TU Delft

Delft University of Technology

Document Version
Final published version

Citation (APA)
Arvanitis, P., Betting, J. H. L. F., Bosman, L. W. J., Al-Ars, Z., & Strydis, C. (2022). WhiskEras 2.0: Fast and Accurate Whisker Tracking in Rodents. In A. Orailoglu, M. Jung, & M. Reichenbach (Eds.), *Embedded Computer Systems: Architectures, Modeling, and Simulation - 21st International Conference, SAMOS 2021, Proceedings* (pp. 210-225). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 13227 LNCS). Springer. https://doi.org/10.1007/978-3-031-04580-6_14

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.

WhiskEras 2.0: Fast and Accurate Whisker Tracking in Rodents

Petros Arvanitis[1,2], Jan-Harm L.F. Betting[1], Laurens W.J. Bosman[1], Zaid Al-Ars[2], and Christos Strydis[1(✉)]

[1] Department of Neuroscience, Erasmus Medical Centre, Rotterdam, The Netherlands
c.strydis@erasmusmc.nl

[2] Department of Quantum and Computer Engineering, Delft University of Technology, Delft, The Netherlands

Abstract. Mice and rats can rapidly move their whiskers when exploring the environment. Accurate description of these movements is important for behavioral studies in neuroscience. Whisker tracking is, however, a notoriously difficult task due to the fast movements and frequent crossings and juxtapositionings among whiskers. We have recently developed WhiskEras, a computer-vision-based algorithm for whisker tracking in untrimmed, head-restrained mice. Although WhiskEras excels in tracking the movements of individual unmarked whiskers over time based on high-speed videos, the initial version of WhiskEras still had two issues preventing its widespread use: it involved tuning a great number of parameters manually to adjust for different experimental setups, and it was slow, processing less than 1 frame per second. To overcome these problems, we present here WhiskEras 2.0, in which the unwieldy stages of the initial algorithm were improved. The enhanced algorithm is more robust, not requiring intense parameter tuning. Furthermore, it was accelerated by first porting the code from MATLAB to C++ and then using advanced parallelization techniques with CUDA and OpenMP to achieve a speedup of at least 75x when processing a challenging whisker video. The improved WhiskEras 2.0 is made publicly available and is ready for processing high-speed videos, thus propelling behavioral research in neuroscience, in particular on sensorimotor integration.

Keywords: Whisker tracking · Algorithmic improvement · Acceleration

1 Introduction

Whiskers, or vibrissae, are tactile hairs found in most mammals [3]. Some rodents, like mice and rats, engage in active touch behavior during which they make fast rhythmic movements with their facial whiskers [10]. The facial whiskers are arranged in a conserved geometric pattern in the skin, which is reflected in

© Springer Nature Switzerland AG 2022
A. Orailoglu et al. (Eds.): SAMOS 2021, LNCS 13227, pp. 210–225, 2022.
https://doi.org/10.1007/978-3-031-04580-6_14

the organization of the primary somatosensory cortex [23]. The behavioral relevance of whisker use and the well-defined anatomy of whisker representation in the brain have made them particularly interesting for neuroscience.

Whisker movements are typically recorded using high-speed cameras with up to 1,000 frames per second, required to faithfully capture whisker movements during *whisking*, back-and-forth movements with frequencies of up to >25 Hz at speeds up to ∼1000 deg./s [5,21]. Whisker movements can be described by the angle of the whisker root relative to the snout. Keeping track of the angle of each whisker over time, however, is challenging. Most current whisker trackers make compromises to provide accurate tracking, like clipping of most of the whiskers (e.g., see [7]), or attaching markers to individual whiskers [13].

Recently developed, WhiskEras [2] is a promising framework for collecting accurate tracking data from untrimmed head-restrained mice, without the need for attaching markers to the whiskers. It was built to increase the performance of the BIOTACT Whisker-Tracking Tool (BWTT) [18]. In particular, BWTT can only *detect* whiskers frame by frame, but does not *track* them through time. For this reason, WhiskEras introduced a tracking module. WhiskEras, however, involves tuning a great number of parameters, owing to the various algorithmic steps it involves. It is expected that, in order to maximize the quality of results, many of these parameters need to be re-tuned for different videos, which is cumbersome and tough to automate. Furthermore, it is slow, processing less than 1 frame per second, which makes it ill-suited for long videos.

This work focuses on studying WhiskEras, identifying and extending its potential and accelerating it. For a complete, detailed description of this work, refer to [1]. The contributions of this paper are summarized as follows:

- Deliver WhiskEras 2.0, an improved version of the original framework which is more robust and easier to tune.
- Accelerate WhiskEras 2.0 by 74.96x by porting the code from MATLAB to C++, exploiting parallel execution on the CPU and GPU and performing several optimizations. The code is available online.[1]
- Pinpoint the limitations of computer-vision based approaches in whisker tracking, steering future endeavors in the field.

This paper is organized as follows: In Sect. 2, related works on whisker tracking are presented. Section 3 outlines the algorithmic stages of WhiskEras. In Sect. 4, the shortcomings of WhiskEras are pinpointed and its performance is analyzed. Then, Sect. 5 contains the implementation details of this work. In Sect. 6, WhiskEras 2.0 is compared to the original WhiskEras in terms of quality and performance, using two benchmark videos. Finally, conclusions of this work and some recommendations for future work are given in Sect. 7.

2 Related Work

In theory, high-speed videography allows accurate and non-invasive detection of whisker movements, but, for instance, crossings and juxtapositions complicate

[1] https://gitlab.com/neurocomputing-lab/whisker/whiskeras-2.0.

tracking. There are two common solutions to this problem. One way is reducing complexity by clipping all but one or a few whiskers, e.g. as is the case with Janelia Whisk [7]. This algorithm uses complex image-processing, statistical and machine-learning methods to follow whisker trajectories. However, it is not very accurate when tracking many whiskers [2]. Also, studies using the DeepLabCut framework, based on modern deep-learning methods, to track whiskers still used whisker clipping [8]. The clipping of whiskers, however, affects animal behaviour and neural processing [15], and should therefore ideally be avoided.

Alternatively, one can detect all whiskers, but without attempting to track individual whiskers over time, as does BWTT [18]. The original version of BWTT was slow, but was accelerated previously to achieve almost real-time processing [17]. A post-processing script, also developed previously, was deployed to track whiskers over time, but it does so with low accuracy [21].

WhiskEras is more promising in terms of tracking whisker movement of unmarked and untrimmed mice [2]. It is an unsupervised algorithm, which uses computer-vision algorithms to detect whiskers, and a machine-learning method to track them. Hence, there is no need for collecting labelled data to feed to a neural network for training, as necessary for (most) deep-learning techniques. In this sense, WhiskEras can be more convenient as it is ready to use on various experimental setups, particularly after the improvements described in this work.

3 WhiskEras Algorithm

WhiskEras comprises two main modules: Detection and Tracking, responsible for *detecting* whiskers and fitting them into a compact representation of several parameters, and for *following* these whiskers, frame-by-frame, respectively (Fig. 1). The system can also be organized into three components:

1. **Whisker-Point Detection** involves preprocessing of each frame in order to remove the background, silhouette and fur of the animal. Its result is an image of bright whiskers on a dark background. Then, Centerline Extraction is used to locate the whisker points on the centerline of each whisker, using Steger's Curvilinear Detector algorithm [22].
2. **Whisker Forming** takes as input the centerline positions of the whisker points in the image and performs Local Clustering to form groups of points which belong to the same whisker. This clustering, however, does not yet result in complete grouping, so that Cluster Stitching is required to unify clusters belonging to the same whisker. Afterwards, Parameter Fitting takes place, where each whisker representation is encoded as a set of four parameters.
3. **Tracking over time** matches whiskers found in the current frame with whiskers found in previous frames. The Tracking - Learning - Detection (TLD) technique [14] was adopted, to achieve consistency across a long sequence of frames. Tracking whiskers, from frame $n-1$ to n is performed using either a Kalman filter or by fitting whisker points to previously detected whiskers.

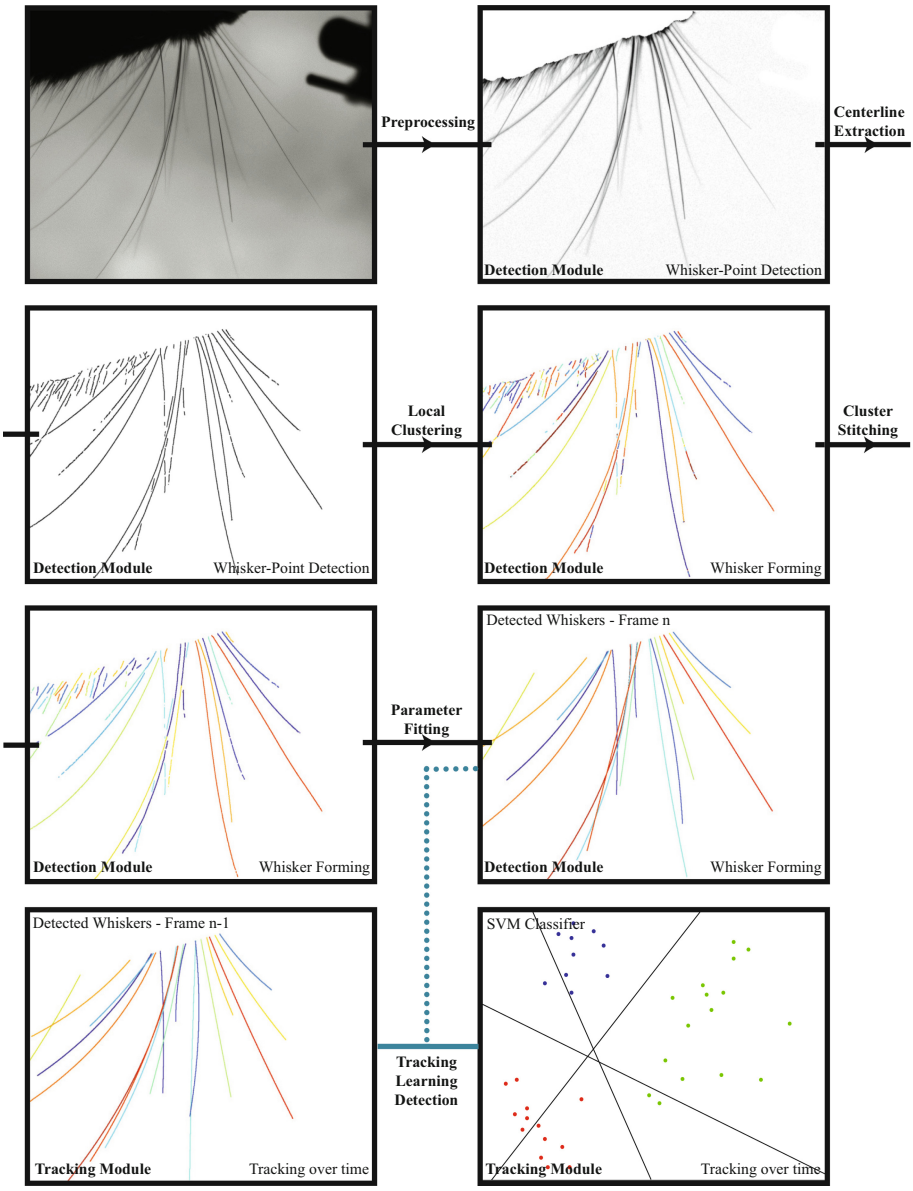
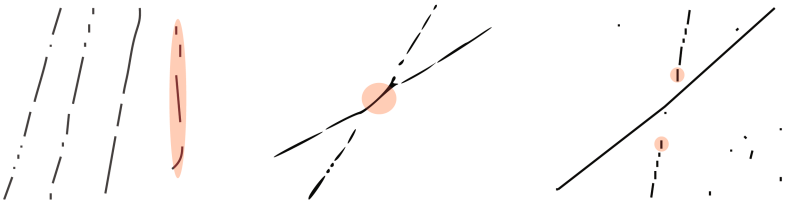


Fig. 1. WhiskEras pipeline overview: algorithmic steps are illustrated between the processing states of a whisker video frame, in the Detection module. The Tracking module makes use of the results from the current and previous frames, as well as an SVM Classifier to track whiskers over time.



(a) Curve on the right is interrupted and noisy

(b) The curve segment on the junction seems to belong to both whiskers.

(c) Whisker is interrupted near intersection.

Fig. 2. Centerline extraction issues

Also, a Support Vector Machine (SVM) Classifier is deployed to *learn* the characteristics (features) of each whisker (Learning) and recognize them in new frames (Detection). Tracking is necessary to collect a minimum number of data to train the SVM on. Then, the SVM is relied upon to identify whiskers in a consistent way, while tracking is used to potentially fit whiskers that were not recognized by the SVM.

4 Application Analysis

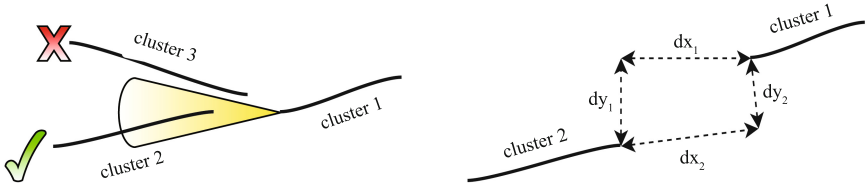
4.1 Quality Limitations

WhiskEras involves many tunable parameters, whose optimal values may differ between recording settings and/or experimental setups. Examples of parameters are the weight attributed to the whisker-point's orientation relative to the distance between neighbours during Local Clustering, or the allowed maximum distance between two clusters for Cluster Stitching. The effect of these parameters on the results was explored and a bounded discrete space was defined for each [1]. It appeared that parameters related to the Cluster-Stitching step were the most relevant, but some of these are incapable of handling a variety of situations, even within a single video.

Also Steger's Curvilinear Detector algorithm, used for Centerline Extraction and Local Clustering, had limitations (Fig. 2). Possibly due to experimental noise, whisker centerlines are not always extracted accurately, in particular when whiskers intersect with each other. Furthermore, whiskers that appear to be fading into the background are not fully recovered. Hence, we made changes to the Cluster-Stitching step and replaced the hard-to-tune parameters with other, more robust ones, and improved the currently employed Unbiased Curvilinear Detector [22] for Centerline Extraction. To overcome the second obstacle, an adaptation of Improved Curve Tracing [20], an expansion on the Curvilinear Detector algorithm, was materialized.

4.2 Performance Profiling

WhiskEras is coded in MATLAB, a high-level programming language, which is generally inefficient for applications that demand large processing power. The



(a) Cluster 2 can be stitched to cluster 1, while cluster 3 cannot.

(b) The parallel and perpendicular distances between two clusters. Depending on the reference cluster, these are different.

Fig. 3. Cluster stitching: maximum angle condition and distance between clusters

implementation of WhiskEras was done nearly optimally, taking advantage of any inherent parallelization present in its algorithmic stages. Centerline Extraction is performed on the GPU, exploiting the massive data-level parallelism (DLP), while some costly steps during Cluster Stitching and Learning are also parallelized, using multiple threads on the CPU. Despite these optimizations, the application takes about 1.3 s to process a single frame. For context, a 50-second-long high-speed video (1,000 fps) would require ∼18 h of processing on a powerful machine.

For acceleration, the code was ported to C++, a high-level language incorporating also low-level features in C, such as efficient memory management, optimized compiler support, and the availability of many fast libraries. Furthermore, the parallelizable portions had to retain multithreading, following the MATLAB implementation. To this end, the OpenMP API and CUDA API were used, respectively for CPUs and GPUs. After this initial acceleration, an iterative process was performed where the code's most expensive portions were pinpointed through profiling, and optimizations were carried out accordingly.

5 Implementation

5.1 Quality Improvements

Pairs of clusters belonging to the same whisker are stitched together based on the distance between the edges of these clusters, and their orientation proximity. Originally, the maximal distance between clusters to allow stitching was a fixed value, as specified a priori by the user, limiting the flexibility to stitch more distant clusters, e.g. of disappearing and re-emerging whiskers (Fig. 2c). Instead, to guard against invalid stitchings, Radon Transform, a technique usually used in tomographic reconstruction in medical imaging [19], was deployed. We implemented first-order, Localized Radon Transform to detect lines between the edges of two clusters. If a line in between them is not found in the unprocessed frame of whiskers, then stitching of this pair is aborted.

In addition, a maximum-angle condition was utilized (Fig. 3a). Clusters that are not rooted within the angle margin, derived from cluster 1's orientation,

cannot be stitched to cluster 1. In addition, the distance between two clusters can be analyzed parallel and perpendicular to the orientation axis of one cluster. These components are dx_1 and dy_1, respectively, if cluster 1 is the reference (Fig. 3b). In WhiskEras, only one cluster was used as reference to compute these distances and invalidate inappropriate stitchings according to a threshold, but now we compute the distances with reference to both clusters. Then, the smallest ones are used to determine if a stitching is valid. This proved to be favorable, as the edge of one of the two clusters belonging to the same whisker is often noisy, not pointing towards the other edge.

In WhiskEras, the stitching score was given by

$$score = \sqrt{dx^2 + dy^2} + c \cdot \beta$$

where β is the angle difference of the edges of the two clusters and c is its weight. However, dy and β should be weighted more heavily than dx, especially as dx grows larger, but dy and β should be allowed to be a bit larger when two clusters are really close to each other, since they are disconnected in the first place due to noise. Taking all these into consideration, the new score is

$$score = \begin{cases} dx + dy + \beta, & \text{if } d \leq d_0 \\ dx + \frac{c}{dx} \cdot dy + c \cdot \beta, & \text{if } d > d_0 \end{cases}$$

Notice that dy and β are weighted as much as dx for small distances, up to d_0 (e.g. $d_0 = 5$ px). When the distance indicates that the clusters are not proximal whisker segments, dy and β should be the prime factors. This necessitates the use of a large value for c (e.g., $c = 200$).

In contrast to the original version of WhiskEras, we consistently opt for Steger's Curvilinear Detector for Centerline Extraction and Local Clustering. Steger's algorithm works as follows: it uses a characteristic 1D line profile to model curves in 2D images by computing the Hessian Matrix of the image and then the first- and second-order derivatives of the image in the direction perpendicular to the curves. A pixel point potentially contains a curve point if the first directional derivative vanishes in its vicinity. The lower the second directional derivative value, the more likely it is that a pixel indeed contains a curve point. This concludes the curve-point detection part of the algorithm. This step is completely retained in Improved Curve Tracing.

The rest of the algorithm finalizes the curve points, while forming clusters, which constitute curves, at the same time. For this purpose, a threshold $tr2$ is used. Pixels with second directional derivative less than $-tr2$, whose first directional derivative vanishes, are marked as curve points. The rest of the curve points are found by extending the curves, starting from these points. A curve is extended neighbour-by-neighbour as long as there is a valid curve point near the last curve point detected, in the curvilinear direction. This part is referred to as *Steger Clustering* in [2].

Improved Curve Tracing enhances Steger Clustering by expanding on how a new neighbour can be discovered by considering the following problematic cases:

multiple and possibly intersecting curves, disappearance and re-emergence of a curve, and tracing curves that fade out. Even though the intuition should be similar in improving the Curvilinear Detector algorithm in whisker tracking, WhiskEras already partially deals with some of these issues by using Cluster Stitching. Improved Curve Tracing was not found to work well with Cluster Stitching. On the other hand, using Improved Curve Tracing and completely eliminating the Cluster Stitching step was also disadvantageous. Thus, an alternative implementation was adopted, that is largely inspired by the Improved Curve Tracing method. In this work, we call it Improved Whisker Tracing.

During Steger Clustering, a neighbour curve point is located by searching in the orientation of the current curve point, which is computed by the Curvilinear Detector algorithm, in a 1-pixel radius (*Steger linking*). In Improved Whisker Tracing, this radius can be >1, as sometimes there is an apparent gap between neighbour whisker points. This is now named *pixel peeking*. Additionally, *beam scanning*, a new neighbour-detecting step was added. If no neighbour is found during pixel peeking, beam scanning is used to search for a neighbour some pixels further, in the orientation of the last whisker point found. A $K = 15$ distance -in pixels- is used but this can also be configured. The addition of these two steps is not trivial, as there is some back-and-forth of extending each whisker cluster from both sides and using pixel peeking and beam scanning interchangeably. Although noisy videos pose challenges to the algorithm, the new methods generally result in larger and fewer initial clusters than using Steger Clustering. This significantly reduces the workload of Cluster Stitching, improving the quality of the final results.

5.2 Acceleration

Accelerating WhiskEras 2.0 was no trivial task due to complex algorithmic stages which required careful implementation to eliminate redundant operations and reduce computational cost. First, the code was ported from MATLAB to C++. This required analyzing many high-level MATLAB functions and writing them in C++, top-down, using imperative statements in an efficient way. Then, the parallelizeable sections were located. Both Preprocessing and Centerline Extraction are performed on an NVidia GPU, using CUDA, exploiting pixel-level parallelism. Furthermore, Parameter Fitting is done using multiple threads on the CPU, one for each whisker to be fitted. The same was done for each pair of whisker classes during the SVM Classifier's one-vs-one training. Finally, properties of the clusters, such as rotation data, necessary in Cluster Stitching, are also computed in parallel using CPU multithreading. After this initial phase of acceleration, several optimizations per algorithmic stage were pursued:

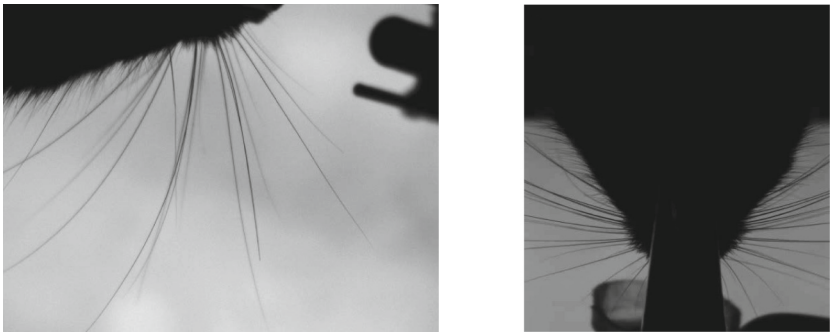
1) **Separable convolution:** Centerline Extraction was a particularly expensive step. Specifically, during this step, five convolutions between the image and Guassian derivative kernels are performed. Initially, this step was the bottleneck, yielding a computational complexity of $\mathcal{O}(n{\cdot}m^2)$, where n are the image dimensions and m is the kernel's width ($m = 20$). It was found that

the kernels used to convolve the image were separable: instead of carrying out normal convolutions, we can perform spatially separable convolutions and obtain the exact same results. The total complexity is now reduced to $\mathcal{O}(n \cdot 2m)$. This implementation is extremely efficient, making use of the GPU's shared memory to minimize memory access latency between consecutive accesses to the same image pixels.

2) **GPU-based sorting:** In the Local-Clustering step, the whisker points with the second directional derivatives are accessed from the lowest to the highest values. Initially, before extending every new curve, an unvisited whisker point with the minimum value of this array was located. This proved to be slow. Thus, this array was sorted before entering the loop, instead. This optimization was even taken one step further by performing the sorting on the GPU, using the highly optimized CUB library.
3) **CPU-GPU-transfer reduction:** Costly data transfers between the host (CPU) and the device (GPU) were minimized in two ways:
   - Constant-sized structures that accommodated variable data per frame are allocated once at the start of processing in the host's pinned memory. This alleviates the need to first transfer the data from the host's pageable memory to pinned memory and then to the GPU, thus maximizing transfer bandwidth [12].
   - The whisker points' orientations and second directional derivative values were originally transferred to the host in arrays with as many entries as image pixels, to be used in the Local Clustering step. This was a naive approach. Instead, another image-sized array was allocated on the device to indicate the presence of whisker points. Then, the CUB library's *DeviceSelect* class was used to discard non-whisker points from the arrays of interest and keep whisker points only. Consequently, the size of these arrays, which are transferred from the GPU to the CPU, decreased dramatically.
4) **Optimal SVM-library selection:** Different libraries were tested on their performance in the demanding SVM training, which can occur e.g. every five frames (user-selectable): a commonly-used SVM library in *libSVM* [6], a GPU SVM library in *gpuSVM* [16], a library using Stochastic Gradient Descent for SVM training in *sgdSVM* [4], with the best choice being an SVM library specialized in linear kernels in *libLinear* [9].

6 Evaluation

The quality and performance of WhiskEras 2.0 is demonstrated in this section by utilizing only two videos (Fig. 4), due to paper space restrictions. However, the videos are markedly different, originating from different experimental setups, thus showcasing the robustness of the WhiskEras 2.0 approach. Video A is focused on one side of the animal's snout, while the whole snout is visible in Video B. The whole Video A (34,133 frames) and a segment from Video B (frames 5,000–30,000) were evaluated qualitatively. The processing power was



(a) Video A (b) Video B

Fig. 4. Representative frames from videos used for evaluation

Table 1. Metrics to compare the quality of WhiskEras vs WhiskEras 2.0

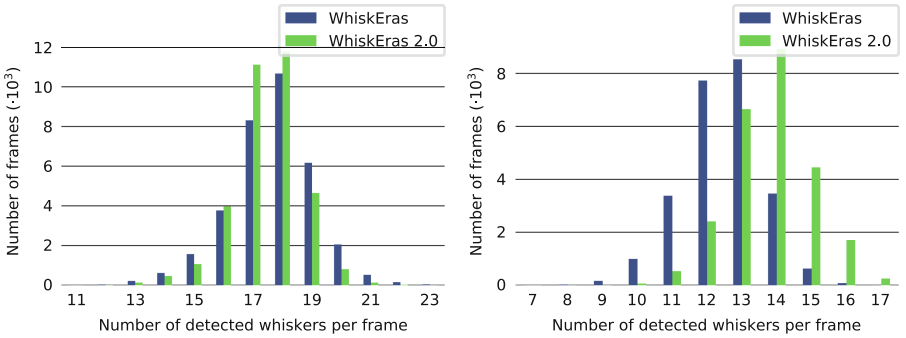
Evaluation metric	Description
Detected whiskers per frame	Number of whiskers detected per frame
Detection ratio	Fraction of the video in which whiskers were detected
Tracking quality	Trajectory of whiskers' angles in time
Signal-to-noise ratio	Ratio of smoothened whisker angle trajectory to (subtracted) noisy whisker angle trajectory (in dB)

measured by taking the average execution time per frame, over 1,000 frames. The hardware used to assess performance included an AMD EPYC 7551 32-Core Processor @ 2.0 GHz with 64 threads and an NVidia Tesla V100-PCIE GPU. The C++ 17 standard was used, compiled with g++ 4.8.5 using the *-O3* flag to perform aggressive optimizations such as loop unrolling and vectorized operations. OpenMP 3.1 and CUDA 11.1 were used for multithreading on the CPU and the GPU, respectively. The OpenCV 4.2 library was used to read video frames and perform out-of-the-box, image-processing functions, while Eigen 3.3 [11] was used to perform fast linear algebra operations.

6.1 Quality Improvements

To evaluate the effect of the improvements, WhiskEras and WhiskEras 2.0 were run on the same video segments, after manually tuning parameters for both versions. In addition, both were configured to follow the same whisker *tracks*, i.e., have the same starting point to measure how well they can follow their trajectories. Video A contains numerous whiskers, which also intersect and often hide behind each other. On the other hand, we only track the whiskers on the right side of the snout in Video B. Thus, Video A poses more challenges which WhiskEras 2.0 is expected to deal with more consistently than WhiskEras. Evaluating quality is no trivial task, since there is no ground truth. Consequently,

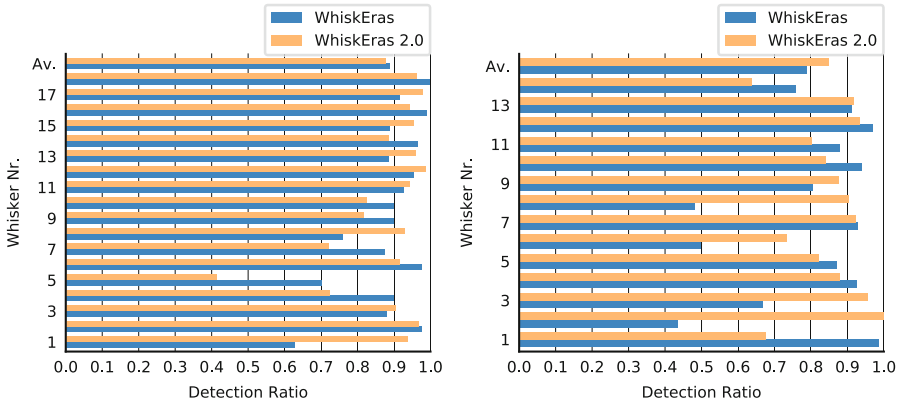
the metrics used to compare the quality of results attempt to quantify tracking quality in an automated way and are the same used in [2], presented in Table 1. The results for each criterion are given below.



(a) Video A: Average and std are 17.66(17.47) and 1.46(1.18) for WhiskEras(WhiskEras 2.0)

(b) Video B: Average and std are 12.47(13.81) and 1.15(1.18) for WhiskEras(WhiskEras 2.0)

Fig. 5. Histogram of detected whiskers per frame

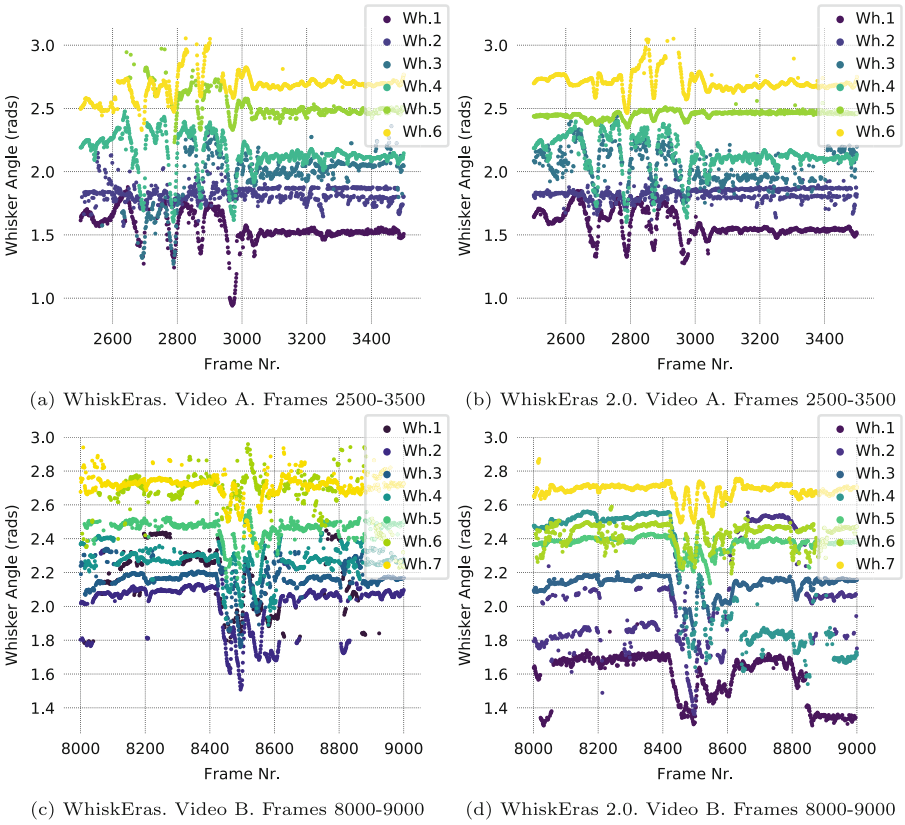


(a) Video A. Average detection ratio is 0.89(0.88) for WhiskEras(WhiskEras 2.0)

(b) Video B. Average detection ratio is 0.79(0.85) for WhiskEras(WhiskEras 2.0)

Fig. 6. Detection ratios

1. **Detected whiskers per frame:** This metric should be fairly stable, meaning a constant number of whiskers is detected throughout a video. Although, whiskers are often hidden, the variance of this metric should be kept to a minimum, indicating algorithmic robustness to whisker movement. A second point of interest is the average number of whiskers detected per frame,



(a) WhiskEras. Video A. Frames 2500-3500

(b) WhiskEras 2.0. Video A. Frames 2500-3500

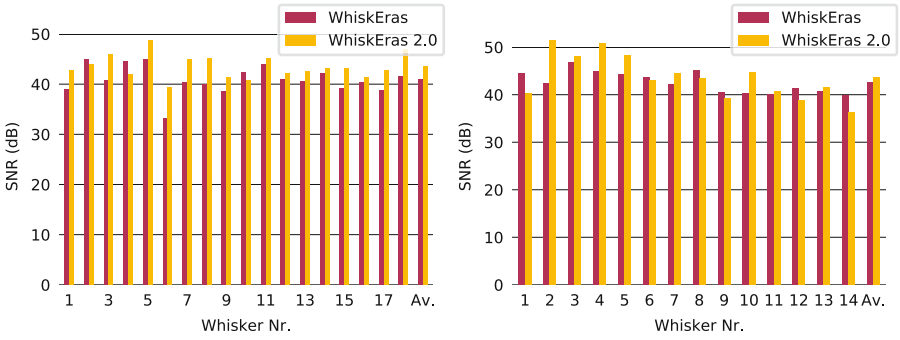
(c) WhiskEras. Video B. Frames 8000-9000

(d) WhiskEras 2.0. Video B. Frames 8000-9000

Fig. 7. Angle tracking

reflecting false positives as well as false negatives. For Video A, Fig. 5a shows a 19% decrease in standard deviation over the number of detected whiskers per frame, which is positive. Yet, the average number of whiskers detected per frame is marginally smaller. The results were quite different for Video B: a higher average and an almost identical standard deviation (Fig. 5b).

2. **Detection ratio:** The detection ratio of each whisker indicates the percentage of the frames wherein a whisker was successfully tracked. Some whiskers have different detection ratios in the two WhiskEras versions, in both videos. It should be mentioned that whisker indexing refers to the starting point of each whisker (first frame), which is identical for both versions. The same whisker indices do not necessarily represent the same whiskers, as these starting points may have evolved to a different trajectory. Overall, however, the average detection ratios are similar in Video A, while Video B shows a 5% increase for WhiskEras 2.0, compared to WhiskEras (Fig. 6). This also aligns with the increased number of detected whiskers, which translates to more successfully tracked whiskers.

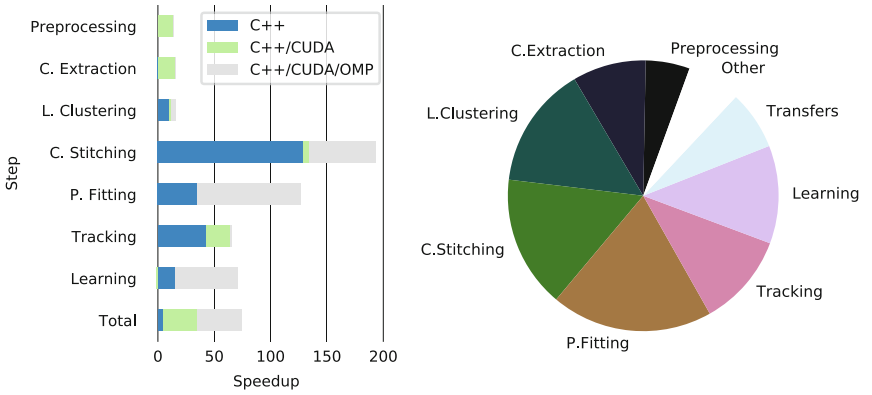


(a) Video A, Average SNR is 40.88(43.46) for WhiskEras(WhiskEras 2.0)

(b) Video B, Average SNR is 42.64(43.66) for WhiskEras(WhiskEras 2.0)

Fig. 8. SNR of trajectories of whiskers' angles through time

3. **Tracking quality:** The whisker angle, relative to the snout, is the most important movement parameter, thus its trajectory was used to evaluate the tracking quality. When a whisker's angle transitions smoothly in time, the whisker is considered to be well-tracked. Two small segments were chosen from each video and only a few of the whiskers' angle-trajectories are presented here, for clarity. For Video A, Fig. 7b (WhiskEras 2.0) illustrates better tracking of most of the whiskers, compared to Fig. 7a (WhiskEras). This is most evident for whiskers 5 and 6. On the other hand, Video B shows much lower quality of tracking for both versions and the results are mixed. For example, whiskers 6 and 7 favour WhiskEras 2.0, while whiskers 2 and 4 show better quality in WhiskEras. Importantly, some of these whiskers do not have a one-to-one correspondence between the two versions, as already stated, thus it is hard to draw any conclusions about which version has the edge in Video B from this metric.
4. **Signal-to-noise ratio (SNR)** represents the quantitative evaluation of the angular change. As in [2], this signal was computed by smoothing the calculated angle of each whisker over time using a Savitzky-Golay filter, which is appropriate for quick variations. A window size of 9 was chosen and the data was fit quadratically. The noise was extracted by subtracting the smoothed data from the actual data. Then, the SNR of each whisker was computed as the ratio of the squared magnitudes of the signal over the noise, in dB. For Video A, WhiskEras 2.0 exhibits a larger SNR for most of the whiskers (Fig. 8a), which was less consistent in Video B (Fig. 8b). Overall, the algorithmic improvements benefited good-quality videos (Video A) more than poor-quality ones (Video B), but even for the latter, gains could be made.



(a) Bar diagram of the accelerated versions' cumulative speedups over MATLAB. Each algorithmic step is shown.

(b) Execution time percentage of each WhiskEras 2.0 accelerated step

Fig. 9. Speedup and execution-time profiling when processing Video A

6.2 Acceleration

The performance of WhiskEras 2.0 was tested in both videos, but here we present the analysis of Video A (see Table 2), due to page limitations. Note that *Other* refers to overhead in between algorithmic steps and OpenCV reading video frames. The speedups obtained are illustrated in Fig. 9a. Simply porting the code to C++ did not provide a massive speedup, due to the lack of execution parallelism in this purely sequential implementation. MATLAB, on the other hand, already made use of CPU and GPU multithreading. The Preprocessing and Centerline Extraction steps, in particular, are much slower in pure C++ than in MATLAB because of the lack of GPU execution, but the C++/CUDA version surpasses both. The optimization involving separable convolution reduced the Centerline Extraction's execution time even more, by a factor of 10x. Furthermore, using OpenMP to take advantage of multiple CPU threads contributed to the final speedup of 74.96x. The execution time per algorithmic step is fairly balanced (Fig. 9b). Further acceleration is *not* achievable without major investments. The processing power of the final version of WhiskEras 2.0 for Video

Table 2. Execution time (in milliseconds) per frame for various development stages of WhiskEras 2.0, as measured when processing Video A

	Pre-process	Centerline extraction	Local cluster	Cluster stitch	Param. fitting	Track	Learn	Trans-fers	Other	Total
MATLAB	12.3	23.0	39.7	520.2	421.3	123.0	140.6	–	4.6	**1284.7**
C++	23.4	232.1	3.9	4.0	12.1	2.9	9.1	0	20.6	**308.1**
C++/CUDA	0.9	1.5	3.2	3.9	12.0	1.9	9.9	2.1	1.2	**36.6**
C++/CUDA/OMP	0.9	1.5	2.5	2.7	3.3	1.9	2.0	1.2	1.1	**17.1**

B was measured to be 82 frames/second, faster than the 58 frames/second for Video A. This difference is mainly attributed to the number of whiskers being detected and tracked (14 in B vs. 18 in A).

7 Conclusions

WhiskEras 2.0 was developed and accelerated to process around 50–120 frames per second, depending on their characteristics, with enhanced quality. Specifically, the whisker-tracking system became more robust in addressing different whisker-video settings and easier to tune. The speedup achieved can be attributed to porting the code to C++, exploiting parallel execution on the CPU and the GPU through multithreading, and a series of optimizations. Future steps include further algorithmic improvements to the Centerline-Extraction step now that the Unbiased Curvilinear Detector's [22] inherent limitations are exposed. Finally, in order to make WhiskEras 2.0 capable of performing online tracking – a desired feature which enables neuroscientists to process whisker videos as they are recorded – more acceleration routes should be explored, such as different/more hardware accelerators.

References

1. Arvanitis, P.: Towards automated and fast whisker tracking in rodents. Master's thesis, Delft University of Technology, January 2021
2. Betting, J.-H.L.F., Romano, V., Al-Ars, Z., Bosman, L.W.J., Strydis, C., De Zeeuw C.I.: WhiskEras: a new algorithm for accurate whisker tracking. Fron. Cell. Neurosci. **14** (2020). https://doi.org/10.3389/fncel.2020.588445. https://www.ncbi.nlm.nih.gov/pmc/issues/351366/. ISSN 1662-5102
3. Bosman, L., Houweling, A., Owens, C., Tanke, N., Shevchouk, O., Rahmati, N., Teunissen, W., Ju, C., Gong, W., Koekkoek, S., De Zeeuw, C.: Anatomical pathways involved in generating and sensing rhythmic whisker movements. Front. Integr. Neurosci. **5**, 53 (2011)
4. Bottou, L.: Stochastic gradient SVM (2007). https://leon.bottou.org/projects/sgd. Accessed 10 Mar 2021
5. Carvell, G., Simons, D.: Biometric analyses of vibrissal tactile discrimination in the rat. J. Neurosci. **10**(8), 2638–2648 (1990)
6. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM TIST **2**, 27:1–27:27 (2011). http://www.csie.ntu.edu.tw/~cjlin/libsvm
7. Clack, N.G., et al.: Automated tracking of whiskers in videos of head fixed rodents. PLoS Comput. Biol. **8**(7), e1002591 (2012)
8. Dooley, J.C., Glanz, R.M., Sokoloff, G., Blumberg, M.S.: Self-generated whisker movements drive state-dependent sensory input to developing barrel cortex. Curr. Biol. **30**(12), 2404–2410.e4 (2020)
9. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: a library for large linear classification. JMLR **9**, 1871–1874 (2008)
10. Grant, R.A., Mitchinson, B., Fox, C.W., Prescott, T.J.: Active touch sensing in the rat: anticipatory and regulatory control of whisker movements during surface exploration. JNP **101**(2), 862–874 (2009)

11. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010). http://eigen.tuxfamily.org. Accessed 10 Mar 2021
12. Harris, M.: How to optimize data transfers in CUDA C/C++ (2012). https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/. Accessed 5 Mar 2021
13. Herfst, L.J., Brecht, M.: Whisker movements evoked by stimulation of single motor neurons in the facial nucleus of the rat. JNP **99**(6), 2821–2832 (2008)
14. Kalal, Z., Mikolajczyk, K., Matas, J.: Tracking-learning-detection. IEEE TPAMI **34**(7), 1409–1422 (2012)
15. Kelly, M.K., Carvell, G.E., Kodger, J.M., Simons, D.J.: Sensory loss by selected whisker removal produces immediate disinhibition in the somatosensory cortex of behaving rats. J. Neurosci. **19**(20), 9117–9125 (1999)
16. Li, Q., Salman, R., Test, E., Strack, R., Kecman, V.: GPUSVM: a comprehensive CUDA based support vector machine package. OCS **1**(4), 387–405 (2011)
17. Ma, Y., et al.: Towards real-time whisker tracking in rodents for studying sensorimotor disorders, July 2017
18. Perkon, I., Košir, A., Itskov, P.M., Tasič, J., Diamond, M.E.: Unsupervised quantification of whisking and head movement in freely moving rodents. JNP **105**(4), 1950–1962 (2011)
19. Radon, J.: On the determination of functions from their integral values along certain manifolds. IEEE T-MI **5**(4), 170–176 (1986)
20. Raghupathy, K., Parks, T.: Improved curve tracing in images. In: 2004 ICASSP. IEEE (2004)
21. Rahmati, N., et al.: Cerebellar potentiation and learning a whisker-based object localization task with a time response window. J. Neurosci. **34**(5), 1949–1962 (2014)
22. Steger, C.: An unbiased detector of curvilinear structures. IEEE TPAMI **20**(2), 113–125 (1998)
23. Woolsey, T.A., Welker, C., Schwartz, R.H.: Comparative anatomical studies of the SmL face cortex with special reference to the occurrence of "barrels" in layer IV. JCN **164**(1), 79–94 (1975)