (1.00)	0.37 (1.00)	0.68 (1.00)	1.28 (1.00)	0.91 (1.00)	0.82 (1.00)
6	1.09 (0.98)	0.12 (1.00)	0.53 (1.00)	1.09 (1.00)	0.69 (1.00)	0.64 (1.00)
7	1.04 (0.95)	1.03 (0.98)	0.24 (1.00)	0.87 (1.00)	0.64 (1.00)	1.48 (0.99)
8	0.81 (0.50)			0.77 (1.00)	1.22 (0.88)	0.90 (0.22)
9	1.95 (0.47)			1.79 (0.98)	2.63 (0.50)	1.05 (0.09)
10				2.50 (0.94)	0.00 (0.04)	
avg.	1.10 (0.88)	0.67 (1.00)	0.67 (1.00)	1.51 (0.99)	1.08 (0.84)	1.15 (0.81)

Table A.2: Standard deviation σ per whisker track for FWTS. Between parentheses, the detection rate.

	A1	A2	A3	B1	B2	B3
1	1.43 (1.00)	1.69 (1.00)	1.50 (1.00)	0.61 (1.00)	0.40 (1.00)	0.44 (1.00)
2	0.68 (1.00)	0.96 (1.00)	0.70 (1.00)	0.49 (1.00)	0.39 (1.00)	0.38 (1.00)
3	0.59 (1.00)	0.66 (1.00)	0.59 (1.00)	0.48 (1.00)	0.38 (1.00)	0.35 (1.00)
4	0.47 (1.00)	0.45 (1.00)	0.40 (1.00)	0.42 (1.00)	0.13 (1.00)	0.34 (1.00)
5	0.47 (1.00)	0.32 (1.00)	0.36 (1.00)	0.42 (1.00)	0.11 (1.00)	0.32 (1.00)
6	0.39 (1.00)	0.29 (1.00)	0.36 (1.00)	0.39 (1.00)	0.08 (1.00)	0.25 (1.00)
7	0.37 (1.00)	0.27 (1.00)	0.32 (1.00)	0.28 (1.00)	0.07 (1.00)	0.11 (1.00)
8	0.32 (1.00)	0.08 (1.00)	0.29 (1.00)	0.94 (0.97)	0.09 (0.81)	0.19 (0.92)
9	0.31 (1.00)	0.06 (1.00)	0.15 (1.00)	3.13 (0.64)	0.98 (0.58)	1.18 (0.71)
10	0.19 (1.00)	0.06 (1.00)	0.10 (1.00)			
11	1.33 (0.67)	0.52 (0.96)	0.44 (0.24)			
avg.	0.60 (0.97)	0.49 (1.00)	0.47 (0.93)	0.80 (0.96)	0.29 (0.93)	0.39 (0.96)

Angle of whiskers tracks over time

B

This appendix contains four plots of different episodes of whisker movement from video A and B, as analyzed by BWTT and FWTS. The plots allow comparison of tracking quality and precision for both systems, as explained in Section 8.4

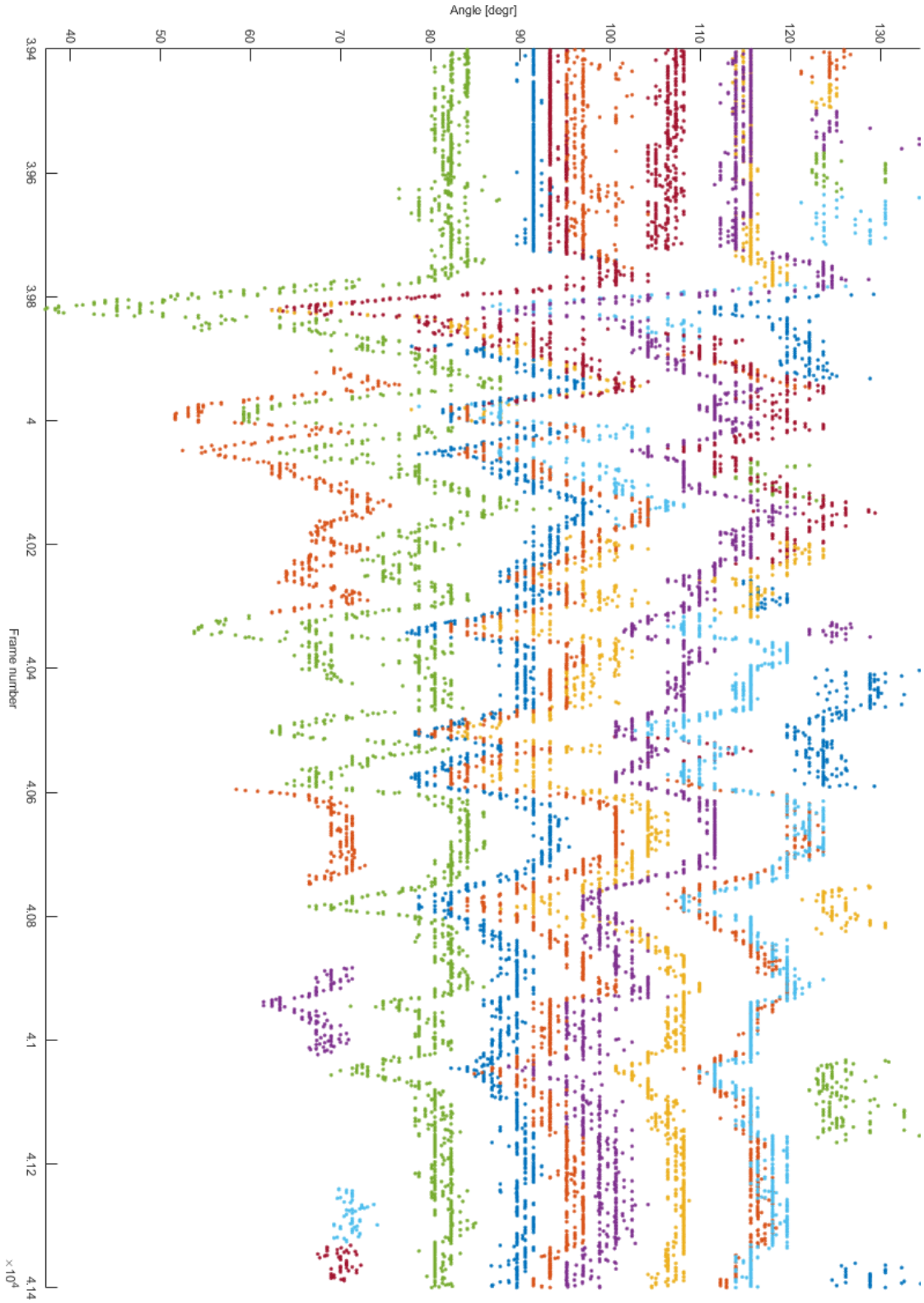


Figure B.1: Detected whiskers and tracks on frames 39400-41400 of video A, as detected by BWTT and Spanke's algorithm. Different colors denote different tracks, but the number of tracks is higher than the number of colors.

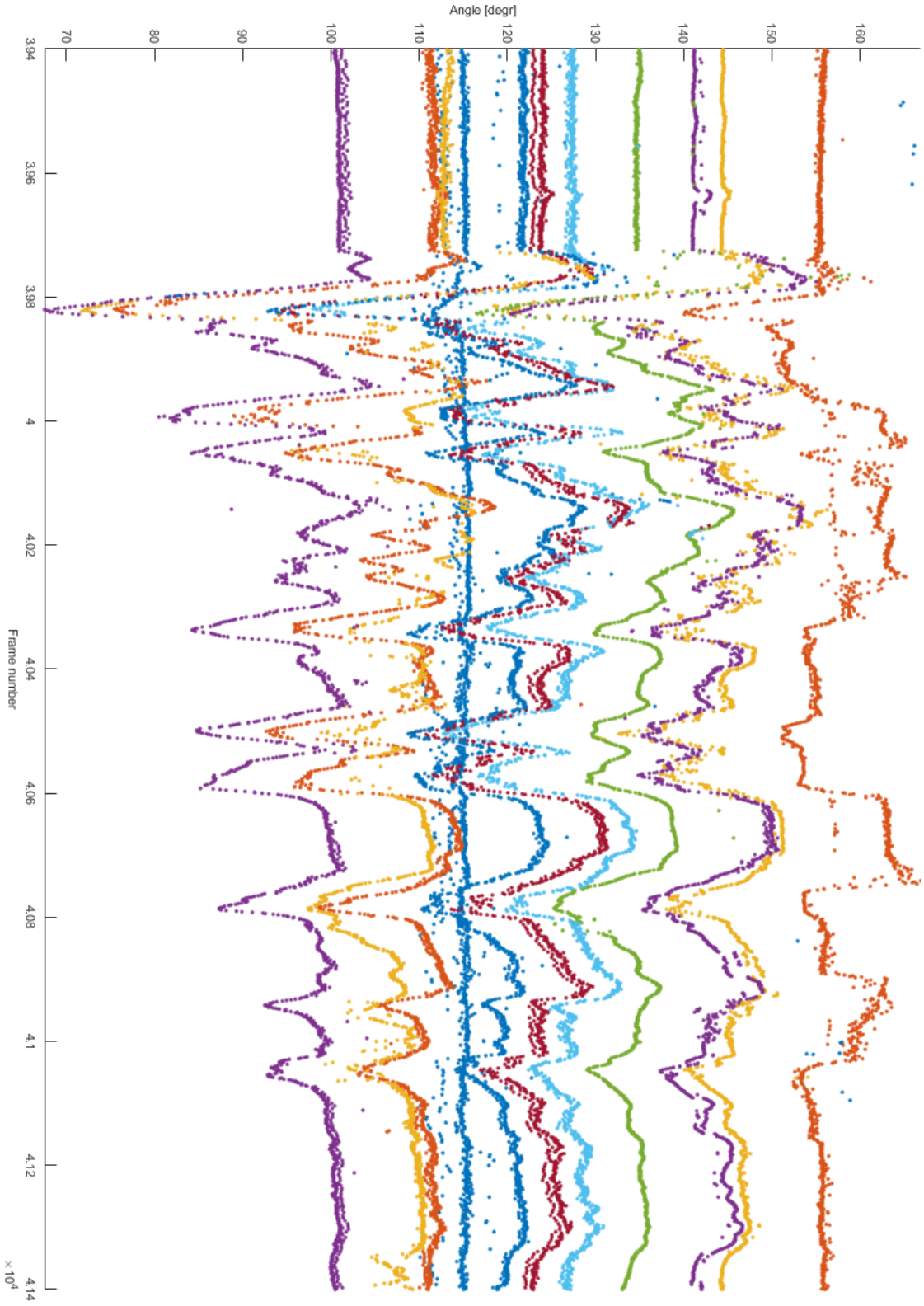


Figure B.2: Detected whiskers on frames 39400-41400 of video A, as detected by FWTS. Different colors denote different tracks.

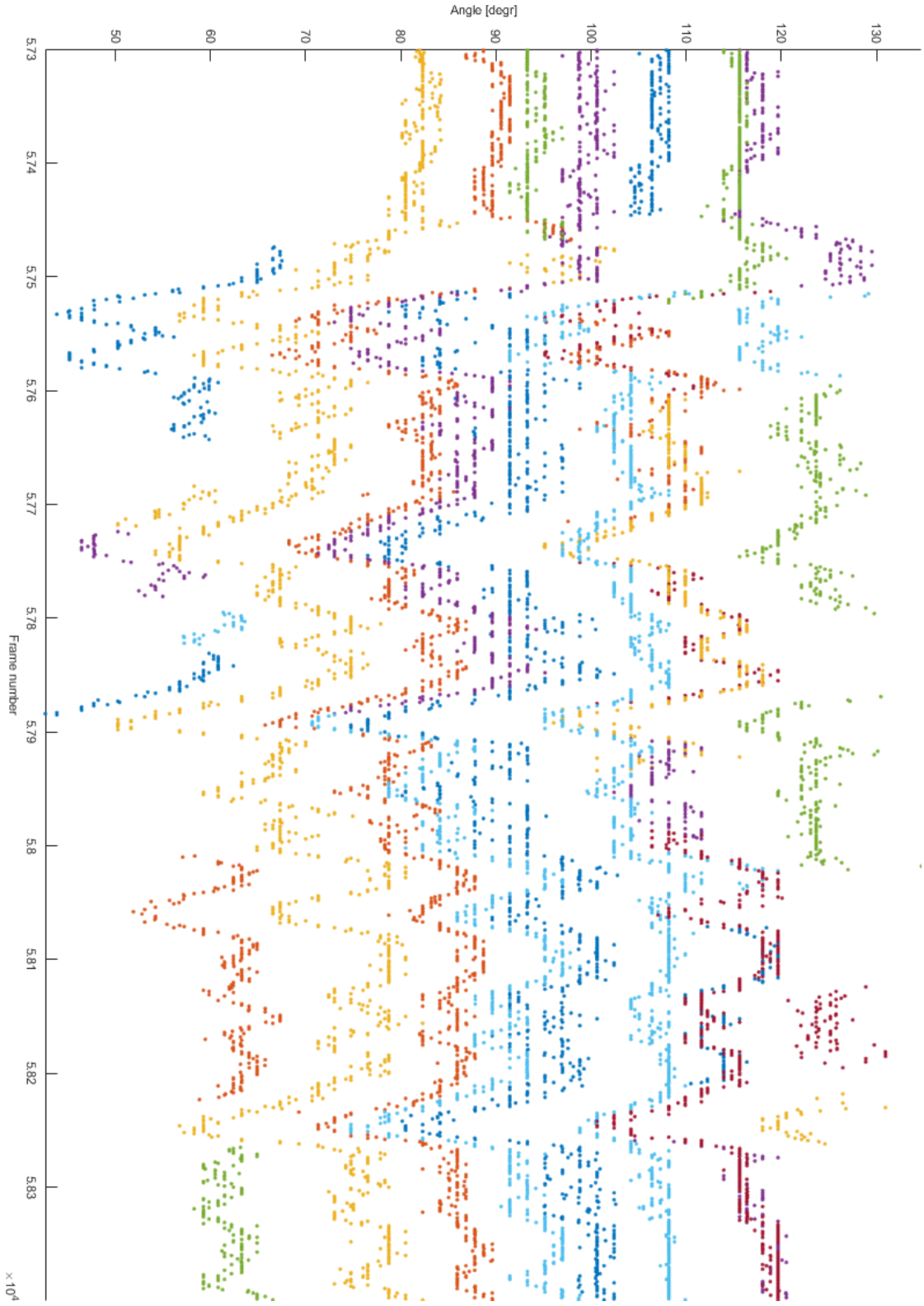


Figure B.3: Detected whiskers and tracks on frames 57300-58400 of video A, as detected by BWTT and Spanke's algorithm. Different colors denote different tracks, but the number of tracks is higher than the number of colors.

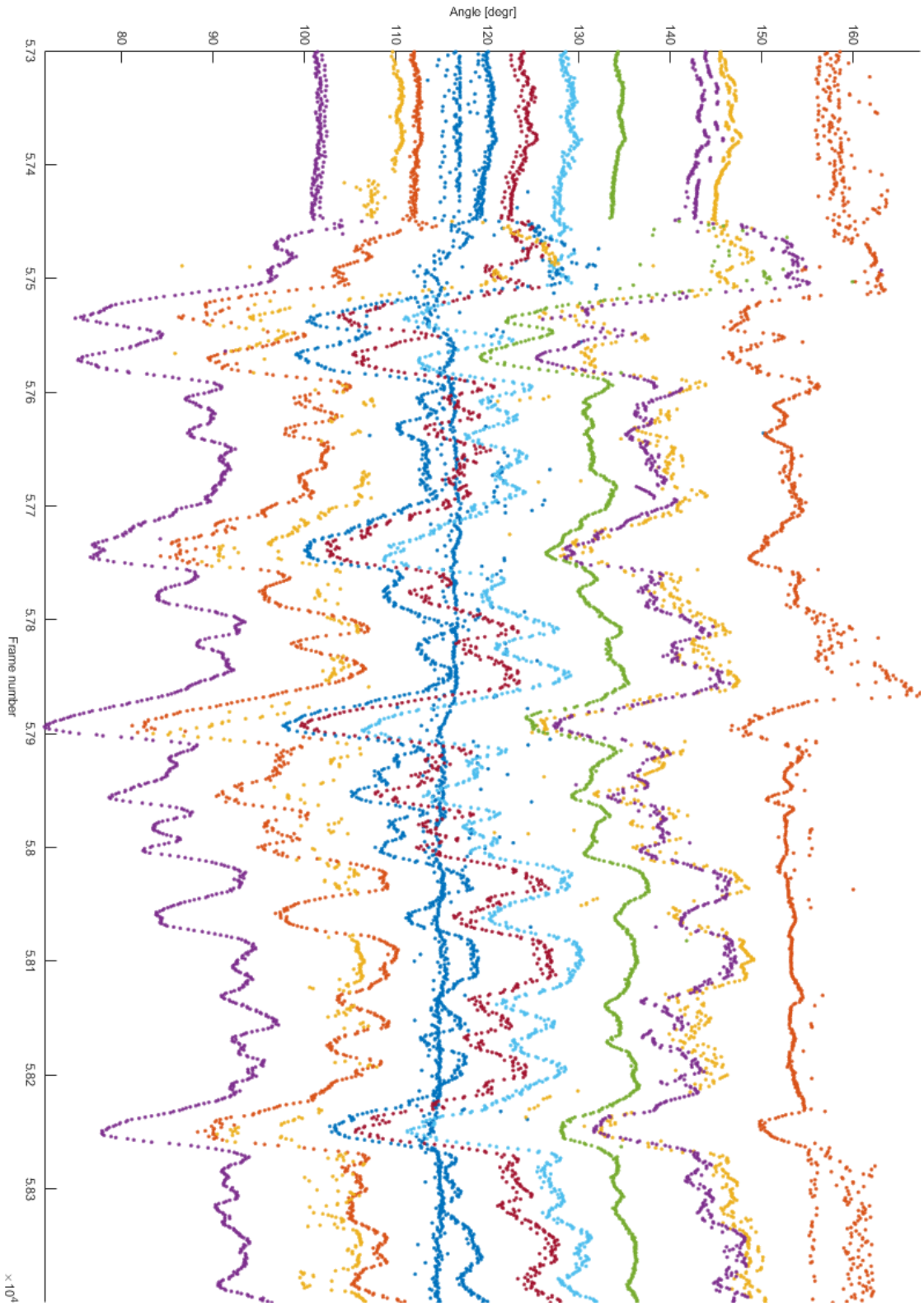


Figure B.4: Detected whiskers on frames 57300-58400 of video A, as detected by FWTS. Different colors denote different tracks.

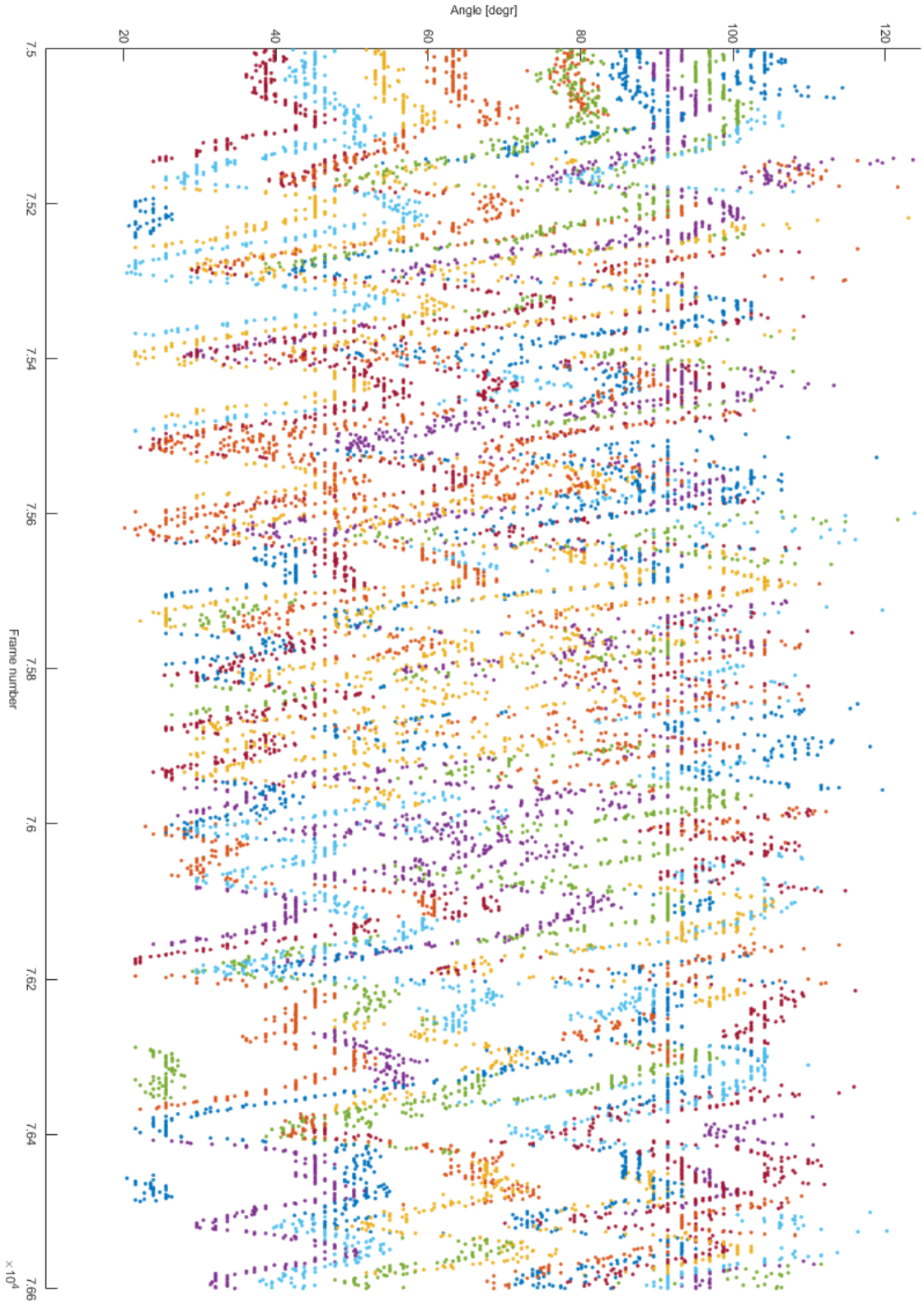


Figure B.5: Detected whiskers and tracks on frames 75000-76600 of video B, as detected by BWTT and Spanke's algorithm. Different colors denote different tracks, but the number of tracks is higher than the number of colors.

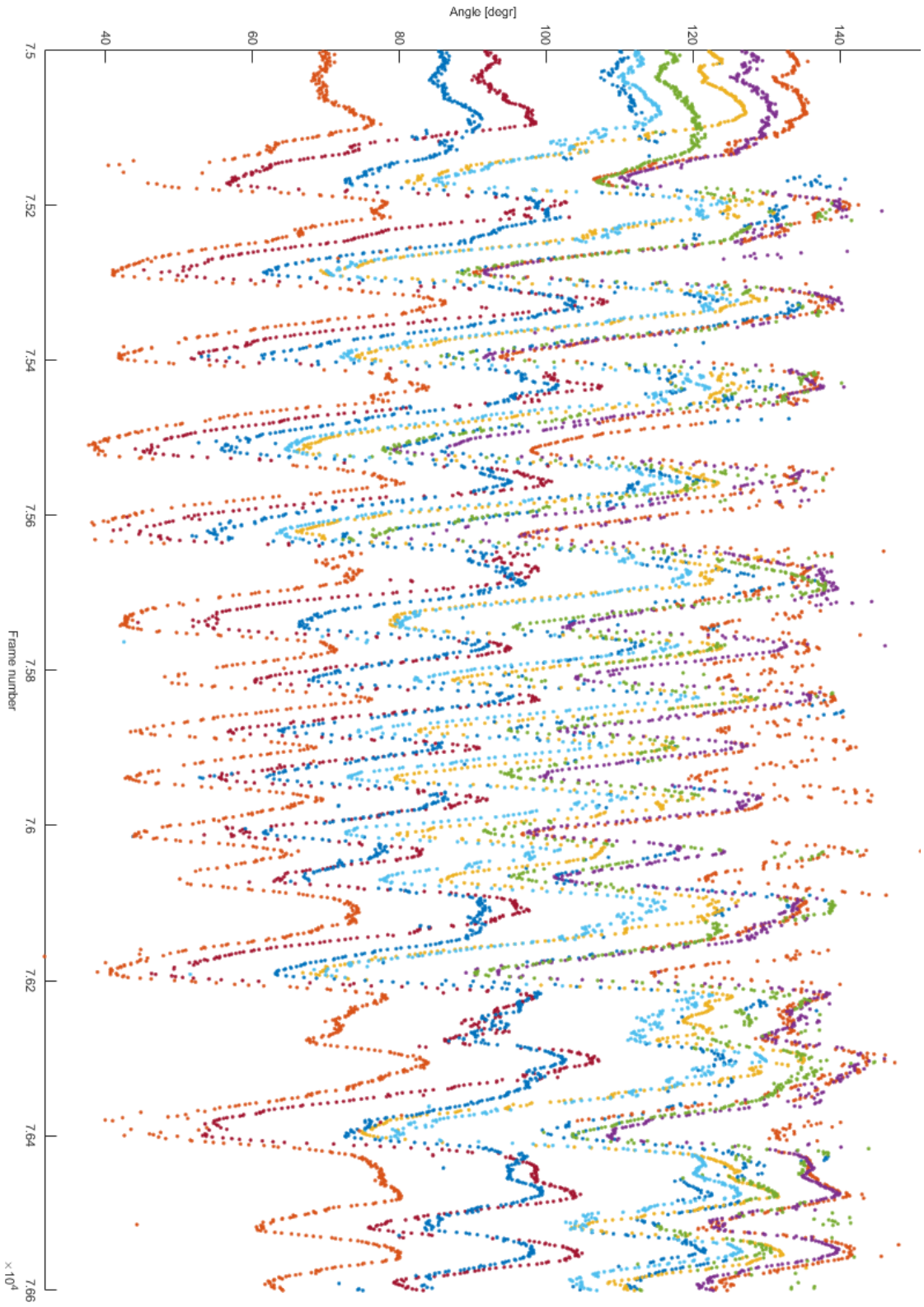


Figure B.6: Detected whiskers on frames 75000-76600 of video B, as detected by FWTS. Different colors denote different tracks.

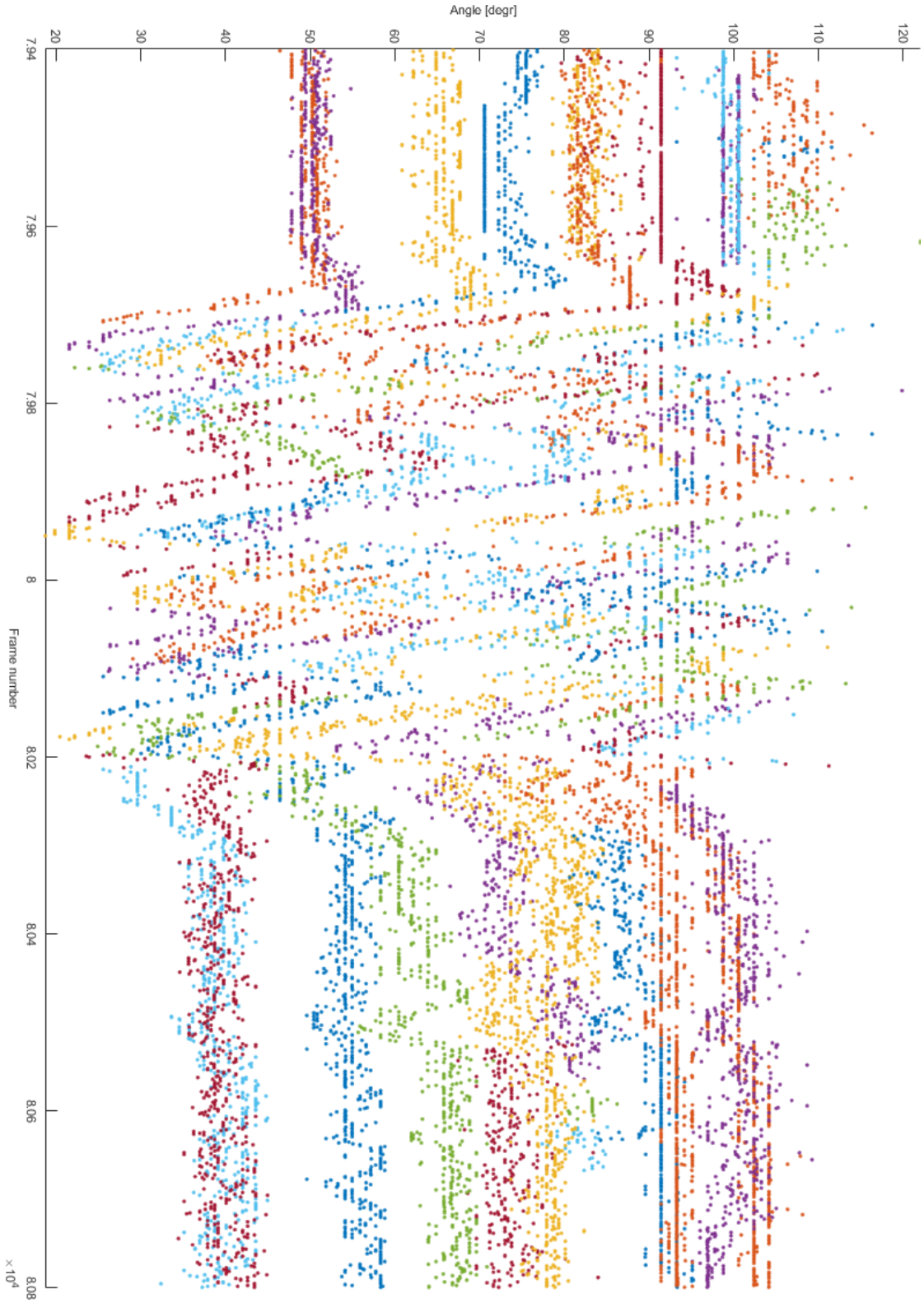


Figure B.7: Detected whiskers and tracks on frames 79400-80800 of video B, as detected by BWTT and Spanke's algorithm. Different colors denote different tracks, but the number of tracks is higher than the number of colors.

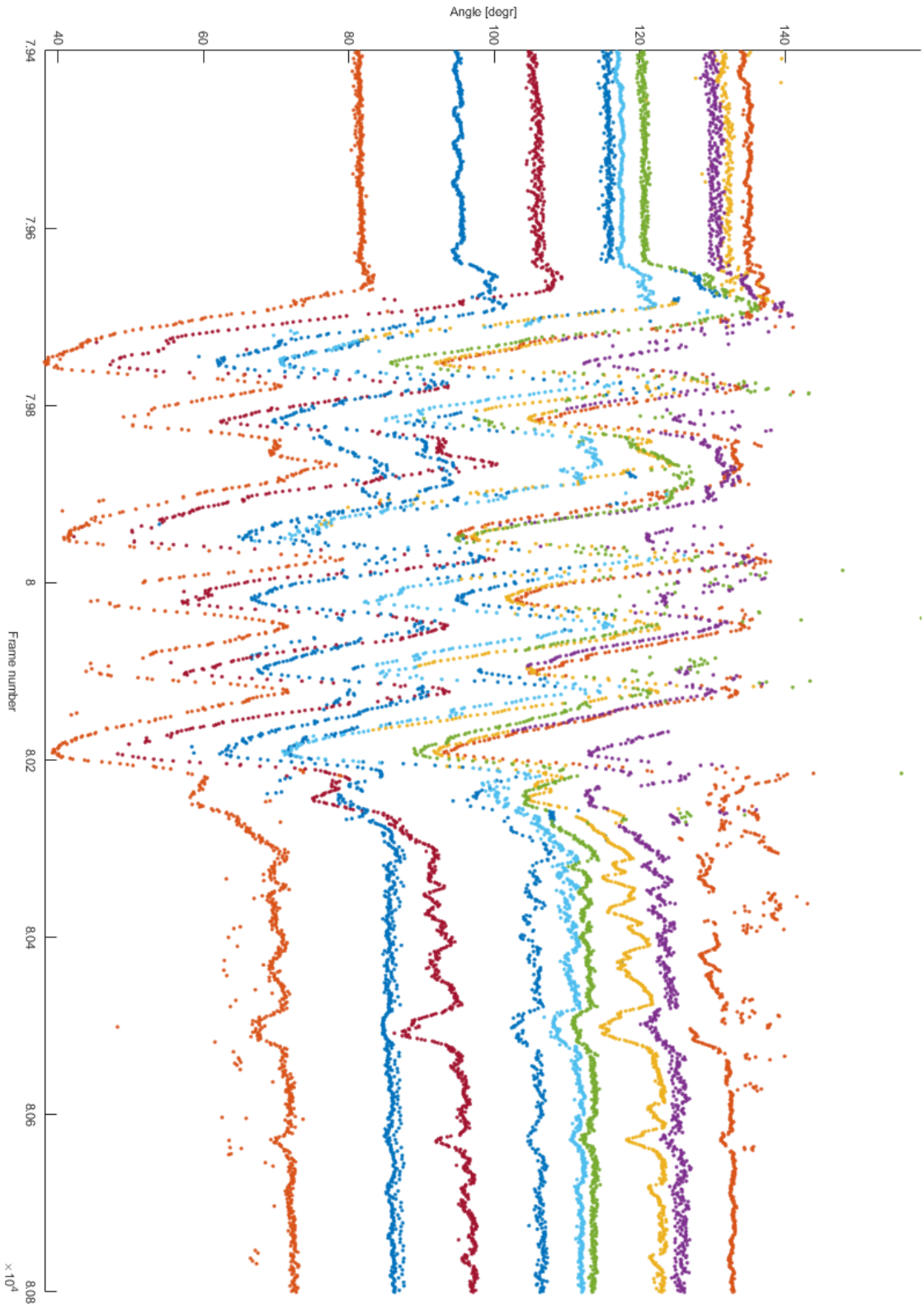


Figure B.8: Detected whiskers on frames 79400-80800 of video B, as detected by FWTS. Different colors denote different tracks.