Lappeenranta University of Technology
School of Industrial Engineering and Management
Degree Program in Computer Science
Major in Intelligent Computing

Master's Thesis

Toni Kuronen

POST-PROCESSING AND ANALYSIS OF TRACKED HAND TRAJECTORIES

Examiners: Professor Heikki Kälviäinen
Professor Lasse Lensu

Supervisor: D.Sc. (Eng.) Tuomas Eerola

ABSTRACT

Lappeenranta University of Technology
School of Industrial Engineering and Management
Degree Program in Computer Science
Major in Intelligent Computing

Toni Kuronen

Post-processing and analysis of tracked hand trajectories

Master's Thesis

2014

89 pages, 34 figures, 12 tables, and 1 appendix.

Examiners:	Professor Heikki Kälviäinen
	Professor Lasse Lensu

Keywords: computer vision, hand tracking, high-speed video, hand trajectories, filtering

In this thesis, the suitability of different trackers for finger tracking in high-speed videos was studied. Tracked finger trajectories from the videos were post-processed and analysed using various filtering and smoothing methods. Position derivatives of the trajectories, speed and acceleration were extracted for the purposes of hand motion analysis. Overall, two methods, Kernelized Correlation Filters and Spatio-Temporal Context Learning tracking, performed better than the others in the tests. Both achieved high accuracy for the selected high-speed videos and also allowed real-time processing, being able to process over 500 frames per second. In addition, the results showed that different filtering methods can be applied to produce more appropriate velocity and acceleration curves calculated from the tracking data. Local Regression filtering and Unscented Kalman Smoother gave the best results in the tests. Furthermore, the results show that tracking and filtering methods are suitable for high-speed hand-tracking and trajectory-data post-processing.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tuotantotalouden tiedekunta
Tietotekniikan koulutusohjelma
Älykkään laskennan pääaine

Toni Kuronen

Käden liikeratojen jälkikäsittely ja analysointi

Diplomityö

2014

89 sivua, 34 kuvaa, 12 taulukkoa ja 1 liite.

Tarkastajat: Professori Heikki Kälviäinen
Professori Lasse Lensu

Hakusanat: tietokonenäkö, käden seuranta, suurnopeusvideo, käden liikeradat, suodatus
Keywords: computer vision, hand tracking, high-speed video, hand trajectories, filtering

Tässä työssä tutkittiin kohteenseuranta-algoritmien soveltuvuutta sormen seuraamiseen suurnopeusvideoiden perusteella. Kädenliikkeiden seurantatietoja analysoitiin suurnopeusvideoista ja prosessoitiin käyttäen erilaisia suodatus- ja pehmennysmenetelmiä. Käden liikeanalyysiä varten laskettiin paikkatiedon derivaattojen nopeuden ja kiihtyvyyden arvot. Kernelized Correlation Filters ja Spatio-Temporal Context Learning -seuranta-algoritmit suoriutuivat testeissä paremmin kuin muut testatut menetelmät. Molempien menetelmien tarkkuus oli hyvä ja ne pystyivät käsittelemään suurnopeusvideoita reaaliajassa, mikä tarkoitti yli 500 kuvan käsittelyä sekunnissa. Tuloksista voitiin nähdä myös, kuinka eri suodatusmenetelmiä voidaan käyttää selkeämpien nopeus- ja kiihtyvyyskuvaajien saavuttamiseksi seuranta-algoritmien tuottamista paikkatiedoista. Local Regression -suodatus ja Unscented Kalman Smoother saavuttivat parhaat tulokset testeissä. Lisäksi tuloksista voitiin havaita seuranta-algoritmien soveltuvuus käden seurantaan suurnopeusvideoissa ja suodatusmenetelmien soveltuvuus liikeratojen jälkikäsittelyyn.

PREFACE

I wish to thank my supervisor, D.Sc. (Eng.) Tuomas Eerola, and the examiners, Professors Heikki Kälviänen and Lasse Lensu, for their valuable support and comments during the thesis process. Also, I would like to thank the Academy of Finland for the financial support to the Copex project.

Finally, I would like to thank my wife, Marika, for the support and understanding during this work.

Lappeenranta, November 18th, 2014

Toni Kuronen

CONTENTS

ABBREVIATIONS AND SYMBOLS

CT	Real-time Compressive Tracking
DOF	Degrees Of Freedom
EKF	Extended Kalman Filter
FFT	Fast Fourier Transform
FCT	Fast Compressive Tracking
fps	Frames Per Second
HCI	Human Computer Interaction
HOG	Histogram of Oriented Gradients
HT	Hough-based Tracking of Non-rigid Objects
IFFT	Inverse Fast Fourier Transform
IVT	Incremental Learning for Robust Visual Tracking
LRS	Log-Euclidean Riemannian Subspace and Block-Division Appearance Model Tracking
KCF	High-Speed Tracking with Kernelized Correlation Filters
KF	Kalman Filter
MIL	Robust Object Tracking with Online Multiple Instance Learning
MVPR	Machine Vision and Pattern Recognition
UI	User Interface
POEM	Psychology of Evolving Media and Technology Research Group
RMSE	Root Mean Squared Error
RSCM	Robust Object Tracking via Sparse Collaborative Appearance Model
SRPCA	Online Object Tracking With Sparse Prototypes
SDC	Sparsity-based Discriminative Classifier
SGM	Sparsity-based Generative Model
STC	Fast Tracking via Spatio-Temporal Context Learning
struck	Structured Output Tracking with Kernels
TLD	Tracking-Learning-Detection
TV	Total Variation
TVD	Total Variation Denoising
UKF	Unscented Kalman Filter

1 INTRODUCTION

Hand tracking and gesture recognition from recorded videos and from real-time streams have been popular research topics since 1990s. Tracking hands in real-time opens new possibilities for different user interfaces in machines, computers, and operation tools, for example. Using gestures and hand movements is natural for people. They are used, for example, when telling stories or perhaps when someone is directed to the nearest automated teller machine. During the last ten years also consumers have observed possible uses of gestures and hand tracking as machines like Microsoft Kinect [1] and Leap Motion [2] have arrived to the market. These machines use different techniques and sensors, aiming to provide more natural human-computer interaction (HCI) than the game pads, keyboards, or other controllers.

This research is aimed towards high-speed video tracking and post-processing the tracking data. The interest to use high-speed videos derives from the fact that the precision of measurements with respect to time, that is, temporal resolution, is better than the measurement precision of videos with standard frame rate. Thus, higher frame rate in camera means that the magnitude of the motion between two frames is smaller than in videos captured with normal speed of around 30 to 60 frames per second (fps), and it is possible to make more accurate measurements. Moreover, increased sharpness of images with fast-moving objects and reduced motion blur is achieved thanks to faster shutter speed while using high-speed cameras. Motion blur can be avoided by using faster shutter speeds for normal speed videos: for high-speed videos this comes naturally because their shutter speed needs to be faster than in the standard frame rate videos. [3]

1.1 Background

Real-time object tracking has been one of the most popular topics in computer vision [4, 5, 6, 7]. Especially hand tracking has received plenty of interest. At the moment, attention is drawn towards a multitude of desktop and tablet computer software, gesture games, and human computer interaction in a general level. Gesture games here are used to describe the games like Microsoft Kinect games [1] where the user is the game controller, without any other input devices. Hand tracking, on the other hand, is a very demanding subject. The reasons behind the complexity are many, but the main reason is that almost every section of the hand can have different appearances and the hand has many degrees of freedom (DOF).

Maggio et al. [3] sum video tracking up as “The process of estimating over time the location of one or more objects using a camera is referred to as video tracking.“ The definition explains the idea of video tracking, namely, following one or more objects in an image sequence. General tracking pipeline includes the following steps: feature extraction, target representation, localisation, and track management. And, as a result of tracking, trajectories can be formed. In the feature extraction phase, the selected features are extracted from the image, and they are used in the next step to represent the target. The following step is to localise the target in the image, and, in the next phases, the object trajectory is maintained and the high-level features from the trajectories can be extracted. [3]

The application field for tracking is broad, as it can be used for media production and augmented reality, medical applications and biological research, surveillance and business intelligence, robotics and unmanned vehicles, tele-collaboration and interactive gaming, and art installations and performances. In the media production field, one can add, modify or remove certain elements of videos or image sequences. For augmented reality cases, one can track movement of a moving animal or a person and make virtual scene changes based on those movements. Medical applications can include analysis of gait of a patient or even a virtual surgery. For surveillance, tracking can be used to detect unusual patterns in the movement of people around some area. In the robotics field and with unmanned vehicles, the results of tracking are essential because it is necessary to know what kinds of objects are around in the vicinity and whether they move and how, or are stationary. [3]

This thesis is a part of the Computational Psychology of Experience in Human-Computer Interaction (COPEX) project [8], which researches human-computer interaction through touch and different gestures. The COPEX project is a collaborative work between the Machine Vision and Pattern Recognition Laboratory (MVPR) [9] of Lappeenranta University of Technology and the Psychology of Evolving Media and Technology Research Group (POEM) [10] of the University of Helsinki. The project aims to analyse and track how different people use their hands while doing various tasks at a computer. Data gathered from the experiments – trajectories, reaction times and accelerations – are then used to analyse and process how different people use novel touch interfaces. Post-processing of the tracking data is one of the main focuses of this research and various methods for extracting features from the tracking data and filtering of data are explored during this research.

1.2 Objectives and Restrictions

The objective of this thesis is to study post-processing and analysis of tracked hand trajectories, such as tracking hand movement, and to filter the hand-tracking data by using different filters and smoothers. It is not the objective of this thesis to develop new methods for these problems but to research and analyse those that are currently available and see if they provide answers for the objectives, which can be seen in Figure 1. On the top row of that figure, tracking is applied to a high-speed video sequence, of which frames 300, 600, and 900 are shown (from left to right). The tracked position in this case is a segmented area of the fingertip, which is illustrated with an area encircled with red colour. On the second row, trajectory features' speed and acceleration are plotted over the tracked sequence. Movement trajectory is highlighted with red and blue colours following the finger tip, velocity is shown in red, green and white on top of another red, green and white acceleration curve. Finally, on the bottom row, filtering is applied to the features of the tracked high-speed sequence. There, the filtered trajectory of the hand movement is plotted with yellow, the filtered velocity curve with green, and finally, the filtered acceleration curve with cyan. Also, the time scale is plotted into the sequences with a span from 0.1 - 1.5 seconds and with 0.1 second intervals.

The practical purpose of the work is to collect applicable methods for post-processing and filtering hand-tracking data so that a method can be developed to estimate the hand trajectory in real-world coordinates. Also, different features, such as velocities, accelerations, curvatures and other applicable features, are computed from hand-tracking data.

The limiting factors in selecting the trackers and the filtering methods are operation environment and the availability of the trackers' source code needed for modifying purposes in this research. In the selection process, more recent tracking methods and those which have performed well in various recent tracker evaluation papers are preferred [11, 12, 13, 4].

1.3 Structure of the Thesis

This thesis is organised as follows. Chapter 2 takes a look at existing object tracking tools and results of some trackers based on various benchmark papers. Also, a short introduction to tracking algorithms and meaningful parameters for the trackers are given. In Chapter 3, the selection of post-processing methods for the hand-tracking data experiments is justified. A look is taken into commonly-used smoothing and filtering methods for digital signals. The signal-filtering toolset used in this research is introduced. Basic

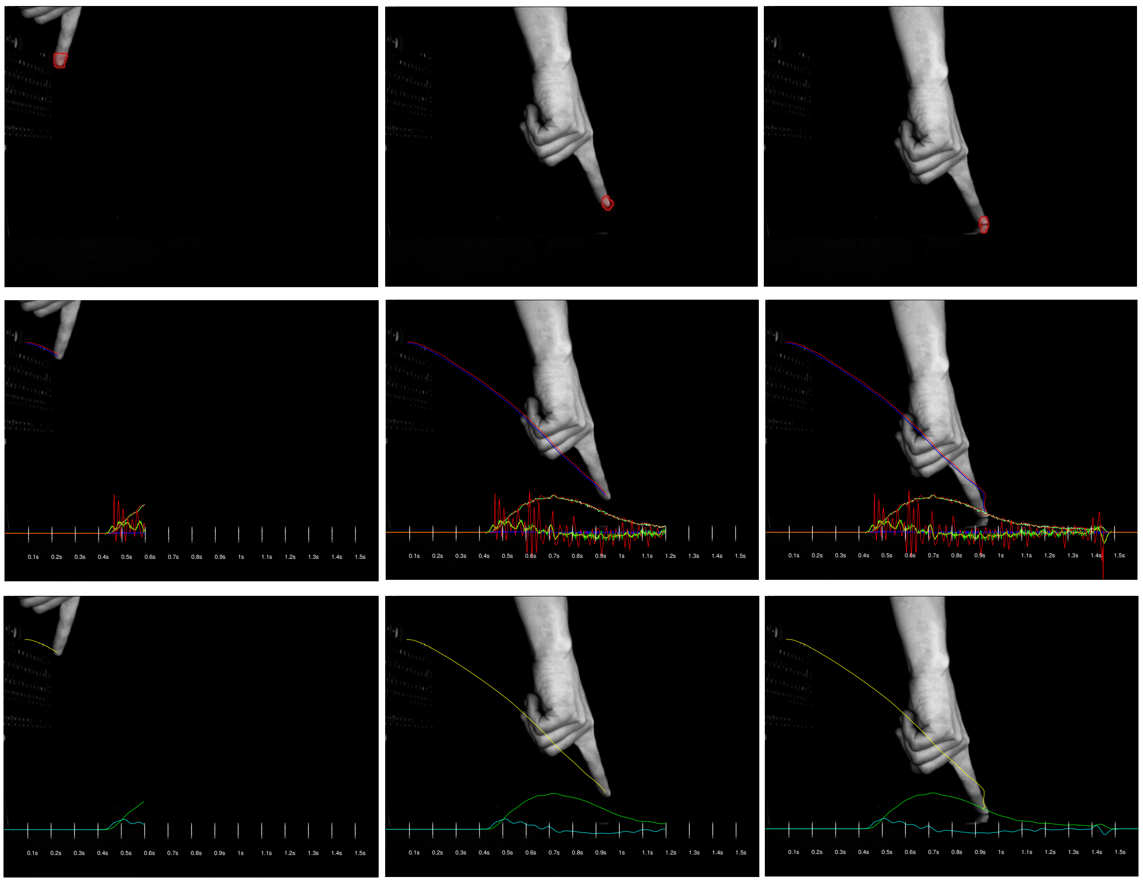


Figure 1. Post-processing of features in a hand-tracking sequence is shown here. From left to right, the frames 300, 600 and 900 of a high-speed video sequence can be seen on all rows. The top row illustrates tracking applied to the sequence. The second row contains the trajectory, velocity and acceleration features which are drawn over the sequence, and, in the third row, filtering is applied and the filtered features are drawn over the sequence.

filtering equations are shown and short introduction to the mathematics behind the filtering is given in this chapter. The reasoning behind the selection of coordinate transform methods is explained. Feature extraction methods are selected and evaluated. A short introduction to the needed features for the trajectory analysis is given, and ideas of any additional features are brought up. The features required include acceleration, reaction times, movement speed, time of touch and path of the movement.

Chapter 4 contains the results and reasons and justifications for the experiments. The first part of the chapter defines the test environment and datasets used. The second part of the chapter defines the evaluation metrics and presents the results for the tracking and filtering experiments. Filtering methods are evaluated against non-processed data and against manually annotated ground-truth trajectories. Camera calibration is tested by calculating how close the measures from tracking are to the real-world coordinates and measurements.

Chapter 5 provides a discussion about the findings and the results of the work conducted during this research and, finally, Chapter 6 provides the concluding thoughts about this research.

2 OBJECT TRACKING

A broad selection of trackers developed for different purposes is available. They are used to track rigid objects like cars and non-rigid objects like people and animals. There are also trackers that can learn different poses for tracked objects and continue tracking, even after lost tracking, by detecting the object again [14]. Only tracking methods which are available with open source implementations were included in this research. Binary trackers were left out because it would have been necessary to modify the implementations to get them equally tested with different initialisation parameters. The tracker selection consisted of the state-of-the-art methods with real-time performance. Robust methods were included for comparison.

2.1 Background

There are many challenges that trackers face with video materials. One group of challenges are variations of appearance which include object pose changes, occlusions, scene changes and sensor noise. The object pose changes can arise from object rotation, translation or deformation. Occlusions happen when other objects in the scene block the view of the tracked object, completely or partially. Scene changes can be due to, for example, weather or illumination: for example, the scene stays the same but the shots or video of the moving object is made in different times of the day or it starts raining in the middle of shooting a video. Background clutter, which means that there are similar objects on the background which can draw attention of the tracker to themselves, creates challenges for most trackers [3]. The object appearance challenges are summed up in Figure 2, which shows the effects of illumination, occlusion, deformation, noise corruption, out-of-plane rotation, and motion blurring for object appearance [5]. Using a high-speed camera brings also other issues for imaging: for example, the demand for light in high-speed imaging increases as exposure time decreases.

Tracker algorithms usually consist of two parts: model and search. The model or representation describes how the features of the object and/or the surroundings like the background are represented. Many models are used, among them local, holistic, template, intensity histogram, binary pattern, principal component analysis (PCA), sparse PCA, sparse representation, discriminative model, and generative model [13]. Similar to different object representation methods, there are also various searching methods, including particle filters, Markov Chain Monte Carlo, local optimum search, and dense sampling search [3]. There are trackers which update the model during tracking based on the object appearance changes, but also trackers which do not update the model during tracking. Using some kind of updating function for object changes is more common because that can enable long-term tracking even when the appearance of the object changes. [13, 15, 7, 16, 4, 3]

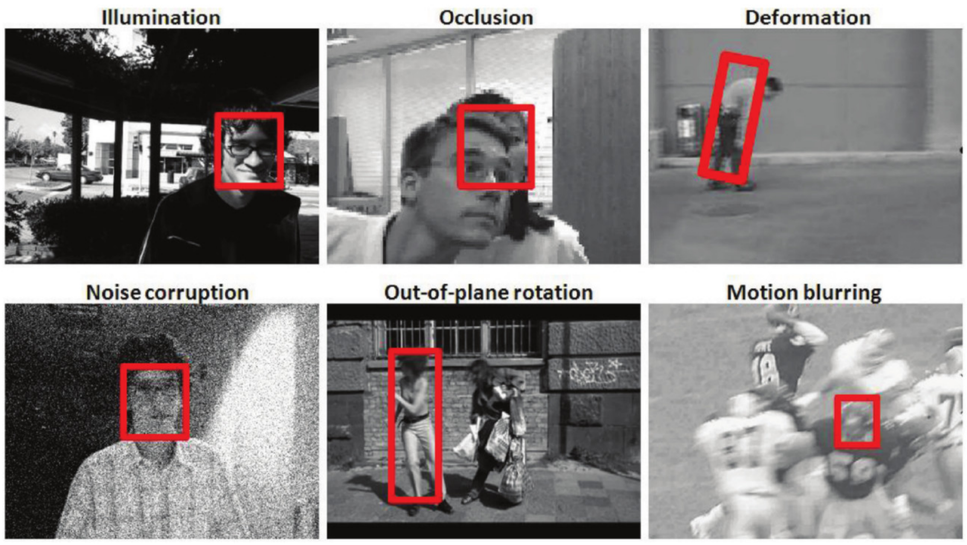


Figure 2. Object tracking challenges. [5]

2.2 Criteria for Selecting Trackers

As Čehovin et al. discuss in article [7], there is still lack of consensus about which performance measures should be preferred in tracker evaluation. TThus, cross-paper tracker comparison can be hard due to unequal performance measures and non-standard video test sets. Tracker selection for this research included trackers for which source code was available, which had appeared in the latest tracking benchmark articles or had provided good results in their original papers. Also, the novelty value of each particular tracker influenced the decision of choosing among them, tracking algorithms having evolved quite rapidly over the last few years. That is why also new trackers like High-Speed Tracking with Kernelized Correlation Filters [11] and Fast Tracking via Spatio-Temporal Context Learning [17] were included in this research, even though there were no collective online benchmark results available for them at the time when the tracker selections were made. The aim of this research was to find a tracker for high-speed videos, and that ruled out some trackers. The selected trackers are shown in Table 1.

Table 1. Trackers selected for the experiments.

Method	Abbreviation	Implementation
Real-time Compressive Tracking [18]	CT	MATLAB [19]
Fast Compressive Tracking [20]	FCT	MATLAB [21]
High-Speed Tracking with Kernelized Correlation Filters [11]	KCF	MATLAB [22]
Hough-based Tracking of Non-Rigid Objects [23]	HT	C++ [24]
Incremental Learning for Robust Visual Tracking [25]	IVT	MATLAB [26]
Robust Object Tracking with Online Multiple Instance Learning [27]	MIL	MATLAB [28]
Tracking Learning Detection [14]	TLD	MATLAB [29]
Robust Object Tracking via Sparsity-based Collaborative Model [30]	RSCM	MATLAB [31]
Fast Tracking via Spatio-Temporal Context Learning [17]	STC	MATLAB [32]
Structured Output Tracking with Kernels [33]	struck	C++ [34]
Single and Multiple Object Tracking Using Log-Euclidean Riemannian Subspace and Block-Division Appearance Model [35]	LRS	MATLAB [36]
Online Object Tracking with Sparse Prototypes [37]	SRPCA	MATLAB [38]

The lack of generalised tracking evaluation sets has given rise to a couple of online object tracking benchmarks [13, 15, 7, 16, 4] for the benefit of tracking algorithm researchers searching for a good online platform with equal measures and easy-to-use tracking evaluation kits. In [13], there are 29 tracking algorithms and 50 fully annotated videos. An evaluation kit for the tracker benchmark is available online, and the team is working on a new version of an online tracker benchmark kit [39].

The Visual Object Tracking (VOT) challenges of 2013 and 2014 consist of a challenging set of video sequences and novel performance measures for tracking, introducing robustness and accuracy plot, as explained in [7]. The challenge benchmark of 2014 includes a rotating bounding box for more accurate object position measures. VOT 2014 contains also a broader range of videos, featuring 25 sequences selected from the initial set of 394 sequences in such a way that the various visual phenomena like occlusion were still strongly present in the selection. [4] The VOT 2013 sequences went through a similar selection process, but the final number of videos was 16 [16]. Performance measures of accuracy and robustness are used for tracking performance evaluation. Accuracy tells how well the trackers predict bounding-box overlaps with the ground-truth bounding-box. Robustness is a measure for losing the target, and in this case target is considered lost when the overlap with the bounding box goes to zero [4]. Single-object, single-camera, model-free, short-term, and causal trackers are included in the VOT challenge. Model-free means that a single training example is provided by the bounding box in the first frame of the sequence. Short-term in this context means that the tracker is not allowed to perform re-detection, as the tracking is considered to fail if it drifts off the target. Once the tracker has lost its target, a complete reset is issued by the visual-object-tracking toolbox. Thus the tracker is not allowed to use any information obtained before the reset. Causality is a requirement in the experiments, and that is why trackers cannot use future frames for pose estimation [4, 16]. VOT challenges are available online, and the team is also working on a new online repository of sequences and results [40].

The work by Hiltunen et al. [41, 42] takes a good look into the tracking algorithms applied to high-speed videos. Building on top of that research, the algorithm introductions in this thesis are kept short for most of the algorithms. Also the scope of this research leans more towards the post-processing of the tracking data. New methods included in this research are High-Speed Tracking with Kernelized Correlation Filters (KCF) [11], Fast Tracking via Spatio-Temporal Context Learning (STC) [17], Robust Object Tracking via Sparsity-based Collaborative Model (RSCM) [30], and Structured Output Tracking with Kernels (struck) [33].

2.3 Tracking Methods

Tracking methods with fixed models of a target usually fail at some point because of the unavoidable changes of appearance. This is why online learning methods that update the representation of a target incrementally over time are used to handle the variations in appearance. The learning methods used by online tracking can be divided into generative and discriminative. In generative online learning methods, the appearance of the model for the target is updated adaptively in response to appearance variations. Generative online learning methods can be subcategorised into methods based on a template, subspace analysis, and sparse representation [43]. Discriminative methods utilise information about the target and background simultaneously and tracking is treated as a classification problem [43].

2.3.1 Real-time Compressive Tracking

Real-time Compressive Tracking (CT) uses a tracking-by-detection methodology, where similar appearance of object is searched from the next frame with the help of the previous frame's appearance model. The classifier is updated by using positive samples near the current target location and negative samples far away from the target location. The location for the next frame is predicted by getting samples from around the last known location and choosing the sample that gets the best classification score. The CT approach is shown in Algorithm 1, and the main ideas and key components of the tracker can be seen in Figures 3 and 4. Wu et al. [44] showed that the CT method could perform even better by implementing motion estimation algorithm into the tracking algorithm with minimal performance costs.

Algorithm 1 : CT tracking algorithm [18]

```
Input: The initialised tracking window for the first frame and the video sequence as
    frames.
Output: Tracked positions.
1: for all frames do
2:     Sample a set of image patches near the position where the target was in the previ-
       ous frame.
3:     Extract features from image patches.
4:     Using classifier, find and update the current target position with the most similar
       features.
5:     Sample positive image patches near the current target position and negative
       patches further away.
6:     Extract the features from image patches.
7:     Update the classifier with positive and negative features.
8: end for
```

2.3.2 Fast Compressive Tracking

Fast Compressive Tracker (FCT) is an improvement over Real-Time Compressive Tracker (CT) [18] and FCT and CT share the same algorithm for most parts. The main ideas and key components of the tracker, which are the same as for the CT algorithm, can be seen in Figures 3 and 4. Speed improvement for this algorithm is gained by using a sparse to a dense search method. First, object search is done by using a sparse detection with a sparse sliding window followed by a dense detection, which is done with better accuracy by using a dense sliding window within a smaller search area. The default parameters for the sparse technique are search radius of 25 pixels and window shifting of four pixels at a time. Dense search is done within 10 pixel radii with window shifting of one pixel at a time [20]. A general level explanation of the steps needed can be seen in Algorithm 2. This sparse to dense technique allows reduced computational complexity in the detection procedure, thus allowing the tracking to be done faster than with an earlier method in [18].

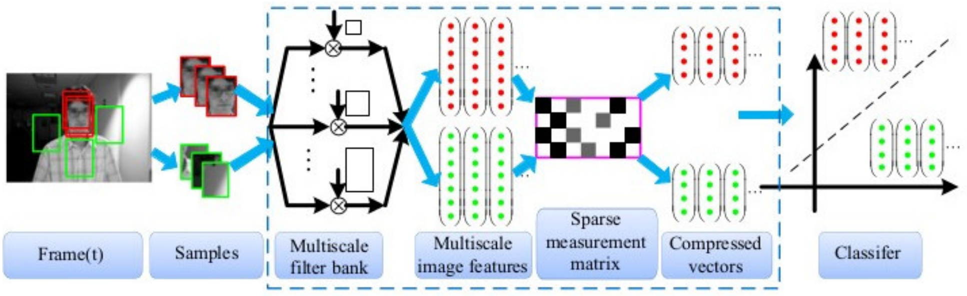


Figure 3. Classifier update at the t-th frame. [20]

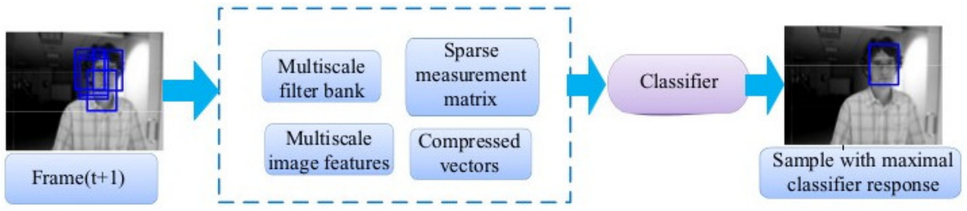

Frame(t+1)
Multiscale filter bank
Sparse measurement matrix
Multiscale image features
Compressed vectors
Classifier
Sample with maximal classifier response

Figure 4. Tracking at the (t+1)-th frame. [20]

2.3.3 High-Speed Tracking with Kernelized Correlation Filters

High-Speed Tracking with Kernelized Correlation Filters (KCF) is an improved version of the kernelized correlation filters introduced in [12]. Henriques et al. [11] showed that by oversampling sliding windows the resulting data matrix can be simplified, the size of the data reduced, and the computation made faster. This can be achieved by taking advantage of Fast Fourier Transform (FFT). [11]

KCF is explained in Algorithm 3. First, a sub-window for the detection at the position from the last frame is obtained and converted into usable features. Then a correlation response is calculated at all possible sliding window shifts. After that, the target location can be found from the maximal response where the correlation is highest. When a new location is found, the object model is updated again. Being approximately three times faster than the top two methods, the KCF algorithm took the 3rd place in visual object tracking challenge of 2014. The KCF algorithm also took the 1st place in overall accuracy rating. [4]

2.3.4 Hough-based Tracking of Non-Rigid Objects

Hough tracking is a tracking-by-detection approach. The method was developed for non-rigid object tracking by Godec et al. [23]. The name Hough-based Tracking of Non-Rigid

Algorithm 2 : FCT tracking algorithm [20]

```
Input: The initialised tracking window for the first frame and the video sequence as frames.
Output: Tracked positions.
 1: for all frames do
 2:     Sample a sparse set of image patches near the position where the target was in the previous frame.
 3:     Extract features from image patches.
 4:     Using classifier, find the current target location with the most similar features.
 5:     Sample a dense set of image patches near the position where the target was found in the previous step.
 6:     Extract features from image patches.
 7:     Using classifier, find and update the current target position with the most similar features.
 8:     Sample positive image patches near the current target position and negative patches further away.
 9:     Extract the features from image patches.
10:     Update the classifier with positive and negative features.
11: end for
```

Algorithm 3 : KCF tracking algorithm [11]

```
Input: The initialised tracking window for the first frame and the video sequence as frames.
Output: Tracked positions.
 1: for all frames do
 2:     Obtain the detection sub-window from the previous target position and convert it into usable features (in Fourier domain).
 3:     Calculate the response of the correlation at all sliding window shifts.
 4:     Find the new target position with the maximal response (the place of the highest correlation) and update the object model.
 5: end for
```

Objects (HT) comes from the fact that the method uses the generalised Hough-transform as a feature extraction technique. The main difference to most of the tracking methods is that HT uses segmentation of the target object region after it is given the bounding box. The fact that the target region is presented as a segmented area instead of the normally used bounding box, helps it to follow also non-rigid objects. The segmentation process is shown in Figure 5, where, on the first frame on top left, the object is surrounded with a bounding box. The object is then segmented and tracked in the subsequent scenes in the sequence [23].The tracking procedure of HT is explained in Algorithm 3.

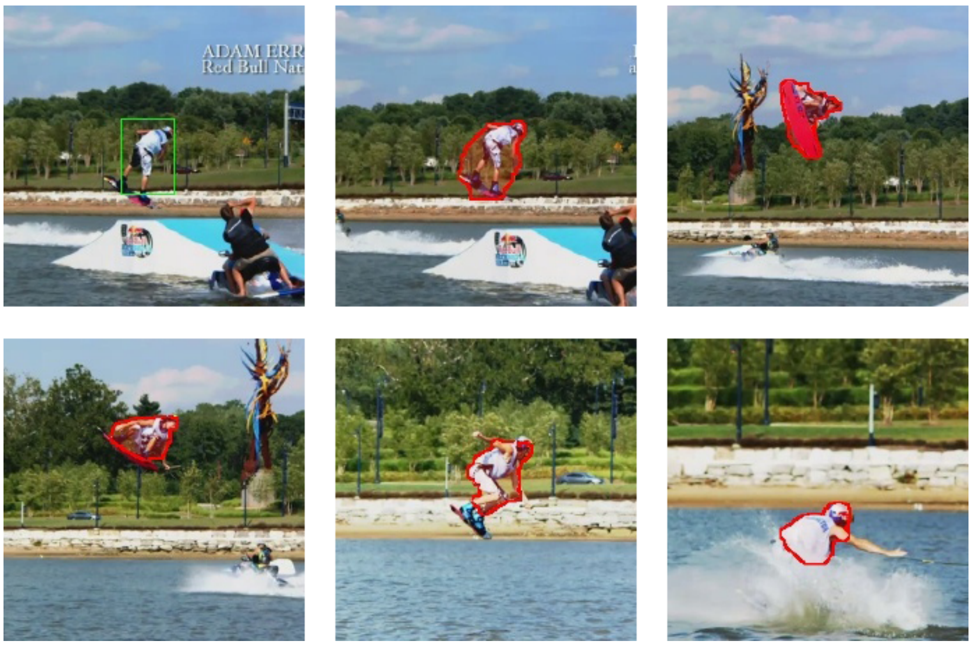


Figure 5. HT segmentation process and tracking. In the first (top left) image, the initial bounding box given is green, and the subsequent images (to the right and down), illustrate the segmentation process with an area encircled with red. [23]

2.3.5 Incremental Learning for Robust Visual Tracking

The focus of Incremental Learning for Robust Visual Tracking (IVT) by Ross et al. [25] is on how to deal with challenging appearance changes of the target. Adaption to changes is achieved by incrementally learning the target representation and adapting online to changes in the appearance of the target. For motion estimation, IVT uses a particle-filtering approach [25], which is a method of estimating where the object will be in the next frame. The method is based on a movement model and probability. The particles in the particle-filtering approach can be thought as a set of possible locations. The method tracks a bounding box that contains the object, and segmentation of the object region is

Algorithm 4 : HT tracking algorithm [23]

```
Input: The initialised tracking window for the first frame and the video sequence as
    frames.
Output: Tracked positions.
 1: Use segmentation process for the target in the tracking window.
 2: Train the classifier.
 3: for all frames do
 4:     Use the classifier for the current frame to find the target again.
 5:     Find the foreground parts of the target.
 6:     Apply the segmentation process to acquire binary segmentation.
 7:     Update the classifier with the new segmentation result.
 8:     Get the target position.
 9: end for
```

not done as in HT [23]. The IVT steps are explained in Algorithm 5.

Algorithm 5 : IVT tracking algorithm [25]

```
Input: The initialised tracking window for the first frame and the video sequence as
    frames.
Output: Tracked positions.
 1: Use a single particle to indicate this location.
 2: repeat
 3:     Draw particles from the particle filter.
 4:     for each particle do
 5:         Extract a window from the current frame, corresponding to the particle.
 6:         Compute the likelihood of the particle, corresponding to the real position.
 7:     end for
 8:     Update the target position with the most likely particle.
 9:     if $n$ frames since last update then
10:         Perform incremental update of the representation.
11:     end if
12: until no more frames
```

2.3.6 Robust Object Tracking with Online Multiple Instance Learning

Multiple Instance Learning (MIL) uses a tracking-by-detection approach for robust object tracking. MIL uses multiple instance learning instead of traditional supervised learning methods and shows improved robustness to inaccuracies of the tracker and to incorrectly labelled training samples [27]. Multiple instance learning differs from traditional supervised learning methods in that it does not contain strictly one positive set of image patches or one positive image patch: it has one positive bag which contains several image patches, of which at least one is the correct image patch containing the true appearance of the target. The main objectives when developing MIL were that the algorithm should be able to cope with partial occlusions and that not too many parameter adjustments would be needed. The target region is presented via multiple bounding-boxes.

Algorithm 6 presents the main steps of the MIL algorithm. For the current frame, to update the model, positive and negative samples are taken from around the predicted object location. In the following frame, by classifying image patches around the predicted location of the object, the tracker location is updated with a patch that best matches the current model. After that, the model is updated with samples from the current location. [27]

Algorithm 6 : MIL tracking algorithm [27]

```
Input: The initialised tracking window for the first frame and the video sequence as
    frames.
Output: Tracked positions.
1: for all frames do
2:     Crop out a set of image patches within a specified search radius of the previous
       target position.
3:     Compute features for each image patch.
4:     Classify image patches.
5:     With the best matching image patch update the tracker position.
6:     Crop out negative and positive sets of image patches.
7:     With the positive set and the negative set, update the appearance model.
8: end for
```

2.3.7 Tracking-Learning-Detection

By decomposing the object tracking task into tracking, learning, and detection subtasks, Kalal et al. [14] aim to realize long-term tracking of objects. The tracking algorithm tries to track an object during the frame sequence while the detector localises all the appearances observed in the frames, and if needed, the detector reinitialises the tracker. The learning subtask estimates the errors of the detector and updates it accordingly. As it can be seen in Figure 6, the learning subtask updates both the tracking and detection subtasks, and the tracking output is a combination of tracking and detection.Tracking-Learning-Detection (TLD) uses a P-N (Positive-Negative) learning method, which estimates the errors. P-N learning defines two types of experts: P-expert is used to estimate missed detections, meaning false negatives, and N-expert to estimate false alarms, meaning false positives. [15, 14]

The main points of the TLD algorithm are shown in Algorithm 7. The tracker part of the algorithm follows the movement and tries to locate the target. The detector tries to locate all similar targets from the image and selects the most probable. If both the tracker and the detector fail to locate the target, it is declared invisible. [15, 14] TLD uses multiple bounding-boxes to represent the target region, as can be seen from Algorithm 7. By using both tracking and detection bounding boxes, the algorithm can easily deal with occlusions if the target-object appearance does not change during occlusion.

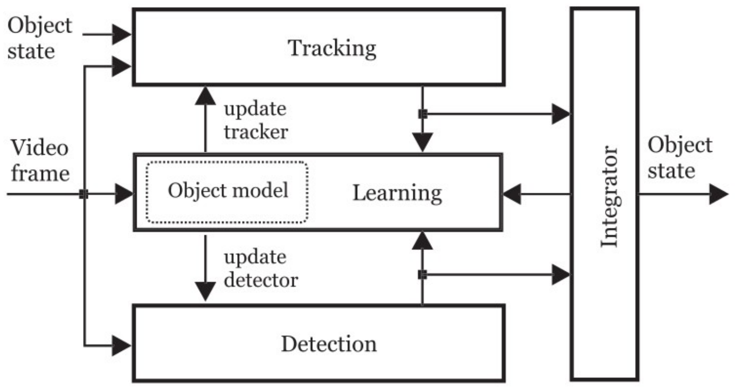


Figure 6. Block diagram visualising the components of the TLD framework. [14]

2.3.8 Robust Object Tracking via Sparsity-based Collaborative Model

Robust Object Tracking via Sparsity-based Collaborative Model (RSCM) by Zhong et al. [30] is based on a sparsity-based discriminative classifier (SDC) and a sparsity-based generative model (SGM). SDC introduces an effective method to compute the confidence value that assigns more weight to the foreground by extracting sparse and determinative features that distinguish foreground and background better. SGM is a histogram-based method that takes the spatial information of each patch into consideration with an occlusion handling scheme [30]. The updater considers the latest observations and also the original template, allowing the tracker to deal with appearance changes. Algorithm 8 simplifies the actual implementation of the RSCM tracker.

2.3.9 Fast Tracking via Spatio-Temporal Context Learning

Figures 7 and 8 and Algorithm 9 illustrate the Fast Tracking via Spatio-Temporal Context Learning (STC) algorithm. Red rectangles represent local context regions, and yellow

Algorithm 7 : TLD tracking algorithm [14]

```
Input: The initialised tracking window for the first frame and the video sequence as
    frames.
Output: Tracked positions.
 1: Use the initial bounding-box as the starting point for the tracker.
 2: Learn an initial classifier and update the object model.
 3: for all frames do
 4:     Estimate object motion by using the tracker.
 5:     if tracking failure detected then
 6:         Do not return a bounding-box.
 7:     else
 8:         Return the tracker bounding-box.
 9:     end if
10:     With the detector, scan the frame by a scanning-window.
11:     Output the detected object bounding-boxes.
12:     Combine the bounding boxes of the tracker and the detector.
13:     if no bounding boxes then
14:         Declare the object invisible.
15:     else
16:         Output the most confident bounding-box position.
17:     end if
18:     Update the detector by using the P- and the N-expert.
19:     if appropriate according to the update strategy then
20:         Update the object model.
21:     end if
22: end for
```

Algorithm 8 : RSCM tracking algorithm [30]

```
Input: The initialised tracking window for the first frame and the video sequence as
    frames.
Output: Tracked positions.
 1: Obtain positive (the target position) and negative samples (near the target) for the
    original template.
 2: for all frames do
 3:     With the particle filter obtain N candidates.
 4:     Calculate the confidence value of each candidate against the original template.
 5:     Obtain M patches for each candidate.
 6:     Calculate similarity between the original template and the candidates.
 7:     Select the candidate with max likelihood based on the confidence and similarity
    and update the target position.
 8: end for
```

rectangles indicate target locations. FFT stands for Fast Fourier Transform, and IFFT is the inverse Fast Fourier Transform. H_t^{stc} and H_{t+1}^{stc} represent spatio-temporal context models in the current frame and in the next frame. h_t^{sc} is the spatial context model. ρ is a learning parameter with values smaller than 1, 0.075 is used as the default value. First, a spatial context model h_t^{sc} between the target and its surrounding background is learnt. In the following step, the learnt model h_t^{sc} is used to update the spatio-temporal context model H_t^{stc} for the following frame H_{t+1}^{stc}. The tracking task is then formulated by convolution as a computing task of a confidence map, and the best object location can be estimated by maximising the confidence map. Computing FFT for the local context region of M x N pixel is of complexity $\mathcal{O}(MN\log(MN))$. This makes the algorithm fast: it reached 350 fps on average in an 18-sequence test set when run on i7 3.40 GHz machine with 8 GB RAM. [17]

Evaluation of the Fast Tracking via Spatio-Temporal Context Learning (STC) algorithm was done with the help of 18 video sequences and against 18 state-of-the-art tracking methods in [17]. The evaluation metrics consisted of Success Rate (SR) and Centre Location Error (CLE). Tracking was considered successful if the overlap of the bounding boxes were more than 50%. [17]

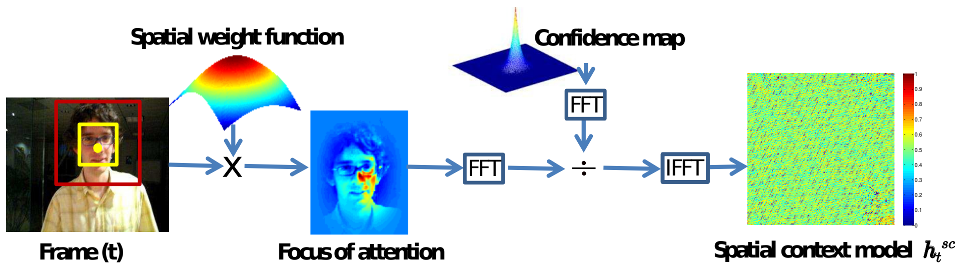


Figure 7. Learning the spatial context model h_t^{sc} at the t-th frame. The yellow rectangle depicts the target location and the red rectangle the local context region. FFT denotes Fast Fourier Transform, and IFFT denotes the Inverse FFT. [17]

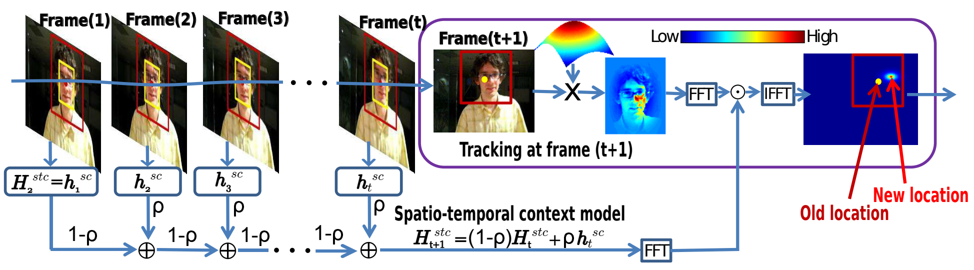


Figure 8. Detecting the object location at the (t+1)-th frame. Target locations are represented with yellow rectangles and local context regions with red rectangles. FFT denotes Fast Fourier Transform, and IFFT denotes the Inverse FFT. [17]

Algorithm 9 : STC tracking algorithm [17]

```
Input: The initialised tracking window for the first frame and the video sequence as frames.
Output: Tracked positions.
1: for all frames do
2:     Learn spatial context model from the target location in the previous frame.
3:     Update the spatio-temporal context model by using the learnt spatial context model.
4:     The object position is determined by maximising the confidence map.
5:     For every Nth frame, update the scale.
6: end for
```

2.3.10 Structured Output Tracking with Kernels

Structured Output Tracking with Kernels (struck) extends the Learning to Localise Objects with Structured Output Regression tracking by Blascko and Lampert [45]. The main idea of the tracker is to create positive samples from areas close to the object, containing the object, and negative samples further away from the object, containing background. It exploits the spatio-temporal context for visual tracking and finds statistical correlation between the object of interest and its local context based on probabilities. Also, it uses a confidence map and obtains the best location by maximising a location likelihood function of an object. In a test video, the positive and negative sampling at one particular moment can be seen in Figure 9. [33]

Steps taken during tracking are explained in Algorithm 10. In the first frame, an object model is created using the initialised tracking window. The target object in the image and areas near the target object, still containing the target object, are classified as positive samples and areas further away containing background and parts of the target are classified as negative samples, Based on these positive and negative samples, the target object in the next frame can be detected by estimating the change in the target object location and finding the most probable transformation based on a Support Vector Machine (SVM) classifier. The classifier is then updated with the current position and budget maintenance is carried out, limiting the number of SVM vectors, if exceeded, to a set amount. This is done by removing the support vector with a minimum impact on the classifier. New samples of the target object and background are taken. By checking the relevance of the new samples against old samples, it is found out if the model needs to be updated. After that, the samples get optimised and the bounding box is returned. [33]

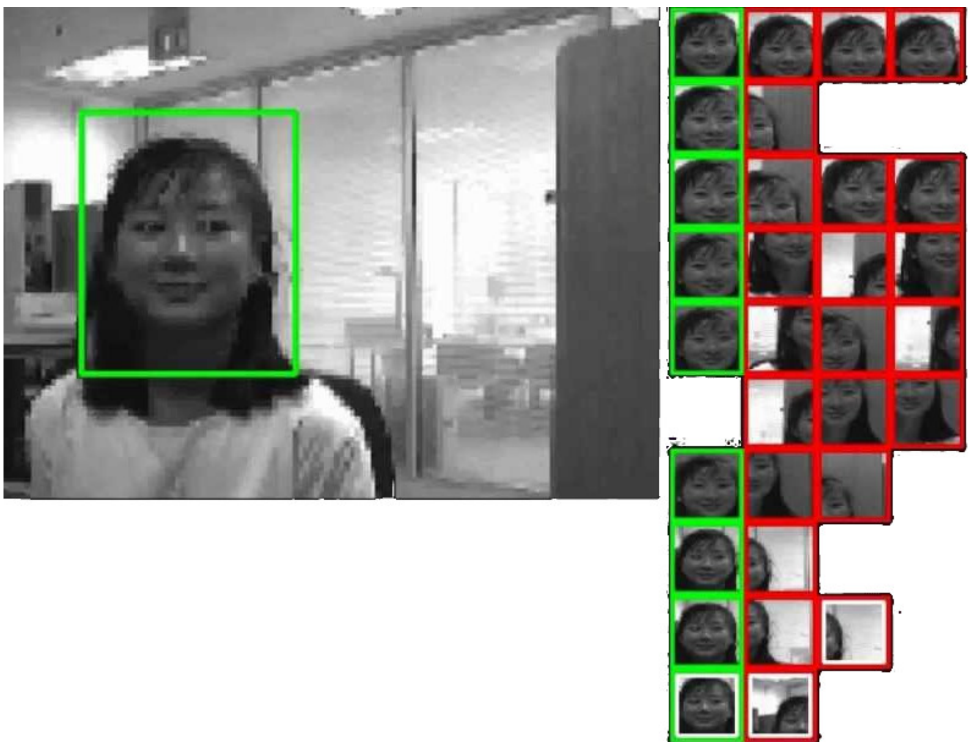


Figure 9. Use of positive (green) and negative samples (red) in struck illustrated. [33]

2.3.11 Single and Multiple Object Tracking Using Log-Euclidean Riemannian Subspace and Block-Division Appearance Model

Like IVT [25], the Single and Multiple Object Tracking Using Log-Euclidean Riemannian Subspace and Block-Division Appearance Model (LRS) algorithm takes into account affine parameters, translation, rotation, scale, aspect ratio, and skew [35]. Including the affine parameters in the tracking framework makes the algorithm more robust to object rotation and scale changes.

The main steps of the method are presented in Algorithm 11. To obtain the optimal object state, an observation model and dynamic model are used. The observation model is used to map the similarity between the image region and the learnt appearance model. The dynamic model is used to update the particle filter, and a particle filtering approach is used to estimate the optimal state. Image region with the optimal state is used to incrementally update the appearance model. During tracking, each image region is warped into a normalised rectangular region by using the estimated affine parameters. [35] The five modes in Figure 10 refer to the five modes of the object appearance: translation, rotation, scale, aspect ratio, and skew.

Algorithm 10 : struck tracking algorithm [33]

```
Input: The initialised tracking window for the first frame and the video sequence as frames.
Output: Tracked positions.
1: for all frames do
2:     Estimate change in the object position.
3:     Find the most probable transformation based on SVM.
4:     With the new position, update the discriminant function.
5:     Use budget maintenance to limit the number of SVM vectors.
6:     Take new samples and check the relevance.
7:     Go through the old samples and optimise them.
8:     Return the last bounding-box position and transformation similarity function.
9: end for
```

Algorithm 11 : LRS tracking algorithm [35]

```
Input: The initialised tracking window for the first frame, and the video sequence as frames.
Output: Tracked positions.
1: for all frames do
2:     Learn the appearance model.
3:     Using the particle filtering approach estimate the optimal state.
4:     With the image region associated with the optimal state, update the appearance model.
5:     Output the optimal state estimated by the particle filter and update the target position.
6: end for
```

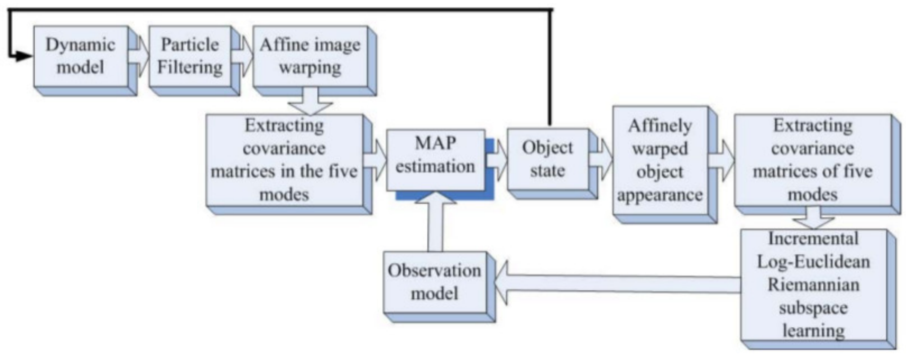

Dynamic model
Particle Filtering
Affine image warping
Extracting covariance matrices in the five modes
MAP estimation
Object state
Affinely warped object appearance
Extracting covariance matrices of five modes
Incremental Log-Euclidean Riemannian subspace learning
Observation model

Figure 10. LRS tracking framework. [35]

2.3.12 Online Object Tracking with Sparse Prototypes

The online object tracking method, Online Object Tracking with Sparse Prototypes (SRPCA), developed by Wang et al. [37] takes into account affine parameters: translation, rotation, scale, aspect ratio, and skew, as do IVT and LRS. Figure 11 and Algorithm 12 summarise the main steps of the SRPCA algorithm. After initialisation, candidate states are sampled and evaluated using the observation model. In the next step, the best candidate is selected and the occlusion map is computed. For the final steps, samples are used to update the observation model. The observation model can be fully updated, partially updated, or not updated at all, according to the occlusion map from the previous step. [37]

Algorithm 12 : SRPCA tracking algorithm [37]

```
Input: The initialised tracking window for the first frame and the video sequence as
    frames.
Output: Tracked positions.
1: for all frames do
2:     Sample candidate states.
3:     Using the observation model, evaluate the candidate states.
4:     Obtain the best candidate and the related occlusion map.
5:     Update the target position.
6:     Based on the samples' occlusion map, decide between full update, partial update,
       and no update to the observation model.
7:     Update the observation model.
8: end for
```

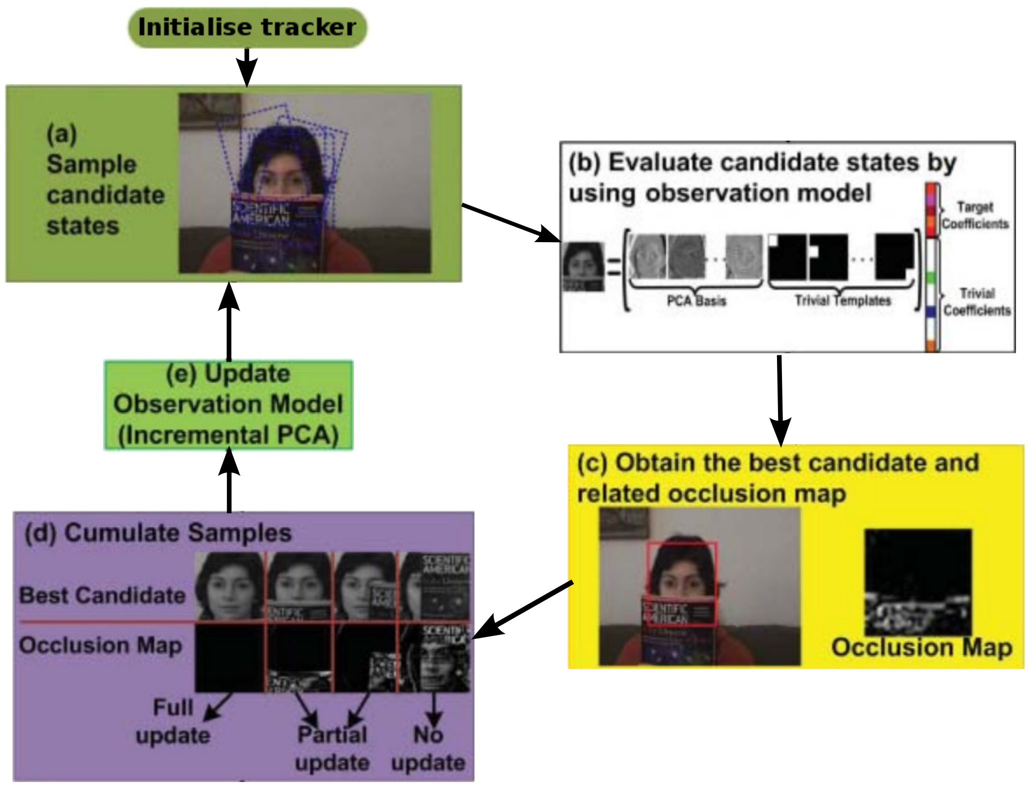

Initialise tracker
(a) Sample candidate states
(b) Evaluate candidate states by using observation model
PCA Basis
Trivial Templates
Target Coefficients
Trivial Coefficients
(c) Obtain the best candidate and related occlusion map
Occlusion Map
(d) Cumulate Samples
Best Candidate
Occlusion Map
Full update
Partial update
No update
(e) Update Observation Model (Incremental PCA)

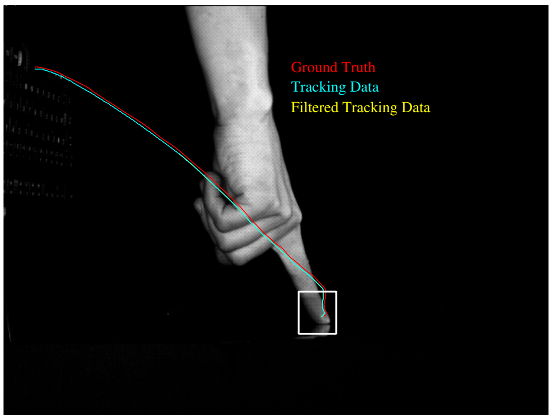
Figure 11. SRPCA tracking algorithms three main parts: dynamic model, observation model, and update module. [37]

2.4 Problems with Tracking Data

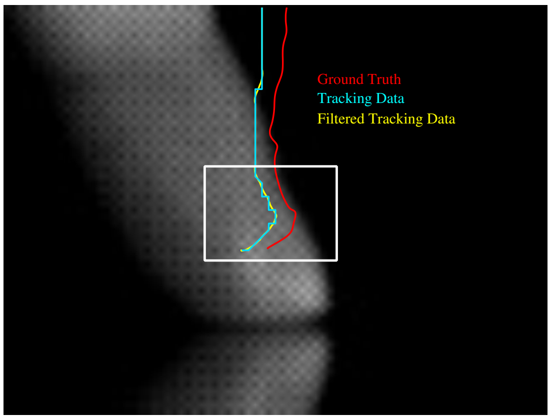
The data provided by trackers is just a collection of locations of moving object(s) and usually contains movement noise that is not included in the real movement of the object. This noise is often staircase-like movement (Figure 12 (c)) which contains a zoomed image of one part of the tracked hand trajectory. Pure tracking data is shown in cyan colour, the ground-truth in white and various filtered results in red, green, and magenta. There is also the question of pixel vs sub-pixel accuracy and how it affects velocity and acceleration calculations. Trackers usually track the object using only pixel accuracy or even less accurate measures. That is one reason why including filtering of trajectories is necessary to accurately represent the trajectory.

Tracking errors can be minimized. If the limiting factor is the number of pixels moved and there is no better material for the sequences, accuracy can be increased by upscaling the current sequence or calculating some probability function to estimate the object's location. A probability function using pixel value changes of the edge pixels can give a better estimate of the true object centre. Another possibility is to upscale the image by resizing, with an upscaling factor, the object region or the whole image. This can lead to more accurate position measures, but it also increases the computational time for most of the trackers. Still, the results would contain rough-edged transforms between the frames, which would affect velocity and acceleration calculations.

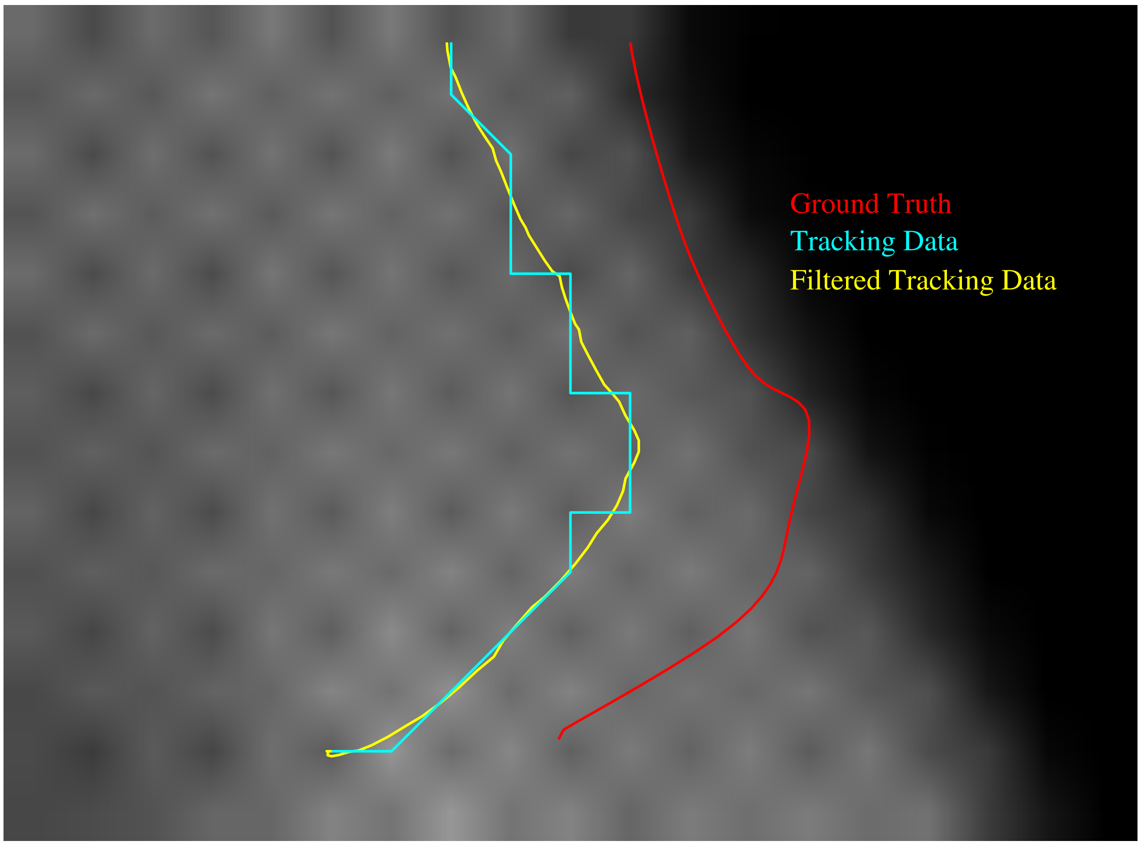
It is also intriguing to see how many milliseconds it takes for a tracking bounding box to move from one pixel region to another during hand movement in the slowest part of the trajectory and also in the fastest part of the trajectory. This becomes an issue when trying to get the accurate position and derivatives of position data out of the high-speed video sequences. It was noted in particle tracking with high-speed cameras that one dust particle needed 30 ms to cross from one pixel to another [46]. So with 500 fps there would have to be 15 frames for the dust particle to move between pixels. This information could be used to drop unmeaningful frames from video sequences, but then one would need to know exactly when the particle moves between pixel regions, and that would cause problems, too.



(a) Full image.



(b) Area marked in (a) zoomed.



(c) Area marked in (b) zoomed.

Figure 12. Tracking data and filtering applied to it. Tracking data shown in cyan, ground-truth in red and filtered tracking data in yellow.

3 POST-PROCESSING AND ANALYSIS OF TRAJECTORY DATA

This chapter focuses on post-processing and analysis of trajectory data obtained from object tracking results. Filtering and smoothing methods such as moving average, Savitzky-Golay, and total variation denoising are introduced. A short introduction to camera calibration, imaging and multiview geometry is given. Feature extraction methods and calculations for computing velocities and accelerations from tracking data are also introduced. The methods presented in this chapter aim to provide smooth trajectory, velocity and acceleration curves. The smoothness of the trajectory can be quantified as a function of jerk, which is the time derivative of acceleration. Thus, jerk is the second time derivative of velocity and third time derivative of position. In other words, jerk is the rate of change in acceleration and defines how rapidly acceleration is perceived.

3.1 Filtering and Smoothing

Extracting higher level features from the tracking results, which are a collection of centre location points collected from tracked image sequence or video, can be challenging. The resulting matrix of centre locations of object over time is usable as such for checking the position of the object at certain time, but, when higher level features such as velocity or acceleration are needed, sub-pixel accuracy must be reached. Sub-pixel accuracy is needed because trackers operate at pixel level accuracy, and in high-speed videos object movement can be smaller than one pixel per frame advance. This causes large errors when calculating velocities or accelerations for the tracked object, resulting in erratic acceleration and deceleration curves, which are hard to read when evaluated.

The main goal for any experiment or test is to extract the information that quantifies the phenomena. Noise is described as random errors that contaminate this information. It is important to remove this noise as much as possible without weakening the underlying signal or information [47]. “Filter” in common usage means a data processing algorithm [48]. Filtering is used for many purposes, usually to remove unwanted noise from measurements. A filter is an electronic circuit or algorithm that removes or attenuates some unwanted component or feature from a signal. Filters are used to eliminate background noise, modifying digital images, and removing specific frequencies in data analysis, among other things. [49]

Most of the signals which are measured in real situations get corrupted in some way or another by some unwanted signals. For the purpose of signal processing and analysis, it is imperative to get rid of these interferences, or at least reduce their effects. This can be achieved by applying signal filtering techniques [49].

3.1.1 Moving Average

The moving average (MA) filter operates by averaging a number of points from the input data to calculate the output. The data point to be filtered can be either from the start of the averaging sequence or from the middle of the sequence so that the group of points to be included in averaging are chosen symmetrically around the output point. Depending on the implementation, end point(s) of the output signal cannot be smoothed because the span cannot be defined for end point(s). Selecting the points symmetrically is more common because it does not introduce a relative shift between the input and output signals. [50]

At the start of the dataset, the points get included one by one into the moving average calculation until the full span window is achieved; similarly, at the end of the dataset, the window for the moving average becomes smaller. At the last point, the moving average covers only that point because there must be an even number of samples on both sides of the span. In general, moving average works by fixed number of points, adding their ordinates together and dividing the result by the number of points. This approach method attenuates peaks from data. One solution to preserve the peaks during smoothing is discussed in Section 3.1.4.

In the moving average, $y_s(i)$ is the smoothed value for the ith data point which is calculated as:

$$y_s(i) = \frac{1}{2N+1}(y(i+N) + y(i+N-1) + \ldots + y(i-N)) \tag{1}$$

where N is the number of neighbouring data points on both sides of $y_s(i)$, and $2N+1$ is the size of the smoothing window, otherwise known as span. [51]

3.1.2 Kalman Filtering and Smoothing

A Kalman filter is an optimal recursive data processing algorithm. The system model of the linear Kalman filter in Figure 13 shows how at each time step the state vector $\mathbf{x}_k$, which is the estimate of the current true state, is extended to the new state estimation $\mathbf{x}_{k+1}$

by multiplication with the constant state transition matrix $\mathbf{A}$. The state vector $\mathbf{x}_{k+1}$ is furthermore influenced by the control input vector $\mathbf{u}_{k+1}$ multiplied by the control matrix $\mathbf{B}$, and the system noise vector $\mathbf{w}_{k+1}$. A system state cannot be measured directly. Measurement vector $\mathbf{z}_k$ consists of the information contained within the state vector $\mathbf{x}_k$ multiplied by the observation matrix $\mathbf{H}$, and the additional measurement noise $\mathbf{v}_k$. [52, 53]

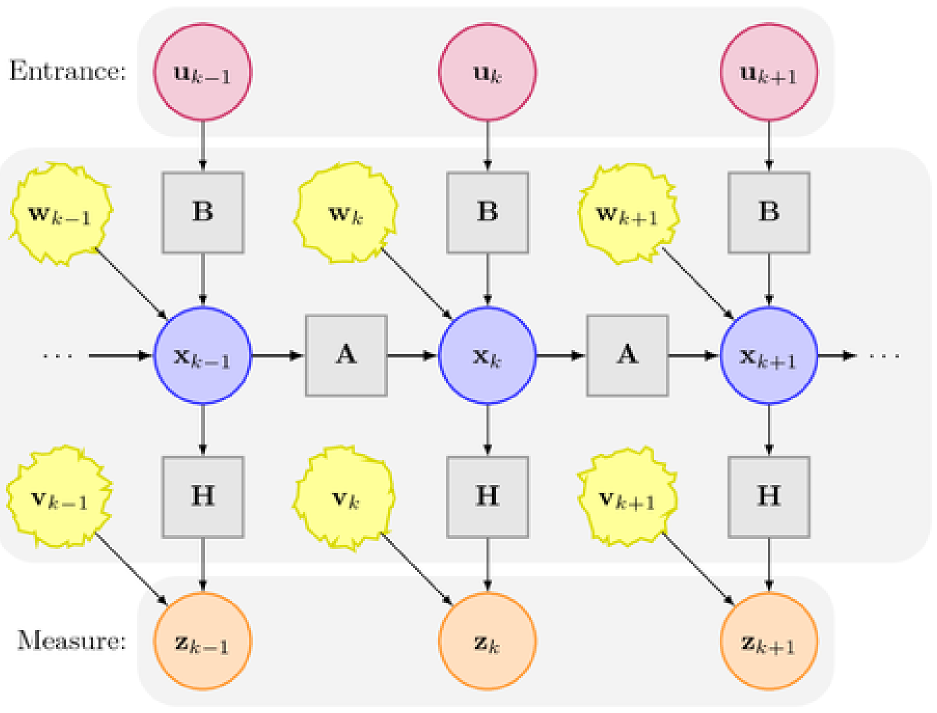


Figure 13. Kalman Filter System Model. A is a state transition matrix, B is a control matrix, x_{k-1} is the latest estimate of the current state, and u_k is control vector [54].

Predicted state estimate $x_{predicted}$ is calculated as

$$x_{predicted} = Ax_{k-1} + Bu_k, \tag{2}$$

where A is a state transition matrix, x_{k-1} is the latest estimate of the current state, B is a control matrix, and u_k is control vector. Predicted covariance $P_{predicted}$ is calculated as

$$P_{predicted} = AP_{k-1}A^T + Q, \tag{3}$$

where P_{k-1} is the latest estimate of the average error for each part of the state, T denotes translation, and Q is the estimated process error covariance. Innovation $\hat{y}$ is calculated as

$$\hat{y} = z_k - Hx_{predicted}, \tag{4}$$

where z_k denotes the measurement at this time step, and H is the observation matrix. Innovation covariance S is calculated as

$$S = HP_{predicted}H^T + R, \tag{5}$$

where R is the estimated measurement error covariance. Kalman gain K is calculated as

$$K = P_{predicted}H^T S^{-1}. \tag{6}$$

Finally, the state update x_k which is the new estimate of current position

$$x_k = x_{predicted} + K\hat{y}, \tag{7}$$

is calculated by adding Kalman gain K into state prediction value $x_{predicted}$. Lastly, the covariance update P_k is calculated as

$$P_k = (I - K_K H)P_{predicted}, \tag{8}$$

where I is identity matrix with ones on the main diagonal and zeros elsewhere.

The predefined values in the Kalman filter equation set are the state transition matrix A, control matrix B, observation matrix H, estimated process error covariance matrix Q, and estimated measurement error covariance matrix R. The inputs for the Kalman filter calculations are the control vector U_k and measurement vector Z_k which contains the measurements. The outputs from the Kalman filter are the estimate of the current true state x_k, and P_k, which is the estimate of the average error for each part of the state. [53]

The predictor predicts parameter values based on the current measurements. The filter estimates parameter values by using the previous and current measurements. The smoothing algorithm estimates parameter values by using the previous, current, and future measurements: that is, all available data can be used for filtering [52]. Future measurements can be used because the Kalman smoother proceeds backward in time. This also means that the Kalman filter needs to be run before running the smoother.

The process model models the current state at time k from the previous state at k-1. Q is the process error covariance which contributes to the overall uncertainty. When Q is large, the Kalman filter tracks large changes in the data more closely than with smaller Q. The measurement error covariance R determines how much measurement information is used. The Kalman filter considers the measurement not to be very accurate if R is high: if R is smaller, then it follows the measurements more closely. [52, 53]

Extended Kalman filter (EKF) is the nonlinear version of the Kalman filter. EKF has been considered as the de-facto standard in nonlinear state estimation [55]. EKF uses first-order terms of the Taylor series expansion of nonlinear functions. Large errors are introduced when the models are highly nonlinear. Local linearity assumption breaks down when the higher order terms become significant. For EKF, the three first steps of the process are: linearising by Jacobian matrix, computing the predicted mean, and predicted covariances. After these three simplified steps, the rest of the Kalman process calculates Kalman gain and updates, with measurements, the state estimate. [55]

In Unscented Kalman Filter (UKF) [56], unscented transformation is used to calculate the statistics of a random variable which undergoes a nonlinear transformation. It is built on the principle that it is easier to approximate a probability distribution than an arbitrary nonlinear function. In UKF, the process starts with sigma point creation. Sigma points are formed by selecting minimal set of carefully chosen samples that represent the state distribution. After the sigma points creation, they are run through the process model and, finally, transformed mean and covariance are computed. After these steps, the rest of the Kalman process is similar to the last three steps of the EKF algorithm. UKF is highly efficient, has almost the same complexity as EKF, and only a constant factor slower in typical practical applications. UKF achieves better linearisation than EKF, and it is accurate in the first two terms of the Taylor expansion while EKF is accurate only in the first term. UKF is derivative-free and there is no need to calculate the Jacobian matrix, but the state estimation for nonlinear systems in UKF is still not optimal. [55, 56, 57]

3.1.3 LOESS, LOWESS and Robust Versions

LOESS (local regression) and LOWESS (locally weighted scatterplot smoothing) were developed to enhance visual information on scatterplots by computing and plotting smoothed points by using locally weighted regression [58]. LOESS and LOWESS are methods to estimate the regression surface through a smoothing procedure. Estimation is done by fitting a function inside a sliding window into the variables. The weight function in LOESS and LOWESS works in such a way that the points closer to the curve play a larger role in the determination of the smoothed values of the curve. Smoothed values are calculated by fitting a polynomial of nth degree by using weighted least squares with certain weight w_i at point x_i. Robust versions of LOESS and LOWESS give less weight to the points further away from the curve than the standard versions. [58, 59]

As in the moving average method, each smoothed value is determined by neighbouring data points defined within the span, and the process is weighted since a regression weight function is applied for the points included within the span. Robust weight function, which makes the process more resistant to outliers, can also be used in addition to the regression weight function. [58, 59]

The methods are separated by the regression model: LOWESS uses a linear polynomial while LOESS uses a quadratic polynomial. This section considers the implementation of LOWESS and LOESS methods proposed in [58]. Detailed explanations can be found in [51]. If there are the same number of neighbouring data points available on each side of smoothed data, the weight function is symmetric: if not, then the function is asymmetric. Thus, unlike in the case of the moving average, the span never changes when using LOESS, LOWESS, or their robust versions. This means that there can be phase changes in the beginning and in the end of the data points before the centre of the window is reached.

In LOESS the weight w_i can be calculated as:

$$w_i = (1 - \left| \frac{x - x_i}{d(x)} \right|^3)^3, \tag{9}$$

where x is the predictor value associated with the response value to be smoothed, x_i are the neighbours of x defined by the span, and $d(x)$ is the distance from x to the most distant predictor within the span [51].

Robust versions of LOESS and LOWESS add weights w_i that are given by the bi-square function defined as:

$$w_i = \begin{cases} (1 - (r_i/6\text{MAD})^2)^2 & , |r_i| < 6\text{MAD} \\ 0 & , |r_i| \geq 6\text{MAD}, \end{cases} \tag{10}$$

where r_i is the residual of the ith data point produced by the regression smoothing procedure, and MAD is the median absolute deviation of the residuals: MAD=$median\ (|r|)$. The median absolute deviation measures how spread-out the residuals are. When r_i is small compared to 6MAD, the robust weight is close to one. If r_i is greater than 6MAD, the weight goes to zero and the data point is excluded from the calculation. [51]

To calculate robust versions, the normal LOESS or LOWESS smoothing is calculated first and then the robust weights are calculated according to Eq. 10. With the calculations done, the data is smoothed again by using the robust weights. The final smoothed value is calculated using both, the local regression weight and the robust weight. Robust weight calculation and smoothing is then repeated for a total of five iterations. [51]

3.1.4 Savitzky-Golay

This section introduces the Savitzky-Golay (S-G) smoothing filter, also called the polynomial smoothing or least-squares smoothing filter [49]. Savitzky-Golay smoothing reduces noise while maintaining the shape and height of peaks. For Savitzky-Golay smoothing to work, there needs to be at least as many data samples as there are coefficients in the polynomial approximation. The Savitzky-Golay filter response of order $N = 0$ or $N = 1$ is identical to the moving average filter response [47].

A Savitzky-Golay smoothing filter fits a polynomial to a set of input samples for each input X_n in a least-squares sense and uses the value of the polynomial at time n as the filter output. A function $f_K(x)$ describes a polynomial of order n:

$$f_K(i) = \sum_{i=0}^{K} b_i x^i = b_0 x^0 + b_1 x^1 + b_2 x^2 + \ldots + b_K x^K, \tag{11}$$

where b_is are the coefficients in the polynomial. If N previous and M future samples are used as the neighbouring samples, then the Savitzky-Golay filter finds b_i coefficients that minimise the term:

$$\sum_{i=-M}^{N} (X_{n-i} - f_K(n-i))^2, \tag{12}$$

where the polynomial value at time n is the filter output $\hat{X}_n = f_K(n)$. Thus, when the polynomial describes the data accurately, there is minimal distortion in the curve. It has been shown by Savitzky and Golay [47] that the filter can be expressed as a weighted moving average filter:

$$\hat{X}_n = \sum_{i=-M}^{N} a_i X_{n-i}, \tag{13}$$

where the filtering coefficients a_i are constants for all X_n. The coefficients a_i can be calculated using available algorithms or using available coefficient tables to check the values for various ranges and polynomial degrees. [47]

Savitzky and Golay showed that, when one fits a polynomial to a set of input samples and then evaluates the resulting polynomial at a point within the fitted interval, it is equivalent to a discrete convolution with a fixed impulse response. Output from the Savitzky-Golay filter is a zero phase so that the features of the signal are not shifted when the filtering is applied. The smoothing effect of the Savitzky-Golay filter is not as aggressive as that of moving average, and the loss of signal information is smaller than with the moving average [47, 49]. The idea behind the Savitzky-Golay smoothing filter is to find a filter that preserves higher-order moments while smoothing the data. In particular positions, heights, and widths of the peaks in the signal waveform are preserved [60]. The degree of smoothing in Savitzky-Golay is controlled by the window size (span) and the degree of the polynomial to be fitted into the data.

3.1.5 Total Variation Denoising

Total variation (TV) of a signal measures the changes in the signal between signal values. Total variation denoising (TVD) output is obtained by minimising a TV-based cost function. It was developed to preserve sharp edges in the underlying signal. [61]

TVD was developed for image filtering and smoothing purposes. It is good for removing noise from constant colour backgrounds but tends to remove also textures from textured areas. Fine details in images can be lost due to filtering [62, 63]. There are versions available which also correct these issues. This is done by adding a second round to the filtering process. The first round is done normally, but on the second round, the filtering is calculated again for the textured regions based on the loss of details. [64, 65]

TVD can introduce a staircase effect to the data on smooth slopes, the staircase effect in this case meaning small flat regions in the slopes. These regions appear because the TV regulariser promotes piecewise-constant behaviour. It is not usually the best denoising method for more general piecewise-smooth signals. [62, 63]

TV for a digital N-point signal $x(n)$, $1 \leq n \leq N$ is

$$TV(x) = \sum_{n=2}^{N} |x(n) - x(n-1)|. \tag{14}$$

Assume that observed signal x is corrupted by additive white Gaussian noise n resulting in signal y,

$$y = x + n, \quad y, x, n \in \mathbb{R}^N. \tag{15}$$

TV denoising approach estimates x by finding the signal x minimising the objective function

$$J(x) = \|y - x\|_2^2 + \lambda \|TV(x)\|_1 , \tag{16}$$

where the degree of smoothing is controlled by parameter $\lambda > 0$. Increasing the λ value gives more weight to the term that measures the fluctuation of the signal. Iteration count is another variable in TVD and controls how many iterations the process will do if the error criterion is not yet met in the algorithm.

3.1.6 Smoothing Trajectory Data

Position data that one gets from the trackers does not change until the object at hand has moved enough to exceed a threshold, in this case, a pixel boundary. With high-speed videos, pixel-based values might not be enough since the object can stay within a single pixel region during multiple frame changes. This can also be the case for a slow moving object in normal videos. These issues become problematic when accurate measurements of velocities and accelerations are needed. There are situations where an object stays in one place for multiple frames and then there is this sudden movement to another pixel region. This shows as a peak in the acceleration and velocity curves. There is a need to smooth these sudden movements, and this is where the filtering or data smoothing can help.

The tracking noise from finger and hand-tracking data is not really real noise, but there is a need to smooth the sharp edges that tracking on a pixel level causes to the tracking data sequence. Sharp edges and whole pixel movements in the tracking data are the points where the derivatives of position, velocity and acceleration start to fluctuate.

Tracked hand movement positions can be thought as a set of measures from the actual hand movement trajectory containing a measurement error that is smaller than some variable n. To get better results without filtering, there is need to get into sub-pixel tracking with some marker which allows the tracker to calculate the exact tracked position at sub-pixel level.

3.2 Camera Calibration

To get accurate measures from the images taken with a camera, there is the need to know the camera parameters that define the image. The parameters are divided into intrinsic (internal) and extrinsic (external) types. The intrinsic parameters contain a focal length, principal point and image sensor format, which define the pixel size in x and y directions. They link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame. These parameters depend only on the camera characteristics. The extrinsic parameters contain rotation and translation. They define the location and orientation of the camera reference frame with respect to a known world reference frame. Extrinsic parameters depend only on the position of the camera. The camera calibration parameters can be calculated by known points in the images by using imaging geometry. [66]

3.2.1 Imaging Geometry

The angle between the two rays back-projected from two points in the image can be determined when a calibrated camera is used. Getting to know the intrinsic and extrinsic parameters of the camera is crucial for calculating the corresponding world points from the images. [67, 66]

The pinhole perspective projection model from the 15th century by Brunelleschi is still mathematically convenient. Brunelleschi invented a small-aperture perspective device in 1425, and in his experiments, he was able to demonstrate that there is a vanishing point in the three-dimensional space where specific lines of perspective converge in a point furthest from the eye. [67]

Figure 15 shows, the basics of the pin-hole camera geometry. A pin-hole camera is described by its optical centre (C), also known as the camera centre, and the image plane. Focal length is the distance from the image plane to the optical centre C. Optical axis, principal axis, or principal ray of the camera is the line perpendicular to the image plane from the camera centre. The principal plane goes through the camera centre parallel to the image plane of the camera. [66]

Figure 15 (a) illustrates the projection of the point M on the image plane. The projection is done by drawing a line through the camera centre C and the point to be projected. Figure 15 (b) shows the same situation in the YZ-plane, showing how similar triangles are used to calculate the position of the projected point m on the image plane. The image point being $p = (u, v)$ and the scene point $P = (X, Y, Z)$, perspective projection equations $u = f \times X/Z$, and $v = f \times Y/Z$ can be derived via the similar triangles rule.

Capital letters X,Y and Z are used to denote the world coordinates, and u and v are used

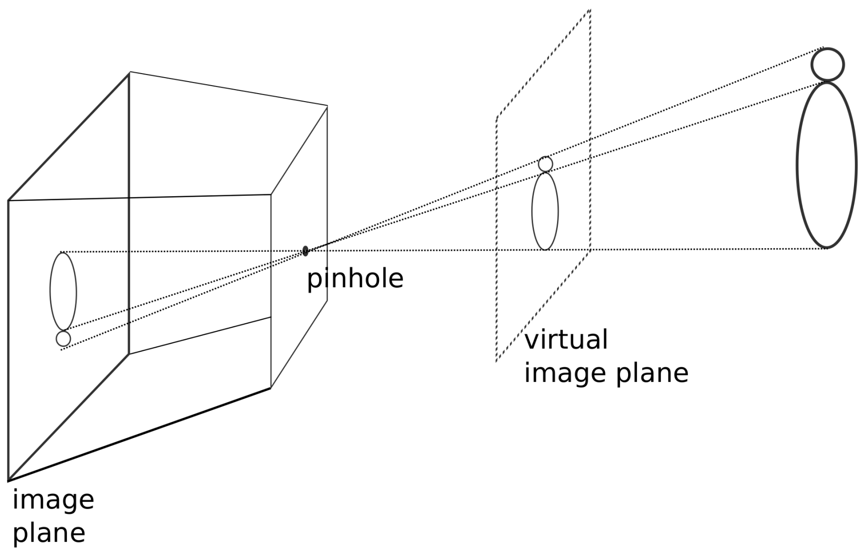


Figure 14. Pinhole projection.

to denote image coordinates. The image plane coordinates can be computed as follows:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \tag{17}$$

where f_x and f_y are the focal lengths in pixel units, c_x and c_y are the coordinates which represent the principal point on the image plane, and X,Y and Z are the corresponding world coordinates. The principal point is usually located at the image centre.

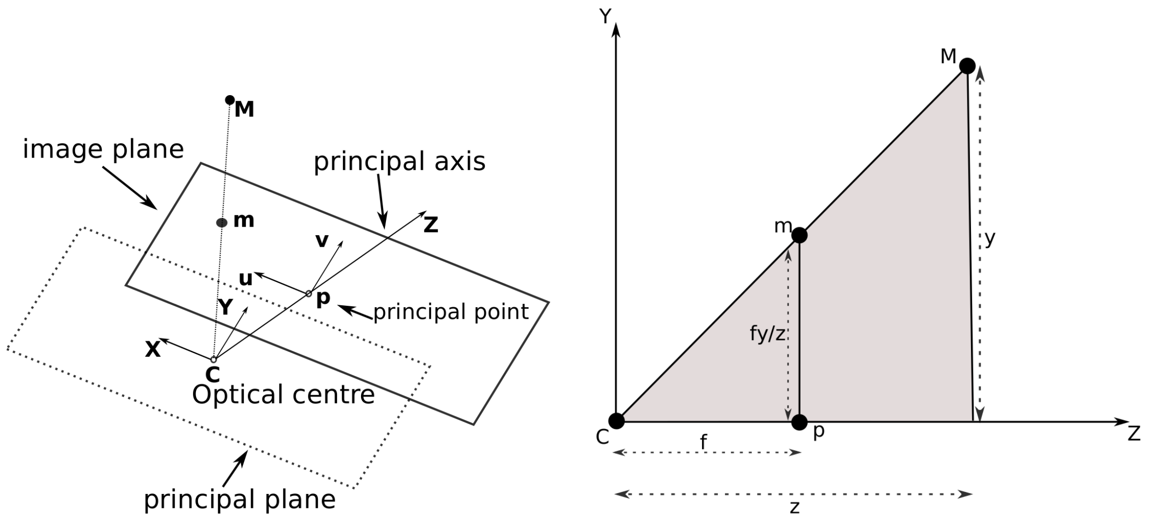
The full camera model containing the intrinsic and extrinsic parameters can be formed as follows:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{11} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{18}$$

$$= C[r_1 r_2 r_3 t]P = C[Rt]P = CAL_{3x4}P$$

where the value of γ contains the skew, rotation parameters r_{nn} define the image rotation, and translation parameters $t_{x,y,z}$ define translations.

Table 2 illustrates various image transforms. Translation contains two degrees of freedom (DOF) and (rigid) Euclidean three, including rotation and translation. With higher degrees of freedoms, Similarity has four DOF, including rotation, translation, and scale. With six



(a) (b)

Figure 15. Pin-hole camera geometry: (a) Projection of the point M on the XY-image plane; (b) Projection of the point M on the YZ-image plane

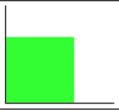
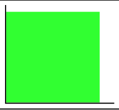
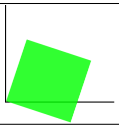
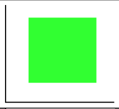
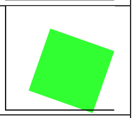
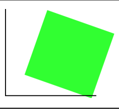
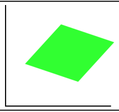
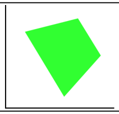
DOF comes Affine transform, including rotation, translation, scale, aspect ratio, and skew. Finally, there is also Projective transform with eight DOF, including rotation, translation, scale, aspect ratio, skew, and perspective warp.

3.2.2 Position

Real world position can be calculated from the image coordinates or at least estimated through different methods. The most commonly used method uses a stereo camera setup where image from the same scene is taken simultaneously by two cameras that are aligned. In this way, it is possible to find also the depth in the image. Because of the way that a normal image is formed, it is impossible to find the depth from a single image. It is only possible to estimate the depth from an image by using some scene points as reference points, but for accurate measurements at least two views from the scene are needed.

The accuracy for the methods can be checked by evaluating the depths calculated from the image against fixed real-world setting where the distances are known. The position data tells where the object moves in real world or in pixel coordinates. With the position information, it is possible to check where the features of interest are located.

Table 2. 2D image transformations.

Transform	DOF	Matrix form	Image
Original Image (I)	-	$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$	
Scale (s)	1	$\begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix} I$	
Rotation (R)	1	$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} I$	
Translation (t)	2	$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} I$	
Euclidean ($R\|t$)	3	$\begin{pmatrix} \cos\theta & \sin\theta & a \\ -\sin\theta & \cos\theta & b \\ 0 & 0 & 1 \end{pmatrix} I$	
Similarity ($sR\|t$)	4	$\begin{pmatrix} \alpha\cos\theta & \beta\cos\theta & \alpha(a\cos\theta - b\sin\theta) \\ -\alpha\sin\theta & \beta\cos\theta & \beta(a\sin\theta + b\cos\theta) \\ 0 & 0 & 1 \end{pmatrix} I$	
Affine (A)	6	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} I$	
Projective (H)	8	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} I$	

3.3 Feature Extraction

Hand-tracking data can provide velocity, acceleration, position and accuracy measures which can be calculated from 2D image points or from translated 3D real-world coordinates. The following sections introduce the methods for calculating velocities, accelerations, and features of interest from the tracking.

3.3.1 Velocity and Acceleration

The velocity of a moving object is calculated as the distance it travelled with respect to time. The velocity in a given point is the movement of the object in pixels between two consecutive frames. To calculate the distance between the sequences of frames the Euclidean distance formula is used. Euclidean distance d in two-dimensional case is calculated as

$$d(x, y) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tag{19}$$

where (x_1, y_1) are the start-point coordinates, and (x_2, y_2) are the end-point coordinates. Velocity can be negative or positive. The magnitude of velocity is speed, which can only be positive. Velocity v can be calculated as

$$v = \frac{\triangle d}{\triangle t} = \frac{displacement}{change\ in\ time}, \tag{20}$$

where $\triangle d$ denotes displacement and $\triangle t$ change in time.

Acceleration is the change of velocity between two measurements. Like velocity, it can be calculated with a longer interval, but then there is also loss in the details of acceleration data. Acceleration is the rate of the change in velocity; it is also a vector, as it implies a direction. Negative acceleration is known as deceleration. Acceleration a can be calculated as

$$a = \frac{\triangle v}{\triangle t} = \frac{change\ in\ velocity\ (or\ speed)}{change\ in\ time}, \tag{21}$$

where $\triangle v$ denotes change in velocity.

3.3.2 Features of Interest for Hand Movement Analysis

Features of interest for hand movement analysis in this thesis come from goal-directed aiming and a multiple-process model of limb control. The model is based on two-component model of speed–accuracy relations in voluntary movement and consequences of a planned movement. To find an accurate and fast movement to the target, the relationship between movement speed and accuracy and the change in aiming behaviour between trials are considered in the model. [68]

nteresting feature points (in time and space) in the hand movement are the moment of the maximum velocity, maximal acceleration and deceleration. Points where deceleration starts, as measured from the end point and/or from the starting position, also provide useful information when defining the smoothness and continuity of the hand movement. Also, the point where the movement stops and the accuracy of the movement hitting the target are interesting. Earlier research on hand movements [68] in pointing actions has shown that, in addition to the primary movement towards the target, there are corrective sub-movements that one can observe from the acceleration or velocity changes during the hand movement.

Figure 16 clearly shows that the movement is divided into two parts: primary and corrective sub-movements. Corrective sub-movement is needed when the target is not reached or is overshot with the primary sub-movement [68]. Timing and relative positions of these sub-movements are one part of the key features for the hand movement analysis. According to earlier research, the deceleration part of the first sub-movement starts approximately 10 centimetres before the target [68].

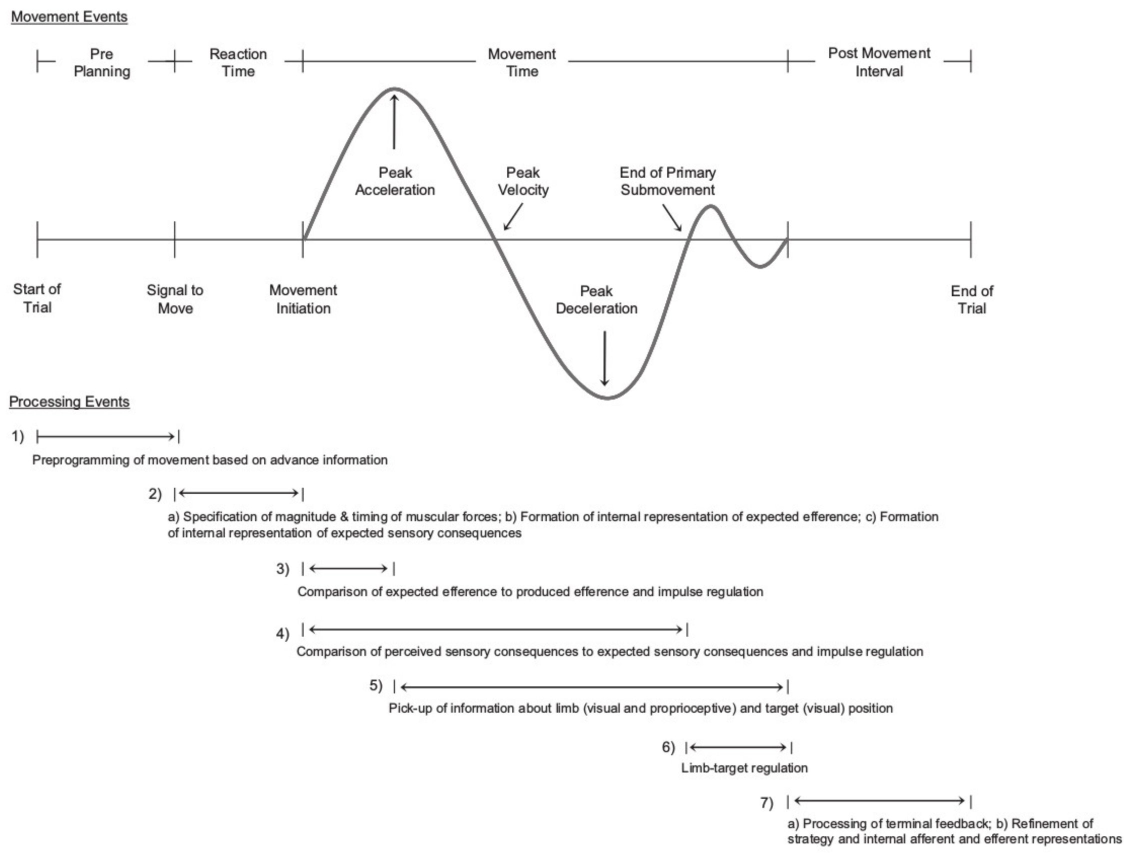

Movement Events
Pre Planning
Reaction Time
Movement Time
Post Movement Interval
Peak Acceleration
Peak Velocity
End of Primary Submovement
Start of Trial
Signal to Move
Movement Initiation
Peak Deceleration
End of Trial
Processing Events
1) Preprogramming of movement based on advance information
2) a) Specification of magnitude & timing of muscular forces; b) Formation of internal representation of expected efference; c) Formation of internal representation of expected sensory consequences
3) Comparison of expected efference to produced efference and impulse regulation
4) Comparison of perceived sensory consequences to expected sensory consequences and impulse regulation
5) Pick-up of information about limb (visual and proprioceptive) and target (visual) position
6) Limb-target regulation
7) a) Processing of terminal feedback; b) Refinement of strategy and internal afferent and efferent representations

Figure 16. Multiple processing events associated with a single goal-directed movement. [68]

4 EXPERIMENTS

In this chapter, the test environments and datasets are introduced. Parameters, performance measures and evaluation methods used for each of the experiments are explained. The results from various experiments conducted with selected evaluation methods and performance metrics are presented. Tracking evaluation metrics are introduced and described and the results are shown with the selected measurements. Filtering and smoothing evaluation is explained and results are shown. The next section explains camera calibration procedure and the results. Finally, the section on measuring trajectory features demonstrates computed trajectories, velocities and accelerations from tracked hand movement.

4.1 Test Environment

Tracking, filtering, and smoothing performance tests were conducted using a desktop computer with Intel i5-4570 CPU, 8 GB of memory, 1 TB Serial ATA (SATA) 3 7200rpm Hard Drive, and a Linux operating system. The first dataset used in this research was collected in various locations with different conditions during earlier high-speed hand-tracking research [41, 42]; the second dataset was collected with a high-speed imaging test-setup explained next. The focus of the second dataset was in the tracking of the finger of the test subject. The high-speed imaging part and planning of the test setup for the second dataset in this research were done in collaboration with POEM [10]. Hand movements made during tests were recorded with a Mega Speed MS50K high-speed camera [69], which was equipped with the Nikon 50mm F1.4D objective and C-mount adaptor. The camera was positioned on the right side of the test setup, and the distance to the screen was approximately 1.5 meters. The lighting was arranged with the help of one overhead LED light rack, with four LED modules including five power LEDs with a total power of 60W, 85 cm above the table surface and 58 cm in depth. The test subject was sitting at the distance of 65 cm from the touchscreen, and the trigger-box was placed 40 cm away from it. The touchscreen was LG T1710 4:3, 17" with 1280$\times$1024 resolution. 3D information was provided as stereoscopic 3D. The trigger-box contained a haptic dot to define an accurate starting position for the test subject. The setup and its 3D construct can be seen in Figure 17. For input sequence triggering and experiment control, a Microsoft Windows operating system running on HP Z820 workstation with 3D GPU was used.

Finding the right settings for the second dataset, captured in the planned environment, was

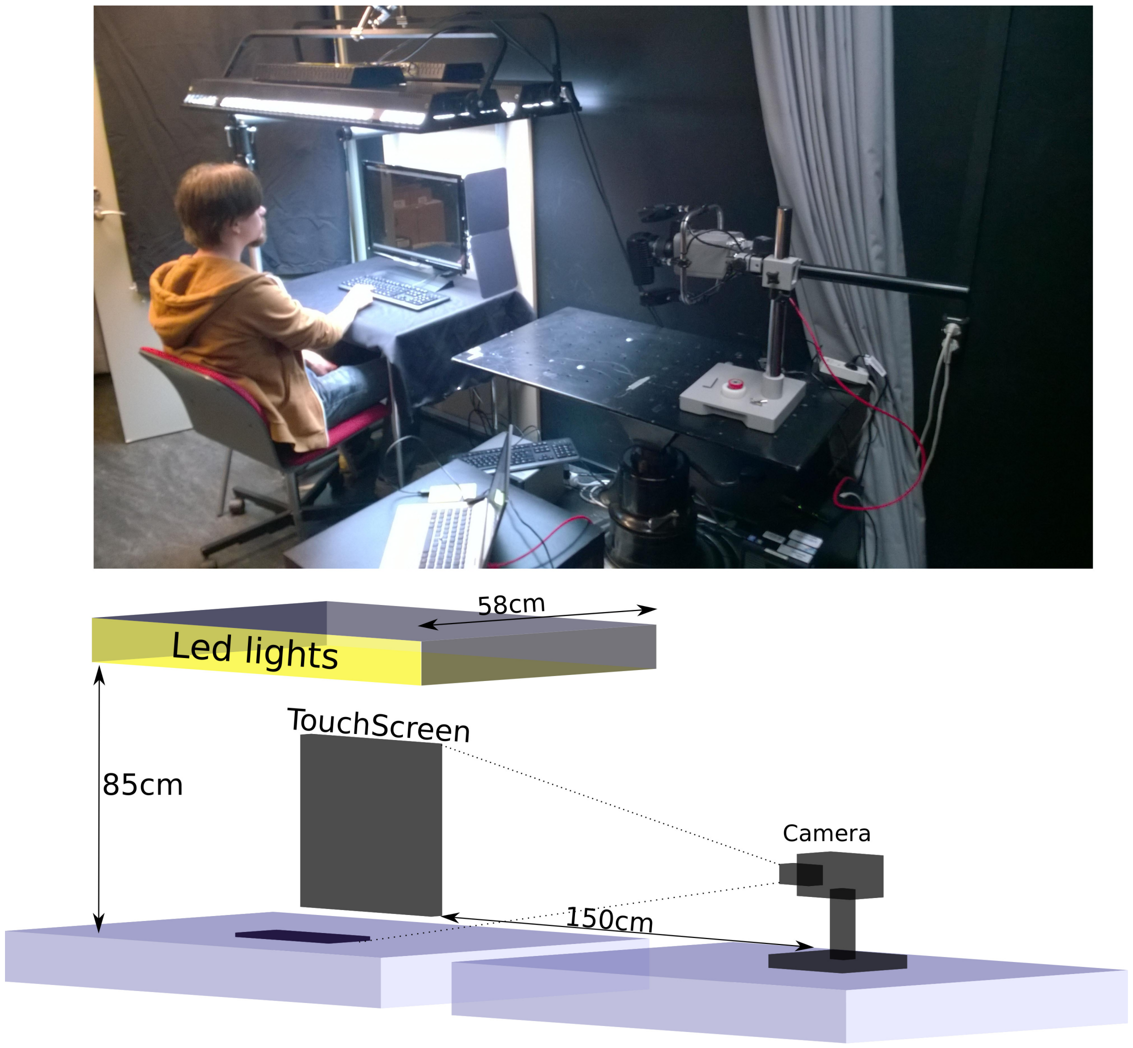


Figure 17. Test setup for high-speed sequence imaging.

done by testing different camera objectives, from 14 mm to 50 mm and zoom. Different video frame rates were tested from 400 to 600 fps. Also, the different resolutions of 800×600 and 1280×1020 were included to find accurate enough testing videos for the research. Light to the screen and around it was kept near 6000 luxes.

4.2 Datasets

In this section, two different datasets are introduced and the annotation process of the ground-truth is explained. Dataset 1 is part of earlier research [41, 42]. Dataset 2 was collected with the imaging environment described in Section 4.1.

4.2.1 Dataset 1

Dataset 1 contained videos provided in [41, 42]. The dataset contains 6 videos with varying frame rates and situations. The video sequences are presented in Table 3. These videos contain different scenes with various tasks and tracked targets varying from a thumbnail, to a hand and a fingertip. The video sequence *drawing* contained originally only 47 annotations. For this research the video was re-annotated to have all the frames included in the annotation. All other videos in this dataset were tested with the original annotations. The number of ground-truth annotations used per video sequence is illustrated in Table 3.

Table 3. Video sequences used in Dataset 1.

Video	Resolution	Fps	Target	Posture changes	Frames/ Annotations	Notes	Colour
drawing [41]	800×600	160	Thumbnail	No	700/700	Drawing a stick figure	No
drawing2 [41]	640×480	30	Thumbnail	No	136/28	*drawing* filmed with normal camera	Yes
touch2 [41]	640×480	80	Hand	Yes	1101/56	Touching an object	Yes
finger2 [41]	720×574	250	Fingertip	No	1328/54	Tablet gaming	No
tablet2 [41]	640×650	250	Fingertip	Minor	400/27	Tablet gaming from different angle	No
cup [41]	640×480	80	cup	Minor	325/33	Grab a cup and drink	Yes

Sample frames of Dataset 1 videos can be seen in Figure 18. The examples were taken 100 frames apart. The top-row images are from the *cup* video sequence, the second row images from the *drawing* video sequence, the third row images from the *finger2* video, and the fourth row images from the *tablet2* video sequence. The bounding-boxes defining the target object region can be seen in the images as green rectangular regions.

4.2.2 Dataset 2

Test environment planning for Dataset 2 was done in collaboration with POEM [10], and the execution of the imaging part was done by POEM using the environment described in Section 4.1. Dataset 2 contained 6 high-speed videos with 800×601 resolution recorded at 500 fps. The 601 in the resolution is the result of one information row on the top of the image. The row contains a time-stamp and other information about the imaging situation. Table 4 summarises the dataset, which consists of videos with dark background containing over 7000 frames of high-speed video material. Sample images of the Dataset 2 videos used in the experiments can be seen in Figure 19. The examples are extracted

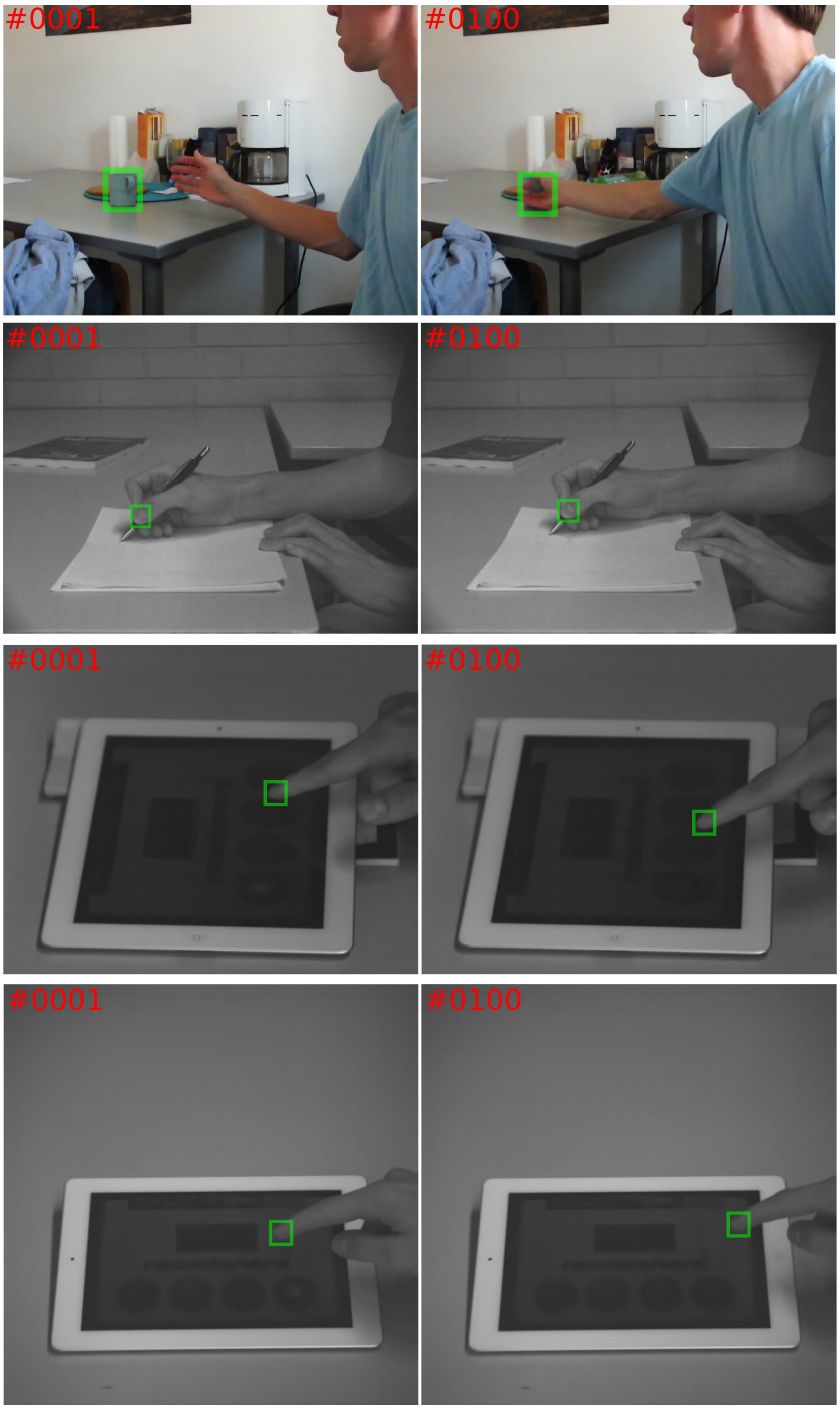


Figure 18. The sample images are from Dataset 1 used in the experiments. The top row images are from the Cup sequence, the second row images from drawing, the third row images from finger2, and the fourth row images from the table2 sequence.

from video sequences at frame 1000 of the respective videos *A, B, C, D, E, and F*. These images illustrate the different end-points of the trajectories in this dataset. The start-point for all the sequences was the same.

Table 4. Video sequences used in Dataset 2.

Video	Resolution	Fps	Target	Posture changes	Frames/Annotations
A	800×601	500	Fingertip	Minor	1407/1407
B	800×601	500	Fingertip	Minor	1319/1319
C	800×601	500	Fingertip	Minor	1246/1246
D	800×601	500	Fingertip	Minor	1208/1208
E	800×601	500	Fingertip	Minor	1271/1271
F	800×601	500	Fingertip	Minor	1086/1086

This dataset was used to test the trackers abilities to cope with a slowly moving object. Videos contained a hand heading towards a monitor with relatively easy dark background but with object movement which was less than one pixel per frame. Filming 1000 frames with 500 fps produces only two seconds of real-time video, and, with 30 fps, which can be considered as a normal speed camera, it would be only 60 frames. The motion between frames in an ideal case should be at least one pixel to some direction so that it would be quantifiable for the trackers. That is not always the case with high-speed videos and can create challenges for the trackers and trajectory analysis.

4.2.3 Ground-Truth Generation

Ground truth is the desired tracking result. Generating the ground-truth is a time-consuming process, and it can be very error prone. It is usually generated manually, but it can also be formed automatically by using some gold-standard method [3], that is, a technique that works nearly flawlessly for the data in question. The ground-truth annotation method described in this section was used for annotating all the Dataset 2 videos and for re-annotating the *drawing* video sequence from Dataset 1. Forming the ground-truth for tracker performance evaluation was done by clicking the desired image region, which was then zoomed in. This was done to make the next step more accurate. After zooming, the fingernail edge line position was annotated as accurately as possible. The accuracy of mouse movement with the selected zoom was 0.06 pixels, but, in a sub-pixel accuracy, it is hard to tell whether the right position in each frame was clicked correctly or not. This is why the accuracy of the ground-truth was assumed to be ±0.5 pixels. In reality, the accuracy of the ground-truth markings was found to be in the range of ±0.2 pixels.

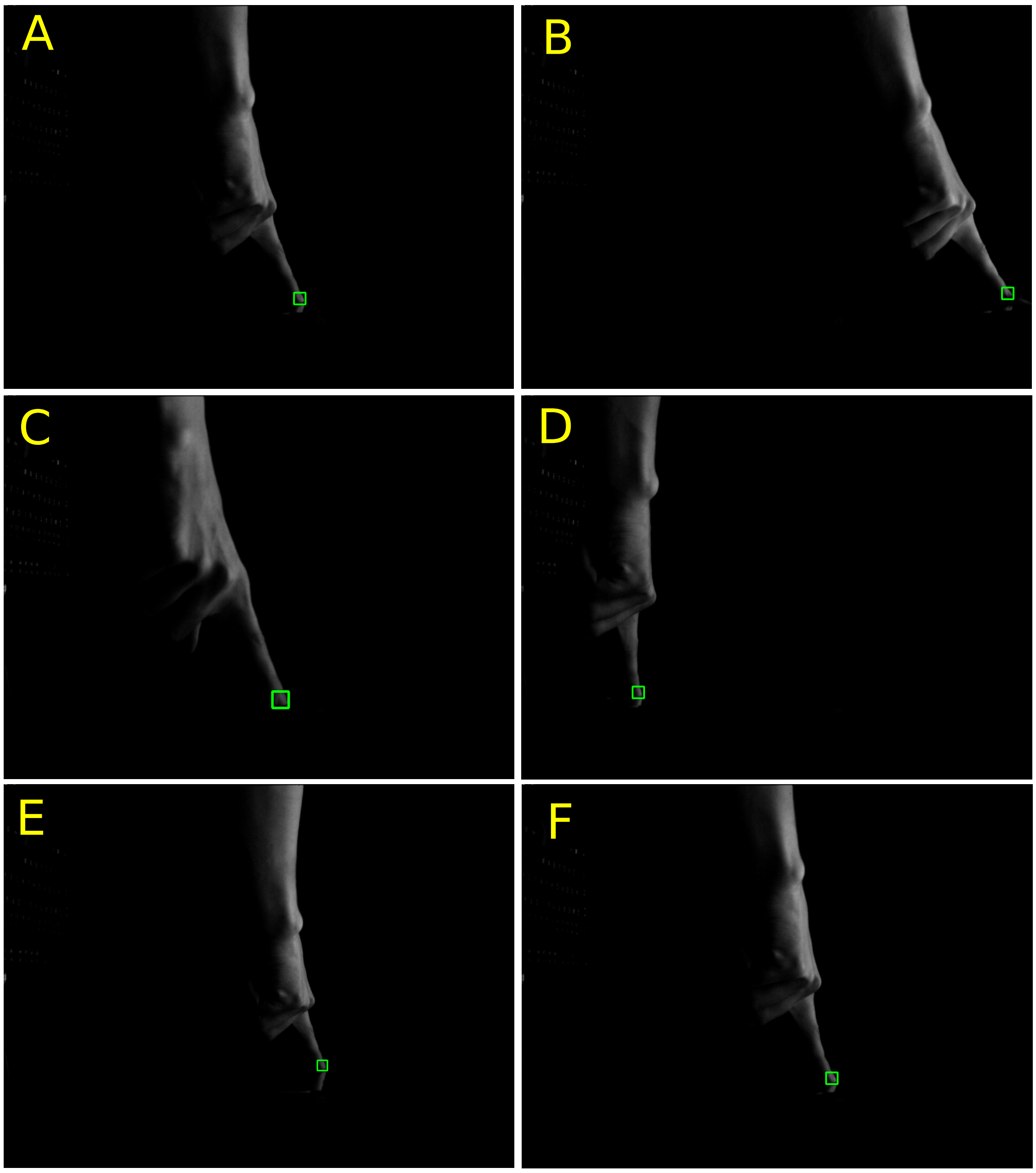


Figure 19. The sample images are from Dataset 2 used in the experiments. The images were all taken from frame number 1000 of respective videos *A, B, C, D, E, and F*. The ground-truth bounding box-can be seen as a green rectangle in the images.

Every 10th frame of the high-speed videos in Dataset 2 and every 5th frame of the Dataset 1 video *drawing* were annotated, and the annotations were interpolated using a spline interpolation method to get the annotations for every frame. This was not done frame by frame because it proved to be harder to see the differences during one frame change and the ground-truth results were worse than with a 10-frame difference, in the sense that there was much more fluctuation in the acceleration and velocity curves when the annotations were made frame by frame. This is mostly explained by the fact that the differences between frames are so small in high-speed video sequences that it is almost impossible to see or pick the position where the object had moved in subsequent frames. Also, the flickering light source added some noise to the image. These changes made the tracking task harder even for the human eye.

4.3 Tracker Evaluation

Tracking a moving object from videos can be done with various tracking methods, but selecting the most accurate and at the same time fast enough tracker for the purposes of this thesis proved to be challenging task. Evaluating trackers for specific sequences is needed when deciding which tracker to use for a specific video. This is because different kinds of trackers usually work best in some particular situations, and a general, all-powerful tracker is yet to be seen.

The performances of the selected trackers were compared against the ground-truth. The comparison was done on position data of the tracked object and its velocity and acceleration, i.e., its different derivatives. The ground-truth was defined as the desired tracking result. Different types of error measures have already been established for tracking evaluation: centre location error, bounding-box area overlap, normalised area overlap or centre location error that takes the object size into account [13, 15, 7, 16, 4]. The most common measure for the accuracy of the trackers is measurement of the distance from the ground-truth centre point to the tracker's centre point. This is known as the centre location error. When success rate or tracking rate measures are used, a frame may be considered correctly tracked if the predicted target centre is within a distance threshold of the ground-truth. Ideas behind the error measures can be seen in Figure 20. The distance measure d demonstrated in Figure 20 (a), explains the centre location error measure. Bounding-box area overlap is demonstrated in Figure 20 (b). Scale changes of the target object, and how they can affect tracking are shown in Figures 20 (a) and (b).

In this research, precision curves and average, mean, and standard deviation of centre

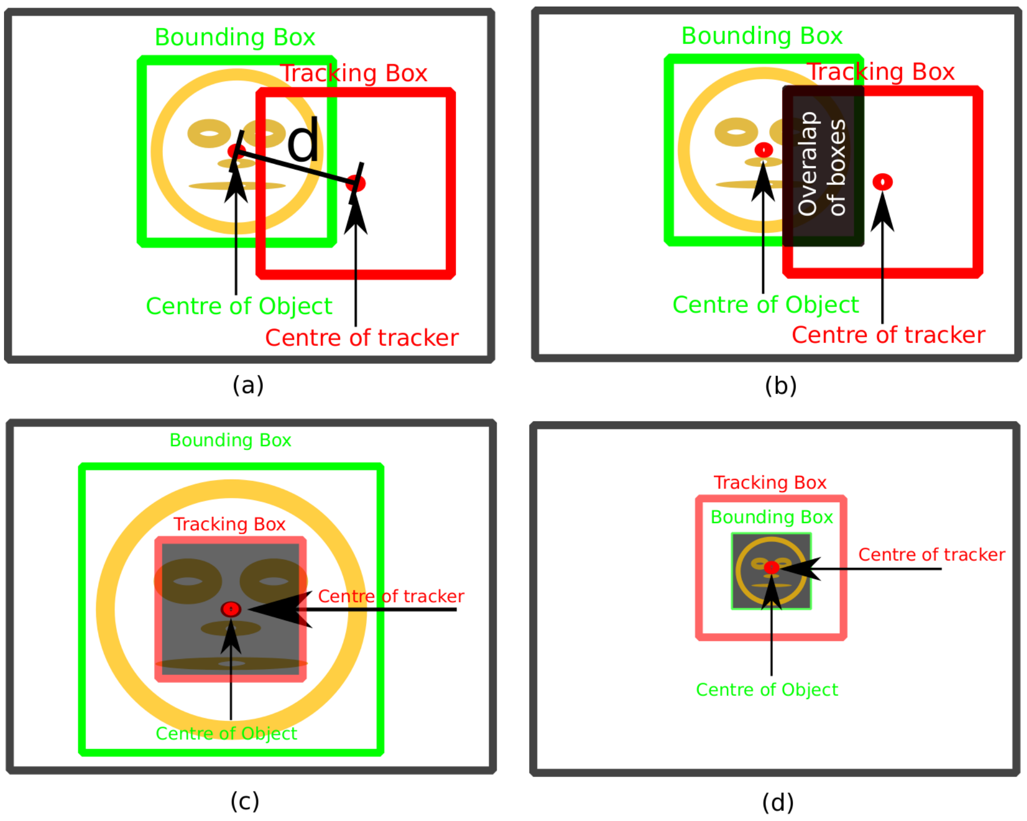


Figure 20. Precision measures: (a) Centre location error; (b) Area overlap; (c) Object upscaling; (d) Object downscaling.

location errors were used as the performance measures. Figure 21, illustrates what one can achieve with precision curves and how different the tracking results shown for them can be. The precision curves are formed with the precision on one axis and the centre location threshold on the other. The centre location threshold starts from 0 and goes up to a set number, in this research either to 10 or 25. This threshold defines the maximum centre location error accepted. On the precision axis, the percentage of correctly tracked frames at a given centre location error point are shown on the scale from zero to one, representing the portion of correctly tracked frames at the given threshold. Precision curves are well suited to situations where one needs to evaluate the accuracy and the tracking performance at the same time. Also, a steep curvature tells that the tracker is relatively accurate, that it is able to follow the tracked object with a minimal centre location error. Recently, precision curves have been employed in several tracking papers [11, 12, 13].

In the evaluation of high-speed videos, the run-time or processing time of the trackers became an issue. Frames per second (fps) measurements for most of the trackers in this research were done by calculating the time needed for tracking while leaving out the image loading, plotting results, or any other activities from the calculation. For some trackers, this was not manageable: they had been implemented using C++, and it would

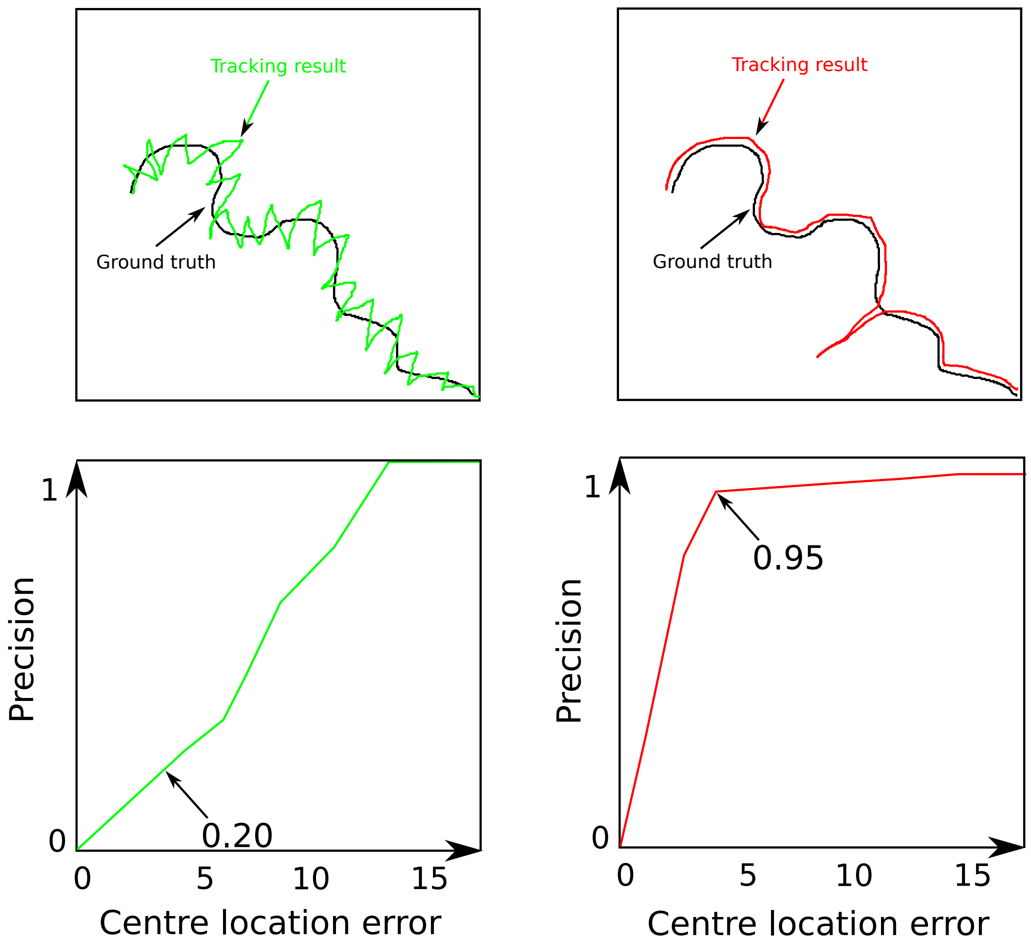


Figure 21. Precision curve example. The top row shows the results of two trackers with the same ground-truth data, and the bottom row shows the precision curves of those two.

have required many changes in the code in order to leave the image loading part outside the time calculations.

During tracking tests, it was noted that, due to different randomisation factors for the tracker initialisation, tracking results can change dramatically between the runs. That is why the results shown in this thesis are averaged by running the trackers three times for each sequence and using the same initialisation values for all the trackers.

4.3.1 Selection of tracker parameters

Most of the trackers were run with their default parameters set by their original authors, but, for those trackers which performed well in the tests, the parameters were further tuned when tests moved to Dataset 2. Explanation and the values used for all the parameters are shown in Table 5. For the C++ implementation of struck, search areas were redefined to speed up the operation for Dataset 2. For the KCF, the Dataset 2 tests were executed by using the gray feature set, which gave faster execution time and also better accuracy. Also, the padding value was decreased to speed up the method.

Table 5. The key parameters of the tracking algorithms.

Method	Important parameters	Used value (range)
CT [18],	*Learning rate*	0.85 (0.7∼0.95)
MIL [27]	*Search window size*	20 (15∼35)
	Max no. of rectangle features	6 (4/6)
HT [23]	*Max scaling factor*	1.0 (-)
IVT [25],	*Number of samples*	400 (-)
LRS [35],	*Affsig*	[4,4,.02,.02,.005,.001] (-)
SRPCA [37]	*Batchsize*	5 (-)
	Forgetting factor	1.0 (0∼1)
	Condenssig	0.01 (0∼1)
KCF [11]	*Kernel*	gaussian (gaussian/linear)
	Kernel Feature	gray / hog (gray/hog)
	cell_size	1 / 4 (1∼10)
	Padding	0.25 / 1.5 (0.25∼2.5)
TLD [14]	*Min_win*	20 (-)
	Patchsize	[17 17] (-)
	Maxbbox	1 (0∼1)
	Update_detector	1 (0/1)
STC [17]	*padding*	1 (0.25∼2.5)
	numer	5 (2∼10)
	rho	0.075 (0.01∼0.5)
RSCM [30]	*numsample*	100 (50∼500)
	affsig	[4,4,.02,.02,.005,.001]
	rho	0.075 (0.01∼0.5)
struck [33]	*searchRadius*	10 (5∼50)
	svmBudgetSize	72 (20∼200)
	feature	haar (haar/raw/histogram)
	kernel	gaussian (gaussian/linear/intersection/chi2)
FCT [20]	*Learning rate*	0.85 (0.7∼0.95)
	Sparse search window size	25 (15∼35)
	Sparse step size	5 (5∼10)
	Dense search window size	10 (5∼15)
	Dense step size	1 (1∼4)
	Max no. of rectangle features	4 (2/4)

4.3.2 Tracking results for Dataset 1

With Dataset 1 the purpose was to evaluate general hand/finger tracking capability of the selected trackers in different environments. From the results shown in Table 6, it can be seen how different sequences affect trackers' performance. Overall, the best performer with this dataset was struck with 96.8% correctly tracked frames, averaged centre location error of 7.4 and standard deviation of errors 4.2 pixels, and average tracking speed of 85.5 frames per second. The second place went to HT with 92.1% correctly tracked frames, and the third place to KCF with 87%. STC was the fastest method with average tracking speed of 474 fps but struggled with almost all the videos in this dataset, managing only 50.6% correctly tracked frames. The second fastest method, KCF, was the third in terms of overall accuracy and achieved 100% tracking rate for four out of six videos. The hardest sequence for the trackers was *touch2*, with which none of the trackers' achieved 100% tracking rate. The difficulty with that sequence is that the appearance of hand in the scene changes a lot and one part of the background, an elastic mattress, can easily confuse trackers when the hand slightly protrudes into the mattress. The easiest sequence for these trackers was the *drawing* video, which is basically the same video as *drawing2* but captured with higher frame rate and containing no colour information.

From the precision curves in Figure 22, the centre location threshold point where the different trackers start to achieve good precision can be seen. For two sequences, *Touch2* and *Cup*, full precision was not achieved with the location error threshold of 25 pixels. In the *drawing* and *tablet2* sequences, full precision was achieved with the centre location error threshold of around 10 pixels for the best methods. For some sequences, the precision curve is steep, and, for some, it is rising up slowly. Steep curves mean that the tracker is accurate with the sequence. Slow rise in the curvature tells that the tracker might have some fluctuation in the position. If 100% tracking result is not met, it indicates that the target was lost.

4.3.3 Tracking results for Dataset 2

After the more general testing of trackers' suitability for hand/finger tracking, Dataset 2 testing was conducted to find the best possible trackers for the purposes of this research. Table 7 shows the results of trackers for Dataset 2. Tracking rate of 100% was met by four of the trackers, struck coming first in overall results with the smallest centre location error of 3.8 and standard deviation of errors 1.2. KCF was the second, with the respective values of 4.7 and 2.4, and STC third with the values of 4.8 and 2.7. Performance-wise, there

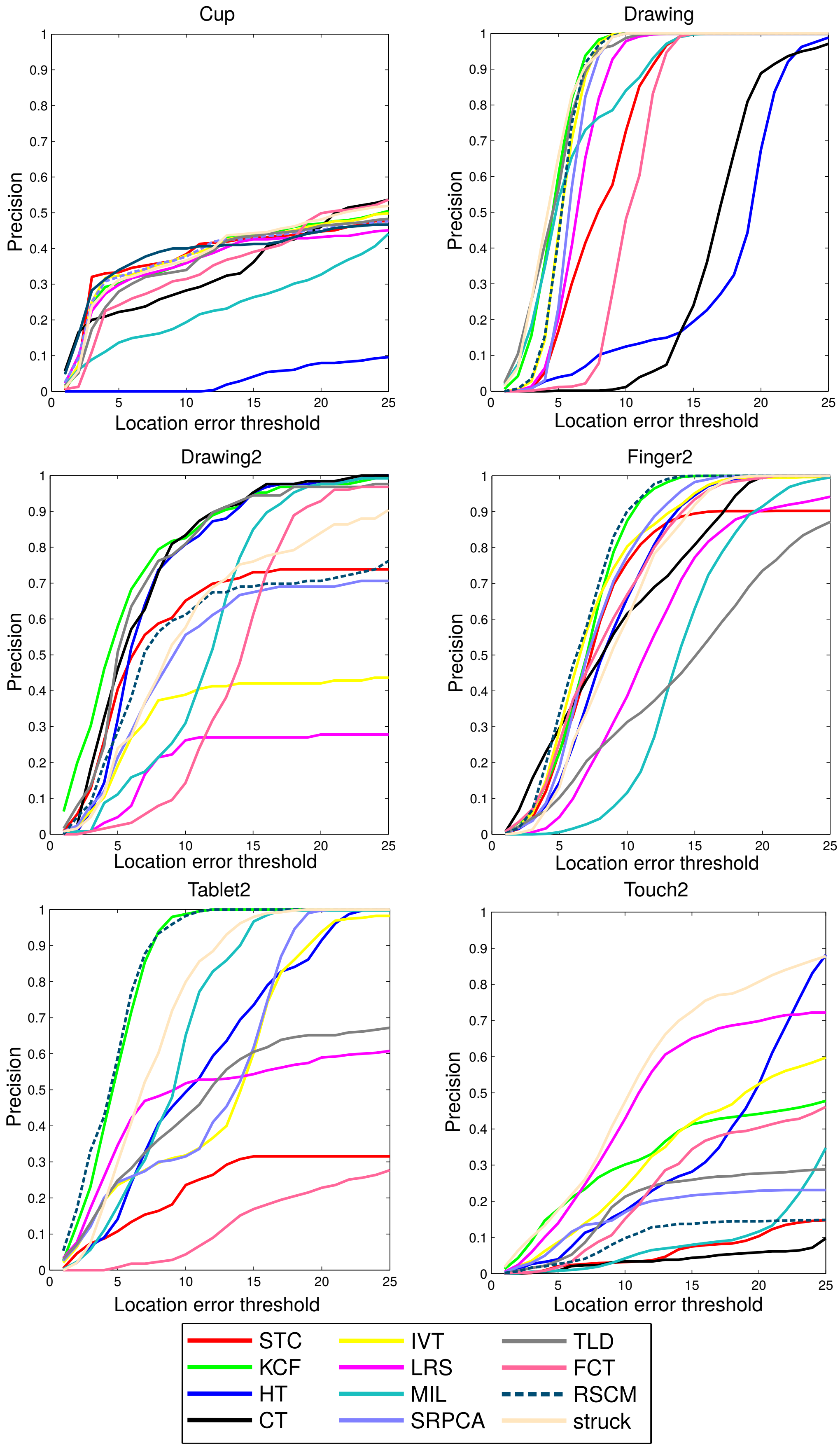


Figure 22. Trackers precision curves on Dataset 1.

Table 6. This table presents tracking results for Dataset 1. Frames per second (top left), percentage of correctly tracked frames (top right), average centre location errors (bottom left), and standard deviations of errors (in parentheses) (bottom right) for each method are shown in the cells. Best results are shown in bold.

	Video						
Method	*cup*	*drawing*	*drawing2*	*finger2*	*tablet2*	*touch2*	*average*
CT [18]	95.6 78.1%	64.0 100%	132 100%	113 100%	107 11.5%	128 25.5%	107 69.2%
	12.8 (12.5)	12.5 (3.2)	7.9 (5.0)	8.9 (5.6)	86.8 (49.8)	80.5 (61.3)	34.9 (22.9)
HT [23]	3.4 100%	3.5 97.8%	4.5 100%	4.4 100%	4.6 100%	3.4 54.5%	4.0 92.1%
	8.4 (**3.2**)	22.4 (5.3)	10.7 (8.4)	7.9 (4.7)	9.0 (5.3)	23.0 (8.7)	13.6 (5.9)
IVT [25]	58.6 25.0%	53.7 100%	58.0 37.1%	60.1 100%	54.3 96.2%	57 41.8%	56.9 66.7%
	96.8 (87.3)	2.1 (**1.3**)	35.3 (24.3)	5.5 (4.0)	15.3 (21.5)	42.7 (42.0)	32.9 (30.1)
LRS [35]	7.5 100%	7.4 100%	6.9 29.6%	6.7 92.5%	6.7 61.5%	6.7 67.3%	7.0 75.1%
	10.1 (6.7)	3.2 (1.7)	51.1 (35.2)	15.1 (9.7)	33.2 (38.4)	47.1 (59.0)	26.6 (25.1)
MIL [27]	0.4 81.3%	0.4 100%	0.4 100%	0.4 100%	0.4 100%	0.4 27.3%	0.4 84.8%
	12.7 (10.8)	8.4 (3.2)	11.8 (5.4)	11.4 (3.4)	8.4 (4.9)	54.3 (42.4)	17.8 (11.7)
SRPCA [37]	8.8 25.0%	10.0 100%	9.3 63%	12.3 100%	12.2 100%	9.0 20%	10.3 68.0%
	36.6 (13.8)	2.3 (1.4)	26.5 (26.6)	5.3 (3.8)	10.5 (6.1)	95.1 (62.2)	29.4 (19.0)
TLD [14]	13.0 34.4%	8.0 100%	11.1 100%	9.7 88.7%	9.5 65.4%	10 25.5%	10.2 69.0%
	37.9 (20.3)	4.5 (2.3)	6.3 (5.3)	15.8 (11.6)	27.6 (34.6)	68.2 (58.0)	26.7 (22.0)
FCT [20]	97.8 78.1%	72.7 100%	107 100%	76.1 100%	135 23.1%	96 45.5%	97.3 74.4%
	16.8 (14.8)	15.0 (2.3)	13.9 (5.5)	9.5 (4.4)	65.5 (39.8)	26.5 (15.0)	24.5 (13.7)
KCF [11]	153 75.0%	239 100%	235 100%	245 100%	242 100%	86 47.3%	200 87.0%
	26.0 (24.7)	3.5 (1.7)	6.3 (5.0)	5.6 (2.6)	3.8 (**2.0**)	52.2 (48.6)	16.2 (14.1)
RSCM [30]	2.2 87.5%	2.6 100%	2.1 70.4%	2.4 100%	2.3 100%	1.8 14.5%	2.2 78.7%
	15.1 (9.4)	**2.0** (**1.3**)	19.2 (23.2)	**3.8** (2.3)	**3.3** (2.2)	111 (65.0)	25.8 (17.2)
STC [17]	316 3.1%	873 100%	816 66.7%	389 88.7%	257 30.8%	194 14.5%	**474** 50.6%
	114 (69.1)	4.8 (2.7)	26.8 (30.5)	20.5 (44.0)	70.7 (63.1)	130 (71.8)	61.2 (46.9)
struck [33]	42.9 87.5%	63.7 100%	68 100%	75.2 100%	51.8 100%	40 92.7%	85.5 **96.7%**
	15.9 (8.5)	2.8 (**1.3**)	**3.0** (**1.2**)	4.8 (**1.6**)	5.3 (4.0)	**12.2** (**8.4**)	**7.4** (**4.2**)

were two methods that take advantage of FFT operations in their tracking algorithms. They were ahead of others by achieving more than real-time performance for the given sequences. STC was the fastest tracker with the average frame rate of 1649 and KCF the second fastest with the average of 887 fps. Drifting from the finger tip to the knuckles at the start of the tracking sequences in videos gave CT and FCT 0% correctly tracked frames as the result. Nevertheless, they were still within reasonable centre location errors and standard deviations as one can see from the results in Table 7. Trackers with major changes in either centre location errors or standard deviation errors usually drift off completely from the finger tip and start chasing some background noise in the image. In the case of CT and FCT, the drift from the finger-tip could have been corrected by carefully adjusting the initial bounding box, but the sensitivity to the initial bounding-box would make those two methods still fail in some videos.

The highest fps was measured for STC, which had over 1600 fps for some video sequences and for KCF which had the peak performance of around 1000 fps. Both achieved processing speeds well over the videos' frame rate, which was for these videos 500 fps. For KCF, grey-scale features and smaller than the default padding and the adaption of the

Table 7. This table presents tracking results for Dataset 2. Frames per second (top left), percentage of correctly tracked frames (top right), average centre location errors (bottom left), and standard deviations of errors (in parentheses) (bottom right) for each method are shown in the cells. Best results are shown in bold.

	Video						
Method	*A*	*B*	*C*	*D*	*E*	*F*	*average*
CT [18]	117 0.0%	77.3 0.0%	66.6 52.4%	84.1 0.0%	89.0 2.4%	72.9 0.0%	84.5 9.1%
	50.1 (6.6)	51.1 (6.6)	33.8 (5.9)	65.3 (11.4)	38.9 (5.2)	51.2 (7.1)	48.4 (7.1)
HT [23]	4.6 100%	4.7 100%	4.8 100%	4.6 100.0%	4.6 100%	4.5 100%	4.6 100%
	4.0 (2.0)	4.4 (3.2)	5.5 (3.0)	4.7 (3.0)	5.4 (2.7)	6.6 (3.75)	5.1 (2.9)
IVT [25]	64.1 65%	62.1 61.1%	52.6 100%	49.2 53.3%	53.4 24.4%	48.0 15.7%	54.9 53.3%
	41.5 (52.0)	41.3 (53.2)	9.2 (3.9)	30.3 (19.7)	361 (234)	448 (236)	155 (99.8)
LRS [35]	8.4 18.6%	8.6 12.2%	8.4 19.4%	7.8 20.0%	7.6 2.4%	7.8 15.7%	8.1 14.7%
	379 (242)	440 (281)	403 (223)	320 (170.1)	317 (180)	455 (227)	386 (221)
MIL [27]	0.4 100%	0.4 100%	0.4 100%	0.4 27.5%	0.4 100%	0.4 100%	0.4 87.9%
	4.9 (2.5)	6.0 (2.6)	4.6 (1.8)	38.0 (23.4)	3.3 (1.5)	4.1 (2.0)	10.1 (5.6)
SRPCA [37]	11.9 27.1%	9.2 100%	11.6 100%	11.2 100%	11.8 24.4%	11.1 68.5%	11.1 70%
	340 (229)	11.1 (10.3)	16.3 (7.0)	17.3 (11.5)	366 (236)	24.7 (15.7)	129 (84.9)
TLD [14]	21.6 100%	8.6 30.5%	12.2 29.8%	13.5 48.3%	18.1 100%	10.9 16.7%	14.1 54.2%
	4.9 (1.9)	115 (75.8)	139 (89.1)	23.7 (17.0)	10.2 (8.3)	110 (49.9)	67.2 (40.3)
FCT [20]	123 0.0%	73.6 0.0%	114 0.0%	102 0.0%	114 0.0%	97.7 0.0%	104 0.0%
	53.4 (8.4)	58.2 (8.35)	67.0 (15.0)	91.8 (23.3)	83.9 (22.1)	53.1 (5.5)	67.9 (13.8)
KCF [11]	1107 100%	813 100%	1147 100%	1131 100%	1240 100%	810 100%	887 100%
	4.2 (1.2)	6.5 (4.4)	6.3 (3.7)	**4.0** (1.9)	**2.4** (1.1)	5.0 (1.9)	4.7 (2.4)
RSCM [30]	3.0 0.0%	2.8 34.4%	1.7 20.2%	2.6 26.7%	2.6 21.3%	2.3 25.9%	2.5 21.4%
	334 (158)	168 (125)	356 (204)	140 (96.1)	258 (167)	69.8 (49.3)	221 (133)
STC [17]	1700 100%	1682 100%	1664 100%	1625 100%	1605 100%	1617 100%	**1649** 100%
	2.8 (1.1)	6.16 (5.10)	6.5 (3.3)	6.3 (3.3)	3.3 (1.7)	**3.7** (1.9)	4.8 (2.7)
struck [33]	129 100%	115 100%	118 100%	96.7 100%	110 100%	110 100%	113 100%
	3.6 (**0.8**)	**4.65** (**1.31**)	**3.9** (**1.5**)	4.1 (**1.5**)	2.5 (**1.0**)	4.1 (**1.0**)	**3.8** (**1.2**)

FFTW (Fastest Fourier Transform in the West) [70] optimisation from STC's source code to optimise MATLAB's Fast Fourier transform handling resulted in higher frame-rates. That gave KCF a boost of around 300 fps for high-speed video sequences of Dataset 2.

The precision curves in Figure 23 for the high-speed video sequence dataset show that, compared to the other dataset, the best trackers achieve 100% tracking rate with low threshold values. Note that the location error threshold for these precision curves changed from the maximum of 25 to 10 pixels because most of the trackers achieved high accuracy with small thresholds in these sequences. For all the sequences, the precision of 100% was achieved with the given location error threshold at least for some trackers. KCF, STC, and struck achieved 100% precision with the smallest centre location error for most of the test sequences in this high-speed video sequence dataset.

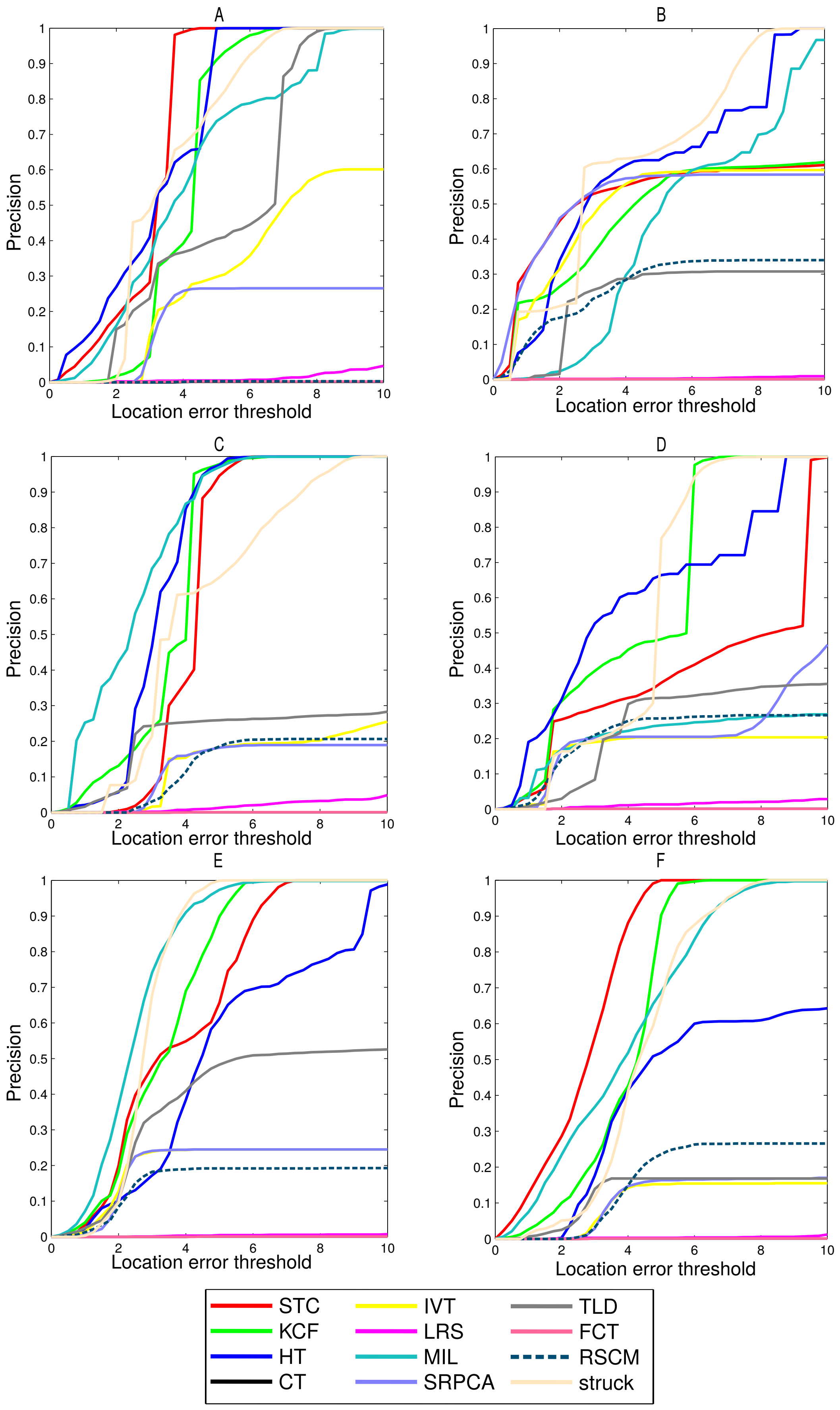


Figure 23. Trackers' precision curves on Dataset 2.

4.3.4 Summary of the Tracking Results

The decision for the selection of the best trackers was made based on the speed and precision. Precision differences in Dataset 2 were quite small, and tracking speed was more important, considering that the purpose here was to track high-speed videos. The algorithm speed for KCF and STC, both of which are based on FFT calculations, was quite high when looking at the results of Dataset 1 and Dataset 2. In Dataset 1, KCF had the average fps of 200 and, in Dataset 2, it was 887. STC had the average fps of 474 in Dataset 1 and 1649 in Dataset 2. The overall winners in Dataset 2 were KCF and STC, both of which performed better than real-time requirement with the high-speed videos, and both tracked all frames of Dataset 2 videos correctly with a given threshold error. KCF was selected for further study in this case because it had also done well in various online benchmark trials like the visual object tracking challenge in 2014 [4]. Also, the general level performance and adaption to different situations, as evidenced with Dataset 1, showed that the KCF algorithm can be used for a broader range of situations.

The fps measure used in the experiments was calculated without including the image loading times in the calculations for all but two of the methods, struck and HT, both of which are written in C++. The time spent on image loading can be considered to be minimal when compared to the tracking procedures, but the error in frames per second measures is hard to calculate because of the different implementations and compiling procedures used. The image load time with MATLAB was on average 2.3 milliseconds. This is not much, but when compared to the frame processing time of KCF algorithm, 1.1 milliseconds, it seems large. The KCF algorithm processing speed drops from an average of 887 fps to 294 fps when image load time is added to the fps measurement. The image load time of 2.3 milliseconds would also mean that the theoretical maximum processing speed is 434 fps if the actual tracking process does not take any time at all.

4.4 Filtering Tests

Accuracy of selected filters was compared against ground-truth measures. The MATLAB implementations of the filters introduced in Section 3.1 were used. Results with different window sizes were compared to the ground-truth position, velocity and acceleration curves. For filtering evaluation, the struck, KCF, and STC trackers were included based on their good tracking results: . Filtering tests were done using Dataset 2. The data provided by the trackers was filtered with eight different filters and 90 filtering-window sizes.

Window size in filtering means the number of samples included for one point-filtering.

The filtering-window size effect for mean location error and mean velocity errors for Dataset 2 video A, tracked with KCF, is demonstrated in Tables 8 and 9. From the results, it can be seen that robust versions of LOWESS and LOESS start to produce large errors with larger filtering-window sizes. This leads to leaving out the robust versions of LOESS and LOWESS from the rest of the experiments with filtering. Also, from the results, it can be seen that some filtering methods produced the best results with the filtering-window size 17, so further test were needed to find the best possible filtering-window sizes for these filtering methods.

Table 8. Filtering window size effect on mean position error on Dataset 2 video A, tracked with KCF. 5.7068. Best results shown in bold and worse than unfiltered results are shown in italic.

	Window size in frames							
method	3	5	7	9	11	13	15	17
MA	5.7017	5.7002	5.6996	**5.6995**	5.6997	5.7002	5.7009	5.7019
LOWESS	5.7068	5.7019	5.7007	5.7	5.6996	**5.6995**	**5.6995**	5.6996
LOESS	5.7068	5.7068	5.7032	5.7021	5.7013	5.7007	5.7002	5.6998
S-G	5.7068	5.7032	5.7019	5.7008	5.7001	5.6997	**5.6994**	**5.6991**
rLOWESS	5.7068	*6.0333*	*25.267*	*34.255*	*34.393*	*34.263*	*34.2397*	*34.0042*
rLOESS	5.7068	5.7068	*25.320*	*27.780*	*34.391*	*34.256*	*34.1687*	*33.7751*

Table 9. Filtering window size effect on mean velocity error on Dataset 2 video A, tracked with KCF. Unfiltered value was 0.2648. Best results shown in bold and worse than unfiltered results are shown in italic.

	Window size in frames							
method	3	5	7	9	11	13	15	17
MA	0.0977	0.0564	0.0461	0.0387	0.0375	0.0345	**0.0336**	0.034
LOWESS	0.2648	0.0979	0.0601	0.0436	0.0359	**0.0328**	**0.0312**	**0.0305**
LOESS	0.2648	0.2648	0.1178	0.0862	0.0644	0.0515	0.0434	0.0381
S-G	0.2648	0.1179	0.079	0.0608	0.0503	0.0444	0.0397	0.038
rLOWESS	0.2648	0.2306	*0.5585*	*0.5276*	*0.6023*	*0.5948*	*0.6072*	*0.5483*
rLOESS	0.2648	0.2648	*0.4938*	*0.4492*	*0.4863*	*0.4821*	*0.5025*	*0.4397*

The results in Table 10 were calculated by averaging the results from all Dataset 2 trajectories tracked with KCF tracker and searching for the best possible filtering-window size in the range of 1 to 90 data points (frames). Filtering with Unscented Kalman Smoother (UKS) and TVD is included for comparison. UKS was selected to represent Kalman smoother algorithms since Extended Kalman Smoother and UKS produced similar results. Unscented Kalman smoother (UKS) results were obtained by making Kalman parameter optimisations by hand, no automatic parameter-search algorithms were used. For TVD, optimal λ-parameter search was conducted before running the filter. Robust versions of LOESS and LOWESS were left out because they did not work well with this large filtering window.

Table 10. Minimal mean and standard deviations of Position Errors (PE), Velocity Errors (VE), and Acceleration Errors (AE) with different filtering methods. In parentheses is the filtering windows size which gave the best result for the filter. Best results shown in bold.

Error	Moving Average	LOWESS	LOESS	Savitzky-Golay	TVD	UKS	unfiltered
Mean PE	4.9384 (3)	4.9383 (6)	**4.9364** (34)	**4.9364** (27)	4.9509	4.9378	4.9410
Mean VE	0.0352 (17)	0.0336 (23)	**0.0331** (43)	0.0339 (37)	0.1569	0.0338	0.1595
Mean AE	0.0091 (83)	**0.0075** (29)	0.0076 (44)	0.0093 (85)	0.2262	0.0076	0.2298
std PE	1.3826 (6)	1.3824 (10)	1.3781 (48)	1.3782 (37)	1.3984	**1.3776**	1.3969
std VE	0.0491 (15)	0.0468 (23)	**0.0456** (43)	0.0467 (35)	0.2615	0.0473	0.2654
std AE	0.0193 (86)	0.0163 (27)	**0.0161** (42)	0.0188 (85)	0.4504	0.0164	0.4542

The effect of using filtered position-data as a base can be seen in Table 11, where comparison is made between unfiltered position-data and optimally filtered position-data while filtering their derivatives. The results of Tables 10 and 11 can be directly compared, as the same position-data was used for both.

Table 11. Minimal mean and standard deviations of Velocity Errors (VE), and Acceleration Errors (AE) with different filtering methods. In parentheses after the name of the filter if applicable is the filtering window size applied to position data, if it is missing then the filtering for position data was not done for position data prior to filtering derivatives of it. Best results shown in bold and in parentheses is the filtering windows size which gave the best result for the filter.

Error	MA	MA (3)	LOWESS	LOWESS (6)	LOESS	LOESS (34)	S-G	S-G (27)
Mean VE	0.0434 (19)	0.0341 (13)	0.0424 (24)	0.0336 (22)	0.0425 (50)	**0.0332** (36)	0.0433 (39)	0.0333 (25)
Mean AE	0.0093 (83)	0.0084 (41)	0.0077 (32)	0.0076 (28)	0.0079 (60)	**0.0075** (27)	0.0097 (85)	0.0076 (29)
Std VE	0.0601 (19)	0.0476 (15)	0.0588 (26)	0.047 (22)	0.0585 (50)	0.0458 (34)	0.0595 (37)	**0.0456** (17)
Std AE	0.0193 (83)	0.0178 (27)	0.0164 (30)	0.0163 (26)	0.0163 (46)	**0.0161** (36)	0.0191 (85)	0.0162 (23)

For the three best tracking methods' position data, optimal filtering windows were searched by minimising the different errors, i.e., the position, velocity, and acceleration error. Those varied from three points which is the size of the minimal filtering-window to 86 points, when 90 was the maximum filtering-window size for the tests committed. Optimal filtering-window sizes varied from 3 to 34 in the case of position error minimisation. For velocity error minimisation the size variation was from 17 to 43. Optimal filtering-window sizes to minimise the acceleration errors were from 29 to 85. Changes in the errors of position, velocity and acceleration relative to filtering-window sizes are illustrated in Figure 24. The results in the figure are calculated, with KCF tracker, by using average errors of all the Dataset 2 videos.

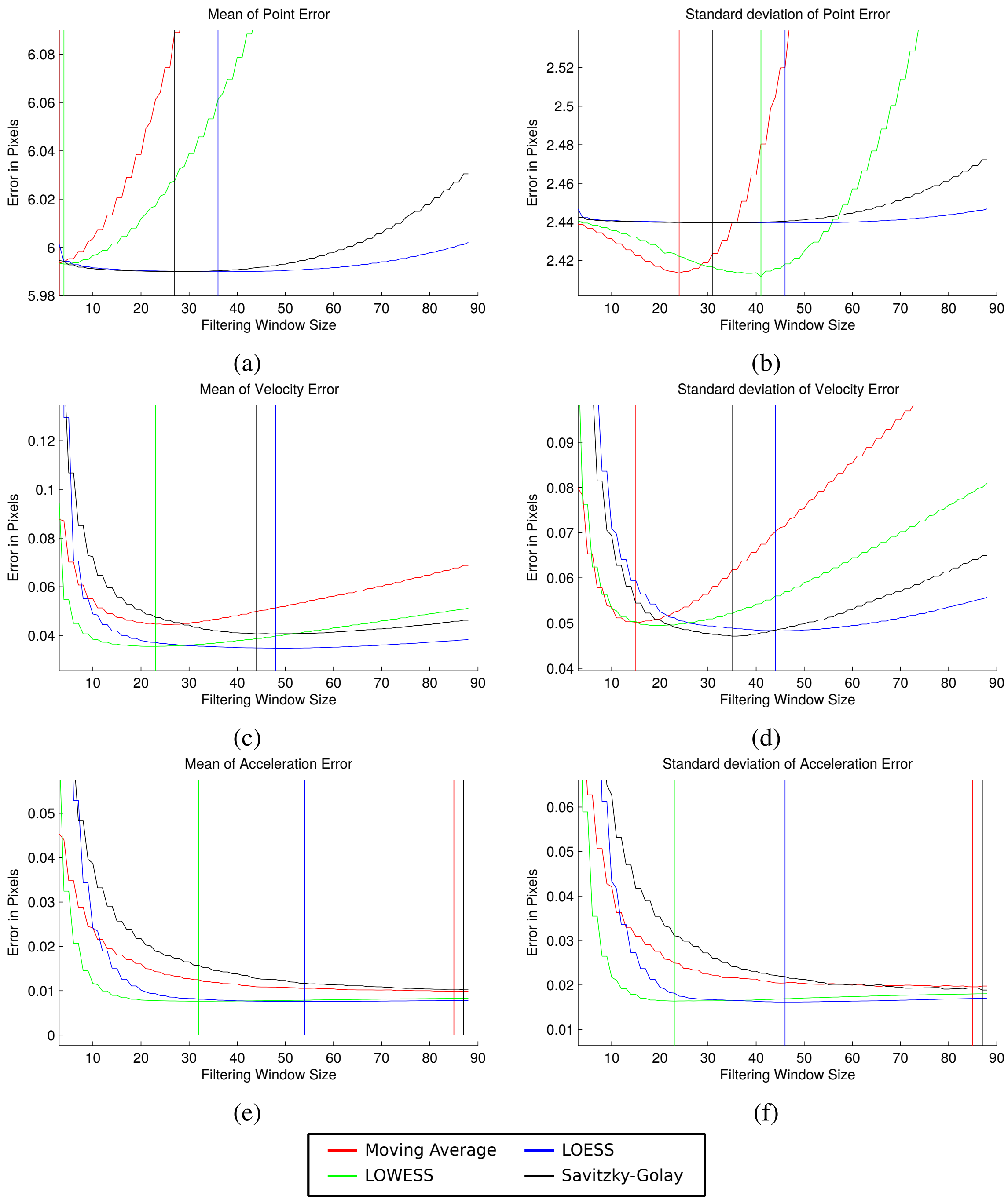


Figure 24. Filtering-window sizes' effect on means and standard deviations of point error, velocity error and acceleration errors. The location of minimum error for each of the methods is indicated with vertical line.

The filtering effects of three different window sizes on velocity curves can be seen in Figure 25. The velocity data comes from a tracked hand movement sequence in Dataset 2 video *C*. The tracker used was KCF. Figure 25 shows how different window sizes affect the velocity curves. The LOESS curves fluctuate the least with these window sizes. Also, note that the curves are taken from the same point of the data but arranged to make it easier to see the differences caused by different amounts of filtering applied. Therefore, the curves are vertically shifted according to the filtering-window size used. The filtering results for position, velocity, and acceleration curves generated by LOESS filtering, with window size of 40 data points with Dataset 2 videos *A, B, C, D, E*, and *F*, are shown in Figure 26.

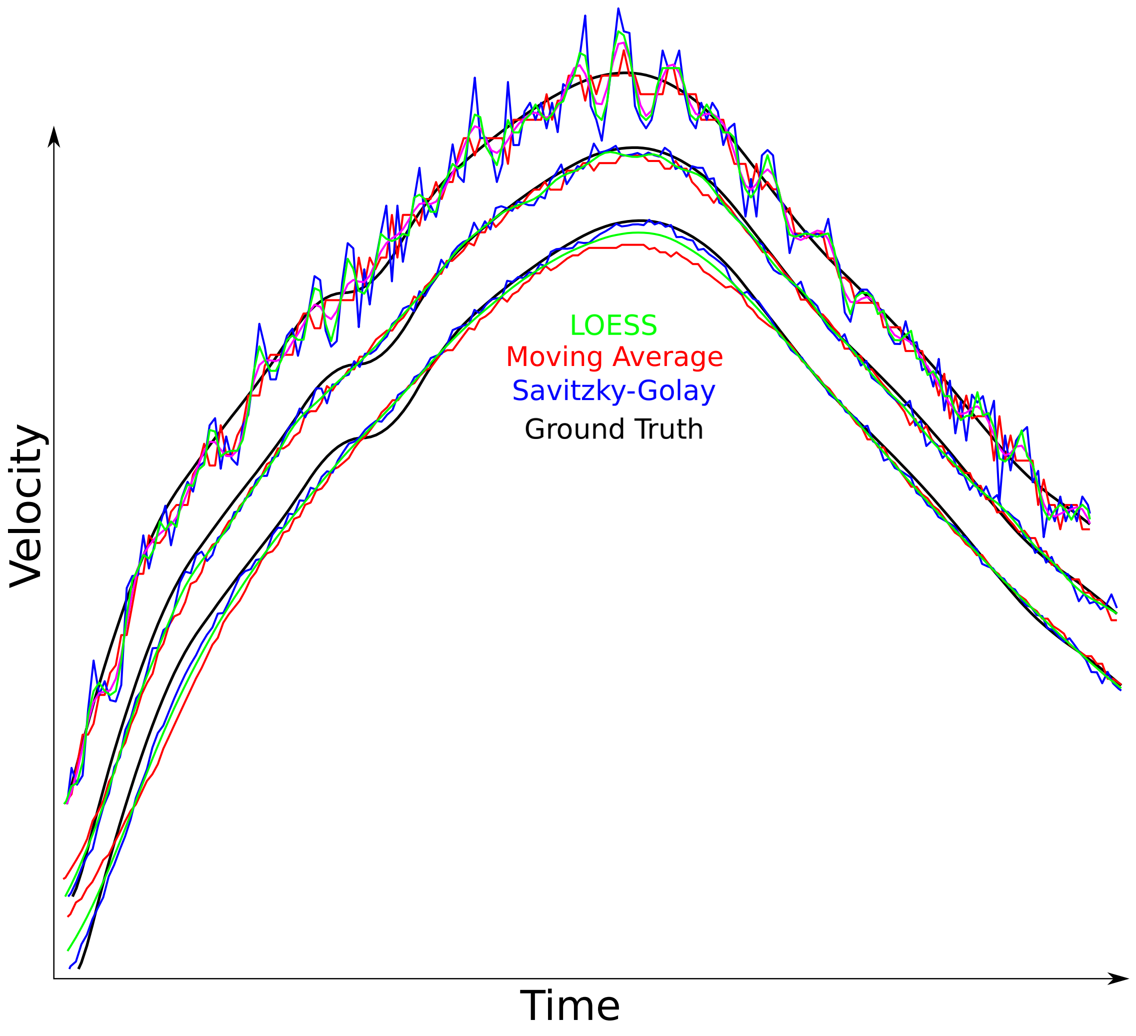


Figure 25. Filtering results of the same velocity curve with three different window sizes. The curve on top has a window size of 7 frames, the curve in the middle has a window size of 23 frames, and the curve on the bottom has a window size of 41 frames.

From the results, it is obvious that different window sizes were optimal for each deriva-

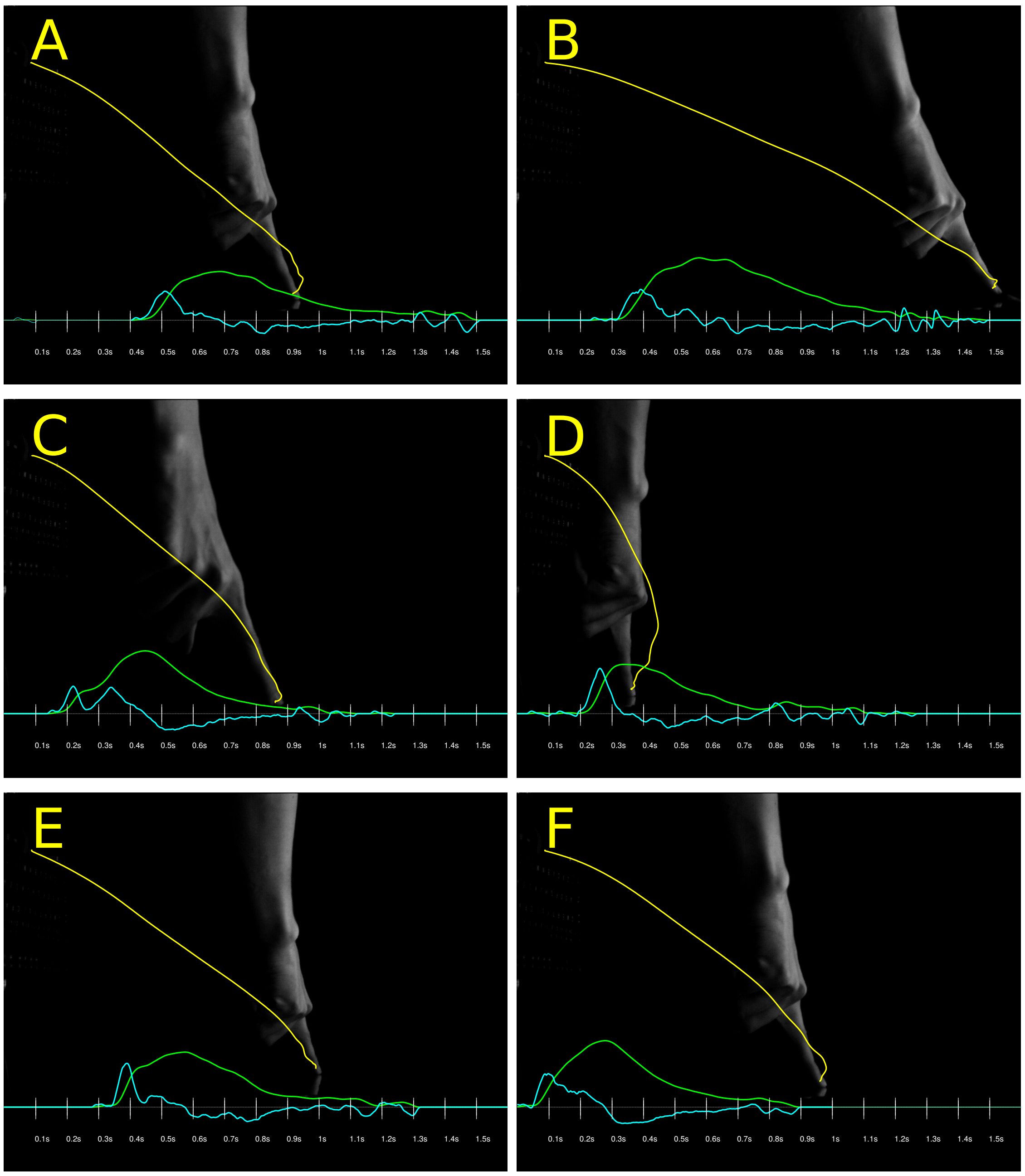


Figure 26. Filtering results with LOESS when using a 40 data points window size. Trajectory is shown in yellow, velocity in green, and acceleration in cyan.

tive of the position. The velocity and acceleration curves needed larger window sizes to get better results than the position feature. LOESS filtering was the most stable in this measure, with optimal filtering results from the filtering-window range of 34 to 48. The problem with a large window size is that the position starts to drift off from ground-truth. Therefore, filtering was performed after each stage of feature extraction. The position was filtered with the average result of position feature optima. Velocities were calculated from the filtered position results and then filtered again with smaller window sizes. For the acceleration, again the same method was used, in other words, it was calculated using the filtered velocity and filtered again using a smaller window size. From the results of filtering it can be seen that most filtering methods gain more out of filtering when the derivatives are filtered separately. This applies to moving average, LOWESS and Savitzky-Golay filtering methods, where the gains are reasonable. In the case of LOESS, the gains were minimal or the results were almost the same.

4.5 Camera Calibration Tests

Camera calibration was done using 10 or 26 images taken from a fixed camera position and A4-sized calibration pattern with a calibration grid where each block was the size of 1×1 inch. The calibration pattern board was held in different angles by a person, during which video was taken using the high-speed camera from the scene so that it was possible to image the calibration pattern from many angles. For the calibration procedure, the toolbox available in [71] was used. The toolbox was constructed with the help of the references available in the toolbox references website [72].

The calibration results shown in Table 12 were obtained using either 10 or 26 different images from the calibration video. The same video sequence is used for both, but fewer images were used from the set of available samples in carrying out 10 image calibration. To make the reading of the results of the camera calibration easier, the following parameters were used: focal Length fc, principal point cc, skew $alpha_c$, distortion kc, and pixel error err. The $alpha_c$ parameter stayed the same 0.00 for each calibration round, meaning that the angle of a pixel was 90 degrees. Distortion kc is a matrix of zeros after initialisation but gets values after optimisation. The differences in the uncertainties reveal that with 26 images the calibration results were more accurate.

The results show that the calibration of the camera was not yet really accurate with 10 images taken from high-speed video. After calibration with 26 images, the uncertainty values approximately halved and the principal point got closer to the centre point. It would

Table 12. The calibration results.

Parameter	Images	Value
fc initial	10	$[2011.48,\ 2011.48]$
cc initial	10	$[399.50,\ 300.00]$
fc optimised	10	$[2038.17,\ 2035.90]\ \pm[44.11,\ 47.83]$
cc optimised	10	$[487.54,\ 237.41]\ \pm[67.86,\ 54.08]$
kc optimised	10	$[-0.128,\ 3.46,\ 0.004,\ 0.012,\ 0.0]$ $\pm[0.213,\ 4.197,\ 0.011,\ 0.013,\ 0.0]$
err optimised	10	$[0.767,\ 0.744]$
fc initial	26	$[1982.92,\ 1982.92]$
cc initial	26	$[399.50,\ 300.00]$
fc optimised	26	$[1999.89,\ 1997.94]\ \pm[17.88,\ 17.88]$
cc optimised	26	$[394.07,\ 333.45]\ \pm[25.53,\ 28.19]$
kc optimised	26	$[-0.023,\ 0.79,\ 0.007,\ 0.003,\ 0.0]$ $\pm[0.0621,\ 1.324,\ 0.005,\ 0.004,\ 0.0]$
err optimised	26	$[0.269,\ 0.192]$

be there if calculating with just pixel values, that is $[400.0, 300.5]$, when the resolution of 800x601 was used. Figure 27 shows from where the actual calibration images were taken in relation to the real world. The extrinsic camera illustration was calculated using the calibration set with 26 images.

The results of camera calibration were used to calculate real-world measurements for the trajectories computed from Dataset 2 videos. The results of speed and acceleration measurements are shown in Appendix 1. The speed and acceleration measurements are shown as a function of frames and as a function of distance to the touchscreen. The distance was measured only on horizontal axis as the distance from the fingertip to the screen surface. The movement starts from a start-point and moves towards an end-point which is the screen surface (0-point). Depth in the sequences was assumed to be constant.

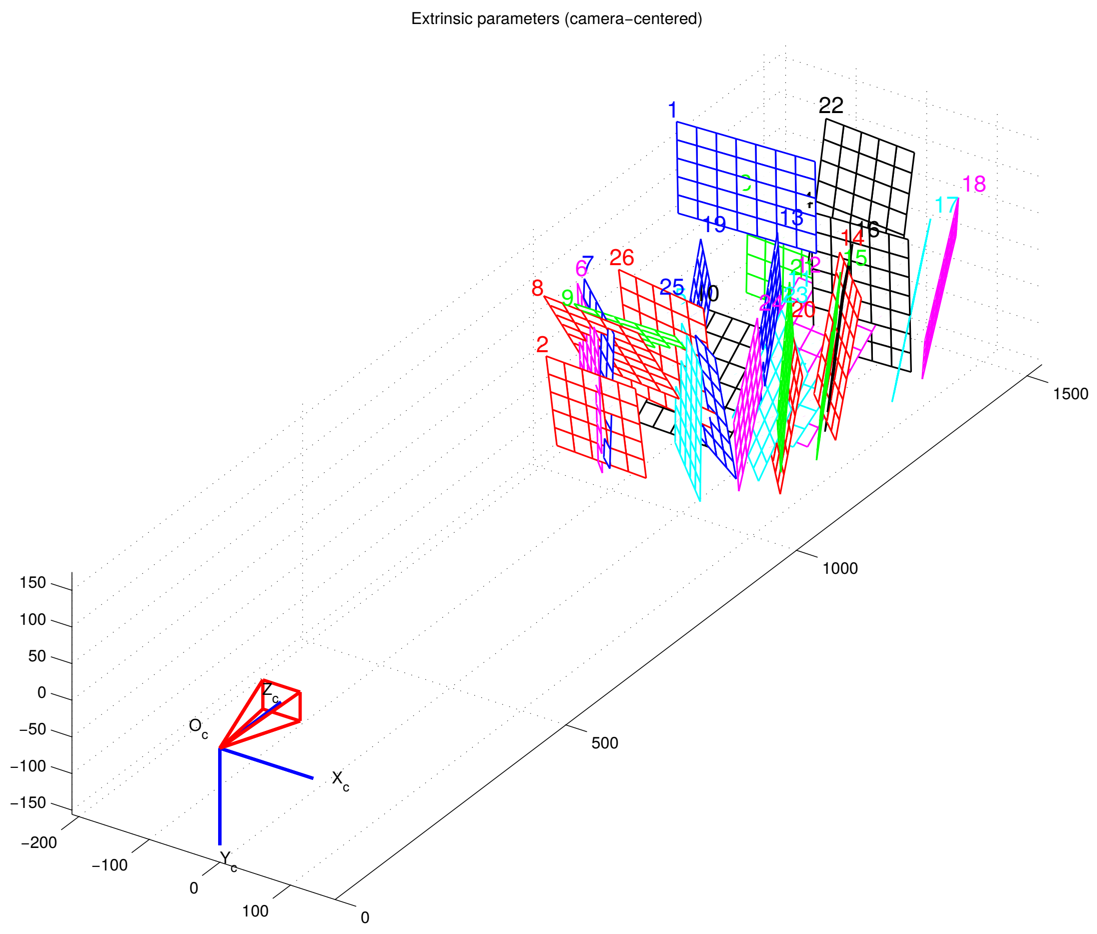


Figure 27. The calibration patterns usage during calibration.

5 DISCUSSION

In this chapter, the results and information gained during this work are discussed. The discussion starts with general notes that were made during the work conducted and then moving to result analysis. Section 5.2 gives ideas for future research on this topic and on how the results from this work could be applied there.

In the object-tracking field, the development of algorithms has seen major improvements during the last 5 years. Trackers are getting more robust and more accurate than before. Still, there are problems with targets that get occluded by some other object or subjected to some dramatic illumination changes in the scene. The overall results of trackers against online benchmarks are quite impressive, and it seems that there are new, faster and better tracking methods coming out frequently. The recent developments in object tracking are shown in various online object tracking challenges such as VOT 2013 and VOT 2014 [16, 4]. These results show that recent advances are significant and that the winners of the challenge in 2013 were only reaching the average results level of the 2014 challenge.

5.1 Results

Tracking is still an unsolved problem for difficult cases like handling occlusions and complicated backgrounds. Also, easier cases where the target is barely moving between the frames proved to be difficult for some trackers trying to predict the movement of the object between frames; when the object does not move between the frames, the tracker looses the target because of these predictions. This is shown, in the experiment part of this work, in high-speed videos with a black background.

Tracking results for Dataset 1 showed that there was no perfect tracker for more general datasets with various scenes. The best performer in this dataset, struck, had 96.7% accuracy on average for all the videos. This required tracking four out of six videos with 100% accuracy, the accuracy of 87.5% for the *cup* video and of 92.7% for the *touch2* video. The *cup* video in this dataset proved to be one of the most challenging videos mostly because, in the start of the video, the cup in the video is almost completely occluded by hand. Dataset 1 showed that the robust tracking methods were not robust for this dataset. This could also have been due to the initialisation parameters used with the IVT, LRS, SRPCA, and RSCM tracking methods because the number of samples used in the experiments was limited to 400 for IVT, LRS, and SRPCA and to 100 with RSCM. This meant faster operation but limited amount of positive and negative samples for the tracking methods to use.

Tracking results for Dataset 2 showed that trackers might still have problems with videos which seem to have a relatively easy dark background and no occlusions or lighting changes. Yet, 4 tracking algorithms, HT, KCF, STC, and struck, achieved a 100% tracking rate for Dataset 2. Parameter tuning for other trackers might have helped the situation, but, as there was already a nice selection of trackers that performed well, the parameter tuning was not done for the others.

Two tracking methods, STC and KCF, used in this work fulfilled the requirements for real-time processing even for high-frame-rate videos. The availability of trackers which satisfy the real-time requirement also for high-speed videos allows instant evaluation of hand movement trajectories if there is enough memory bandwidth to get the data. The high-speed videos presented challenges in the trackers' accuracy when evaluating their performance with different derivatives of position, velocity and acceleration. The question of how to make the ground-truth as accurate as possible for high-speed videos came up. Higher resolutions and precise markers for tracked positions would help the manual ground-truth annotating process. Without high resolution footage and markers, selecting the exact same position of the finger for each tracked frame proved to be a challenging task and could have the accuracy of only 0.5 pixels. With the Dataset 2 videos, this would roughly translate into 0.5-1 millimetres accuracy in real world units.

Filtering with the derivatives of position helps to get more meaningful results out of the trajectory. Without filtering, the acceleration curves of trajectories were unclear and obscured the force behind the real movement. When enough filtering was applied to the data, the results started to correspond with the ground-truth results and made a lot more sense to look at, thus helping the analysis of the real accelerations behind the movement.

From the experiments, it can be seen that with enough filtering applied to tracking data one can find the underlying acceleration and velocity values. Filtering position-data only was not enough for most of the filtering algorithms, but when filtering was applied also to the velocity data obtained from filtered position-data the velocity curve smoothed out much closer to the ground-truth data, which itself had some fluctuation. The best filtering methods for tracking data were LOESS and UKS, both of which got good results out of the tracking data even without additional velocity or acceleration component re-filtering.

The position of maximal speed on the trajectory provides information about how confident the hand movement is. The acceleration curve tells us how the target was reached and

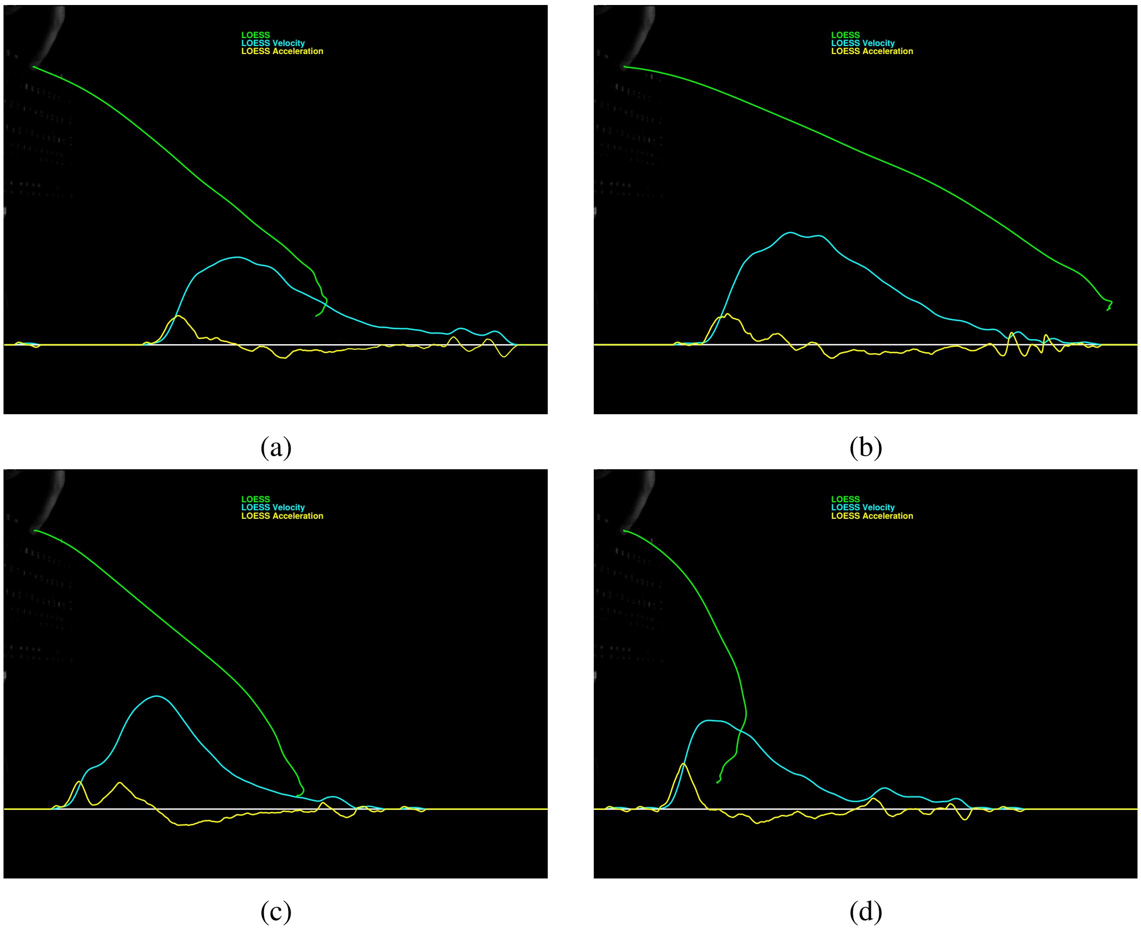


Figure 28. Four images with filtered trajectories with velocity and acceleration curves. Trajectory is plotted with green, velocity with cyan, and acceleration with yellow colour: (a) Video A; (b) Video B; (c) Video C; (d) Video D

whether corrective sub-movements in the trajectory were needed. What was interesting to see from the results was how different target positions affect the commitment of the hand movement towards the screen and how accurate the hits on the target position were. Corrective sub-movements tell about hesitation in the trajectory. The results in Figure 28 show that, at the end of the trajectories, there are corrective sub-movements which can be seen as changes in acceleration in the images.

5.2 Future work

The tracking performance of two trackers in case of Dataset 2 surpassed real-time requirements when the image-loading times were not included into the calculations. It would be interesting to see how Solid-State Drive (SSD) would change the MATLAB image-loading time, which was 2.3 milliseconds with normal 1TB 7200rpm HDD. Also, it would be interesting to see whether the use of SSD would enable real-time tracking of high-speed videos when the image-loading times are included.

Smoothing the trajectories produced by the trackers gave good results for the derivatives of the position, velocity and acceleration, but sub-pixel accuracy for video sequences which require precision could be another alternative way. Sub-pixel accuracy usually requires some kind of marker from which one can calculate an accurate position for the object. By having more accurate positions of the object, one would not need to smooth the trajectories and more accurate results also for the velocities and accelerations of the moving object would be generated.

Ground-truth annotation process proved to be a hard undertaking. Some gold-standard method for sub-pixel tracking with the use of a marker would probably make the annotations better and more accurate. There could be room also for a new tracking performance measure such as normalised sum or average of bounding-box corners' distance from the ground-truth. That could help differentiate trackers which really follow the state change of the object and would help to identify trackers that can update the scale of the object being tracked. The videos used in this work did not have large scale changes, but adapting to the scale changes on sub-pixel level could help to make the tracking process even more accurate.

Automatic Kalman filter parameter selection was out of the scope of this research. It would be interesting to see if the Kalman filter or smoother with optimal parameters would perform better than it did with manually selected parameters in this research. With automatic parameter selection, Kalman filtering could be easier to use than LOESS. The Kalman filter would enable filtering the results in real-time.

Including a multi-camera setup for research like this would make 3D data option available, and it would be interesting to see how filtering and feature extraction would work with 3D data, and how much the hand would move in the depth direction during the corrective sub-movement part. In this work, the movement was assumed to be linear in the depth direction.

6 CONCLUSION

In this thesis hand tracking in high-speed videos, and post-processing and analysis of the tracked hand trajectories were studied. The results showed that objects in high-speed video feeds with almost black background can be tracked in real-time with two current trackers. For this research, this meant reaching speeds of 887 (KCF) and 1649 (STC) frames per second on average for Dataset 2 test video sequences which were recorded at 500 frames per second. Thus, the trackers reached speeds well over real-time speeds. Even though the performance evaluation for trackers in this setup did not include the image-loading times, 2.3 milliseconds on average per image with MATLAB, the results are still impressive.

Filtering helps to find smooth acceleration curves to allow us clearly see where the moments of maximum and minimal acceleration are. With a right amount of filtering, the velocity and acceleration features of the trajectories got closer to the ground-truth. Two filtering methods, LOESS and UKS, produced the most consistent results for all the tests. Selecting one method as the winner came down to the question of which is the simpler one to use, and that happened to be LOESS. To conclude, with filtering and smoothing the hand-tracking data, it is possible to get to the underlying characteristics of the real movement sequence.

The results provide observations about the suitability of tracking methods for high-speed hand tracking and about how filtering can be applied to produce more appropriate velocity and acceleration curves calculated from the tracking data.

REFERENCES

[1] Microsoft. Xbox - Kinect. `http://www.xbox.com/en-US/kinect`. [Online]. Accessed: May 2014.

[2] Leap motion. `https://www.leapmotion.com/product`. [online]. Accessed: May 2014.

[3] Emilio Maggio and Andrea Cavallaro. *Video Tracking: Theory and Practice*. Wiley Publishing, 1st edition, 2011.

[4] Matej Kristan, Roman Pflugfelder, Aleš Leonardis, Jiri Matas, Fatih Porikli, Luka Čehovin, Georg Nebehay, Gustavo Fernandez, Tomas Vojir, A Gatt, A khajenezhad, A Salahledin, A Soltani-Farani, A Zarezade, A Petrosino, A Milton, B. Bozorgtabar, Bo Li, Chee Seng Chan, Cherkeng Heng, D. Ward, D. Kearney, D. Monekosso, H.C. Karaimer, H.R. Rabiee, Jianke Zhu, Jin Gao, Jingjing Xiao, Junge Zhang, Junliang Xing, Kaiqi Huang, K. Lebeda, Lijun Cao, M.E. Maresca, Mei Kuan Lim, M. El Helw, M. Felsberg, P. Remagnino, R. Bowden, R. Goecke, R. Stolkin, S.Y. Lim, S. Maher, S. Poullot, S. Wong, S. Satoh, Weihua Chen, Weiming Hu, Xiaoqin Zhang, Yang Li, and ZhiHeng Niu. The Visual Object Tracking VOT2014 challenge results. In *Proceedings of the European Conferance on Computer Vision (ECCV)*, pages 1–27, 2014.

[5] Xi Li, Weiming Hu, Chunhua Shen, Zhongfei Zhang, Anthony Dick, and Anton Van Den Hengel. A survey of appearance models in visual object tracking. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(4):58:1–58:48, October 2013.

[6] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4), December 2006.

[7] Luka Čehovin, Matej Kristan, and Aleš Leonardis. Is my new tracker really better than yours? In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 540–547. IEEE, 2014.

[8] Lappeenranta University of Technology. The Copex (Computational Psychology of Experience in Human-Computer Interaction) project. `http://www2.it.lut.fi/project/copex/index.shtml`.

[9] Lappeenranta University of Technology. Machine Vision and Pattern Recognition Laboratory. `http://www.it.lut.fi/mvpr/`.

[10] University of Helsinki. Visual Cognition Research Group. `http://www.helsinki.fi/psychology/groups/visualcognition/`.

[11] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-Speed Tracking with Kernelized Correlation Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Published online 2014.

[12] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. Exploiting the Circulant Structure of Tracking-by-detection with Kernels. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 702–715, 2012.

[13] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online Object Tracking: A Benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2411–2418, 2013.

[14] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, 2012.

[15] Arnold Smeulder, Dung Chu, Rita Cucchiara, Simone Calderara, Afshin Deghan, and Mubarak Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1442–1468, July 2014.

[16] Matej Kristan, Roman Pflugfelder, Aleš Leonardis, Jiri Matas, Fatih Porikli, Luka Čehovin, Georg Nebehay, Gustavo Fernandez, Tomas Vojir, A Gatt, A khajenezhad, A Salahledin, A Soltani-Farani, A Zarezade, A Petrosino, A Milton, B. Bozorgtabar, Bo Li, Chee Seng Chan, Cherkeng Heng, D. Ward, D. Kearney, D. Monekosso, H.C. Karaimer, H.R. Rabiee, Jianke Zhu, Jin Gao, Jingjing Xiao, Junge Zhang, Junliang Xing, Kaiqi Huang, K. Lebeda, Lijun Cao, M.E. Maresca, Mei Kuan Lim, M. El Helw, M. Felsberg, P. Remagnino, R. Bowden, R. Goecke, R. Stolkin, S.Y. Lim, S. Maher, S. Poullot, S. Wong, S. Satoh, Weihua Chen, Weiming Hu, Xiaoqin Zhang, Yang Li, and ZhiHeng Niu. The Visual Object Tracking VOT2013 Challenge Results. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 98–111, 2013.

[17] Kaihua Zhang, Lei Zhang, Qingshan Liu, David Zhang, and Ming-Hsuan Yang. Fast visual tracking via dense spatio-temporal context learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 127–141, 2014.

[18] Kaihua Zhang, Lei Zhang, and Ming-Hsuan Yang. Real-time compressive tracking. In *Proceedings of the European Conferance on Computer Vision (ECCV)*, pages 864–877, 2012.

[19] Real-Time Compressive Tracking Website. `http://www4.comp.polyu.edu.hk/~cslzhang/CT/CT.htm`. [online]. Accessed: May 2014.

[20] Kaihua Zhang, Lei Zhang, and Ming-Hsuan Yang. Fast compressive tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):2002–2015, Oct 2014.

[21] Fast Compressive Tracking Website. `http://www4.comp.polyu.edu.hk/~cslzhang/FCT/FCT.htm`. [online]. Accessed: May 2014.

[22] High-Speed Tracking with Kernelized Correlation Filters Website. `http://www.isr.uc.pt/~henriques/circulant/`. [online]. Accessed: June 2014.

[23] Martin Godec, Peter M. Roth, and Horst Bischof. Hough-based tracking of non-rigid objects. *Computer Vision and Image Understanding*, 117(10):1245–1256, 2012.

[24] Hough-based Tracking of Non-Rigid Objects Website. `http://lrs.icg.tugraz.at/research/houghtrack/`. [online]. Accessed: June 2014.

[25] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental Learning for Robust Visual Tracking. *International Journal of Computer Vision*, 77(1-3):125–141, 2008.

[26] Incremental Learning for Robust Visual Tracking Website. `http://www.cs.toronto.edu/~dross/ivt/`. [online]. Accessed: May 2014.

[27] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1619–1632, 2011.

[28] Robust Object Tracking with Online Multiple Instance Learning Website. `http://www.cs.toronto.edu/~dross/ivt/`. [online]. Accessed: May 2014.

[29] Tracking Learning Detection Website. `http://personal.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html`. [online]. Accessed: May 2014.

[30] Zhong Wei, Lu Huchuan, and Yang Ming-Hsuan. Robust Object Tracking via Sparse Collaborative Appearance Model. *IEEE Transactions on Image Processing*, 23(5):2356–2368, 2014.

[31] Tracking via Sparsity-based Collaborative Model. `https://github.com/gnebehay/SCM`. [online]. Accessed: October 2014.

[32] Fast Tracking via Spatio-Temporal Context Learning Website. `http://www4.comp.polyu.edu.hk/~cslzhang/STC/STC.htm`. [online]. Accessed: June 2014.

[33] Sam Hare, Amir Saffari, and Philip H. S. Torr. Struck: Structured output tracking with kernels. In *IEEE International Conference on Computer Vision (ICCV)*, pages 263–270, 2011.

[34] Struck Website. `http://www.samhare.net/research/struck/code`. [online]. Accessed: June 2014.

[35] Weiming Hu, Xi Li, Wenhan Luo, Xiaoqin Zhang, Stephen Maybank, and Zhongfei Zhang. Single and multiple object tracking using log-Euclidean Riemannian subspace and block-division appearance model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2420–2440, 2012.

[36] Tracker Based on Riemannian Subspace Learning Website. `http://www.iis.ee.ic.ac.uk/~whluo/code.html`. [online]. Accessed: May 2014.

[37] Dong Wang, Huchuan Lu, and Ming-Hsuan Yang. Online Object Tracking With Sparse Prototypes. *IEEE Transactions on Image Processing*, 22(1):314–325, 2013.

[38] Online Object Tracking with Sparse Prototypes Website. `http://ice.dlut.edu.cn/lu/Project/TIP12-SP/TIP12-SP.htm`. [online]. Accessed: May 2014.

[39] Online Object Tracking: A Benchmark Website. `http://visual-tracking.net/`. [online]. Accessed: October 2014.

[40] Visual Object Tracking Website. `http://www.votchallenge.net/`. [online]. Accessed: October 2014.

[41] Ville Hiltunen. Hand tracking in high-speed camera videos. Master's thesis, Lappeenranta University of Technology, 2013.

[42] Ville Hiltunen, Tuomas Eerola, Lasse Lensu, and Heikki Kälviäinen. Comparison of general object trackers for hand tracking in high-speed videos. In *International Conference on Pattern Recognition (ICPR)*, pages 2215–2220, 2014.

[43] Qi Liu, Xiaoguang Zhao, and Zengguang Hou. Survey of single-target visual tracking methods based on online learning. *Institution of Engineering and Technology (IET) Computer Vision*, 8(5):419–428, 2014.

[44] Jiayun Wu, Daquan Chen, and Rui Yi. Real-time compressive tracking with motion estimation. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2374–2379, 2013.

[45] Matthew B. Blaschko and Christoph H. Lampert. Learning to localize objects with structured output regression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 2–15. Springer-Verlag, 2008.

[46] Yan Feng, John Goree, and Bin Liu. Errors in particle tracking velocimetry with high-speed cameras. *Review of Scientific Instruments*, 82(5):–, 2011.

[47] Abraham Savitzky and Marcel J. E. Golay. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[48] Peter S. Maybeck. *Stochastic Models, Estimation and Control*. Mathematics in Science and Engineering. Academic Press, 1978.

[49] Sophocles J. Orfanidis. *Introduction to Signal Processing*. Prentice Hall international editions. Prentice Hall, 1996-2009.

[50] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[51] Matlab Documentation Filtering and Smoothing Data. `http://www.mathworks.se/help/curvefit/smoothing-data.html`. [online]. Accessed: September 2014.

[52] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter: SIGGRAPH 2001 Course 8. In *Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques*, pages 12–17, 2001.

[53] Greg Welch and Gary Bishop. An introduction to the Kalman filter. Technical report, Department of Computer Science, University of North Carolina, 1995.

[54] Kalman Filter System Model by Burkart Lingner. `http://www.texample.net/tikz/examples/kalman-filter/`. [online]. Accessed: October 2014.

[55] Michael Montemerlo and Sebastian Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1985–1991 vol.2, 2003.

[56] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Proceedings of The International Society for Optics and Photonics (SPIE) AeroSense: International Symposium on Aerospace/Defense Sensing, Simulations and Controls*, 1997.

[57] Rudolph Van Der Merwe and Eric A. Wan. The unscented kalman filter for nonlinear estimation. pages 153–158, 2000.

[58] William S. Cleveland. Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979.

[59] William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):pp. 596–610, 1988.

[60] Ronald W. Schafer. What is a Savitzky-Golay filter? [lecture notes]. *IEEE Signal Processing Magazine*, 28(4):111–117, 2011.

[61] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20(1-2):89–97, 2004.

[62] Ivan W. Selesnick and Ilker Bayram. Total variation filtering. *Lecture Notes*, February 4 2010.

[63] Laurent Condat. A direct algorithm for 1-d total variation denoising. *IEEE Signal Processing Letters*, 20(11):1054–1057, 2013.

[64] Jian Zhang, Shaohui Liu, Ruiqin Xiong, Siwei Ma, and Debin Zhao. Improved total variation based image compressive sensing recovery by nonlocal regularization. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2836–2839, 2013.

[65] Guy Gilboa, Yehoshua Y. Zeevi, and Nir Sochen. Texture Preserving Variational Denoising Using an Adaptive Fidelity Term. Proceedings of the Variational, Geometric, and Level Set Methods in Computer Vision (VLSM), 2003.

[66] Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[67] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[68] Digby Elliott, Steve Hansen, Lawrence E. M. Grierson, James Lyons, Simon J. Bennett, and Spencer J. Hayes. Goal-directed aiming: two components but multiple processes. *Psychological Bulletin*, 136(6):1023–44, 2010.

[69] Mega Speed MS50K. `http://www.megaspeed.ca/index.php?option=com_content&view=article&id=7&Itemid=10`. [online]. Accessed: October 2014.

[70] Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, volume 3, pages 1381–1384. IEEE, 1998.

[71] Camera Calibration Toolbox for Matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`. [online]. Accessed: June 2014.

[72] References for Camera Calibration Toolbox for Matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/ref.html`. [online]. Accessed: June 2014.

Appendix 1. Dataset 2 Results

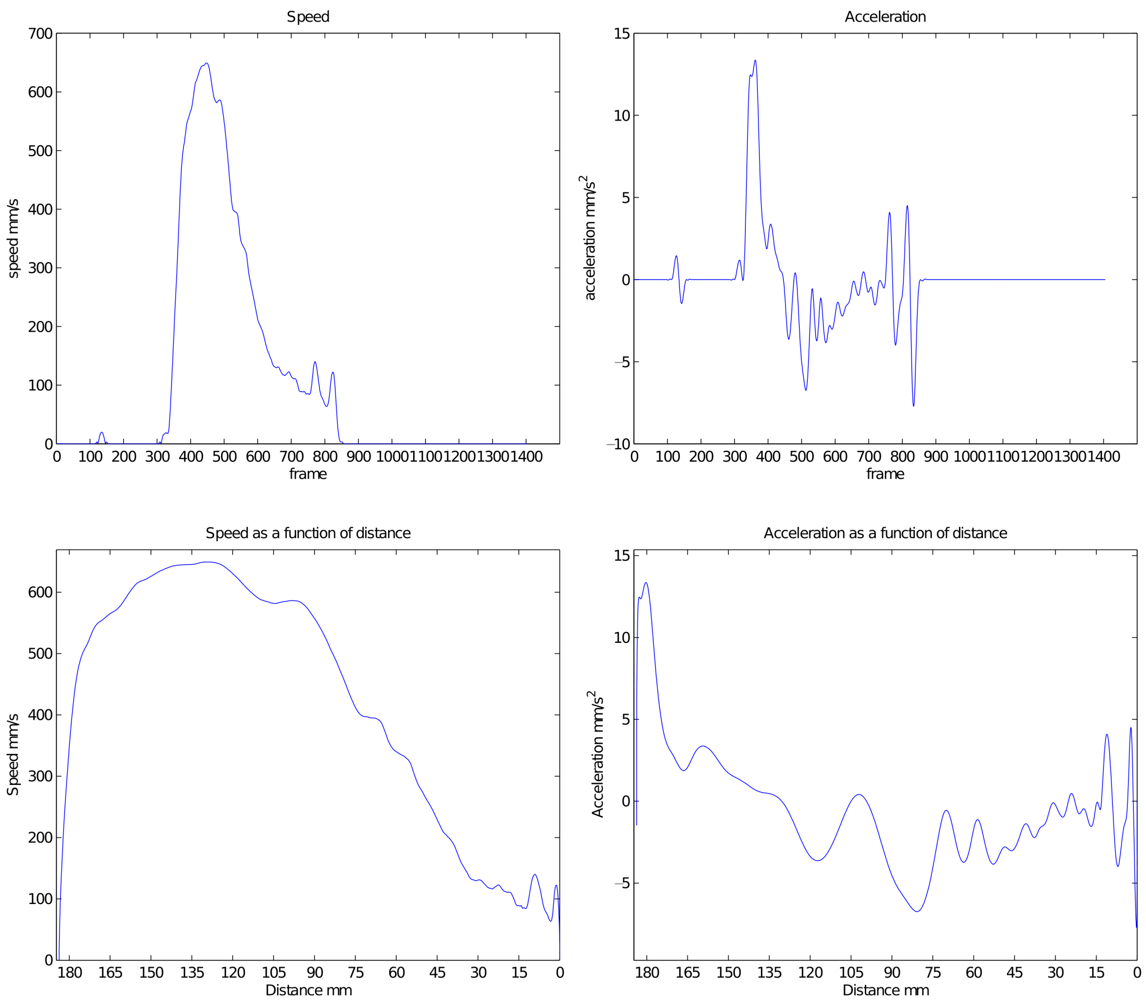


Figure A1.1. Dataset 2 video A: speed and acceleration curves as a function of frames and distance to the screen.

(continues)

Appendix 1. (continued)

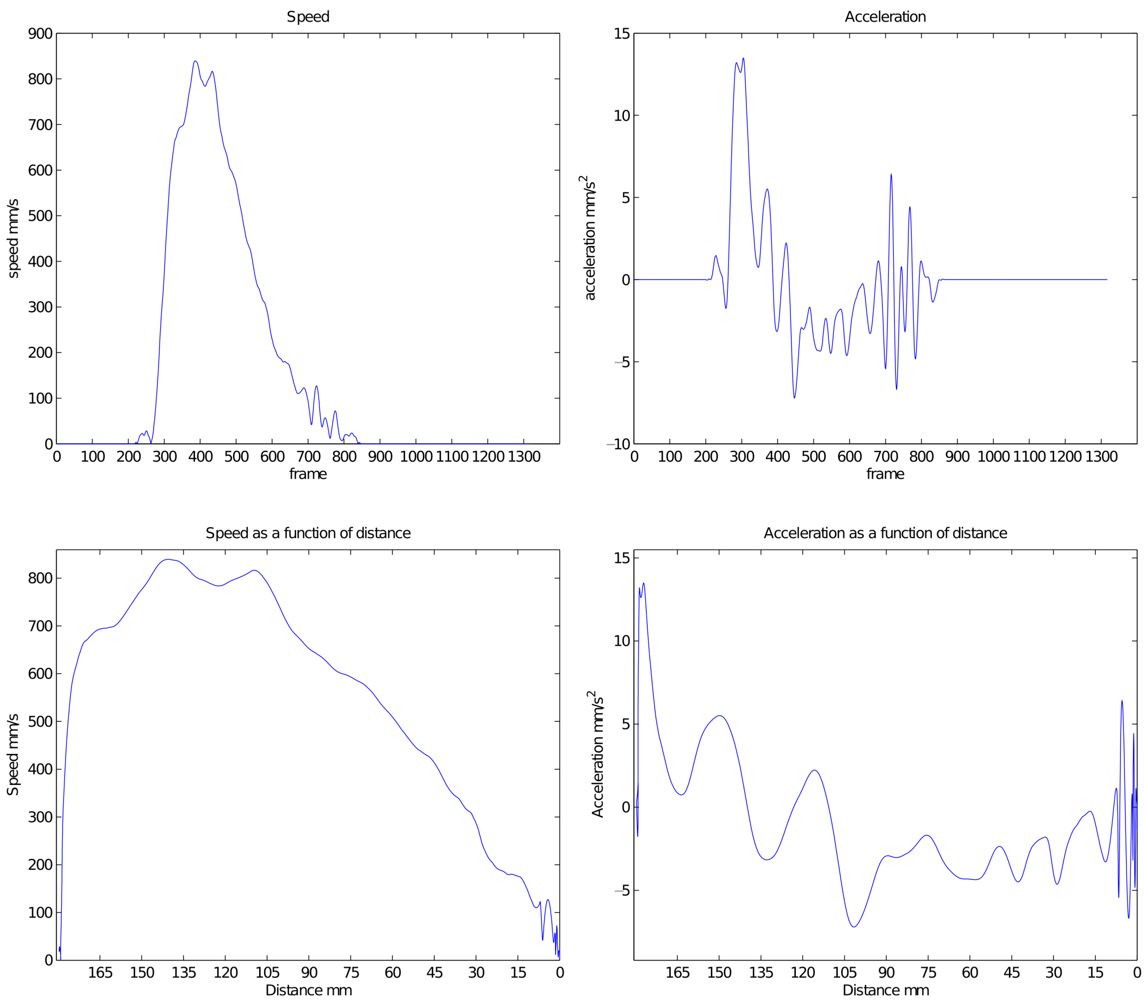


Figure A1.2. Dataset 2 video B: speed and acceleration curves as a function of frames and distance to the screen.

(continues)

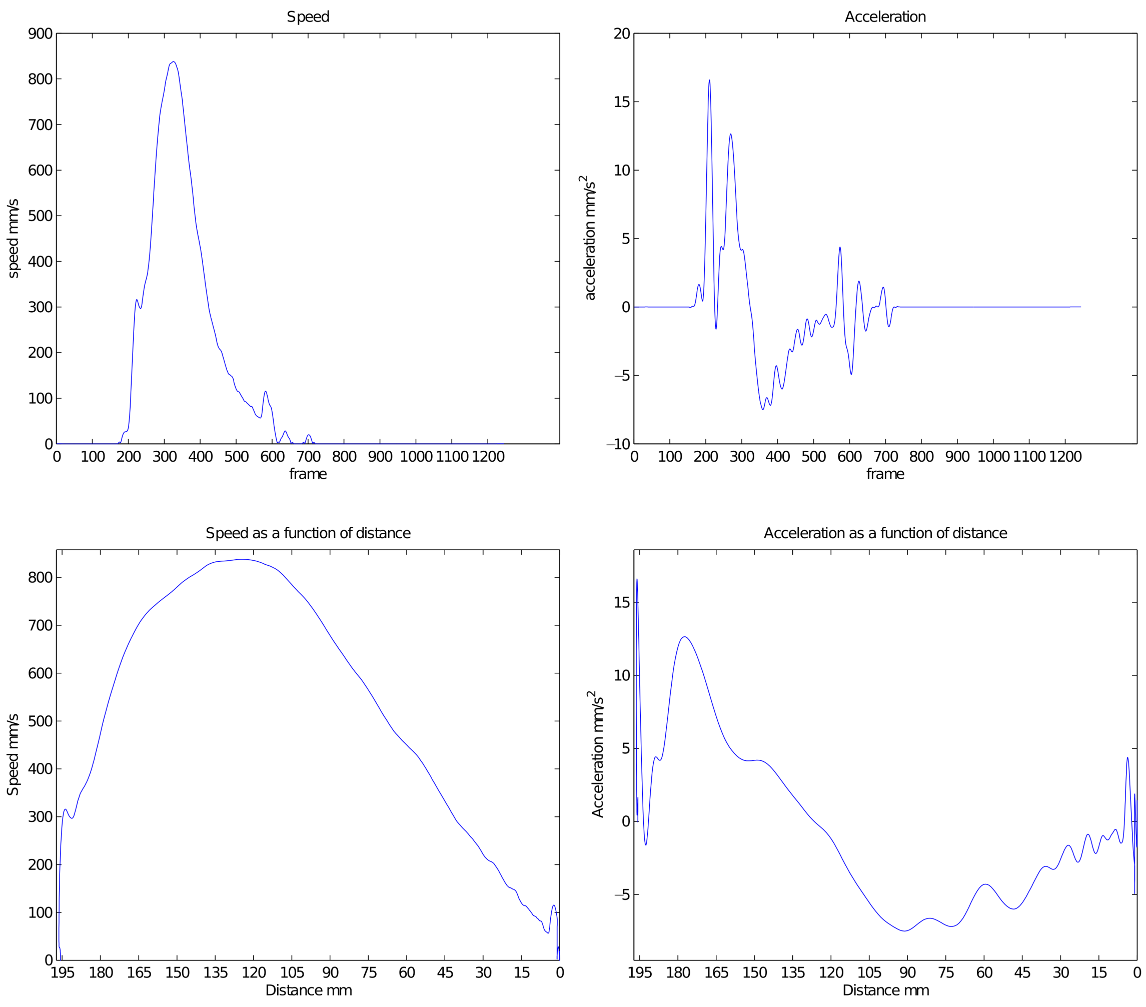


Figure A1.3. Dataset 2 video C: speed and acceleration curves as a function of frames and distance to the screen.

(continues)

Appendix 1. (continued)

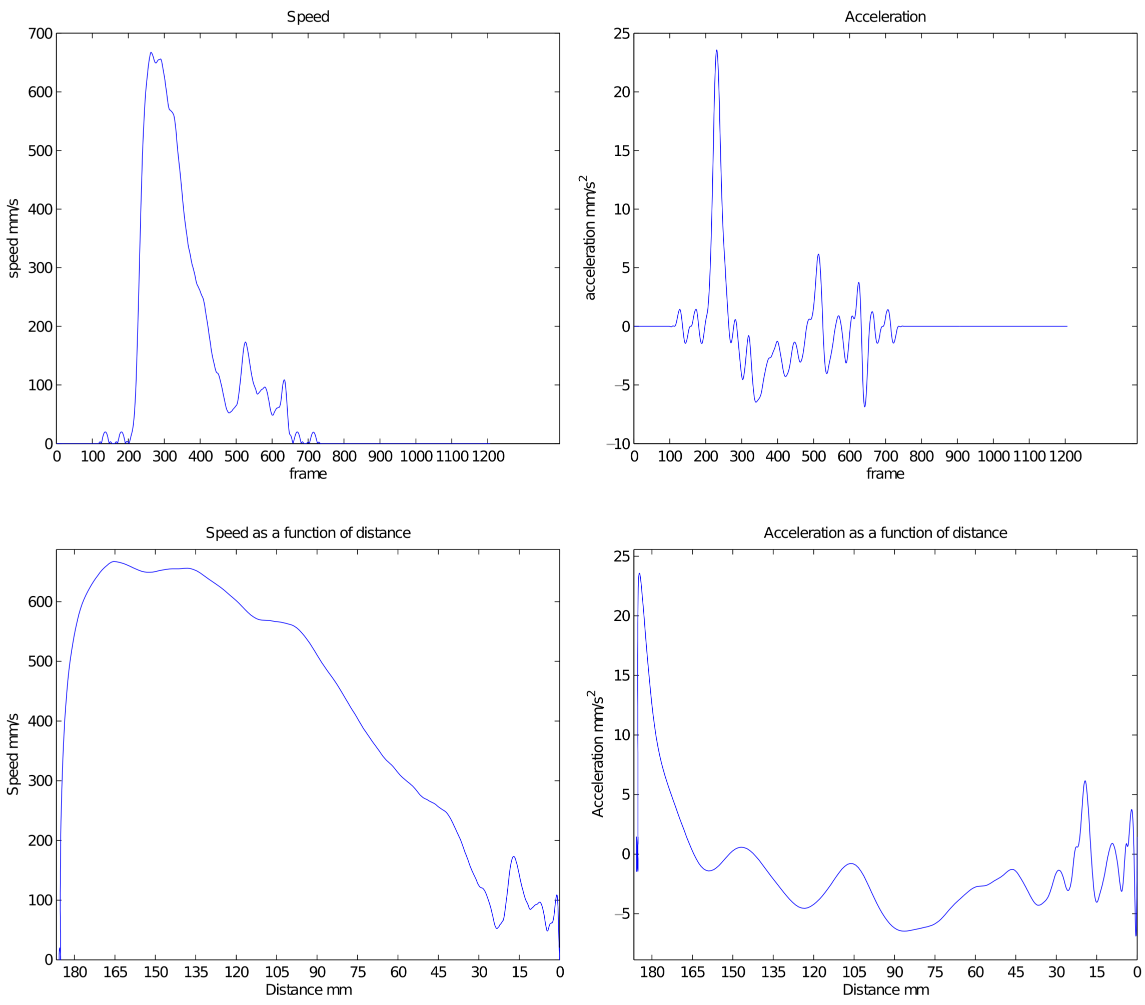


Figure A1.4. Dataset 2 video D: speed and acceleration curves as a function of frames and distance to the screen.

(continues)

Appendix 1. (continued)

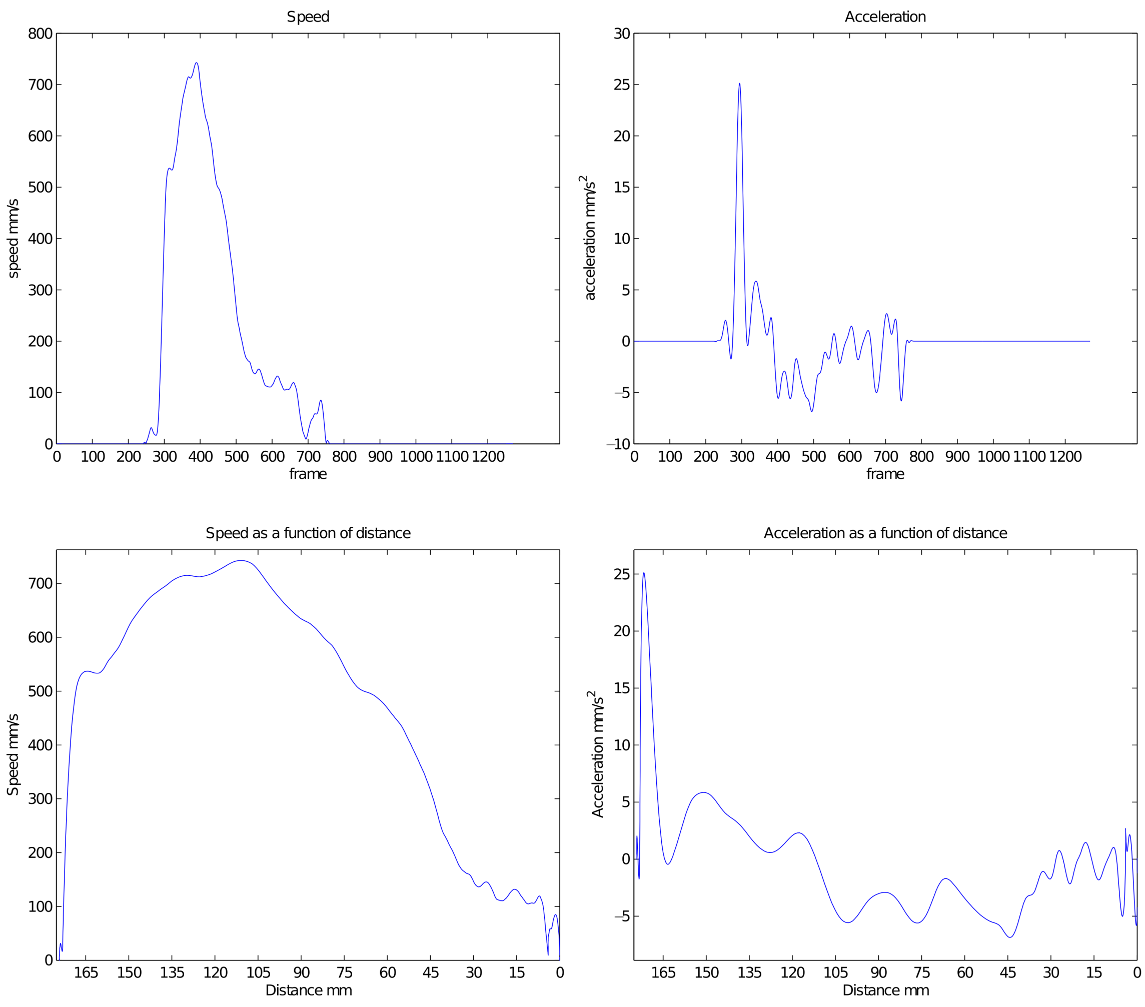


Figure A1.5. Dataset 2 video E: speed and acceleration curves as a function of frames and distance to the screen.

(continues)

Appendix 1. (continued)

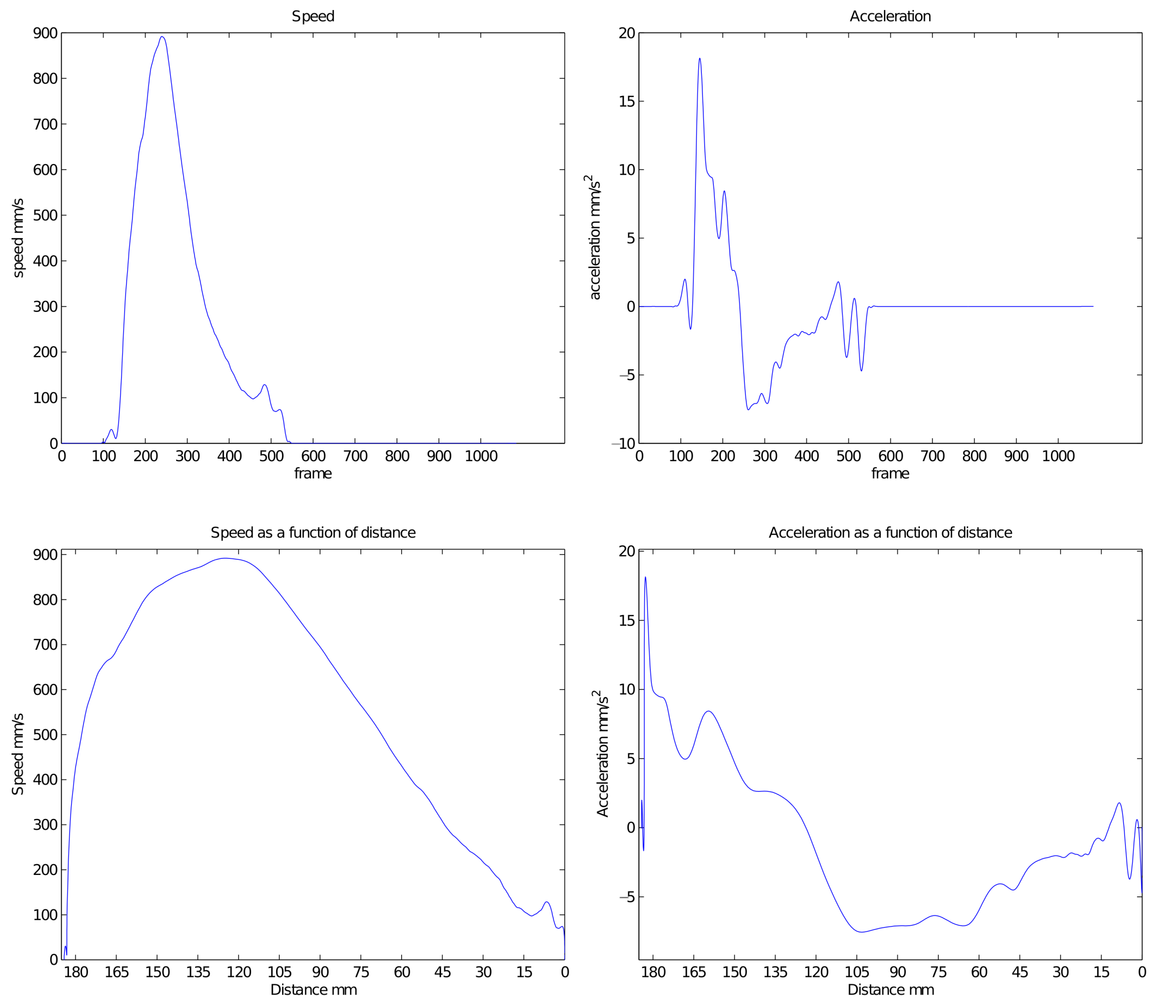


Figure A1.6. Dataset 2 video F: speed and acceleration curves as a function of frames and distance to the screen.