Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Extending Simple Online Realtime Tracking with Sparse Optical Flow

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science

eingereicht von:	Falko Becker
geboren am:	03.03.1980
geboren in:	Magdeburg
Gutachter:	Prof. Dr.-Ing. Peter Eisert (Humboldt-Universität zu Berlin) M.Sc. Nils Kornfeld (Deutsches Zentrum für Luft- und Raumfahrt e.V.)
eingereicht am:	..... verteidigt am: .....

Abstract

Detection and tracking of objects in video data is a common topic in computer vision and widely used for traffic surveillance. Simple Online and Realtime Tracking (SORT) is a Multi-Object Tracking (MOT) algorithm which performs in realtime by using efficient algorithms. SORT's tracking performance is based on detection quality and may suffer due to detection failures. This work proposes an extension to SORT's Kalman filter measurement model with Robust Local Optical Flow to add a velocity component to the filter's update vector and improve robustness against detection failures and decrease track id switches. The elaborated extension, termed SORT_OF, is tested and benchmarked on parts of the MOT15 tracking benchmark as well as on a self created benchmark representing a DLR real world scenario. The proposed tracking method still works in realtime.

Contents

List of Figures

List of Tables

> *"I've seen things you people wouldn't believe. Attack ships on fire off shoulder of Orion. I watched c-beams glitter in the dark near the Tannhäuser Gate. All those moments will be lost in time, like tears in rain."*
>
> - Roy Batty to Rick Deckard in Ridley Scott's *Blade Runner*, 1982

1. Introduction

As part of ongoing projects a research group at the German Aerospace Center, Institute of Transportation Systems (Deutsches Zentrum für Luft- und Raumfahrt e.V., Institut für Verkehrssystemtechnik, DLR), deals with video data for traffic surveillance (figure 1). The video data is used to detect and track traffic participants on a certain scene, usually a crossroad or intersection. With the gathered data from the tracks new simulations can be run, for example to predict the effect of the traffic flow on the environment, analyze critical situations between traffic participants to improve traffic safety as well as for statistical reasons. The detection and tracking (see 2.1) is done live or offline, but in the latter case still in real time on already collected video data. To accomplish this task, a combination of

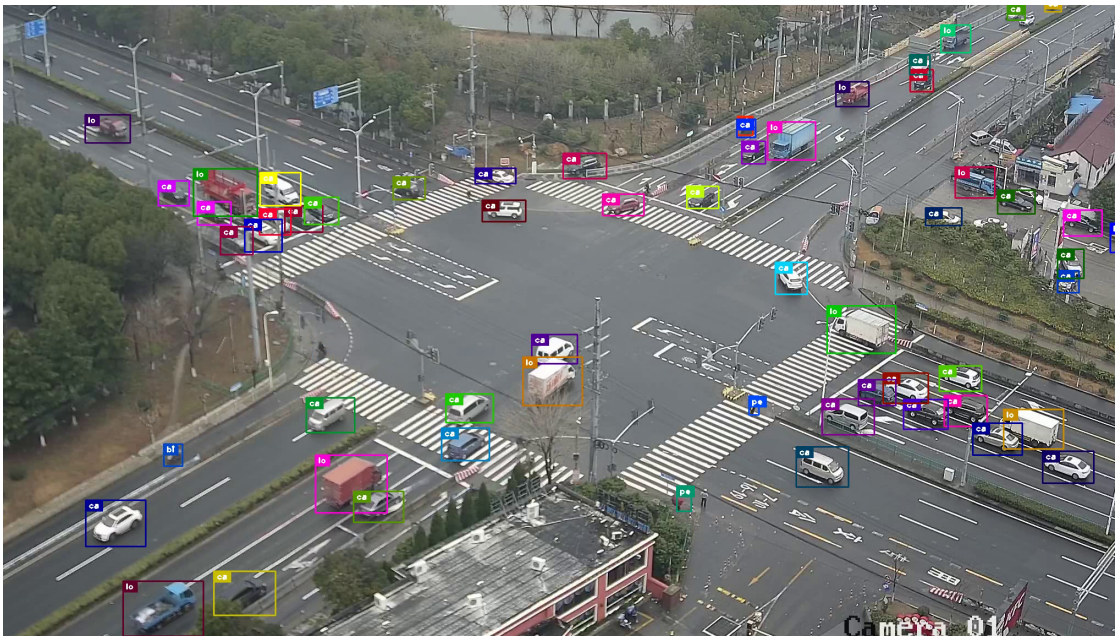


Figure 1: A typical DLR traffic scene

both object detection and tracking methods is used. Object detection is a task in computer vision, which uses an image classifier to figure out what is present in the image and where. This can be accomplished through the use of Convolutional Neural Networks (CNNs) which makes it possible to detect and classify multiple

objects in a single pass of an image. The detected object is fully enclosed in a bounding box, describing its location in the image. With the help of a tracking algorithm one can then identify the detected object over a period of time, usually while it is present in the scene.

Tracking of objects in visual data is a challenging and complex task, especially when done online and in real world scenarios. Missing tracks, re-identification of objects after occlusions or reassigning a new id to an already tracked object, even without any occlusions, are only some challenges to overcome. Janz showed in his thesis [1] that especially track id switches in the area of traffic surveillance are of high significance and should be as small as possible. Online applications which are supposed to run in realtime share the problem of not having any information about future events. Although tracking objects in an offline scenario is still a demanding task, it offers the advantage to leverage from future information as well.

The SORT [2] method is a tracking algorithm which runs in real time using only the provided detections, represented as bounding boxes in 2D image space, from a CNN. SORT's tracking performance relies heavily on the provided detections and missing detections can lead to lost tracks or track id switches.

But tracking can not only be done with the information about bounding boxes, there are also approaches to use visual information from the provided image directly. The concept of optical flow (see 2.6) provides motion vectors which can be exploited to track previously calculated keypoints (see 2.5). Although the calculation of optical flow can be computationally expensive there are approaches to work in realtime (see 4.2).

The proposed method uses the Robust Local Optical Flow [3] method to track feature points, calculated by OpenCVs[1] Oriented FAST and Rotation Aware BRIEF [4] key point detector. The derived motion vectors are then combined with the SORT tracking method to achieve a higher accuracy and be more failsafe against track id switches. The extended tracking algorithm is benchmarked and evaluated on parts of the MOT15 [5] benchmark and a self created benchmark based on DLR data.

1.1. Contributions

With this thesis, the author contributes the following:

- An implementation of an extension with sparse optical flow to the SORT tracking algorithm 5.2.
- A data set with ground truth for tracks on a DLR real world scenario 6.1 for benchmarking.

[1] `https://www.opencv.org`

- Evaluation of the elaborated extension on the MOT15 [5] and DLR data sets 6.2.
- An implementation of the original SORT in C++ 5.2.

1.2. Outline

The thesis is organised as follows: section 2 introduces key concepts and relevant theory, laying the foundation of this work. The SORT tracking algorithm is described in chapter 3. The research question is then defined in section 4 which also gives a brief overview on recent work on multiple object tracking, optical flow and feature point detection. Followed by the implementation details described in section 5, the used metrics and benchmarks to evaluate the implementation and the results are described in section 6. Chapter 7 presents the conclusion as well as ideas for future work.

2. Preliminaries

This section covers basic terms and concepts as well as algorithms and methods which lay the foundation of this thesis.

2.1. Tracking

Tracking objects in image data is the process of locating an object over time in this data and if there is more than one object to track in the scene it is referred to as multiple object tracking (MOT). The source material can be provided for example by a camera, a video file, a radar or lidar. In the scope of this thesis the source material are single images as a sequence. For us humans it is pretty easy to tell if the red car which passes from left to right in a video sequence is the same all the way. For a computer system this is a difficult task with several problems to solve, depending on the complexity of the scene. Compared to tracking only one object, MOT can be really challenging as each object in the scene has to be detected first and then assigned to its own track. For example, tracking a red dot which moves on a white surface is less demanding than tracking multiple participants in a traffic scenario with changing illumination and occlusion situations.

Tracking algorithms can be divided into two fields: batch and online methods. While batch methods can exploit future information, online methods on the other hand can only make use of present and past information.

Tracking by Detection To determine which objects need to be tracked the objects need to be detected first. Doing this by hand is not applicable to a real world use case, especially in the case of MOT, because new objects can enter or leave the scene rather quickly. But with the improvement of visual object detection through the use of deep learning and neural networks over the last years, recent research in MOT has focused on the so called tracking-by-detection paradigm. Within this paradigm, objects are detected by a properly trained neural network first and the detections are then used for further processing. The detections are usually provided as bounding boxes, a rectangle around the detected object described in 2D coordinates in image space.

2.2. The Kalman Filter

This section only provides a very brief introduction to the Kalman filter, for a more detailed insight, see Welch *et al.* [6].

The Kalman filter[7] is an algorithm to produce estimates of unknown variables. To accomplish this task the algorithm uses a series of measurements over time, which contain statistical noise. The estimated variables tend to be more accurate than those based on a single measurement alone. The algorithm works in a two-step process. First, in the prediction step (also known as time update), the filter calculates estimates of the current state variables out of the previous state and the motion model. Second, in the update step (also known as correction), the estimates are calculated again, but with the aid of the measurements.

The Kalman filter consists of several matrices, which will be described a bit more detailed. First one needs to describe the state of the problem x_k at time step k. This multidimensional vector contains observable values as well as values one can't observe but which describe the state of the model. Further matrices defining the Kalman filtering conditions are [6]:

- F, the state transition matrix (sometimes called A, also called the transfer matrix or transition matrix). This matrix describes "where" the state vector moves from one time step k to the next. When x_k is the n-dimensional state vector, F is $n \times n$ dimensional.

- H, the measurement matrix describes which values of x_k will be updated by measurements. When x_k is the n-dimensional state vector and z_k is the m-dimensional measurement vector, H is $m \times n$ dimensional.

- Q, the process noise matrix, describing how much the estimated state is

allowed to differ from the model between steps. When x_k is the n-dimensional state vector, Q is $n \times n$ dimensional.

- R, the measurement noise matrix, describing the uncertainty within the measurement process. When z_k is the m-dimensional measurement vector, R is $m \times m$ dimensional.

- P, the covariance matrix. For the initial step, this matrix is set with high uncertainty. When x_k is the n-dimensional state vector, P is $n \times n$ dimensional.

In section 5.2 as well as in appendix A it is shown how these matrices are designed in practice . The complete operation of the Kalman filter is shown in figure 2. The variable K, the Kalman gain, is not described here, but think of it as a factor which tells the filter how to weigh new measurements against what the filter already "knows". The loop shown in figure 2 can also run without the update step. This is usually the case when no measurement update is available, for example due to sensor failure. But of course it will lead to higher uncertainties in P.

As one can see the Kalman filter uses only the previously calculated estimates and the present input measurements, thus no additional information from the past is required, making it possible to run the filter online.

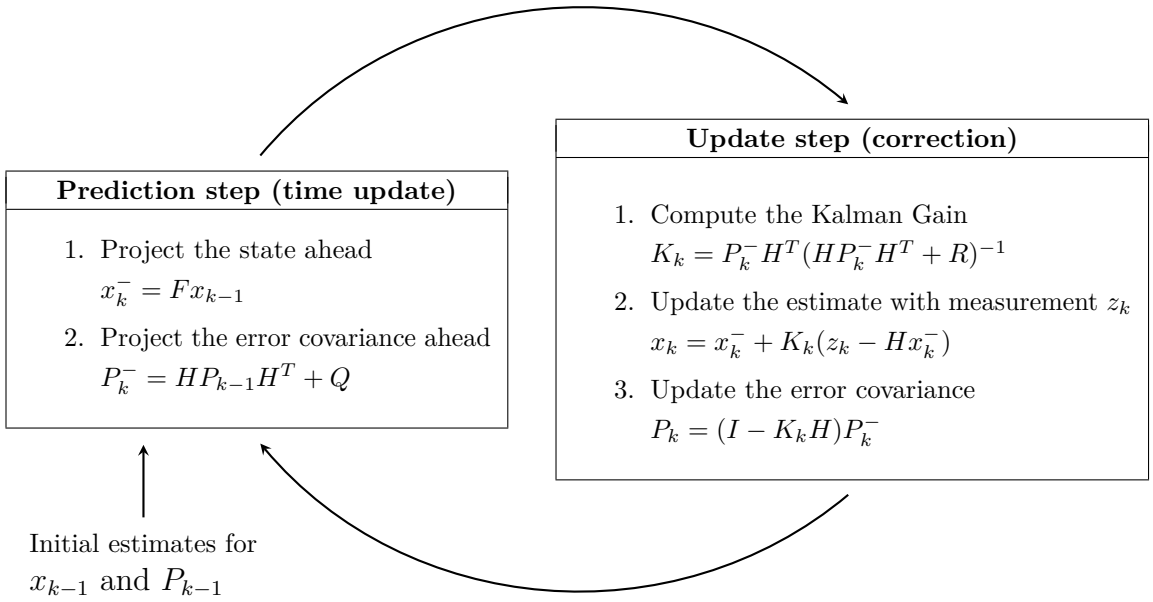


Figure 2: Operation of the Kalman filter
Scheme after Welch *et al.* [6, p. 6], x_k^- is the *a priori* state vector and P_k^- the *a priori* estimate error covariance.

Mahalanobis Distance Since the Kalman filter itself provides no way to reject bad measurements a metric to accomplish this task is needed. The Mahalanobis distance is a statistical measure of the distance of a point from a Gaussian distribution, introduced in 1930 [8]. Since the state in the Kalman filter is nothing else than such a distribution, the Mahalanobis distance gives a decent metric to distinguish between good and bad measurements. It is defined [9] as

$$D_m = \sqrt{(x-y)^T S^{-1}(x-y)}$$

with x as the observation, y the mean and S the covariance matrix of the collected data.

2.3. Linear Assignment Problem

When it comes to multiple object tracking with the tracking-by-detection paradigm one major problem is to assign the incoming detections to existing tracks. Or, if this is not possible, to create new tracks. This association problem can be formulated as a linear assignment problem [2] which then can be solved using suitable algorithms. Informally, one can describe the assignment problem as follows: a given $n \times n$ cost matrix $C = (c_{ji})$ and one wants to match each row to a different column in such a way that the sum of the corresponding entries is minimized. In other words, when C represents detections and tracks, respectively, each detection is assigned to a track. The next section explains with what values, the costs, the matrix can be filled.

One common method to solve this problem is the Hungarian method[10] with a time complexity of $O(n^3)$ [11]. A less common algorithm to solve the linear assignment problem but with uniformly lower computation time is the Jonker-Volgenant method[11].

2.4. Intersection-Over-Union

Intersection over union (IOU), also known as the Jaccard index, is, generally spoken, a measure for the similarity of quantities. Given two finite sets A, B, the index J is calculated by dividing the size of the intersection by the size of the union of the sets:

$$J(A,B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid}$$

In the given scenario let A and B be the areas of the bounding boxes provided by the object detection, illustrated in figure 3. The intersection over union can be

used as a metric and the calculated value can then be used to fill the cost matrix from section 2.3.

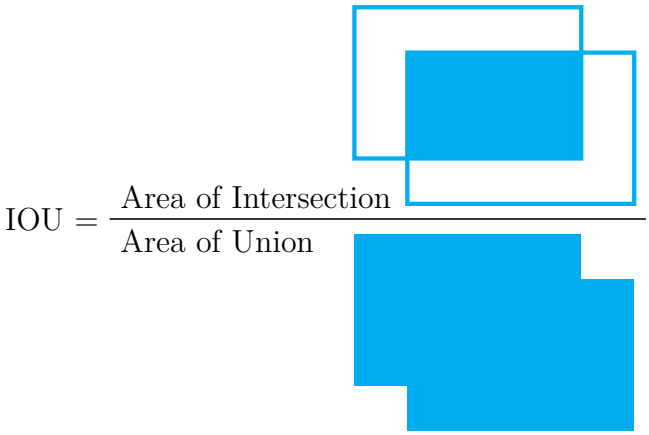


Figure 3: Intersection over union with bounding boxes

2.5. Keypoints

A keypoint, also called a feature point or feature, is a small portion of an image that, for one reason or another, is unusually distinctive, describing an "interesting" region in the image. Keypoints can be used to create a signature for the image and to locate the "interesting" region in another, related image. Distinctive points are usually corners, edges or areas which vary in color or intensity.

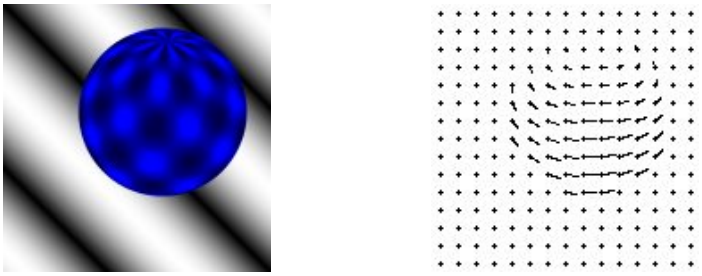
2.6. Optical Flow

The concept of optical flow describes the pattern of motion of objects in an image sequence caused by the relative motion between the observer and the scene. Typical use cases are motion estimation of an object in a scene, image stabilizing of video cameras or motion compensation in video compression algorithms.

In terms of computer vision, you can say that for each pixel in image one, you want to know where the pixel in image two moved to. Common optical flow algorithms assume that the observed object persists in time, meaning that the intensity of a small region in two consecutive frames remain the same. This can be formulated with

$$I(x, y, t) = I(x + u\delta t, y + v\delta t, t + \delta t)$$

where $I(x, y, t)$ is the image intensity, x, y the pixel position, u, v the corresponding motion vector and t the frame number. Chapter 4.2 gives a brief overview on different research approaches to solve this equation.



(a) Rotating sphere (b) Optical flow field

Figure 4: Visualization of the optical flow
Images courtesy of Computer Vision Research Group, Department of Computer Science, University of Otago, Dunedin, New Zealand, `http://of-eval.sourceforge.net/`, accessed: October 1, 2020

Furthermore, a distinction is made between the so called dense and sparse optical flow. The former calculates the motion vector for every pixel in the image while the latter track only a subset of certain feature points. Figure 4 illustrates the motion vector field calculated by dense optical flow for a rotating sphere. The computational cost for dense optical flow is quite high, hence it is usually used in academic fields or for use cases where online functionality is not an issue. As for sparse optical flow, computational cost compared to dense optical flow algorithms is low, making it a suitable technique for practical applications in realtime scenarios. As already stated in the title, this work makes use of sparse optical flow.

3. SORT

This section describes the 2016 proposed *Simple Online and Realtime Tracking* method (SORT) [2] by Bewley *et al.*, a tracking framework to address the MOT problem in realtime scenarios.

Basic Procedure SORT works by the tracking-by-detection paradigm (see 2.1), hence it highly depends on the quality of the provided output of the object detection. It is an online tracking algorithm (see 2.1), that is has no access to future information of the scenario. The detection output is provided by a convolutional neural network (CNN) detector in the form of bounding boxes around the detected objects. Coordinates of these bounding boxes are in 2D image space. Using key concepts and algorithms described in 2.1, 2.2, 2.3 and 2.4 the basic procedure of SORT is:

- Get detections from the CNN. If this is is the very first frame then only new tracks will be created. Each track has its own Kalman filter.

- Propagate tracks with the Kalman filter in the next frame.
- Compute intersection-over-union of all detections and propagated tracks.
- Associate detections to tracks based on the IOU distance.
- Update the track's Kalman filter with the associated detection.
- Filter out lost tracks, tracks with low IOU distance or create new tracks.

Estimation Model As stated above, each track (also called target) has its own Kalman filter. The state of each target in the Kalman filter is modeled as:

$$x_k = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}].$$

In this state, u and v represent the center of the target's bounding box as pixel coordinates, s is the scale (area) and r is the aspect ratio of that bounding box. $\dot{u}, \dot{v}, \dot{s}$ are the velocity components. The aspect ratio is considered to be constant. When a tracker is associated with a bounding box from the CNN the tracker's state is updated with that detection. The velocity components are solved by the Kalman filter. If a target has no associated detection, the state is simply predicted without the measurement step (see 2.2, 2).

Data Association To address the assignment problem between existing tracks and detections (see 2.3) SORT uses the intersection-over-union between each detection and every existing target. These values are used to fill a cost matrix which then is solved by the Hungarian algorithm[10] to assign each track a proper detection. The authors also impose a minimum IOU threshold to reject assignments where the overlap between bounding box and target is less than $IOU_{\min}$.

Deletion and Creation of Tracks To create a new track, the state in the Kalman filter for the new track is initialized with the geometry of the bounding box. The velocity is set to zero as it is not observable at this point. To describe this uncertainty to the Kalman filter the covariance of the velocity component is initialized with high values (see appendix A). And finally to prevent tracking of false positives the new candidate needs to go through a probationary period of assigned detections. Targets are deleted if no detection is assigned for a set threshold of T_{Lost} frames.

Summary The SORT method makes a pragmatic and simple approach to deal with multiple object tracking using classical yet efficient methods like the Kalman filter [7] and the Hungarian method [10]. With this practice the algorithm achieves high speed and reasonable accuracy ([2], also see 6.2), making it suitable for realtime applications and scenarios. For example, on an AMD Ryzen7 3700X 4GHz machine with 64GB memory, which is the machine used in this thesis for evaluation, the original Python implementation[2] runs at 1100 FPS on the MOT15 [5] benchmark. Nevertheless the results are highly dependent on the performance of the used detection framework, that is to say if it comes to detection failures SORT will loose it's accuracy. But the authors also claim to not focus on object re-identification or being robust against detection errors.

4. Research

This section defines the research question and takes a glimpse on previous approaches and developments in multiple object tracking, optical flow and key point detection.

4.1. Research Question

This thesis deals with the question if and how it is possible to enhance the *Simple Online and Realtime Tracking* [2] algorithm with sparse optical flow in a way that accuracy is improved, especially with regard to track id switches, but realtime functionality is still maintained.

To accomplish this task, research focuses on similar tracking methods, different optical flow and key point detection algorithms. With this as a basis a suitable extension is elaborated and implemented.

4.2. Related Work

Multiple Object Tracking With the improvement of visual object detection over the last years, recent research in MOT has focused on the tracking-by-detection paradigm. Within this paradigm, object trajectories are usually found in a global optimization problem that processes entire video batches at once.

Multiple Object Tracking Using K-Shortest Paths Optimization [12] for example uses network flows to accomplish this task. But due to batch processing this and similar methods are not applicable to our online scenario.

[2] https://github.com/abewley/sort

Online processing methods on the other hand try to solve the data association problem for example determinative (for example with the Hungarian [10] algorithm in [2], [13] or with greedy association [14]). *Tracking by Decision Making* [15] aims to build appearance models to re-identify objects over a period of time. In addition to appearance models, motion is often incorporated to assist association detections to tracklets, described in *The Way They Move* [16].

DeepSORT [13] for example extends the original SORT method to integrate appearance information based on a deep appearance descriptor. By this addition the tracker becomes less prone to occlusion problems or id switches caused by occlusion or missing detections. The downside of this approach is an offline pre-training stage for the association metric and the complex computation of the convolutional neural network. Yet this approach still operates in real time.

High-Speed Tracking-by-Detection Without Using Image Information [14] is a simple intersection-over-union tracker which only uses an IOU metric from the provided detections. Hence it is extremely fast but relies highly on the object detection which leads to accuracy problems in realtime environments. Yet this method has been improved in *Extending IOU Based Multi-Object Tracking by Visual Information* [17] by adding visual information in form of the KCF tracker, proposed in *High-Speed Tracking with Kernelized Correlation Filters* [18].

Optical Flow As a common topic in computer vision ideas for the computation of optical flow and different approaches have evolved over the past years. As described in section 2.6, a common technique is to assume that brightness is constant in two consecutive frames. This is formulated with:

$$I(x, y, t) = I(x + u\delta t, y + v\delta t, t + \delta t)$$

The 1981 proposed method by Horn *et al.* [19] solved this equation by expanding it as a first order Taylor series and adding a soft spatial coherence as a second assumption to the brightness constancy. This approach is also called a *global* approach. The also in 1981 proposed method from Lucas *et al.* [20] replaces the global constraint with a local one in which the motion vectors are expected to be constant in a small region. Both methods assume that the motion between two consecutive frames is very small which is often not the case in real world scenarios. To address this problem a hierarchical procedure is used, where the flow is first calculated on a coarser scale and then successively refined on a finer scale. This technique is also used in OpenCV's implementation of the Lucas-Kanade method, described by Bouguet[21]. FOLKI [22], proposed in 2005 by Le Besnerais *et al.*,

like Lucas-Kanade, uses a window-based registration, but with a specific Taylor expansion. The 2013 proposed *Large Displacement Optical Flow from Nearest Neighbor Fields* [23] also addresses this problem. In *Robust Local Optical Flow for Feature Tracking* [3] (RLOF) from 2012 Senst *et al.* use a modified Hampel estimator instead of the Least Squares estimator as Lucas-Kanade did, improving robustness against outliers with only minimal impact to computational costs. The RLOF method subsequently was improved, among other things with regard to illumination changes and large displacements [24]–[26].

Other approaches to calculate the flow field are for example Farnebäck's 2003 [27] proposed method which uses polynomial expansion. The TV-L1 [28] method by Zach *et al.* from 2007 uses variational methods and the L^1 norm to increase robustness against illumination changes, occlusions and image noise. *Fast Optical Flow Using Dense Inverse Search* [29] by Kroeger *et al.* from 2016 focuses on low time complexity for the calculation of a dense flow field, to make dense flow suitable for real-time applications.

With the recent development in machine learning, especially for computer vision tasks, the problem to determine the optical flow has been transferred to this research field as well [30]–[33]. *DeepFlow: Large displacement optical flow with deep matching* [30] from 2013 by Weinzaepfel *et al.* proposes a descriptor matching algorithm build upon a deep convolutional network structure. The 2020 proposed *RAFT: Recurrent All-Pairs Field Transforms for Optical Flow* by Teed *et al.* proposes a deep network architecture for optical flow, similar to *Probabilistic Pixel-Adaptive Refinement Networks* [32], proposed in 2020 by Wannenwetsch *et al.* Although these methods show promising results in terms of accuracy and robustness, they lack realtime performance. For example RAFT needs 550ms per frame on a 1080p video [33, p. 14] and deepflow takes 20 seconds for flow computation on the MPI-Sintel [30, p. 8] test set.

This work uses the Robust Local Optical Flow for tracking feature points.

Feature Point Detection As described in section 2.5 keypoints are distinctive points in an image such as corners or edges. Early work focused on concepts like corner detection and the most commonly used definition of a *corner* was provided by Harris [34]. Harris calculated the pixel intensity for the image with an autocorrelation function and corners are defined as places in the image where the autocorrelation matrix has two large eigenvalues. Since the calculation of eigenvalues is expensive Harris defined a function, also known as the *Harris measure*, which compares the eigenvalues without requiring their explicit computation. The corners

can then be found as local maximum of that function.

Shi and Tomasi [35] later found that "good" corners can be determined if the smaller of the two eigenvalues was greater than a minimum threshold. This variant proved to be sufficient and gave in many cases even better results than the Harris feature detector and is often referred to as *Good Features to Track*.

Both Harris and Shi-Tomasi corner detections are not scale-invariant. This problem was addressed with *Scale Invariant Feature Transform* (SIFT) [36]. The algorithm can also create a feature point descriptor, which simply describes how feature points differ from each other. Although SIFT offers a rich set on information for found feature points it is relatively slow. The *Speeded-Up Robust Features* (SURF) [37] improves the speed of keypoint and descriptor detection and is around three times faster than SIFT but with comparable performance in terms of detection. Nevertheless SURF is still too computationally expensive for real-time applications. It should also be noted that both SIFT and SURF are patented.

The 2006 proposed *Features from Accelerated Segments Test* (FAST) [38] uses a direct comparison between a point p and a set of points on a small circle around p. The points on the ring are classified as either darker than p, lighter than p, or similar to p. If the number of contiguous points in the arc containing only lighter or darker points is more than half the total number of points defining the ring, p is considered to be a feature point. To avoid the detection of multiple interest points in adjacent locations, non-maximum suppression is applied. FAST, like the Harris detector and the Shi-Tomasi detector does not calculate a feature point descriptor.

The *Oriented FAST and Rotation Aware BRIEF* feature (ORB) [4] from 2011 is OpenCV's[3] own development to come up with a feature point detector alternative to SIFT and SURF. The algorithm uses FAST for the feature point detection and calculates the Harris measure for the located points. This is done to overcome the problem that feature points computed from FAST respond to edges as well as corners. The Harris measure is also used as a metric for feature quality and can be used to choose the strongest feature points in an image. ORB also adds an orientation to the located feature points, an attribute FAST is missing. In a second stage ORB computes the feature descriptors with a variation of the BRIEF [39] algorithm.

This work uses FAST and the first stage of ORB for feature point detection.

[3] `https://www.opencv.org`

5. Implementation - SORT_OF

This section covers the two implementation approaches. The first approach is described with a brief methodology overview followed by its disadvantages. The second attempt is described in more detail, explaining design decisions made during the development process and technical details. In the first approach the target's Kalman filter model was tried to replace while in the second approach the measurement model was extended. In both cases the solving of the data association problem kept its original functionality.

5.1. First Approach - Replacing the Kalman Filter with Tracked Feature Points

As described in section 3, SORT's tracking quality highly depends on detection quality. In this first attempt it was tried to always provide SORT with bounding boxes for active tracks, eliminating missing detections from the CNN. The general idea for this approach was to completely remove SORT's Kalman filter model (see 3) and replace it with a self developed algorithm. The provided detection from the CNN was only used as an initial bounding box, a region of interest to detect feature points in.

Brief Methodology Overview Firstly, feature points had to be detected in each new emerging bounding box in the scene provided by the object detection. Then the optical flow of these points had to be calculated to track them, which is basically what the Kanade-Lucas-Tomasi (KLT) feature tracker [40] does. Additionally the new position of each feature point is predicted with a Kalman filter model. The state of a feature point was modeled as:

$$x_k = [x, y, \dot{x}, \dot{y}],$$

where x, y are the pixel coordinates of the feature point and $\dot{x}, \dot{y}$ their respective velocities. This step was made to ensure that the tracked point didn't get stuck at visual barriers in the scene. With these steps combined, a predicted bounding box for calculating the intersection over union is always provided, even if detection output is missing. This could easily be achieved as the tracked feature points provide a velocity and only that velocity was added to the 2D coordinates of the initial bounding box.

Disadvantages But this attempt has a lot of drawbacks. First of all, there is the detection of feature points, for which the Shi-Tomasi corner detector [35] was used. The downside was that there was no guarantee that every detected feature point was part of the object of interest (figure 5) which implies that it had to be somehow tested if every feature points is part of the object. Secondly, the case that

Figure 5: False positives from Shi-Tomasi
The red dots marking the detected feature points which in the best case should be all on the object.

a feature point got stuck on a visual barrier had to be handled. As already said, every tracked feature point had its own Kalman filter which was updated every frame with the new pixel position $[x, y]$ calculated by the optical flow. After the update process, the Mahalanobis distance against a certain threshold was checked. If this threshold was exceeded, the actual point coordinates were switched against the predicted ones for the optical flow calculation in the next frame. Although it was achieved that the feature points were more robust against visual barriers, the question on how to set the threshold for testing against the Mahalanobis distance remained unanswered. Also, switching the feature points if they got stuck, will add bad measurements into the Kalman filter framework, something one usually wants to avoid.

Furthermore, with calculating the bounding box out of the tracked feature points there was no influence on the aspect ratio of the bounding box. For example imagine the following scenario: a camera watches a t-shaped crossroad from a bird-eye view like angle and a car passes by from left to right. The bounding box around the car would be the shape of a rectangle with a 4:3 aspect ratio. But if the car turns, the aspect ratio of the bounding box should change.

Additionally, the method had no measurement or metric to decide which one of the tracked feature points should be used to update the bounding box in its position in image space. Instead, the average velocity of all feature points was used, which, however, could be strongly distorted by the problems mentioned above.

And finally it is undefined when a track ends. SORT deletes an active track if no detections can be associated in a given amount of time steps. In this approach it was not defined if and how a track can end in image space. The track was only deleted if the tracked key points moved outside the visual scene.

Summary It quickly became obvious that this approach had too many uncertainties and corner cases to cover, making the implementation too complicated. Because of this there was no evaluation done.

5.2. Second Approach - Extending the Kalman Filter Measurement Model

With the experience from the first approach a more generic way was elaborated. Instead of replacing SORT's Kalman filter, the measurement model of SORT's Kalman filter is extended for a velocity component.

Methodology The following paragraphs describe the proposed method by explaining key steps, decisions in the development process and changes made to the original SORT algorithm. This includes the extension of the measurement model, determine feature points, calculation of the optical flow and the update process.

Extending the Measurement Model As described in section 3, SORT's state

$$x_k = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]$$

is updated with the center of the bounding box u, v, the scale s and the aspect ratio r. Equation 2 shows the resulting measurement vector. The state variable x has dimensions 7×1 and z has dimensions 4×1. By multiplying the matrices the size of H can be deduced to 4×7, as shown in equation 1.

Since the optical flow provides a velocity component the intention here is to use this velocity as a measurement to additionally update $\dot{u}, \dot{v}$. With the velocity components for u, v the update vector z is extended to $[u, v, r, s, \dot{u}, \dot{v}]$ and the measurement matrix H now becomes a 6×7 matrix, shown in equations 3 and 4.

$$H_{SORT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (1) \qquad z_{SORT} = [u, v, s, r] \quad (2)$$

$$H_{SORT\_OF} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (3) \qquad z_{SORT\_OF} = [u, v, r, s, \dot{u}, \dot{v}] \quad (4)$$

Determine Feature Points Since the use of sparse optical flow was intended, as a first step feature points to track had to be found. The first attempt used the Shi-Tomasi corner detector [35]. This often lead to false positives, feature points that are in the region of interest, but are not part of the actual object (figure 6).

This time the ORB feature detector [4] was used. As mentioned in section 4.2, ORB tends to be more robust and also provides a metric to only output the n-strongest corners. Although this does not imply that the n-strongest feature points will be on the object of interest in any case, they will be ranked later to determine the best for the update process. This ranking process, described in the next but one paragraph, is computationally demanding and to keep the effort as low as possible the number of calculated feature points are therefore limited. It should also be added that with a higher number of feature points, the effort required to calculate the optical flow increases. ORB is used only to detect key points, but not to calculate the descriptors. Even though ORB tends do be more robust and

(a) ORB (b) Shi-Tomasi (c) FAST

Figure 6: ORB vs. Shi-Tomasi vs. FAST

the derived feature points are of decent quality, it is still possible that ORB fails and a found key point is not part of the object. Therefore the feature points are calculated new for every active track in every time step, also meaning that feature points are tracked by the optical flow over two frames only.

In the case that ORB fails and provides no feature points, the FAST [38] key point detector is used as backup. Due to the lack of a metric to measure feature point quality FAST outputs a lot of false positives (figure 6) and is in this case considered to be a "hit it by all means" - method. If FAST returns no feature points as well, no optical flow for this track will be calculated in the current time step.

Optical Flow Calculation Although there are a lot of interesting approaches to determine the optical flow (see 4.2), in the end it was decided to go with the Robust Local Optical Flow (RLOF) method [3]. It is already implemented in the OpenCV library[4] which is used in this thesis for several reasons:

- It provides convenient ways of handling video data.
- It provides implementations of the ORB and FAST feature point detectors.
- It provides implementations of various optical flow algorithms.
- It provides an implementation of the Kalman filter algorithm.
- It has a Python API.

Since the original SORT implementation is in Python, the intention was to use it as a basis in combination with OpenCV's Python API. Unfortunately I had to discover a bug[5] in OpenCV's implementation of the RLOF method when using the Python API which could not be fixed in the processing time of this work. Hence the implementation is written in C++, which is also OpenCV's main development language. For better comparability and to exclude any errors based on different types or internal processes a reimplementation of the original SORT method in C++ was done, too. The code for this reimplementation is publicly available[6].

The implementation of RLOF in OpenCV returns a set of points p_{next} for a given set of feature points p_{prev} for which the optical flow was found on two sequential images. Since the time step between those two images is $dt = 1$, the velocity of the points can then be calculated as $v_{next} = p_{next} - p_{prev}$.

In this implementation the feature points from the recent frame are the input to calculate the optical flow to the previous frame, meaning the flow is calculated "backwards". Hence, the leading sign has to be changed to get the velocity from the frame before to the recent one. If no optical flow is found, the track will not be updated with a velocity, but at least with a bounding box, if associated, which is SORT's original behaviour.

Detecting Bad Measurements This implementation approach also does not provide a way to detect and reject bad measurements. Instead the measurements to update the Kalman filter are ranked. This reduces the probability to update with a bad measurement but of course does not completely prevents it. In this case a bad measurement is the optical flow calculated from a feature point which is not part of the object, got stuck on a visual barrier or an outlier. That is, if all detected feature points are not part of the object, the target will be updated

[4] `https://www.opencv.org`
[5] `https://github.com/opencv/opencv_contrib/issues/2663#issuecomment-708056333`
[6] `https://github.com/tylernewnoise/SORT_in_Cpp`

nevertheless with one velocity, which in this case will be false.

Therefore the Mahalanobis distance D_m is calculated (see 2.2) for each new calculated velocity against the velocity in the Kalman filter from the frame before. Equations 5 and 6 show how the mean mean vector y and the covariance matrix S are derived. The velocity with the smallest distance is then used to update the Kalman filter. With this technique the algorithm does not depend on a threshold.

$$x_k^- = [u, v, r, s, \dot{u}, \dot{v}, \dot{r}] \Longrightarrow y = [\dot{u}, \dot{v}] \tag{5}$$

Derivation of mean y from state x_k^-

$$P_{SORT\_OF} = \begin{bmatrix} \sigma_u^2 & \sigma_{uv} & \sigma_{us} & \sigma_{ur} & \sigma_{u\dot{u}} & \sigma_{u\dot{v}} & \sigma_{u\dot{s}} \\ \sigma_{vu} & \sigma_v^2 & \sigma_{vs} & \sigma_{vr} & \sigma_{v\dot{u}} & \sigma_{v\dot{v}} & \sigma_{v\dot{s}} \\ \sigma_{su} & \sigma_{sv} & \sigma_s^2 & \sigma_{sr} & \sigma_{s\dot{u}} & \sigma_{s\dot{v}} & \sigma_{s\dot{s}} \\ \sigma_{ru} & \sigma_{rv} & \sigma_{rs} & \sigma_r^2 & \sigma_{r\dot{u}} & \sigma_{r\dot{v}} & \sigma_{r\dot{s}} \\ \sigma_{\dot{u}u} & \sigma_{\dot{u}v} & \sigma_{\dot{u}s} & \sigma_{\dot{u}r} & \sigma_{\dot{u}}^2 & \sigma_{\dot{u}\dot{v}} & \sigma_{\dot{u}\dot{s}} \\ \sigma_{\dot{v}u} & \sigma_{\dot{v}v} & \sigma_{\dot{v}s} & \sigma_{\dot{v}r} & \sigma_{\dot{v}\dot{u}} & \sigma_{\dot{v}}^2 & \sigma_{\dot{v}\dot{s}} \\ \sigma_{\dot{s}u} & \sigma_{\dot{s}v} & \sigma_{\dot{s}s} & \sigma_{\dot{s}r} & \sigma_{\dot{s}\dot{u}} & \sigma_{\dot{s}\dot{v}} & \sigma_{\dot{s}}^2 \end{bmatrix} \Longrightarrow S = \begin{bmatrix} \sigma_{\dot{u}}^2 & \sigma_{\dot{u}\dot{v}} \\ \sigma_{\dot{v}\dot{u}} & \sigma_{\dot{v}}^2 \end{bmatrix} \tag{6}$$

Derivation of S from P

Update Process The update of the target's Kalman filter is the final step and involves multiple scenarios. Table 1 gives a matrix-like overview on which occasion which update is chosen. Figure 7 shows the different measurement functions used for the update process.

The following list describes the cases and their reasons:

- A bounding box and a velocity is provided. This is the standard scenario where a detection is associated with the target, feature points are detected and a velocity from the optical flow is calculated.
- Only a velocity is provided. This indicates that no detection was associated with the target. In this case the detection of feature points and the succeeding flow calculation are done for the predicted bounding box of the Kalman filter. The update of the latter is done only with the velocity component, if found.
- Only a bounding box is provided. In this case a detection is associated to the target but, for whatever reason no feature points could be detected or no optical flow could be calculated.

- None of the above is provided, the Kalman filter of the target gets no update. This will happen if no detection is associated and in the predicted bounding box the feature point detection or the optical flow calculation failed.

	Detection is associated with target	No detection is associated, use prediction instead
Optical flow is found	Update with detection and velocity	Update with velocity only
Feature point detection or optical flow calculation failed	Update with detection only	No update

Table 1: Kalman filter update scenarios and how they are processed

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(a) Update with detection and velocity

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(b) Update with velocity only

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) Update with detection only

Figure 7: Measurement matrices for different update cases

Additional Corner Cases The implementation up to this point had only one corner case left, namely when the bounding box of the target moves, even partly, outside of image space. If this happens, feature point detection could not be done on the complete region of interest represented by the target's bounding box. This is covered by always checking if the target's bounding box is in image space. If not, an additional test checks if an IOU with the remaining part and a detection in image space is still possible within the given IOU threshold $IOU_{\min}$ (see 3). The track is deleted if the test fails, otherwise the track's bounding box is resized to fit in image space and feature point detection is done in this resized bounding box.

Summary Putting it all together, listing 1 displays the basic procedure. With this approach the extension of SORT with optical flow was kept simple and generic, extending only the measurement model of the target's Kalman filter. The number of corner cases to cover was kept to a manageable amount and the integration

depends only on two sequential frames. Deletion and creation of tracks keeps its original functionality as well as the data association - assigning detections to tracks based on an IOU metric (see 3, 2.4). The final code is available at a public Github[7] repository.

Algorithm 1 SORT_OF

```
for Every frame in grayscale do
    for all tracks do
        Predict track bbox
    end for
    Assign detections to tracks
    Collect and delete inactive tracks
    for all targets do
        if ROI is inside image space or not too small then
            Detect feature points with ORB or FAST
        end if
    end for
    Delete targets which moved outside image space
    for all feature points do
        calculate Robust Local Optical Flow
    end for
    for all targets with a calculated velocity do
        Rank calculated velocity based on Mahalanobis distance
    end for
    for all targets do
        Update the target's Kalman filter
    end for
    Create and initialize new targets for unmatched detections
    return Active targets
end for
```

[7] https://github.com/tylernewnoise/SORT_OF

6. Evaluation

This section explains the metrics used to benchmark and evaluate the algorithm and presents the results. Furthermore, a brief look at ethical aspects is given and a conclusion is drawn.

6.1. Benchmark and Metrics

The proposed tracking algorithm is evaluated on the 2008 proposed CLEAR MOT [41] metrics MOTA and MOTP along with some additional metrics used in the MOTChallenge [5] benchmark. The following list describes the evaluation measures:

- **MOTA** Multi-Object Tracking Accuracy is computed as:

$$MOTA = 1 - \frac{\sum_t (fn_t + fp_t + id\_s_t)}{\sum_t g_t}$$

  The measure combines three error sources for the frame t with fn_t the number of false negatives, fp_t the number of false positives and id_s_t the number of identity switches. Higher is better and a perfect tracker has a MOTA value of 100%.
- **MOTP** Multi-Object Tracking Precision is computed with:

$$MOTP = \frac{\sum_{i,t} d_{t,i}}{\sum_t c_t}$$

  MOTP measures the alignment of predicted bounding boxes and the ground truth where c_t is the number of matches for frame t and $d_{t,i}$ is the bounding box overlap of target i with its assigned ground truth object. Higher is better and a perfect tracker has s MOTP value of 100%.
- **FP** The total number of false positives. Lower is better, 0 is perfect.
- **FN** The total number of false negatives, i.e. missed tracks. Lower is better, 0 is perfect.
- **ID Sw.** Number of identity switches, i.e. the number of times an already tracked object is assigned a new ID. Lower is better, 0 is perfect.
- **MT** Mostly tracked targets is the ratio of ground-truth tracks that are assigned the same label for at least 80% of the given image sequence.
- **ML** Mostly lost targets is the ratio of ground-truth tracks that are assigned the same label for at most 20% of the given image sequence. Lower is better, 0% is perfect.

Although SORT as well as SORT_OF do not rely on any visual information in terms of form and shape about the tracked object, the extension is tested and compared on a real world DLR scenario. To achieve this a simple benchmark was created from DLR data, termed SHAN-1. This data was taken from the actual scenario the tracking algorithm is used on. The data set contains 449 images with a 2560×1440 resolution, the ground truth detections of 68 tracks derived from a 25 FPS video from a non moving camera as well as detections for evaluation. The latter detections are created with the help of the tensorflow framework for machine learning[8]. A Faster R-CNN inception V2 from the tensorflow model zoo[9] was used, which was pretrained on the COCO dataset[10]. The convolutional network was retrained with 335 images of the scenario. The data set with detections, ground truth tracks and images is available online[11].

Evaluation was also done on the MOT15 [5] training set, but only on scenarios with a static camera setup. The provided detections for the MOT15 benchmark are taken from the authors of SORT [2, pp. 2–3]. The several setups and their properties are listed in appendix B.

6.2. Results

All test runs were done on an AMD Ryzen7 3700X 4GHz machine with 64GB ram. Results are calculated by py-motmetrics[12], a Python implementation of metrics for benchmarking multiple object trackers. Table 2 shows the overall results for the MOT15 benchmark with static camera setups. For both benchmarks the IOU threshold $IOU_{\min}$ for both SORT and SORT_OF was set to 0.3, as the original implementation does as well. The maximum number of feature points per detection (as described in 5.2) for SORT_OF is set to 50. Since there is an interest in the results for tracking over longer periods without matching detections the T_{Lost} (see 3) parameter is set to 1 and 25. The overall results for the MOT15 dataset for $T_{Lost} = 1$ show no mentionable improvement. But looking at the detailed results from the MOT15 data sets in appendix C and depending on the scenario, results show an improvement. Even on $T_{Lost} = 1$ the proposed method could reduce ID switches by ~10%. Although, setting a higher T_{Lost} value did not always improve performance significantly.

[8] www.tensorflow.org
[9] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md
[10] https://cocodataset.org/#home
[11] https://bit.ly/ShanTrackBench
[12] https://github.com/cheind/py-motmetrics

$T_{Lost} = 1$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
SORT	39.8%	72.1%	257	**3209**	7861	34	10
SORT_OF	**40.2%**	72.1%	**247**	3230	**7789**	34	**9**

$T_{Lost} = 25$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
SORT	16.0%	**71.1%**	**215**	9147	6458	48	9
SORT_OF	**18.2%**	70.9%	221	**8933**	**6238**	48	9

Table 2: MOT15 overall results for $T_{Lost} = 1$ and $T_{Lost} = 25$

For the SHAN-1 scenario evaluation was done for T_{Lost} from 1–25 and complete results can be found in appendix D. This finer granular evaluation was done for better comparison and to find an optimized value for T_{Lost}. The MOTA metric was chosen as reference and figure 8 plots the comparison of MOTA vs. ID switches over T_{Lost}. As one can see, with a small T_{Lost}, SORT and SORT_OF are almost on the same level with respect to ID switches and MOTA. But as T_{Lost} increases, SORT_OF is always better than SORT with respect to the characteristics. Then again, it is not practical to set T_{Lost} to huge values. For example, setting $T_{Lost} = 50$ would mean that a tracked object could be without detection for two seconds on a video sequence with 25 frames per second, but could be already out of sight for the detection framework for more than 30 frames and would increase false positives. The plot also shows that SORT has a maximum MOTA with $T_{Lost} = 10$ with 59 ID switches. SORT_OF on the other hand reaches its MOTA maximum at $T_{Lost} = 12$ and has 48 ID switches at this point, showing a reduction of ID switches of 18.6% for this scenario. Table 3 shows the detailed results for both T_{Lost} values.

$T_{Lost} = 10$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
SORT	81.4%	83.0%	59	1544	3998	**63**	7
SORT_OF	**81.9%**	**83.1%**	**53**	**1469**	**3913**	62	7

$T_{Lost} = 12$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
SORT	81.3%	83.0%	54	1680	3891	**63**	7
SORT_OF	**82.0%**	83.0%	**48**	**1569**	**3795**	62	7

Table 3: SHAN-1 results for $T_{Lost} = 10$, $T_{Lost} = 12$

Runtime performance for the two tracking methods is shown in table 4. On the MOT15 data set it is for both algorithms way above realtime requirements although the performance impact on SORT_OF is of factor 500 in comparison to SORT. For the runtime performance on the SHAN-1 data set SORT_OF is with 25FPS still able to run online. The performance loss in this scenario is around factor 100 for SORT_OF compared to SORT.

	MOT15	SHAN-1
SORT	~27000FPS	~2900FPS
SORT_OF	~50FPS	~25FPS

Table 4: Runtime performances

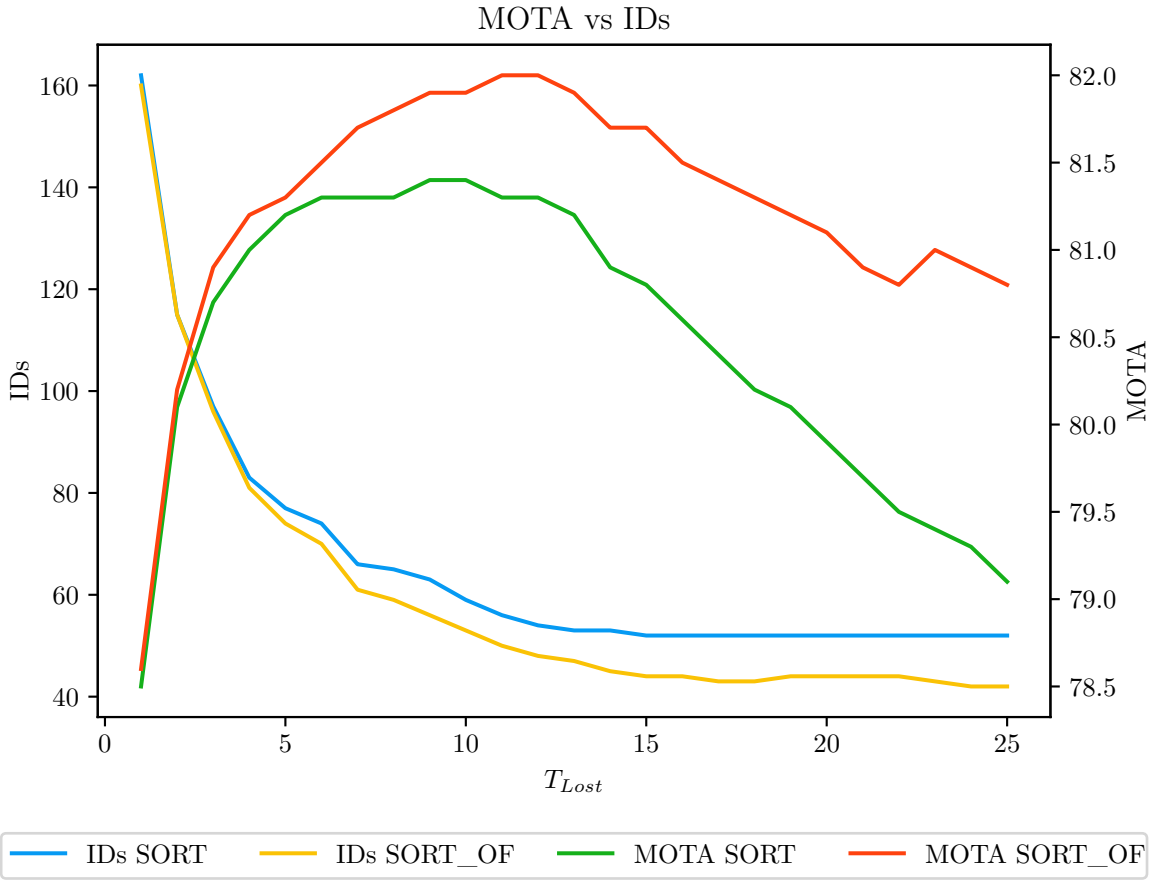


Figure 8: MOTA vs. IDs over T_{Lost} on SHAN-1 data set

6.3. Ethics

Multi object tracking is often related to surveillance and surveillance again is often associated with the constant tracking of individuals, for example by a government or a company. While there are obviously privacy issues it is possible to argue that tracking in video data is not the most effective way to do so in contrast to make use of peoples cell phones for example.

Although this work is made with the focus on a specific scenario with a static camera, multiple object tracking can also be used in robotics or autonomous driving, in which the perception of images would be from a moving point of view. For the first case one could argue that the work on multiple object tracking is helpful to make robots more sensitive to their environment, hence make them more supportive. In the case of autonomous driving the goal is to improve safety in road traffic. But also, if not already in use or developed, it is likely that tracking algorithms are used in weapon systems, like missile guidance or target localization. For this reason the original developer of the object detection framework YOLO [42] has discontinued all of his computer vision research[13].

As always it is unfortunate that much of the technology which was developed to improve peoples lives can also be used in a destructive manner. Still, the hope is that fast and accurate tracking algorithms could help and improve in certain scenarios and situations and the benefits will outweigh the potential downsides.

7. Conclusion

The present thesis elaborated two methods to extend the SORT algorithm with sparse optical flow. But in the scope of this work only the extension which simply extends the Kalman filter measurement model was implemented and examined. SORT_OF's computational cost is of factor 100 higher than SORT, but SORT_OF also lead to noticeable improvements against track id switches on the SHAN-1 dataset. Yet this extension is still capable to perform in realtime with at least 25 frames per second, even on images of size 2560×1440 pixels. The problem of re-identification remains but it is also not in the scope of SORT or this thesis.

For future work it might be interesting to see if the other approach that was described in 5.1 could be evolved in a way to have less corner cases to take care of. It may be more sufficient to not replace or remove the Kalman filter of SORT completely but only if detections are missing. Another aspect to have a look at is

[13] `https://twitter.com/pjreddie/status/1230524770350817280`

the use of dense optical flow, maybe not for the complete image, but for certain regions of interest to keep the performance impact low. Similar to [17] it may be also interesting to see if adding the KCF tracker [18] to SORT might be an improvement. Also, like Janz showed in *Post-Processing of Multi-Target Trajectories for Traffic Security Analysis* [1], an additional post-processing step or combination with a min-cost flow problem formulation, as described by Zhang *et al.* in *Global Data Association for Multi-Object Tracking Using Network Flows* [43] may lead to better results.

Another way to improve computational performance could be a GPU-based optical flow calculation. Object detection with a CNN is usually done on a GPU, so the image is already present in GPU memory, which makes further processing less expensive.

On the other hand it is possible, that the addition of optical flow to SORT becomes obsolete, as the development of further improved object detection frameworks continues. And since the detection quality is of great importance for SORT's tracking accuracy it may be more efficient to focus on a better CNN.

Appendices

The complete source code for SORT_OF and SORT's C++ reimplementation as well as the SHAN-1 data set is made publicly available online:

- SORT_OF: `https://github.com/tylernewnoise/SORT_OF`
- SORT in C++: `https://github.com/tylernewnoise/SORT_in_Cpp`
- SHAN-1: `https://bit.ly/ShanTrackBench`

A. Appendix: Kalman Filter Matrices

$$F_{SORT\_OF} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{SORT\_OF} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Q_{SORT\_OF} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.0001 \end{bmatrix}$$

$$P_{SORT\_OF} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10000 \end{bmatrix}$$

Depending on the update case (see 5.2) H_{SORT_OF} is one of:

- Update with detection and velocity:

$$H_{SORT\_OF} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Update with velocity only:

$$H_{SORT\_OF} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Update with detection only:

$$H_{SORT\_OF} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

B. Appendix: Benchmark Scenarios Descriptions

Name	FPS	Resolution	Length	Tracks	Boxes	Description
ADL-Rundle-6	30	1920 × 1080	525	24	5009	A pedestrian street scene filmed from a low angle.
TUD-Stadtmitte	25	640 × 480	179	10	1156	A static camera at about 2 meters height shows walking people on the street.
KITTI-17	10	1224 × 370	145	9	683	Walking pedestrians on a sunny day, static camera.
Venice-2	30	1920 × 1080	600	26	7141	People walking around a large square.
PETS09-S2L1	7	768 × 576	795	19	4476	A widely used sequence showing up to 8 walking pedestrians, partly in unusual patterns.
TUD-Campus	25	640 × 480	71	8	359	A short sequence with side-view pedestrians.

Table 5: Properties MOT15 data sets (static camera)

Name	FPS	Resolution	Length	Tracks	Boxes	Description
SHAN-1	25	2560 × 1440	449	68	30041	A crossroad scene with all kinds of traffic participants in Shanghai observed from a high angle, static camera.

Table 6: Properties SHAN-1 set

C. Appendix: MOT15 Detailed Results

$T_{Lost} = 1$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
ADL-Rundle-6	38.2%	74.6%	75	**919**	2101	**7**	2
TUD-Stadtmitte	72.4%	75.2%	10	22	287	6	0
KITTI-17	**63.0%**	**72.0%**	8	**37**	**208**	1	0
Venice-2	18.1%	73.6%	**59**	**1698**	4089	8	8
PETS09-S2L1	62.5%	67.7%	100	515	1062	8	0
TUD-Campus	**61.8%**	73.5%	5	**18**	114	4	0

$T_{Lost} = 25$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
ADL-Rundle-6	14.8%	**73.0%**	**73**	**2527**	1669	9	2
TUD-Stadtmitte	59.7%	**74.4%**	11	237	218	7	0
KITTI-17	15.8%	**71.9%**	**7**	370	198	3	0
Venice-2	-1.7%	**72.1%**	45	3622	3593	8	7
PETS09-S2L1	33.2%	67.3%	**77**	2197	**716**	**15**	0
TUD-Campus	27.6%	**72.9%**	**2**	194	64	6	0

Table 7: SORT on MOT15 data set (static camera) detailed results

$T_{Lost} = 1$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
ADL-Rundle-6	**38.5%**	**74.7%**	**67**	921	**2095**	6	2
TUD-Stadtmitte	72.4%	**75.3%**	10	22	287	6	0
KITTI-17	61.8%	71.8%	8	43	210	1	0
Venice-2	**18.2%**	73.6%	61	1723	**4057**	8	**7**
PETS09-S2L1	**63.7%**	**67.8%**	**96**	**502**	**1025**	**9**	0
TUD-Campus	61.3%	**73.6%**	5	19	115	4	0

$T_{Lost} = 25$

Sequence	MOTA	MOTP	IDs	FP	FN	MT	ML
ADL-Rundle-6	**15.8%**	72.1%	77	2554	**1589**	9	2
TUD-Stadtmitte	**61.6%**	74.2%	**9**	**218**	**217**	7	0
KITTI-17	**40.3%**	71.1%	10	**232**	**166**	**4**	0
Venice-2	**-0.1%**	71.9%	**41**	**3617**	**3488**	8	7
PETS09-S2L1	**34.3%**	**67.4%**	80	**2140**	719	14	0
TUD-Campus	**34.5%**	72.5%	4	**172**	**59**	6	0

Table 8: SORT_OF on MOT15 data set (static camera) detailed results

D. Appendix: SHAN-1 Detailed Results

	SORT							SORT_OF						
T_{Lost}	MOTA	MOTP	IDs	FP	FN	MT	ML	MOTA	MOTP	IDs	FP	FN	MT	ML
1	78.5%	83.5%	162	769	5519	56	10	**78.6%**	83.5%	**160**	**767**	**5512**	56	10
2	80.1%	83.4%	115	890	4972	59	9	**80.2%**	83.4%	115	**882**	**4966**	59	9
3	80.7%	83.3%	97	994	4703	60	7	**80.9%**	83.3%	**96**	**978**	**4677**	60	7
4	81.0%	83.3%	83	1085	4530	60	7	**81.2%**	83.3%	**81**	**1069**	**4496**	60	7
5	81.2%	83.2%	77	1161	4424	61	7	**81.3%**	**83.3%**	**74**	**1149**	**4383**	61	7
6	81.3%	83.2%	74	1228	4317	61	7	**81.5%**	83.2%	**70**	**1203**	**4270**	61	7
7	81.3%	83.2%	66	1327	4217	**62**	7	**81.7%**	83.2%	**61**	**1285**	**4156**	61	7
8	81.3%	83.1%	65	1401	4137	**63**	7	**81.8%**	**83.2%**	**59**	**1347**	**4075**	62	7
9	81.4%	83.1%	63	1465	4068	**63**	7	**81.9%**	83.1%	**56**	**1398**	**3982**	62	7
10	81.4%	83.0%	59	1544	3998	**63**	7	**81.9%**	**83.1%**	**53**	**1469**	**3913**	62	7
11	81.3%	83.0%	56	1619	3933	**63**	7	**82.0%**	83.0%	**50**	**1521**	**3843**	62	7
12	81.3%	83.0%	54	1680	3891	**63**	7	**82.0%**	83.0%	**48**	**1569**	**3795**	62	7
13	81.2%	83.0%	53	1748	3861	63	7	**81.9%**	83.0%	**47**	**1632**	**3764**	63	7
14	80.9%	83.0%	53	1833	3844	63	6	**81.7%**	83.0%	**45**	**1701**	**3739**	63	6
15	80.8%	83.0%	52	1897	3832	63	6	**81.7%**	83.0%	**44**	**1751**	**3717**	63	6
16	80.6%	83.0%	52	1958	3823	63	6	**81.5%**	83.0%	**44**	**1803**	**3703**	63	6
17	80.4%	83.0%	52	2028	3806	63	6	**81.4%**	83.0%	**43**	**1854**	**3677**	63	6
18	80.2%	**83.0%**	52	2089	3797	63	6	**81.3%**	82.9%	**43**	**1905**	**3663**	63	6
19	80.1%	82.9%	52	2150	3788	63	6	**81.2%**	82.9%	**44**	**1955**	**3635**	63	6
20	79.9%	82.9%	52	2211	3779	63	6	**81.1%**	82.9%	**44**	**2006**	**3622**	63	6
21	79.7%	82.9%	52	2288	3770	63	6	**80.9%**	82.9%	**44**	**2071**	**3609**	63	6
22	79.5%	82.9%	52	2349	3761	63	6	**80.8%**	82.9%	**44**	**2118**	**3594**	63	6
23	79.4%	82.9%	52	2409	3714	63	6	**81.0%**	82.9%	**43**	**2162**	**3513**	**64**	6
24	79.3%	**82.9%**	52	2469	3704	63	6	**80.9%**	82.8%	**42**	**2209**	**3495**	**64**	6
25	79.1%	**82.9%**	52	2529	3695	63	6	**80.8%**	82.8%	**42**	**2255**	**3481**	**64**	6

Table 9: SORT vs. SORT_OF on SHAN-1 data set

Bibliography

[1] T. Janz, "Post-processing of multi-target trajectories for traffic security analysis," M.S. thesis, Georg-August-Universit ä t G ö ttingen, Dec. 2017. [Online]. Available: `https://elib.dlr.de/139881/`.

[2] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3464–3468.

[3] T. Senst, V. Eiselein, and T. Sikora, "Robust local optical flow for feature tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1377–1387, Sep. 2012.

[4] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," Nov. 2011, pp. 2564–2571. DOI: `10.1109/ICCV.2011.6126544`.

[5] L. Leal-Taix´ e, A. Milan, I. Reid, S. Roth, and K. Schindler, "Motchallenge 2015: Towards a benchmark for multi-target tracking," *CoRR*, vol. abs/1504.01942, Apr. 2015. [Online]. Available: `http://arxiv.org/abs/1504.01942`.

[6] G. Welch and G. Bishop, "An introduction to the kalman filter," University of North Carolina, Chapel Hill, USA, Tech. Rep. TR 95-041 Departement of Computer Science, 1995.

[7] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[8] P. C. Mahalanobis, "On test and measures of group divergence, part i: Theoretical formulae," in *Journal and Proceedings of Asiatic Society of Bengal (New series)*, vol. 26, 1930, pp. 541–588.

[9] R. Gnanadesikan and J. R. Kettenring, "Robust estimates, residuals, and outlier detection with multiresponse data," *Biometrics*, vol. 28, no. 1, pp. 81–124, 1972, ISSN: 0006341X, 15410420. [Online]. Available: `http://www.jstor.org/stable/2528963`.

[10] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[11] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, pp. 325–340, 1986.

[12] J. Berclaz, F. Fleuret, E. Türetken, and P. Fua, "Multiple object tracking using k-shortest paths optimization," vol. 33, 2011, pp. 1806–1819.

[13] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3645–3649.

[14] E. Bochinski, V. Eiselein, and T. Sikora, "High-speed tracking-by-detection without using image information," in *International Workshop on Traffic and Street Surveillance for Safety and Security at IEEE AVSS 2017*, Aug. 2017.

[15] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 4705–4713.

[16] C. Dicle, O. I. Camps, and M. Sznaier, "The way they move: Tracking multiple targets with similar appearance," in *2013 IEEE International Conference on Computer Vision*, 2013, pp. 2304–2311.

[17] E. Bochinski, T. Senst, and T. Sikora, "Extending iou based multi-object tracking by visual information," in *IEEE International Conference on Advanced Video and Signals-based Surveillance*, Nov. 2018, pp. 441–446.

[18] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015. DOI: 10.1109/TPAMI.2014.2345390.

[19] B. K. Horn and B. G. Schunck, "Determining Optical Flow," in *Artificial Intelligence*, vol. 17, 1981, pp. 185–203.

[20] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of Imaging Understanding Workshop*, 1981, pp. 121–130.

[21] J.-Y. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker, description of the algorithm," Intel Corporation, Microprocessor Research Labs, 1999.

[22] G. Le Besnerais and F. Champagnat, "Dense optical flow by iterative local window registration," vol. 1, Oct. 2005, pp. I–137, ISBN: 0-7803-9134-9. DOI: 10.1109/ICIP.2005.1529706.

[23] Z. Chen, H. Jin, Z. Lin, S. Cohen, and Y. Wu, "Large displacement optical flow from nearest neighbor fields," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2443–2450. DOI: 10.1109/CVPR.2013.316.

[24] T. Senst, I. Keller, and T. Sikora, "Robust local optical flow estimation using bilinear equations for year = 2013 , pages = 2499-2503 , sparse motion estimation," in *2013 IEEE International Conference on Image Processing*. DOI: 10.1109/ICIP.2013.6738515.

[25] T. Senst, T. Borgmann, I. Keller, and T. Sikora, "Cross based robust local optical flow," Oct. 2014, pp. 1967–1971. DOI: 10.1109/ICIP.2014.7025394.

[26] T. Senst, J. Geistert, and T. Sikora, "Robust local optical flow: Long-range motions and varying illuminations," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 4478–4482. DOI: 10.1109/ICIP.2016.7533207.

[27] G. Farneb ä ck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370, ISBN: 978-3-540-45103-7.

[28] C. Zach, T. Pock, and H. Bischof, "A duality based approach for realtime tv-l1 optical flow," vol. 4713, Sep. 2007, pp. 214–223, ISBN: 978-3-540-74933-2. DOI: 10.1007/978-3-540-74936-3_22.

[29] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool, "Fast optical flow using dense inverse search," in *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, pp. 71–488.

[30] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, "Deepflow: Large displacement optical flow with deep matching," in *IEEE Intenational Conference on Computer Vision (ICCV)*, Sydney, Australia, Dec. 2013. [Online]. Available: http://hal.inria.fr/hal-00873592.

[31] J. Hur and S. Roth, "Iterative residual refinement for joint optical flow and occlusion estimation," *CoRR*, vol. abs/1904.05290, 2019.

[32] A. S. Wannenwetsch and S. Roth, "Probabilistic pixel-adaptive refinement networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[33] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *Computer Vision – ECCV 2020*, Springer International Publishing, 2020, pp. 402–419.

[34] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147–151.

[35] J. Shi and C. Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

[36] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[37] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[38] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," Engineering Department, Machine Intelligence Laboratory, University of Cambrdige, Tech. Rep., 2006.

[39] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision (ECCV 2010)*, Springer Berlin Heidelberg, Sep. 2010, pp. 778–792.

[40] C. Tomasi and T. Kanade, "Detection and tracking of point features," in *Carnegie Mellon University Technical Report CMU-CS-91-132*, Apr. 1991.

[41] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The clear mot metrics," *EURASIP Journal on Image and Video Processing*, 2008.

[42] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[43] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, 2008, pp. 1–8.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 17. Februar 2021 .....