Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik

Bachelorarbeit Medieninformatik

Automatic Semantic Mapping of Mobile Eye-Tracking Data through Keypoint Matching and Tracking

Jannik Hofmann

28.02.2019

Gutachter

Prof. Dr. Enkelejda Kasneci
Lehrstuhl Perception Engineering
Fachbereich Informatik
Universität Tübingen

Betreuer

Thiago Santini
Lehrstuhl Perception Engineering
Fachbereich Informatik
Universität Tübingen

Hofmann, Jannik
Automatic Semantic Mapping of Mobile Eye-Tracking Data through Keypoint Matching and Tracking
Bachelorarbeit Medieninformatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 01.11.18 - 28.02.19

Abstract

This work demonstrates that it is possible to reliably remap gaze data collected by a head-mounted eye tracker, even when the person wearing it is freely moving their head and body, to a reference frame. The gaze point coordinates are originally relative to a video of the field of view in front of the participant, distorted by a fisheye lens. The gaze data is then automatically remapped to an undistorted projection and tracked along a user-defined video section using the SURF [Bay et al., 2006] and FLANN algorithms [Muja and Lowe, 2009], resulting in a collection of time-related coordinates that are relative to a region of interest which can be selected from a frame in the tracked video section. Once the data gaze data is relative to the static keyframe, it is outputted in Eyetrace format [Kübler at al., 2015] and traditional eye tracking methods can be applied, for example to create a heatmap of gaze points on a 2-dimensional object.

Keywords

computer vision, eye tracking, opencv, surf, flann, gaze mapping

Contents

1 Introduction

In a wide range of scenarios, researchers are interested in the attention people pay to various aspects of an experience. In order to be able to quantify this question, it is helpful to analyze the eye movements of participants and create statistics of what parts of the experience people looked at the most and where their gaze remained longer. If this is done with a screen-based eye tracker, the fixation data can be easily analyzed and interpreted [Pontillo et al., 2010].

However, wearable eye trackers, which record not only the eye movements, but also the participant's field of view, produce data that is more difficult to analyze. While participants have the advantage of being able to freely look around and move through a three-dimensional space, the collected gaze data has the participant's field of view as a reference frame and only indicates the relative view direction [MacInnes et al., 2018]. The video of the field of view can then be used to translate those relative gaze directions into absolute data, therefore enabling semantic interpretation of the data.

The software presented in this work provides a user interface to easily process the data collected by an eye tracker during such an experiment. The user can select a video section that shows an object of interest and define a quadrilateral region of interest (ROI) in a frame within that section. This region then works as frame of reference for all gaze points. The coordinates collected by the eye tracker need to be transformed to the ROI. For this process, the algorithm first uses camera calibration data collected by intrinsic parameter estimation to undistort the effect of the fisheye lens and then tracks the selected video section using keypoint detection in "Speeded up robust features" (SURF) [Bay et al., 2006] and "fast library for approximate nearest neighbors" (FLANN) for matching the keypoints [Muja and Lowe, 2009].

1.1 Objectives

The objective of this work is to discuss an algorithm that can reliably track a visual region over a video section and map the participant's eye movement to that region using data and video recorded by a head-mounted eye tracker. The goal of this algorithm is to provide a user-interface, where a user can select a part of the video and define a region within that section. The video is used to calculate the movement of the field of view, by detecting and matching keypoints between the frames. Then the gaze coordinates collected by the eye tracker are translated to the region of interest and outputted to a file in Eyetrace format [Kübler at al., 2015].

This results in gaze data that is relative to a static reference frame, so it can be more easily processed and analyzed by traditional eye tracking methods, for example, to display a heatmap of the participant's gaze points on different objects without having to analyze each frame in a video of the field of view individually.

1.2 Outline

The following chapters are structured as follows. In chapter 2, related work in this field, as well as the interfaces and algorithms are discussed. This provides an outline into what technologies and approaches were used to create the application, which is discussed in this work. Chapter 3 contains a detailed description of the approach and implementation that was used for this algorithm. It contains an overview of what exactly is happening before and during the tracking process, followed by comprehensive explanations of these steps. A practical application of this algorithm is described in chapter 4. Here, the setup and results of eye-tracking data from a specific experiment is described and analyzed. The discussion and interpretation of these results follows in chapter 5. A further investigation into the shortcomings of this algorithm, followed by approaches to improve the application takes place in chapter 6.

2 Related research and interfaces

This chapter provides an explanation as to which interfaces and algorithms were used to develop the application for tracking the video frames and calculating transformed gaze points.

2.1 Related Work

For experiments that take place in large spaces, the location of the participant can also be determined and mapped with GPS [Kiefer et al., 2012].

A different approach for tracking the image, is to create a three-dimensional model of the environment with a stereo camera system on the eye tracker [Badino and Kanade, 2011]. This not only provides another layer of depth-related data for more accurate tracking but can also be used to distinguish objects in the video that are moving in different directions.

De Beugher et al. utilize object detection; the area around the gaze point is processed with "Scale-invariant feature transform (SIFT)" and compared to the keypoints of handpicked images, which have already been pre-processed [De Beugher et al., 2012]. This algorithm can then recognize any objects that have been stored in image form and detect faces and people in front of the eye tracker. Toyama et al. followed a similar approach, they also used SIFT to compare the participant's field of view to a database of images of objects from the experiment. This algorithm was used to demonstrate a prototype for an interactive museum guide that acts in real-time based on the visitors' gaze [Toyama et al., 2012].

Takemura et al. also estimate the three-dimensional point-of-regard with just the video of the participant's field of view. They employ "Simultaneous localization and mapping (SLAM)," which was originally developed in robotics research "for estimating robot pose in unknown environments" [Takemura et al., 2010]. This algorithm can be used to create a three-dimensional model of keypoints around the camera but has limitations in its usage [Smith and Cheeseman, 1986]. For example, it employs closed loops that require the algorithm to know when the camera is in a location where it has been before. It could also cause issues when keypoints on moving objects (like people around the participant) are freely moving around.

Another related project has an objective similar to this work. Bykowski and Kupiński also aim to map the gaze points of video from a head-mounted eye tracker to a static reference frame using OpenCV [Bykowski and Kupiński, 2018]. The results of the algorithm still needed to be supervised, the authors suggested to use automatic validation in the future. Their project corresponds with the keyframe selection and tracking of gaze data in this work, which is covered in the section 3.3.3, as well as chapters 3.5 and 3.6. The difference between their project and this one is that they use AKAZE for keypoint detection and matching, instead of SURF and FLANN. They also did not implement a measure to automatically detect and correct tracking mistakes, which is detailed in chapter 3.4 of this work. This approach aims to make the tracking process more reliable, so that no more supervision of the results is necessary in the future.

2.2 OpenCV

“OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library” that has “C++, Python, Java and Matlab interfaces and supports Windows, Linux, Android and Mac OS.” It "was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products" [OpenCV, 2019]. Its BSD license makes the main library free to use and modify, for both private and commercial use.

In this work, OpenCV is used for the general framework to easily load image and execute calculations based on their data. Images are saved in the form of 3-dimensional matrices, which makes it easy to manipulate them or draw simple lines on them. Specifically, the application utilizes OpenCV features for extracting video frames, storing and loading image files, drawing visualizations, as well as for matrix transformations in many steps throughout the image processing. In addition to these features, OpenCV also includes a model for images distorted by fisheye lenses, which is used to easily undistort the video frame into images from a pinhole camera model [Kannala and Brandt, 2006].

2.3 SURF and FLANN

OpenCV includes many functions for image processing natively. However, several features have to be added manually. These are found in a separate Github repository, which is “intended for development of so-called "extra" modules, contributed functionality.” [Github, 2019a] These modules are parts of OpenCV that are “not well-tested” or “do not have a stable API,” so they are published in this repository first and moved to the main release at a later point [Github, 2019a].

This repository for contributions also contains so-called “nonfree” features, which cannot be released under the BSD license and are therefore not part of the default OpenCV library. One of these features is the patented “SURF” (“Speeded up robust features”) algorithm, which may not be used for commercial purposes, unless permitted by the copyright holder [SURF, 2019].

The SURF (Speeded Up Robust Features) algorithm is described as a “scale and rotation-invariant interest point detector and descriptor” and is multiple times faster than the SIFT (scale-invariant feature transform) descriptor, which it is based on [Bay et al., 2006]. Instead of using Gaussian filters to detect keypoints, SURF utilizes box filters to approximate second order Gaussian derivatives, which can be evaluated significantly faster. These interest points should be scale and rotation-invariant, so they often have a blob shape. The orientation of these points is determined by computing a feature vector from the neighborhood of each point. This vector is distinctive and robust to minor changes in the image. The distance between these vectors are then used to match the two images and compute a transformation matrix.

The Fast Library for Approximate Nearest Neighbors (FLANN) was introduced in 2009 by Muja and Lowe. It was written in C++ with the goal to quickly perform an approximate nearest neighbor search in high-dimensional spaces and can be used with C++, C, MATLAB, and Python [Muja and Lowe, 2009]. This approximation is orders of magnitude faster than calculating the nearest neighbor values directly. When applying FLANN to the keypoints that were detected with SURF, the result can be used to calculate a transformation matrix between two images.

3 Approach

3.1 Overview of the pipeline

The application developed for this work was written in C++ and uses Windows Forms for the user interface, to guide the user through the steps necessary to track the gaze points. The steps presented here match the pipeline for processing the main video of the participant's field of view, which was recorded by the eye-tracking device.

At first, the user selects such a video file, the file containing the gaze point data from the same recording session as well as the calibration data for the eye tracker. The application then converts the video into single frames, so that frames can be previewed and individually processed later. Next, the user selects the section of video that is relevant for his research. The frames in this section get undistorted from a fisheye-view using the eye trackers calibration data. Now the user defines a keyframe within this section and, if applicable, a region of interest (ROI) by selecting four points in that keyframe. As soon as the keyframe is selected, a folder for that tracking session will be created. It is identified by the start-, end- and keyframe numbers and stores every file that the application creates in that session, except for the cached tracking results. Finally, the application tracks the keyframe over the video section and translates the original gaze points into coordinates relative to the keyframe. If a valid ROI was selected, the points will also be converted to the coordinate system defined by this region. The application also displays and stores a visualization of the movement of camera and gaze points for each frame. Frame extraction, undistortion and tracking are done in separate threads, so that the user can still use the application’s interface, prepare the next steps, and cancel actions while these processes are running.

3.2 Reversing lens distortions

The original video of the participants’ field of view is recorded by an eye-tracking device. These devices usually use fisheye lenses, which distorts the image, in order to capture a field of view that is wide enough to contain the most relevant area of the participant’s field of view.

The tracking that is done later will use projective transformations, which can only be applied to images that preserve collinearity [Kannala and Brandt, 2006]. Therefore, the frames captured by the fisheye lens need to be undistorted. Otherwise, the tracking algorithm would have to

take into account in what area of the field of view the keypoints are, and in which way those points would be distorted, before it could match the frames and calculate the camera movement.

To undo the distortion, OpenCV uses a generic camera model for fisheye lenses [Kannala and Brandt, 2006][1] to estimate a transformation matrix for the lens distortion, which is then applied to each individual video frame of the selected section. The data for the undistortion was obtained during the experiment with a typical camera calibration procedure using a chessboard pattern. It consists of the following components, demonstrated with the values from the experiment described in chapter 4.1.

$$\text{Camera matrix } K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} 810.54 & 0 & 642.97 \\ 0 & 811.54 & 342.28 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{input vector of distortion coefficients } D = (k_1, k_2, k_3, k_4) \approx (-0.149, -0.048, 0.075, -0.034)$$

$$\text{image\_size} = \text{Size}(\text{width}, \text{height}) = \text{Size}(1280, 720)$$

$$\text{rectification transformation in the object space } R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{new camera matrix } P \approx \begin{bmatrix} 288.95 & 0 & 651.78 \\ 0 & 289.3 & 332.57 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{balance} = 1, \quad \text{new\_size} = \text{image\_size}, \quad \text{fov\_scale} = 1$$

Figure 3.1: *The same video frame before and after reversing the fisheye lens distortion.*

[1] The usage of this camera model in OpenCV is documented on [Bouguet, 2015a] and [Bouguet, 2015b]. Github user *Ilya-Krylov*, who is responsible for implementing the fisheye camera model in OpenCV, confirmed this fact in a comment on Sep 1, 2014 [Github, 2018b].

3.3 Tracking frames and calculating camera movement

3.3.1 Preparations for the tracking process

As soon as the user has selected a keyframe within a corresponding section and optionally a region of interest for the keyframe, the tracking process can be started. When that happens, all tracking-related values set by the user are copied to prevent changes, so any adjustments made by the user will not affect the currently running tracking process. This is done in order to avoid errors and inconsistencies, which would be difficult to accommodate by the program and might lead to confusing tracking results.

3.3.2 Image enhancements

Before the tracking algorithm itself is applied to two frames, the images from those frames are converted to grayscale versions and enhanced with sharpening and local histogram equalization. The sharpening is done by subtracting a blurred version of the image from the unblurred original. To further improve the detection of keypoints in the image, the contrast is enhanced locally using contrast limited adaptive histogram equalization (CLAHE) with a clip limit threshold of 2, which was empirically defined. This technique was developed for enhancing low-contrast images while limiting noise amplification [Pizer et al., 1987].

Figure 3.2: *Video frame before and after image enhancements (cropped for better visibility of the central field of view)*

3.3.3 Calculating transformation matrices

The keypoint detection itself is done with the “Speeded up robust features” (SURF) algorithm, a “scale and rotation-invariant interest point detector and descriptor” [Bay et al., 2006] and the matching uses “fast library for approximate nearest neighbors” (FLANN) [Muja and Lowe, 2009]. A detailed description of these algorithms is found in chapter 2.3 of this work. They are used to calculate a transformation matrix for the movement of the image content from one frame to the next. If this algorithm is unable to track two frames, the error detection will catch the result and try to calculate the movement by tracking a different combination of frames.

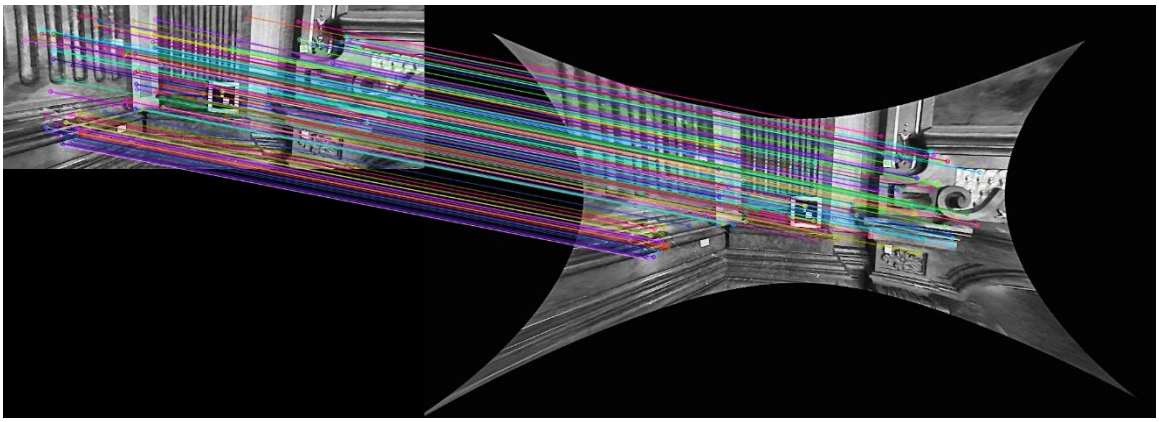

Figure 3.3: *Visualization of keypoint matching. The keypoints detected by SURF were matched using FLANN. The colored lines indicate positions of matched corresponding keypoints in each image.*

The undistorted images all have a concave pincushion shape, surrounded by black areas, which are always in the same place. Because of this, the SURF algorithm would identify the high-contrast areas at the edge of these areas as keypoints for tracking, so it might determine no movement as result. To prevent this issue, one of the images is cropped to a rectangular central view area, which only contains pixels from the original video. After matching this cropped area with the other full image, another transformation is added to the result, in order to translate the cropped coordinates back to the full image and neutralize the effect of the cropping on the tracking result.

All of the calculated tracking matrices are cached in .csv-files for later use. So instead of recalculating the same matrices again, they are simply retrieved from that file. This mostly saves time, when the same frames are tracked again, for example with a different keyframe and ROI. The user also has the possibility to calculate all of the tracking matrices in advance for the whole video in advance, and later analyze various sections and ROIs significantly faster.

3.3.4 Visualization

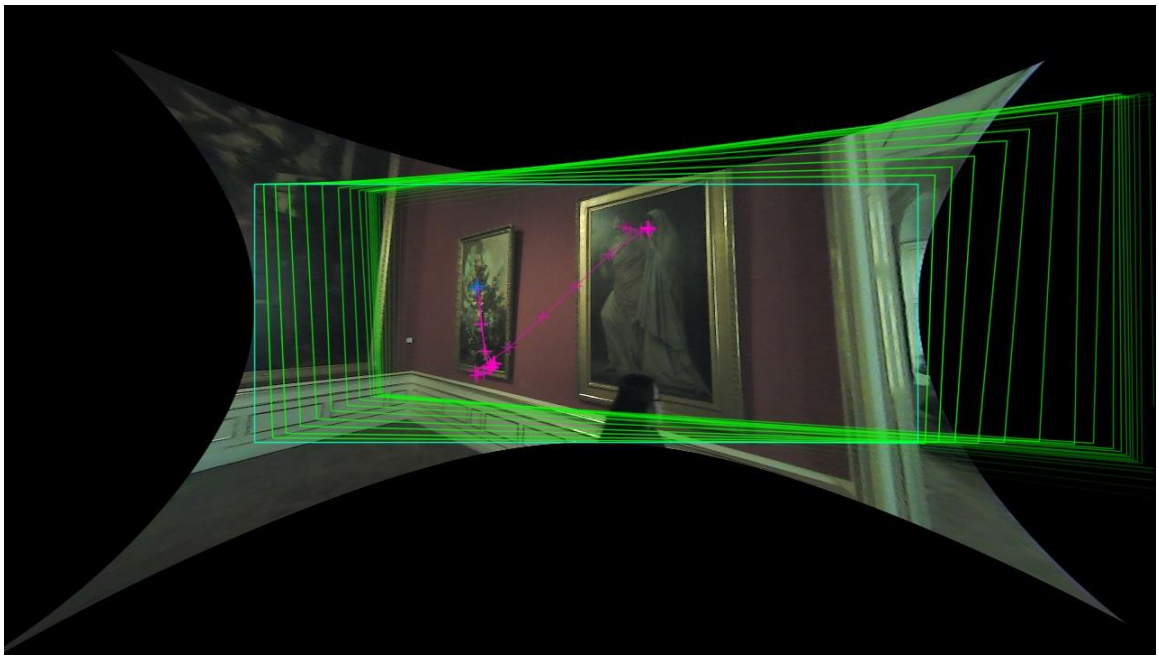

Figure 3.4: *Visualization of the tracking process (cyan rectangle: central field of view; green polygons: locations of that rectangle from previous frames, showing the tracked camera movement;*
blue crosses: gaze points of the current frame, magenta: gaze points of previous frames;
ROI not shown in this example)

The visualization gets updated every frame; it is shown to the user during the tracking process and stored for later reference. It shows the camera movement, gaze positions and any tracking errors. The camera movement is represented by displaying the previous positions of the rectangular area of the central field of view in green, the unmodified current area one is highlighted in cyan. The gaze points of the current frame are indicated by blue crosses, while the gaze points corresponding to previous frames are shown in magenta. Gaze points that are consecutive and not interrupted by gaze values outside of the undistorted field of view are connected with each other. In order to improve performance and visual discernibility, the elements slowly fade out and become invisible after twenty frames. If the keyframe has already been reached, and the current frame still belongs to its section, the keyframe's ROI is displayed in amber, it does not fade out. The visual elements from previous frames are moved along with the image tracking before being displayed in the next frame.

3.4 Error detection and correction in tracking

3.4.1 Cause of tracking errors

Sometimes the tracking calculation does not produce a result. This usually happens when a frame does not have enough easily distinguishable keypoints or if the keypoints match in a way that does not lead to a conclusive result. However, if a conclusive transformation is found and returned as result, there is a chance that it is inaccurate and produces a wrong tracking result. This happens if the image is grainy, dark or has low contrast, the eye-tracking camera has a low image quality in general or if it produces unfocused images. Tracking also proves difficult, when one frame greatly differs from the previous one, for example when a nearby camera flash is fired at the same time as the shutter of the eye tracker's video camera is open, which colors the frame white – or parts of it, when the eye tracker uses a rolling shutter. And if various parts of the image move in different directions, the tracking might also be unable to provide a conclusive result. This happens if people walk in front of the participant, or when the participant moves with distinct objects in foreground and background, like when walking through an open doorway.

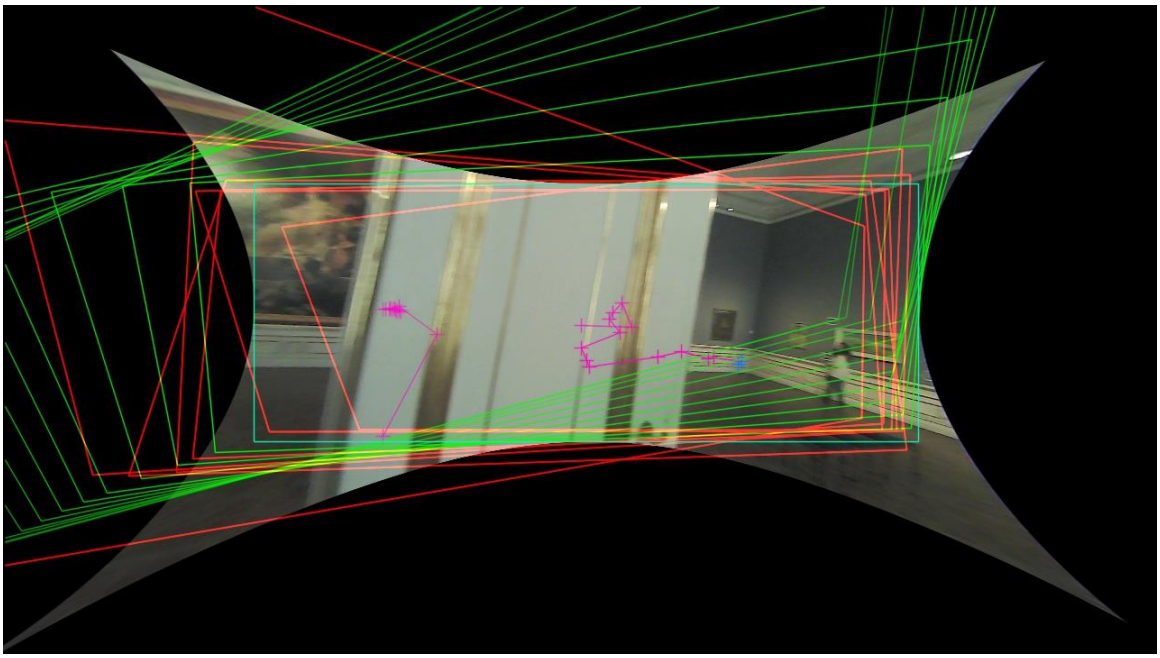

Figure 3.5: *Example of problems when tracking a blurred frame, which also has a foreground element (white-golden doorway) moving faster compared to the background.*
Different results of tracking attempts shown in red.

3.4.2 Strategy for error detection

To combat these issues, the application includes several measures to check whether a result is reasonable or more calculations need to be made. The transformed coordinates of the four corners of the central field of view are used for these calculations.

For instance, if the camera transformation has a high deviation to the previous one, this would mean that the camera suddenly changes direction or velocity in its angular movement and the participant had an exceedingly high acceleration in their head movement. Above a certain threshold set by the algorithm, these results are discarded as the movements would be unrealistic or even physically impossible. This threshold is defined in the implementation and can be dynamically changed depending on how strict the error detection should be under the circumstances.

Another measure against wrong tracking results checks the shape of a rectangle of the central field of view after transformation. When it differs too much from the original rectangle, the result is discarded as well, since it would be impossible for a participant to enact such a high velocity on the eye-tracking device while wearing it. If the corners of that transformed rectangle are in the wrong order, for example when a flipped or rotated frame is detected, the result is discarded, as well. These calculations are based on the fact that the camera has an average framerate of 30.5 frames per second, so the camera movement between the frames cannot physically change by a high degree within this timeframe under normal use cases with a head-mounted eye tracker.

The exact values and calculations to perform these checks were empirically defined and are explained in more detail in chapter 3.4.3.

3.4.3 Implementation

Let T be the transformed rectangle defining the transformation, with A, B, C, D as corner coordinates.
Let T' be the transformed rectangle defining the previous transformation, with A', B', C', D' as corners.
Corner coordinates are always in the following order: top left, top right, bottom right and bottom left.

$$valid(T) = \begin{cases} 0 \; if & \left(\begin{array}{c} (|A_x - D_x| > 150) \vee \\ (|B_x - C_x| > 150) \vee \\ (|A_y - B_y| > 100) \vee \\ (|C_y - D_y| > 100) \vee \\ \\ \left(\frac{|A_x - C_x|}{|B_x - D_x|} > 1.5\right) \vee \\ \left(\frac{|A_y - D_y|}{|C_y - B_y|} > 1.5\right) \vee \\ \left(\frac{|A_x - C_x|}{|B_x - D_x|} < \frac{2}{3}\right) \vee \\ \left(\frac{|A_y - D_y|}{|C_y - B_y|} < \frac{2}{3}\right) \vee \\ \\ (\min\{B_x, C_x\} < \max\{A_x, D_x\}) \vee \\ (\min\{C_y, D_y\} < \max\{A_y, B_y\}) \vee \\ \\ \left(\begin{array}{l} |A'_x - A_x| + |A'_y - A_y| + \\ |B'_x - B_x| + |B'_y - B_y| + \\ |C'_x - C_x| + |C'_y - C_y| + \\ |D'_x - D_x| + |D'_y - D_y| \end{array} \right) > \Delta mov\_threshold \end{array} \right) \\ 1 \; otherwise \end{cases}$$

Figure 3.6: *Formula for deciding if a transformation result is accepted initially.*
Values are adjusted for a rectangle with a width of 737 pixels, and a height of 286 pixels[2] before being transformed. $\Delta mov\_threshold$ is first equal to 20 px, with another round of checks allowing a $\Delta mov\_threshold$ = 40 px.

[2] The size of this rectangle is based on the largest possible area in the undistorted image that only contains pixels that were captured by the camera. Any areas that were outside of the field of view before the undistortion process should not be included in that rectangle. Otherwise, the tracking process would produce more inaccurate results, as it would try to track the static border between the black areas and the image. This is discussed in more detail in chapter 3.3.3.

Whenever a tracking result is discarded, alternative calculations take place. Instead of comparing the current frame with the previous one, these calculations work through up to 5 nearby frames in either direction until an acceptable result is found. So, in total, up to 10 frames, other than the two that are supposed to be tracked, can be used in this process. This nearby frame is compared to both the current and the previous frame. Because the camera transformations are transitive, these transformations can be added up to calculate the transformation between the previous and the current frame.

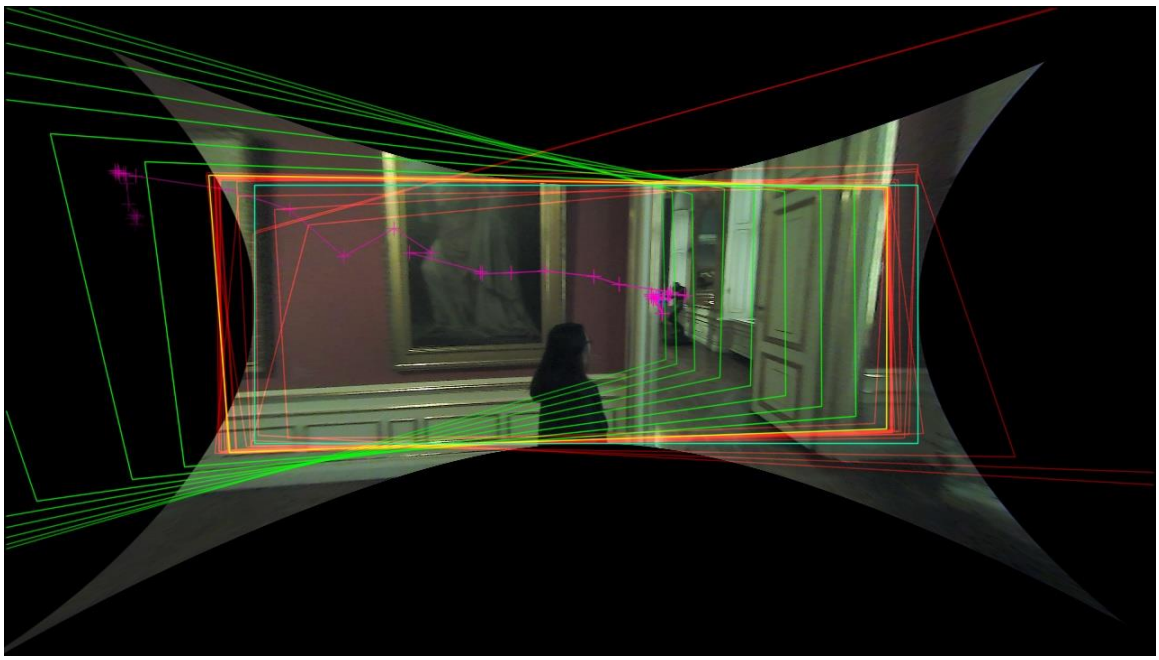

Figure 3.7: *Error detection catching multiple incorrect transformations that have shapes which are unsuitable for camera movement or deviate too much from the previous movement (red). The error correction then decides on a correct one (yellow, in the middle of most of the red ones). This can be confirmed by visually comparing the position of the gaze points to the previous frame*

If none of these results were within the threshold for accepted camera transformations with $\Delta mov\_threshold$ = 20 px, the calculations are checked again with a lower threshold of 40 px. In the case that the lowered threshold still does not produce an acceptable result, the program falls back to using the transformation from the previous frame as an approximation for the camera movement. In order to avoid too many frames in a row copying the same transformation, for every time that the previous transformation has been used consecutively, the current calculations allow a maximum $\Delta mov\_threshold$ of +40 px. That means, for example, if one transformation has been used for four consecutive frames, because for the last three frames, no transformation result was acceptable, the error detection for the next frame allows a maximum $\Delta mov\_threshold$ of 160 px.

3.5 Keyframe and region of interest (ROI)

Before the tracking takes place, the user selects a section of the video and chooses a keyframe in that section. The user can also define a rectangular ROI for the keyframe. This happens by clicking on the undistort keyframe to add a corner at that position or, if four corners have already been created, to move the closest corner to that point. Once the four corners have been created and whenever they are re-adjusted, they are used to create a quadrilateral ROI and the application validates whether a projective transformation from the keyframe to the ROI can be created. If the transformation produces a result without throwing an error, the ROI selection is accepted and will be used for translating the gaze points. A detailed explanation of that process is found in the next subchapter, 3.6.

The preview of keyframe and ROI selection is also saved, so that the user can easily interpret and visualize the tracking results using a reference image.

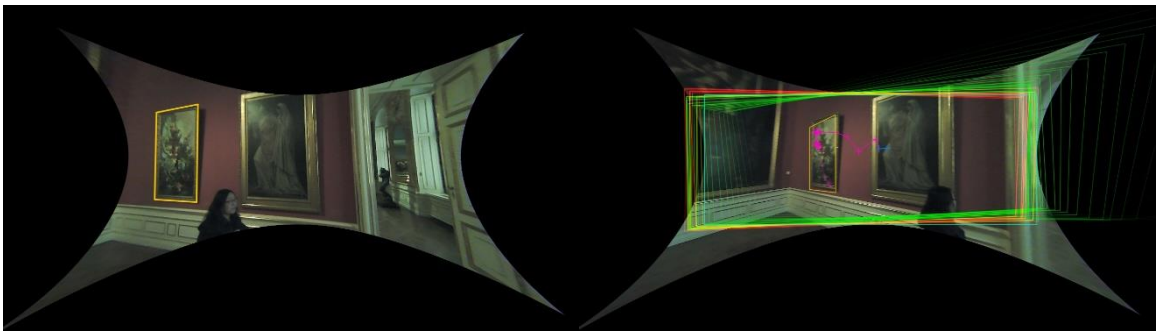

Figure 3.8: *Example of a keyframe with ROI selection, stored as preview, and a tracked frame with the transformed ROI from the same session, 38 frames after the keyframe. ROI in yellow*

For every transformation between frames, the transformation matrix that was accepted by the error detection system is added to an array. If the keyframe was already reached, then the same matrix is also applied to the coordinates of the field of view of the keyframe. These corner values are continuously changed, once per transformation, until the end of the video section. Because of the transitive nature of these matrices, the transformations from each frame to the next can simply be applied on top of each other. This way, the algorithm can easily transform the gaze points from the current frame to the keyframe. The transformation from that keyframe to its ROI, if a valid one was selected by the user, is then carried out using the ROI corner coordinates.

If the keyframe has not been reached yet, the array of transformation matrices acts as a storage, so that the transformations can be backtracked once the keyframe is reached. The array is also used to draw up to twenty previous camera directions and gaze points as visualization for the user.

Once the tracking process reaches the keyframe, all previous frame transformations are added up in reverse and applied to the gaze points that have timestamps corresponding to each previously tracked frame. These gaze points are then stored by ascending timestamp in the output file. And, if applicable, they are translated to coordinates along the ROI quadrilateral and stored in a separate file specifically created for ROI-relative coordinates.

3.6 Translating gaze points

The coordinates of the participants' gaze direction during the experiment were recorded by the eye-tracking device. However, these coordinates are relative to the video of the participants field of view, which was captured using a fisheye-lens. Therefore, the coordinates need to be undistorted in the same manner as the original video. During this undistortion process, a transformation matrix for the specific setup and resolution is created, as described in detail in chapter 3.2. This matrix then acts as a lookup table for which pixels values are used to create the resulting pixels of the undistorted view. And because this lookup table is continuous for a fisheye transformation, a straightforward algorithm to close in on the distorted gaze coordinates can be used to approximate values for the reverse of the distortion transformation. The position of the original gaze coordinate in the lookup able then directly translates to the same gaze point in the undistorted image at the frame that corresponds to the coordinates timestamp, after being scaled by the same scaling factor as the frames when they were being undistorted.

The application goes through each frame of the selected video section and calculates the transformation matrix from that frame to the keyframe. Each recorded gaze point with a timestamp that corresponds to that frame, is then undistorted, and transformed with that matrix. Now that the coordinates are relative to the undistorted keyframe image, it can be stored in the final output files.

If the user selected a rectangular region of interest in the keyframe, the resulting gaze coordinates also will be transformed with a matrix calculated with OpenCV's perspective transformation tools and stored in a separate ROI-relative output file. An ROI-relative coordinate of (0, 0) would be at the top-left corner of the ROI, while (1, 1) would be in the bottom right. This means, any coordinates with x and y values both between 0 and 1 represent the participant looking directly at the region, while any other coordinates are outside of that region.

Because projective transformations in the undistorted pinhole camera model preserve collinearity [Kannala and Brandt, 2006], those gaze points can be directly transformed to a coordinate system along a 2-dimensional ROI. So, the resulting ROI-relative coordinates are independent of the eye tracker movement in the 3D-space and can be easily interpreted and visualized. For example, a simple heatmap of the participants gaze can be created using this data and a photograph of the original region of interest.

The final coordinates are outputted in one or two files formatted for Eyetrace, depending on whether the user selected an ROI. This format was chosen to facilitate further analysis of the data with the Eyetrace tool, which bundles "a variety of different evaluation methods for a large share of eye trackers supporting scientific work and medical diagnosis." [Kübler at al., 2015].

3.7 Performance and storage requirements

During the tracking, the system-wide processor usage was constantly used to capacity. On average, the application used about 50% of the system's processor performance. This was tested on a computer running Windows 10 with an Intel® Core™ i5-7300HQ CPU, at 2.50GHz with 4 Cores. The performance on other systems may vary greatly.

With these resources, extracting one minute of original video, and storing those 1830 frames, takes about 140.7 seconds. When the distortion values are initialized, undistorting these frames takes another 516.1 seconds, and tracking without precalculated matrices takes about 34 seconds per frame on average, which adds up to 17 hours and 17 minutes for one minute of original footage. If the distortion values are already cached, the tracking speed generally increases, and one frame takes between 2 and 60 seconds to track. The performance of this process now highly depends on the error correction. When the frames are easy to match, only one cached transformation needs to be loaded per frame. However, if the error detection repeatedly rejects tracking results, a lot of calculations are still needed in order to provide the most accurate results possible.

Over a long-term tracking session of more than 240 minutes, the application used up to 190 MB of memory on a computer that has a total of 16 GB RAM. When simultaneously starting the video extraction and undistortion processes, the maximum memory usage reached 203.5 MB. After additionally requesting several gaze point previews, the memory usage reached a peak of 254.8 MB. These values were determined with the application running in debug mode via Visual Studio using a source video that has a size of 2.39 GB.

The current implementation of this algorithm stores usually three images per processed frame in JPEG-format with the same resolution as the source video. On average, one minute of source video, encoded in mp4-format with a resolution of 1280 x 720 pixels, has a size of about 2.8 MB and contains 1830 frames. If all of those frames are completely processed, each frame is stored once when extracted, once after being undistorted and once again for each tracking pass, meaning 5490 images would have to be saved. Together, these take up about 900 MB of storage space, with the exact value depending on the source video content. It is noted that the exact size varies, depending on the video content, because the images are compressed when stored. In addition to the images, the application also saves varies data in different files and stores individual previews of frames with gaze points if the user enables the preview. The storage space required for these files is negligible compared to the space needed for processing the video frames.

4 Experiment

4.1 Setup

An experiment designed by [Santini et al., 2018a] developed an eye-tracking system that was provided to visitors in the Austrian gallery Belvedere on January 23, 2018. As part of the study, their eye movements and field of view were recorded by a Pupil Labs eye-tracking device, which sampled about 2.06 gaze points per frame on average and recorded the video with 30.5 frames per second in HD resolution. The data was recorded with EyeRecToo [Santini et al., 2017a], using PuReST [Santini et al., 2018b] to tack the pupil and CalibMe [Santini et al., 2017b] for calibration.

For this work, the distortion and calibration data, videos of the participant's field of view and each of their eye movements, as well as timelines of the coordinates of their gaze relative to that field of view and timestamp data for synchronization for sixteen participants were provided. Each session lasted between approximately 9 and 29 minutes. In addition to this data, manual annotations for parts of the session were provided by the University of Vienna. These files contain a list of 48 distinct painting identifiers and in in which timestamp ranges the participant looked at them. They also distinguish if the participant was looking at markers near the paintings or at other people who were walking through the museum, sometimes in front of the paintings.

In the next section, I am comparing the results of the tracking algorithm with these manual annotations in order to determine how reliable the algorithm is. Annotations were only provided for sessions s009 and s011, which span approximately 15 minutes of video material. As described in chapter 3.7, the algorithm takes over half a minute to process each frame. Therefore, for this work, a section of 2586 frames was tracked, analyzed, and compared to the annotation data.[3]

To test the tracking algorithm, section, keyframe and ROI had to selected manually for each painting. The identification codes from the manual annotations also have to be assigned to the paintings seen in the videos. The results that are presented and analyzed in this work show the reliability of the tracking algorithm and gaze point transformation.

[3] This section contains the frames 8415 - 11000 of session s009 in the Belvedere experiment described above, which corresponds with the field timestamps 905708 - 992332. It was randomly chosen before processing the data or viewing the video, in order to provide unbiased data for the analysis.

4.2 Results

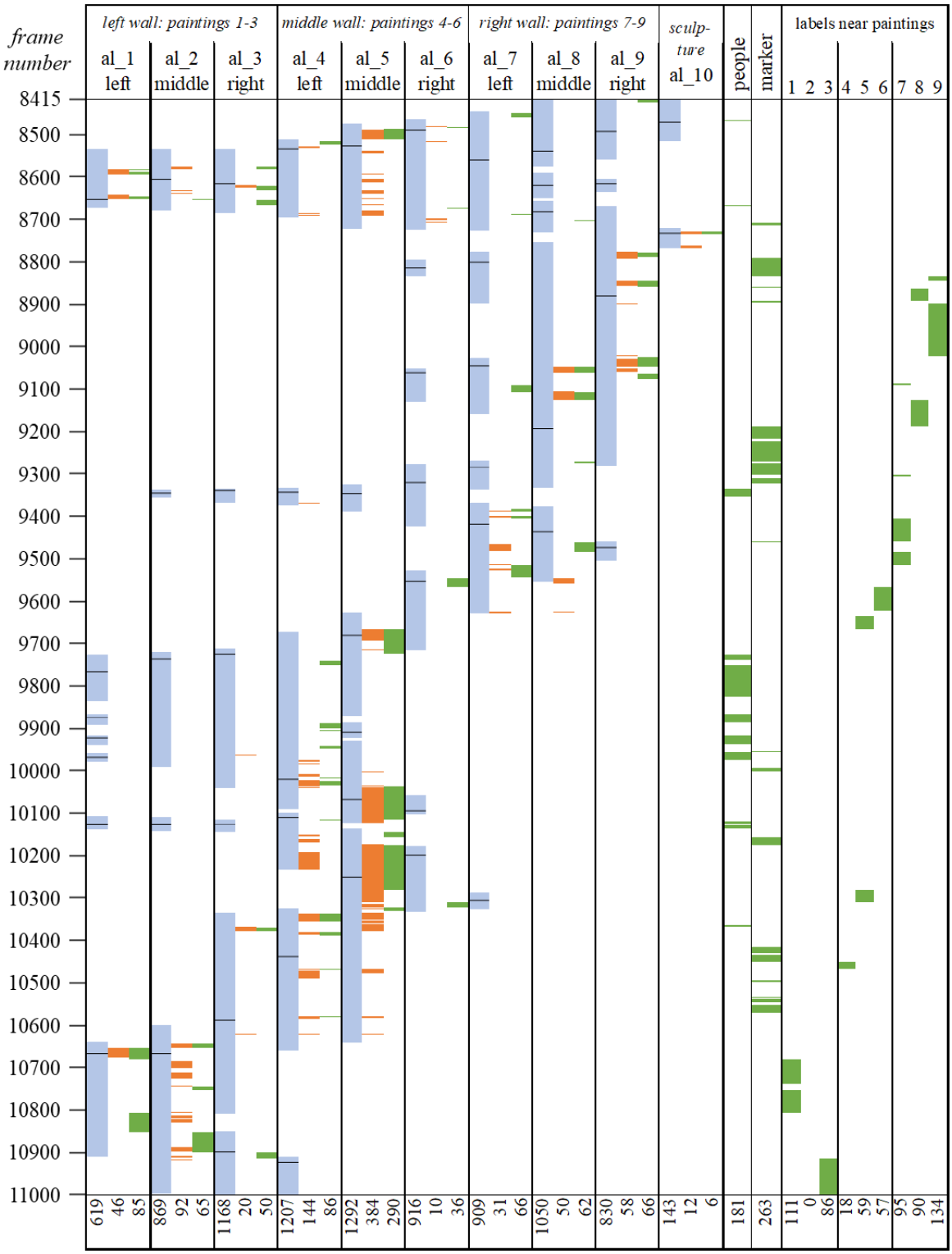

Figure 4.1: *Comparison of the tracking algorithm with manual annotations.*

The individual tracking sections are shown in light blue, with each keyframe location highlighted in black. Orange areas illustrate when the participant looked directly at a certain object according to the tracking results. The manual annotations, displayed in green, show in which frames the participant was in fact looking at an object, people, marker, or artwork description. Numbers in the bottom row indicate how many frames fulfil the conditions for each selection. For example, for the sculpture, two sections were defined, which contain 143 frames in total. Tracking results show that in 12 of these frames, which are all located in the second section in this case, the participant was looking directly at the sculpture. However, according to annotations, this happened only in 6 frames.

4.2.1 General properties of the results

54 tracking sections were chosen, based on when any part of each of the 10 objects was visible in the camera's field of view. Within each section, a keyframe has been picked in a way that the four ROI corners could easily be located to select the ROI. After the sections were tracked individually, it was determined when the participant's gaze was directed at which object by interpreting the ROI-relative tracking results. This information can be directly compared to the manual annotations which provide the ground truth for analysis. The annotations also included descriptors for people, markers on the walls, and labels to describe the galleries artwork which were located on the wall outside of the object's region. This extra information is displayed in the green columns on the right part of the diagram and was not used for analysis, as no correlating information was produced with the tracking process.

Overall, 2586 frames were analyzed by tracking ten distinct objects. Most of the frames were tracked multiple times within different sections. In that case, the cached transformation calculations could be reused for better performance as described in detail in chapter 3.3.3.

When tracking the 54 sections, a total of 9003 frames was processed. According to the results, in 847 of these frames, the gaze point was on one of the objects. The annotations indicate 812 frames on one of the objects, 7 of which were not within the selected tracking sections.

When comparing the tracking results within the sections with the manual annotations for those frames, 436 frames are true positives, 411 false positives, 376 false negatives and 7780 true negatives.

This results in a precision (positive predictive value) of 51.48% and a recall (true positive rate) of 53.69%. The true negative rate is 94.98%, balanced accuracy is at 74.34%. The tracking algorithm produces positive results at a predicted positive condition rate of 9.41%.

The results contain the information of the gaze data from each section, transformed to the coordinate system of the section's keyframe, which amounts to 20 956 gaze point coordinates. All these values are additionally stored as coordinates relative to the keyframe's ROI in separate files. Because the manual annotations only indicate when the gaze points were inside of the ROI, without providing exact coordinates, the analysis in the next chapter will mostly focus on the interpretation of the results in a dichotomic way when being compared with the annotations.

It is noted that all percentage values were rounded to two decimal points and are based on the results of the experiment described above. The sections were selected in a way that they start as soon as any part of the object enters the camera's field of view and end when the object has completely left that field of view. Different circumstances, input data or strategies for section and keyframe selection may lead to rates that vary from these values.

4.2.2 Examples for the position of ROIs near the end of a section

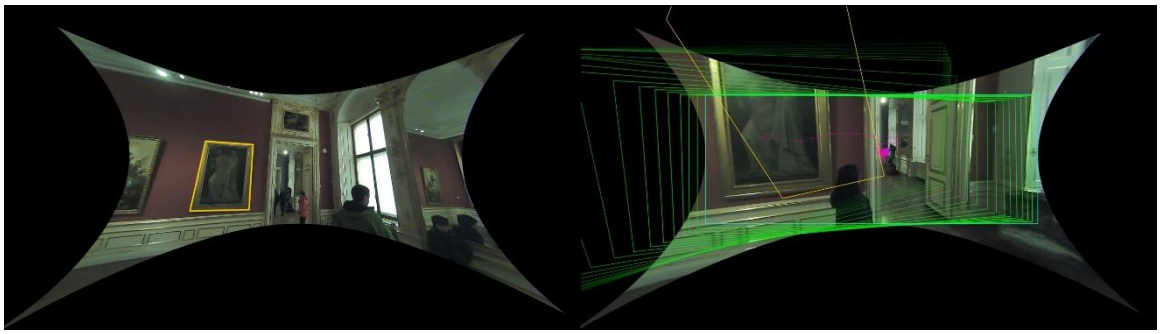

Figure 4.2: *Initial ROI selection and position of the same ROI during tracking, 275 frames after the keyframe. The ROI gets distorted but remains mostly in the same place.*

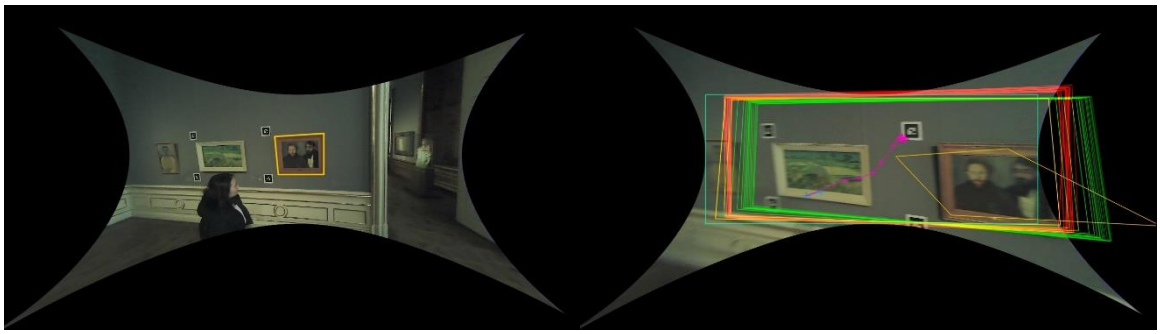

Figure 4.3: *Initial ROI selection and position of the same ROI during tracking, 393 frames after the keyframe. The ROI gets distorted but remains mostly in the same place.*

Most of the sections from this experiment were tracked this way. The ROI got distorted, but still remained in the same area after several dozen to a few hundred frame transformations.

Here are a few examples of the error detection failing and providing the algorithm with wrong transformation matrices, which leads to the ROI moving to a wrong place:

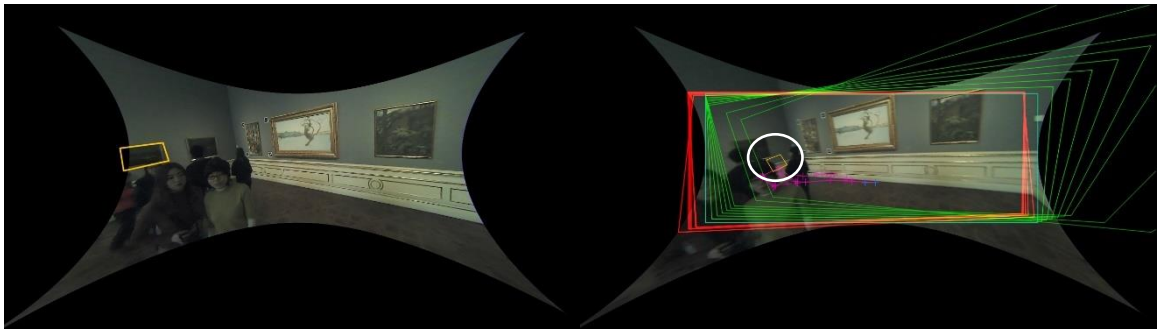

Figure 4.4: *Initial ROI selection and position of the same ROI during tracking, 9 frames after the keyframe. The ROI moved to an incorrect place almost immediately due to tracking errors. The ROI in the right image is highlighted with a white ellipse.*

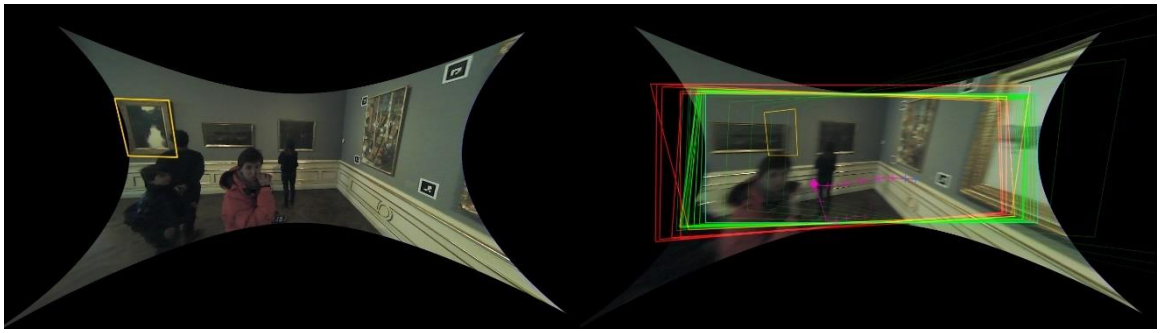

Figure 4.5: *Initial ROI selection and position of the same ROI during tracking, 17 frames after the keyframe. Another example of it quickly moving away from the original location due to tracking errors.*

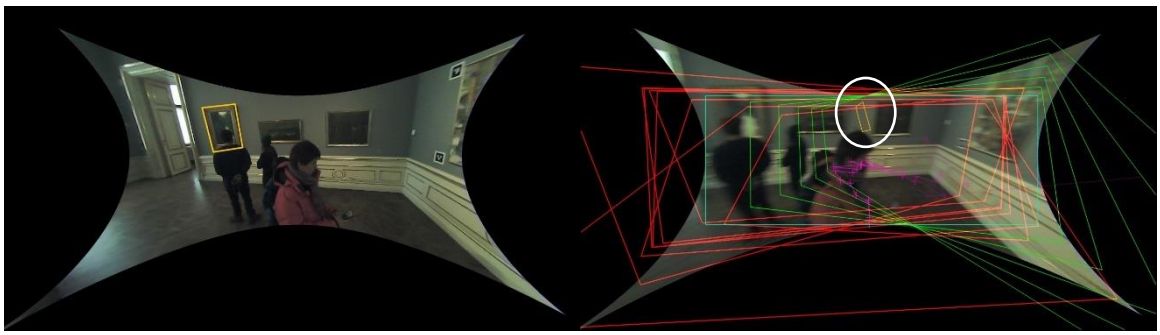

Figure 4.6: *Initial ROI selection and position of the same ROI during tracking, 9 frames after the keyframe. Many wrong transformations are rejected, and the error correction system repeats the same transformation several times in a row. Unfortunately, this transformation contains fast rotational movement (see green polygons), therefore the ROI moves to a completely wrong location within a few frames. The ROI in the right image is highlighted with a white ellipse.*

5 Discussion

5.1 Drifting of ROI and gaze points

Whenever a frame transformation is calculated, there is a possibility that the result of that calculation is wrong. That means, the longer a section is and the further away the section's keyframe is, the more calculation errors accumulate. This can lead to incorrect output data, for example when the selected ROI drifts away from the moving image that it should be anchored to.

The effect of this error accumulation can be seen in the visualization of ROIs at the end of each section, shown in chapter 4.2.2.

Several measures take place to minimize this effect as much as possible, for example the error detection described in chapter 3.4, image enhancements from chapter 3.3.2 and the visualization which provides a way for the user to easily review the tracking results during and after the tracking process.

As can be seen in the results, these strategies are not sufficient for providing a reliable algorithm that does not have to be supervised. Depending on the accuracy that is needed from the resulting data, it is suggested to choose keyframes towards the middle of their section and to select shorter sections for tracking, so that keyframe and ROI are regularly redefined with accurate values.

Other general approaches to improve this algorithm will be discussed in chapter 6.

5.2 False positives in error detection

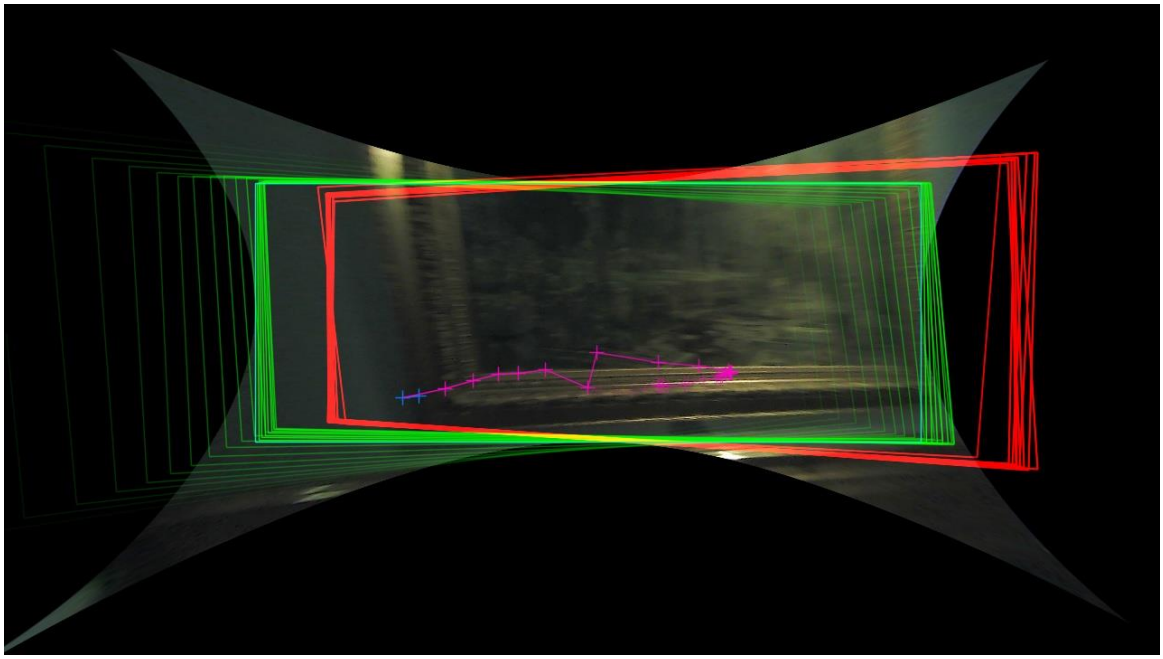

Figure 5.1: *Example or error detection wrongly rejecting correct tracking results and copying the same transformation for several consecutive times.*

Setting the movement threshold for the error detection algorithm usually results in a compromise between performance, stable tracking for longer sections and preventing drifting of ROI and keyframe locations. If the threshold is high, the error detection is more tolerant towards anomalous tracking results. This means that the first result is accepted more often, resulting in better performance and a higher probability of unusual camera movement being accepted, if it has been detected correctly.

A lower threshold on the other hand prevents radical changes in camera movement, which usually keeps the ROI from suddenly being in a completely wrong location and assumes a more predictable camera movement. However, this also means that sometimes a correct tracking result is rejected when it does not fall below the threshold. This can lead to repeatedly copying a previous transformation instead of using more correct tracking result, which means the location of ROI and keyframe slowly drifts away from its correct position in some cases.

Specific approaches for a more sophisticated error detection will be discussed in more detail in section 6.2.

5.3 Effect of various circumstances on tracking quality

The quality and accuracy of tracking varies greatly, as it highly depends on the quality of the image captured by the eye tracker's camera. If the participant barely moves, the transformations are quickly calculated and accepted and do not cause any major drift in the ROI position relative to the image. However, if the participant's head moves quickly, it produces a less clear image due to motion blur, therefore making it more difficult to detect distinct keypoints and calculate movement. It also might cause problems with the error detection, as it accepts calculations based on previous camera movement. Depending on the implementation of the movement threshold, if the movement changes to rapidly, the correct transformation could be falsely flagged as incorrect and the previous movement result is repeated. This is especially problematic when the actual movement is quite different to previous one, further facilitating the drifting of the ROI location from the correct location in the image.

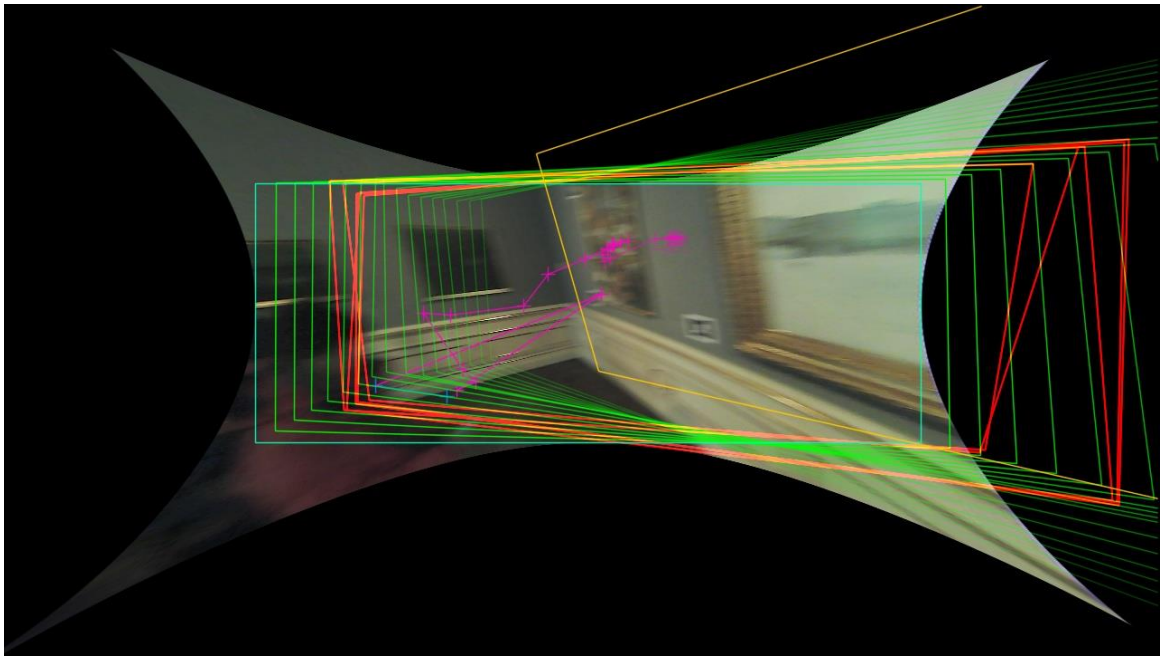

Figure 5.2: *Example of a frame, where the camera was moved quickly. The image is blurred and harder to track and the error detection does not accept any transformation with this much movement (red), so it continuously copies a previous, slower transformation (green). This causes the ROI (yellow) to shift away from its original location (the bright painting with a golden frame) and lags behind the movement of the image content.*
A higher movement threshold would cause the correct transformations (red) to likely be accepted, but would also mean that wrong transformations at other frames are also more likely to pass through error detection and move the ROI to a completely wrong location.

During the part of the experiment that is analyzed in this work, rarely any people walked in front of the camera while the participant was looking directly at an ROI. Only 9 frames were identified, in which the participant looked at an ROI according to the tracking, while the annotated data simultaneously indicated that the participant was looked at another person.

In a busy environment, where it is expected that people walk in front of the participant more often, the contradicting movement in the video will not only lead to less accurate results, but the results also do not indicate whether the gaze toward the ROI was interrupted by anything between the object and the participant. The coordinates are simply transformed to the ROI position and visual information from the video is only used for the tracking itself. The possibility of adding a visual comparison of the gaze information in the video to the tracked object is further explored in chapter 6.3.

5.4 Reliability and supervision

The results presented in this work demonstrate that the approach to track data collected with a head-mounted eye tracker to a static reference frame is feasible and produces results that correlate with manually gathered data.

However, the reliability of this algorithm still has possibilities to be improved. When sections are selected to be as long, as the objects are visible in the frames, the program in its current version does not produce results that are reliable enough for automated processing, which is also indicated by the 51.48% precision and 53.69% recall values that were calculated from the analyzed results.

The data produced by the current version of the program requires supervision and should be confirmed at least by checking the tracking visualization before being further used in scientific analysis. If the movement of gaze points and tracking polygons correlates with the movement of the image content and if the ROI stays at the same position relative to the captured video, the data can be accepted for further analysis. Otherwise, it is recommended that the section is split up and tracked with more manually defined keyframes.

Possible approaches and strategies to improve these results in the future are discussed in the following chapter.

6 Further Investigations and Outlook

6.1 Experiment setup and environmental conditions

In this work, only the data from video sections that had manual annotations associated with them were analyzed to determine the reliability of the algorithm. These videos were all shot on the same eye-tracking device with similar lighting conditions in the same space. The algorithm was developed accordingly, in order to achieve reliable results with this type of data. It may produce different and less accurate results in different environments and lighting conditions. Therefore, the algorithm should be tested with supervised data before applying it to data from experiments with another setup or different conditions, in order to confirm that it works reliably in those conditions. Some parameters may need to be adjusted to particular conditions, especially settings related to undistortion and keypoint detection.

If the eye-tracking device is equipped with a camera of higher quality, it could also aid in tracking, since noisy and blurred images are a major cause of tracking errors. In the preprocessing step, the enhancement calculations also facilitate the amplification of noise. Another filter, to reduce this noise and only enhance high-contrast edges might also help with tracking accuracy. And, if the saturation of the image is enhanced, the color channels could be tracked separately to combine their results. This would be especially useful in colorful environments, where the captured video has a high color contrast.

6.2 Error detection and correction during tracking

The algorithm used in this work utilizes basic error detection and tries to correct wrong tracking calculations. It does this by combining the tracking results with other frames nearby if the direct comparison of two frames provides inconclusive results, as described in chapter 3.4. This basic error detection could be improved with more sophisticated approaches, which would decrease the frequency of occurrence of tracking errors and improve overall tracking reliability, resulting in more accurate data. Right now, it is recommended that the results of this algorithm are supervised, as they occasionally drift into a wrong direction, especially for longer tracking sections.

If the speed of the algorithm is negligible, each frame comparison could be done several times using different parameters or more sophisticated tracking algorithms, also taking the current velocity of the camera into account. When these results are compared to each other, only transformations matching with several others could then be accepted as final tracking result between the frames. This approach would minimize the chance of an error occurring but would also increase the speed of the algorithm by about an order of magnitude, with the exact increase depending on the exact implementation. In addition to this drawback, issues can arise when the individual calculations have multiple equally improbable results and cannot reliably agree on one transformation. In this case, the error cannot be corrected automatically, and a human decision might need to be made, if the error rate has to be kept as low as possible.

To mitigate the effect of errors on the rest of the video section and improve overall accuracy and reliability, a new type of frame could be introduced (m-frame). Calculating the transformation of m-frames towards the keyframe takes more time than a conventional tracking calculation, because it is checked with multiple tracking variations against the keyframe and other m-frames, so that their positions are known with a high certainty. The m-frames should appear regularly, with the interval between them being dynamically chosen. If a certain frame cannot be tracked well, for example because it is blurry or overexposed due to a camera flash being fired at the same time, another frame nearby should be chosen as m-frame, as to not jeopardize the accuracy of m-frames. All frames between m-frames will be calculated in the quick, conventional way and may just be checked for inconsistencies against the upcoming m-frame. If an undetected tracking error occurs in those calculations, it will be limited to that section and repaired with the next m-frame. This approach would greatly increase the reliability of the algorithm, without disproportionally compromising performance.

6.3 ROI detection

In the current implementation, the user has to manually select a section, keyframe and ROI for each object they are interested in. In addition, no checks are carried out, which control whether the ROI location still shows anything resembling the originally selected region. So, if the experiment takes place in a space with a complex 3-dimensional configuration, different ROIs might overlap each other, even if they are not visible in the real world at the same time. This happens, for example, in the setup discussed in this work, when a participant moves to another room in the gallery. In this case, every room needs to be processed in an individual video section. So, every ROI may need to be selected multiple times within one video session.

This repetitive task could be aided by the same computer vision algorithms that are used for the tracking itself. The application could then compare the selected keyframe to other, possibly randomly selected frames in a different part of the video or in a different video from the same experiment. If it finds similarities it can re-identify the paintings, for example, when a person visits the room again a few minutes later, several thousand frames apart. In that case, a researcher would only need to select each painting once for the whole experiment.

This matching can then also be used to compare the area around the gaze point in the video to the keyframe's ROI. If those areas do not match well enough, then the view to that point might be obstructed, for example, by a person walking in front of the object. This observation should be included in the gaze point results, so that the user can work with more comprehensive results.

Also, the accuracy of ROI selection could be improved with minor corrections using edge detection or by zooming into the point that the user selected, so they can make fine adjustments to the exact position of the ROI corners.

6.4 Further UI and stability improvements

The graphical user interface of the application could be further improved to be more intuitive for non-technical users. In the current version, the user is guided through the process with descriptions and buttons, but they can skip any processing steps that might be already completed. In a future version, those operations should be checked before they can be skipped to avoid errors later in the process, for example if frames for the chosen section have not been undistorted.

A file management system for loading videos and configuration data from a chosen directory would also make the application easier to use. It could also help prevent experiment data and precalculated values being incorrectly associated with videos from another session, in case the user forgets to copy or move the files from a previous session. Configuration and tracking settings may even be stored to pause and continue the process without having to select section, keyframe and ROI again.

The code of the application itself could also be designed in a more robust manner, to better handle exceptions and edge cases. In most cases, if an error appears due to an unconventional use case, improperly formatted input data or missing files, the application just cancels the calculations, informs the user, and stores details in a logfile. Instead, the application should analyze these exceptions and either try to continue with the processes that are possible or give the user more comprehensive information about what needs to be done in order to fix the issue.

6.5 Different approaches for calculating the camera position

Currently, ROIs should be of a two-dimensional object, for example a wall or painting, complex shapes or objects, that can be viewed from multiple directions are difficult to track. Such an object may need to be defined with multiple ROIs, video sections showing them would have to be divided up into smaller sections for different viewing directions.

In a more sophisticated approach, stereo cameras on the eye tracker can create a three-dimensional point cloud of the space around the participant [Badino and Kanade, 2011]. This model can then be used to recreate the shapes of objects and scene and to estimate the current camera position in that space. The gaze data from the eye-tracking device can then be used to calculate the exact point that the participant is looking at.

With another approach, the eye-tracking device could be equipped with inertial sensors when setting up the experiment. This way, the position and orientation can be calculated using only the sensors, without having to process and track the video. However, the estimates based on inertial sensor data "suffer from integration drift over longer time scales" [Kok et al., 2017]. If the data is combined with the visual tracking information, the sensors can provide general movement data and visual information would only be used to minimize the value drift that occurs with inertial sensors and correct the results in a small scale. This would prevent the algorithm from accepting tracking transformations that differ too much from the correct result, and greatly improve the accuracy and reliability of the algorithm.

References

[Badino and Kanade, 2011] Badino, H., and Kanade, T. (2011, June). A Head-Wearable Short-Baseline Stereo System for the Simultaneous Estimation of Structure and Motion. In *MVA* (pp. 185-189).

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006, May). Surf: Speeded up robust features. In *European conference on computer vision* (pp. 404-417). Springer, Berlin, Heidelberg.

[Bouguet, 2015a] Bouguet, Jean-Yves (2015). Camera Calibration Toolbox for Matlab. Retrieved February 25, 2019 from http://www.vision.caltech.edu/bouguetj/calib_doc/

[Bouguet, 2015b] Bouguet, Jean-Yves (2015). Camera Calibration Toolbox for Matlab. A few links related to camera calibration. Retrieved February 25, 2019 from http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/links.html

[Bykowski and Kupiński, 2018] Bykowski, A., and Kupiński, S. (2018, June). Automatic mapping of gaze position coordinates of eye-tracking glasses video on a common static reference image. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research and Applications* (p. 84). ACM.

[De Beugher et al., 2012] De Beugher, S., Ichiche, Y., Brône, G., and Goedemé, T. (2012, September). Automatic analysis of eye-tracking data using object detection algorithms. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (pp. 677-680). ACM.

[Github, 2019a] Github, Inc. (2019). Opencv/opencv_contrib: Repository for OpenCV's extra modules. Retrieved February 25, 2019 from https://github.com/opencv/opencv_contrib

[Github, 2019b] Github, Inc. (2019). Fisheye camera model for calibration and stereo rectification #2889. Retrieved February 25, 2019 from https://github.com/opencv/opencv/pull/2889#issuecomment-54018497

[Kannala and Brandt, 2006] Kannala, J., and Brandt, S. S. (2006). A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, *28*(8), 1335-1340.

[Kiefer et al., 2012] Kiefer, P., Straub, F., and Raubal, M. (2012, March). Towards location-aware mobile eye tracking. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (pp. 313-316). ACM.

[Kok et al., 2017] Kok, M., Hol, J. D., and Schön, T. B. (2017). Using inertial sensors for position and orientation estimation. *arXiv preprint arXiv:1704.06053*.

[Kübler at al., 2015] Kübler, T. C., Sippel, K., Fuhl, W., Schievelbein, G., Aufreiter, J., Rosenberg, R., Rosenstiel, W. and Kasneci, E. (2015, January). Analysis of eye movements with eyetrace. In *International Joint Conference on Biomedical Engineering Systems and Technologies* (pp. 458-471). Springer, Cham.

[MacInnes et al., 2018] MacInnes, J. J., Iqbal, S., Pearson, J., and Johnson, E. N. (2018). Wearable Eye-tracking for Research: Automated dynamic gaze mapping and accuracy/precision comparisons across devices. *bioRxiv*, 299925.

[Muja and Lowe, 2009] Muja, M., and Lowe, D. (2009). Flann-fast library for approximate nearest neighbors user manual. Computer Science Department, University of British Columbia, Vancouver, BC, Canada.

[OpenCV, 2019] OpenCV team (2019). About - OpenCV library. Retrieved February 25, 2019 from https://opencv.org/about.html

[Pizer et al., 1987] Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., Greer, T., ... and Zuiderveld, K. (1987). Adaptive histogram equalization and its variations. Computer vision, graphics, and image processing, 39(3), 355-368.

[Pontillo et al., 2010] Pontillo, D. F., Kinsman, T. B., and Pelz, J. B. (2010, March). SemantiCode: Using content similarity and database-driven matching to code wearable eyetracker gaze data. In *Proceedings of the 2010 Symposium on Eye-Tracking Research and Applications* (pp. 267-270). ACM.

[Santini et al., 2017a] Santini, T., Fuhl, W., Geisler, D., and Kasneci, E. (2017). EyeRecToo: Open-source Software for Real-time Pervasive Head-mounted Eye Tracking. In *VISIGRAPP (6: VISAPP)* (pp. 96-101).

[Santini et al., 2017b] Santini, T., Fuhl, W., and Kasneci, E. (2017, May). Calibme: Fast and unsupervised eye tracker calibration for gaze-based pervasive human-computer interaction. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (pp. 2594-2605). ACM.

[Santini et al., 2018a] Santini, T., Brinkmann, H., Reitstätter, L., Leder, H., Rosenberg, R., Rosenstiel, W., and Kasneci, E. (2018, June). The art of pervasive eye tracking: unconstrained eye tracking in the Austrian Gallery Belvedere. In *Proceedings of the 7th Workshop on Pervasive Eye Tracking and Mobile Eye-Based Interaction* (p. 5). ACM.

[Santini et al., 2018b] Santini, T., Fuhl, W., and Kasneci, E. (2018, June). PuReST: robust pupil tracking for real-time pervasive eye tracking. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research and Applications* (p. 61). ACM.

[Smith and Cheeseman, 1986] Smith, R. C., and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. The international journal of Robotics Research, 5(4), 56-68.

[SURF, 2019] SURF: speeded up robust features. (n.d.) Retrieved February 25, 2019 from http://www.vision.ee.ethz.ch/~surf/download.html

[Takemura et al., 2010] Takemura, K., Kohashi, Y., Suenaga, T., Takamatsu, J., and Ogasawara, T. (2010, March). Estimating 3D point-of-regard and visualizing gaze trajectories under natural head movements. In *Proceedings of the 2010 Symposium on Eye-Tracking Research and Applications* (pp. 157-160). ACM.

[Toyama et al., 2012] Toyama, T., Kieninger, T., Shafait, F., and Dengel, A. (2012, March). Gaze guided object recognition using a head-mounted eye tracker. In Proceedings of the Symposium on Eye Tracking Research and Applications (pp. 91-98). ACM.

Erklärung

Hiermit erkläre ich, dass ich diese schriftliche Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe.

Ort, Datum Unterschrift