Video stabilization algorithm from low frame rate video for hyperlapse applications

Kim Tengbom Zetterman
Björn Hansson

Master's thesis
2015:E14

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Video Stabilization Algorithm from Low Frame Rate Video for Hyperlapse Applications

Kim Tengbom Zetterman
Björn Hansson

June 1, 2015

Abstract

There are several methods that one can use to visualize image sequences. One such method, called timelapse, is based on synthesizing a video from the image sequence. One sub category of timelapses is the so-called hyperlapse, which is defined as a timelapse with a camera movement over great space. A problem with combining camera movement with speeding up the frame rate per second is that camera shakes appear magnified. One way to minimize this problem is to stabilize the video, using estimated relative camera movement. Such estimates can be obtained using computer vision methods based on epipolar geometry. Choosing how to compensate for camera shakes and calculate a new, more smooth camera path is essential to the video stabilization algorithm. One aim of this thesis is to create such a video stabilization algorithm. Another aim is to examine how performance degrades with decreased frame rate for the input sequence. Along with this thesis we have collected a set of benchmark image sequences. Several different video stabilization algorithms have been developed in the project. These have all been tested on the benchmark data sets and evaluated with promising results.

Contents

Chapter 1

Introduction

1.1 Motivation

Narrative has developed a wearable camera that takes an image twice every minute [22]. This is a different and a convenient way to get your life documented and to capture the precious moments without having to bring forward your camera or mobile phone. These images can then be displayed in a slide show. However, if the images were captured with a higher frame rate than twice every minute, would it be possible to present them as a short video that still is pleasant to look at? This is what we are trying to determine in this thesis. A video camera could of course be used and then the video could be sped up, but a video camera requires more from the hardware in the camera. Since one of the many advantages with the product is that it is small we want to investigate if it is possible to get a nice summary of a longer sequence of events without having to enlarge the memory and battery.

Within the video and movie making industry there has for a long time been a need for creating shorter video sequences from longer sequences of events. A technique that is called timelapse can be used in situations when presenting a longer sequence of events using a shorter video. Timelapses are basically created speeding up the video by, for example, keeping every tenth frame resulting in a ten times faster/shorter video than the original sequence of events. If walking when doing a timelapse, or in another way moving the camera over great distances, it is formally a hyperlapse as according to the following definition:

> *Hyperlapse is an exposure technique in time-lapse photography, in which the position of the camera is being changed between each exposure in order to create a tracking shot in timelapse sequences. In contrast to a simple motion timelapse – dolly shots, which are realized with short camera sliders; in hyperlapse photography, the camera is being moved through very long distances.*
>
> *-Wikipedia, May 2015*

A hyperlapse in its naive implementation often is not pleasant to look at if the video is recorded while, for example, walking since every camera shake appears amplified. By the use of different techniques to take care of the camera shakes it is in many cases possible to present a smooth video. Hyperlapses used to

require a lot of planning before making. Nowadays, for example Microsoft and Instagram are among those who have taken the step of making the hyperlapse more automated and thus can be made without planning ahead while filming, see section 1.2. The drawback with these implementations is that these previous algorithms is built to require an entire video (often somewhere between 24 to 30 frames per second) as input. In contrast we want to develop an algorithm that uses as few input frames as possible and also try to determine what is the lowest possible frame rate that can do.

1.2 Related Work

Microsoft's paper on hyperlapse videos [14] was published in 2014 from material taken with a GoPro camera. They manages to obtain very nice results. Using all information in every frame, they build a 3D-representation of the scene and then find a new way through this 3D-representation that is smooth and passes near all of the input frames. Eventually they finish by stitching the frames to new ones that will correctly capture what would have been on the video had it been taken carefully on the new path. Though the method manage to obtain very nice results, it costs a lot in computation making it unsuitable for situations where computation power is limited.

A hyperlapse-app from Instagram was released in 2014 and this works quite well for handheld cameras. It is similar to already existing video stabilization techniques in the sense that it warps each frame but instead of relying on image analysis to determine the rotation of each frame it uses the camera's built-in gyroscope. If the video is too shaky the output will not be good since it crops the frames instead of stitching new ones together as Microsoft does, see [13] and [15].

EgoSampling [24] is a technique that after skipping the static parts in a video (for example standing still while waiting for a red light) selects the frames that are in a similar viewing direction which gives a result that looks as if the camera was on rails. This was published in December 2014. One drawback is that the number of output frames is not fixed. Another limitation is the handling of videos when the wearer is static in long periods.

Auto-directed video stabilization with robust L1 optimal camera paths [9] is a way of finding a smooth camera path by formulating the problem as a minimization of the first three derivatives and solve it using linear programming. Since it uses cropping of the frames some information may be cut out which may not be appreciated by the user. The algorithm was published in 2011.

Fitting B-Spline Curves to Point Clouds by Curvature-Based Squared Distance Minimization [27] is a method to fit B-spline curves [28] to a set of unorganized and possibly noisy data points that is faster in convergence and more stable than earlier methods.

In Video stabilization using epipolar geometry [8] the authors propose a video stabilization algorithm, that rather than doing a full 3D reconstruction, does a projective reconstruction using relations between epipolar lines and image points. Basically they trace points in different images through a video sequence then smooth the resulting trajectory using a Gaussian kernel. Eventually they estimate fundamental matrices relating the original trajectory and the smooth one. The result is an algorithm with the physical correctness of a full 3D re-

construction but with the computational costs and robustness of a simple 2D reconstruction.

1.3 The Contributions of This Thesis

There are a lot of different solutions to the problem of presenting a video stabilization algorithm. The common feature is that the different approaches assume a much higher input frame rate than this thesis is aiming for, as mentioned in Section 1.1. In other words we aim to contribute in the field of low input frame rate video stabilizing and also, by doing this, examine how low frame rate we can handle without failure of our algorithm.

We present a stabilization algorithm that mainly consists of three parts; estimating the epipolar geometry, calculating a new camera path and transforming the image sequence as to match the new camera path. This outline is, generally speaking, common to the work of for example [14] and [8].

Regarding the reconstruction of the original camera movement using epipolar geometry a few different approaches have been proposed. Microsoft, [14] are using a full 3D reconstruction while [8] do something that can be considered falling between 2D and 3D reconstruction. The method used by [13] is not based on epipolar geometry at all but instead uses gyroscope and accelerometer to reconstruct the original camera movement. This thesis does a full 3D reconstruction but uses subspaces to the reconstructed 3D space to get a simple and intuitive feel for the algorithm.

In order to calculate a new camera path there have also been some different suggestions, such as proposed by [14], [24] and [13]. Since Microsoft to a greater extent are aiming for manipulating frames they can be more free to choose an optimal camera path. In other work a minimization of movement is behind the camera path estimation. Our work is based on fitting different degree polynomials to our data set, consisting of estimated camera centers. Then using minimization over the L2 norm to find the closest point on the curve for each camera center.

Transforming images are mainly based on image warping in many stabilization techniques, see [8] and [13]. Microsoft uses a different technique where they stitch a series of different images together, see [14]. One other suggestion is to just choose the final video to consist of frames with similar viewing orientaion, see [24]. Our work is based on projective transformation to simulate a new view orientation.

Chapter 2

Theory

In this chapter we will give an overview of the theory essential to this thesis. The most fundamental parts of the theory are regarding *epipolar geometry* and how to describe images mathematically. Epipolar geometry captures the geometric relationship between pairs of matched images.

After describing the basic theory we touch upon more thesis specific theory. We describe how to relate a lot of images to each other when having done pairwise matching. Also we describe how to do *projective transformations*, a way of digitally simulating a rotation of a camera.

Lastly we talk about theory that helps us in general with getting a robust system. More specifically we describe different methods of *feature detection* and how *RANSAC* allows us to find the best results in a set of data that contains so-called "outliers".

To find a new camera path we will use techniques to fit a curve to a set of data points, we will however not go into further details of the theory behind curve fitting.

2.1 Epipolar Geometry

This section is reviewing the basic theory regarding epipolar geometry. According to its definition, epipolar geometry describes the relations in a stereo vision. In other words epipolar geometry is used when having two images of a common three-dimensional (3D) scene. First we define the scene and the camera taking an image of the scene mathematically. To do this it is practical to introduce different spaces that we later use to relate an image point to a real-world scene. It will be shown that it is possible to describe this mapping from a real-world 3D point to a two-dimensional (2D) point in pixel coordinates as a single matrix transformation, called the *camera matrix* (see Section 2.1.2). The camera matrix is fundamental when trying to recreate a 3D model of the original real-world coordinates. To obtain the camera matrix from a stereo pair of images one must compute where different real-world points projects in the different images. This later relation can be described by the *fundamental matrix* (see Section 2.1.3), or as the *essential matrix* (see Section 2.1.5) in the case of calibrated cameras.

2.1.1 From Pinhole Camera to Describing Images Mathematically

Even though today's cameras are a lot more advanced than the pinhole camera, the pinhole camera model serves well when trying to describe images mathematically, see for example [7].

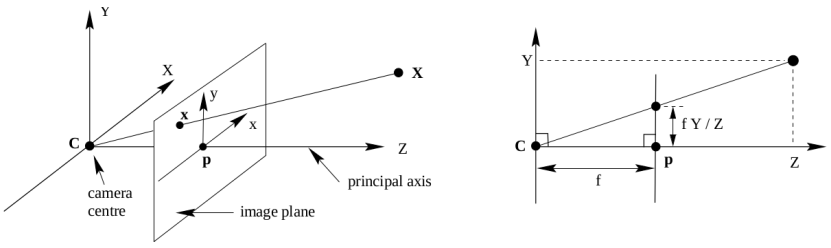

Figure 2.1: A simple pinhole camera model described mathematically as according to [10].

In Figure 2.1 an overview of a simple pinhole camera model can be seen. From this model we define three spaces; camera space, projection space and image space. We shall now show the difference between these spaces.

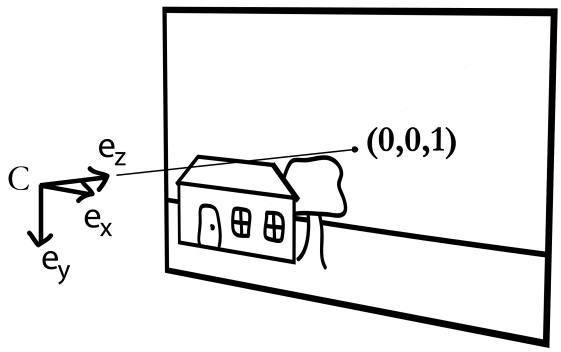

Figure 2.2: Camera space is a 3D-space with the origin in the camera centre and the e_z aligned with the view orientation. An image in camera space have a horizontal alignment with e_x and vertical alignment with e_y. The middle of the image is defined as the coordinate $U = (0, 0, 1)$.

Camera space: We define the camera space coordinates as $U = (x, y, z)$ with the camera center, C, in the origin and e_z along the viewing direction of the camera. We also say that an actual image in our model is positioned at $z = 1$, see Figure 2.2.

Projection space: The projection space coordinates is defined as $U = (x, y, 1)$, in other words it is defined as the camera space mentioned above with $z = 1$. Defined this way means that the projection space is a 2D-space defined along the actual image-axes but, rather as described by pixels in a discrete manor, this space is defined in a continuous way. The origin is defined as the middle of the image, see Figure 2.3.

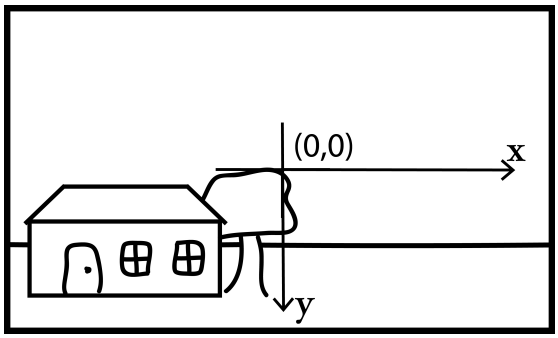

Figure 2.3: Projection space is defined as the plane in camera space where $z = 1$. This makes the origin of projection space to be $U = (0, 0)$.

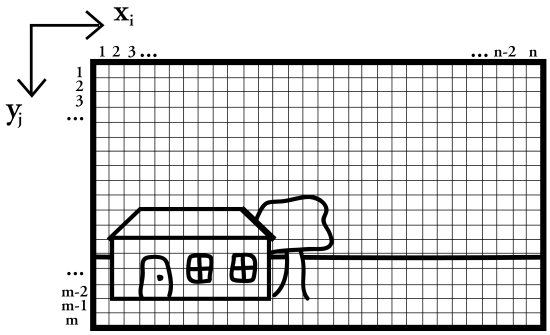

Figure 2.4: Image space is basically the transform of the continuous projection space into a discrete one. Coordinates in image space correspond to pixels in a real image. Note that the origin is positioned in the upper left corner.

Image space: Lastly, the image space is defined similar to the projection space with a few modifications. The space is defined as having discrete values so that a coordinate pair $u = (u_x, u_y, 1)$ denotes a pixel in an actual image. As can be seen in Figure 2.4, the origin is in the top left corner in the image. Depending on the resolution of the actual image the maximum values, m and n, is changed accordingly.

Finally we note that observing the relation between a camera center and a point, U, from an *xz-plane* perspective will give an approximate equivalent perspective as if you were to relate the camera center and the point in a real-world map, see Figure 2.5. This assumes that you align both the x-axis and the z-axis to the real-world horizon. The same goes for *zy-plane* where you can get an intuitive feel of the height relation between the camera center and a point.

Now, having defined the various spaces associated with the camera model we continue. Assuming a point, U, given in camera space with coordinates $U = (U'_x, U'_y, U'_z)$ the viewing direction from the camera center towards the point is given by $U - C$ where C is the origin. Parametrization gives in turn

$$C + s(U - C) = sU, s \in \mathbb{R}, \tag{2.1}$$

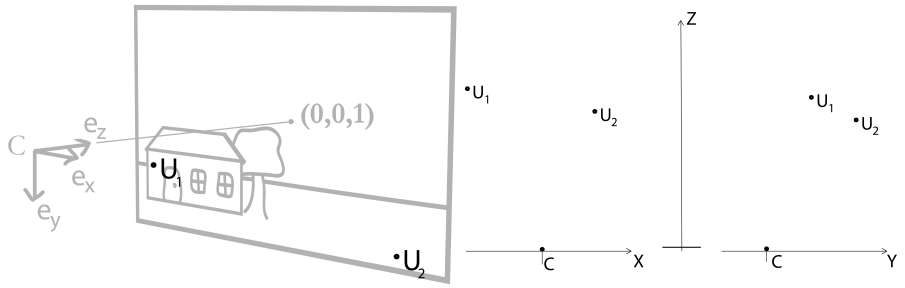

Figure 2.5: This figure illustrates different perspectives used in this thesis to evaluate our different results. To the left the view of an image of a scene is shown in camera space where U_1 and U_2 is two feature points. To the right there is a two view plot of the same scene seen from "above" and from the "side". These two views will be referred to as the xz-plane and the zy-plane respectively.

where we have used the fact that C is positioned in the origin, $C = (0, 0, 0)$. To calculate where in an image a projected point is seen we seek the parameter, s, where the third coordinate is equal to one. This implies

$$s = \frac{1}{U'_z} \Leftrightarrow u = \begin{bmatrix} \frac{U'_x}{U'_z} \\ \frac{U'_y}{U'_z} \\ 1 \end{bmatrix}. \tag{2.2}$$

In other words, this choice of the parameter, s, describes projection into projection space of the point, U. For an even more thoroughly review on this topic see [10], [23] and [7].

2.1.2 Camera Matrix

One crucial part of further extending the model of describing images and cameras mathematically is to introduce the so called camera matrix. The camera matrix describes the full process of going from camera space to image space. This implies that the following relation can be obtained, up to scale,

$$u \sim PU. \tag{2.3}$$

where the camera matrix, P, is a 3 by 4 matrix. It can be shown that the camera matrix can be written as

$$P = K\,[R\ t]\,, \tag{2.4}$$

where K is a 3 by 3 matrix, R is a 3 by 3 matrix and t is a 3 by 1 vector.

The K-matrix is called the *camera calibration matrix* that holds the so called *inner parameters* of the camera. We will have reason to revisiting the camera calibration in just a little while, see Section 2.1.4. The R-matrix is a rotation matrix that fulfills the orthogonality property and the t-vector describes a motion in 3D-space called a translation.

Although we do not present a strict proof of why the camera matrix can be written as in Equation 2.4 we shall present a short intuitive reasoning around the

involved matrices. By thinking about what movements a camera can undergo it is obvious that, in real-world applications, it is limited to exactly a rotation and a translation as proposed. Furthermore, the K-matrix is basically describing the mapping from projection space to image space, as suggested in [23].

Moving on, it can also be shown that we can easily extract the camera center, C, and the view orientation, VO, as follows

$$C = -R^{-1}t, \tag{2.5}$$

$$VO = R[0\ 0\ 1]^T. \tag{2.6}$$

This section has provided some information about and defined the basic properties of the camera matrix. The next step is to examine how to estimate the camera matrix. It will be shown that, based mainly on movement between two cameras and relating corresponding image points to one another, this can be done. The hands-on approach obtaining the camera matrix includes, as a first step, the estimation of the fundamental matrix. The fundamental matrix says something about the epipolar geometry in a scene including a pair of cameras. For a more extensive view on epipolar geometry there is a lot of literature covering this topic, see for example [10].

2.1.3 The Fundamental Matrix

The fundamental matrix is a 3 by 3 matrix of rank 2 that relates corresponding image points in two images. This matrix is later used, as mentioned in Section 2.1.2, together with the intrinsic parameters to eventually obtain the camera matrices of stereo images. It can be shown that the following relation holds

$$\bar{u}^T F u = 0, \tag{2.7}$$

where u is the image point in the first image and $\bar{u}$ is the image point in the second image and u and $\bar{u}$ are projections of the same 3D point, as first suggested by [3]. See also [10].

By triangulation one can reconstruct the world points but only up to a projective reconstruction. This means that the triangulated points may seem distorted from what they actually look like, see Figure 2.6 and [23].

To get a desired euclidean reconstruction we must include the intrinsic parameters of the camera in the reconstruction process. When doing this we remove some degrees of freedom and thus some of the ambiguity in the reconstruction.

2.1.4 Intrinsic Parameters & Camera Calibration

In our strive off an euclidean reconstruction we must find the intrinsic parameters that were mentioned together with the K-matrix in Section 2.1.2. A camera, as seen before, can be described as a rotation followed by a translation. If just described as a rotation and a translation, we can not move between image space and projection space. Introducing the intrinsic parameters of the camera, we can remove some of the arbitrariness and tell something about the actual mapping from camera space to the actual pixel coordinates in image space. We start by defining the camera calibration matrix, K, as follows

Figure 2.6: To the right we see a projective reconstruction and to the left a euclidean reconstruction of the Arch of Triumph in Paris. Note that projective reconstruction seems somewhat distorted from what is expected. Both reconstructions give the same projections. Not knowing the intrinsic parameters of the camera only enables projective reconstruction due to the degrees of freedom of the fundamental matrix. Image courtesy of [23].

$$K = \begin{bmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.8}$$

where f is the focal point of the camera and (x_0, y_0) is the principal point. We shall further note that this definition assumes quadratic pixels. The elements of the camera calibration matrix are the intrinsic parameters of the camera.

To find the intrinsic parameters, we could do a so-called camera calibration or have a look at technical specification from the camera manufacturer.

In an image with resolution m by n, the principal point however can be easily found as

$$(x_0, y_0) = (\frac{n}{2}, \frac{m}{2}). \tag{2.9}$$

More on this topic can be found in [10] and [23]. Even though this thesis does not use this explicitly, there is a possibility determining the intrinsic parameters as proposed by [11].

2.1.5 Essential Matrix

As in the case with the fundamental matrix the essential matrix is a matrix describing the relation between point correspondences in stereo images. In other words the following relation holds

$$\bar{u}^T E u = 0. \tag{2.10}$$

It is obvious that the fundamental matrix and the essential matrix are closely related. It can be shown that the relationship between the fundamental matrix and the essential matrix is

$$E = K^T F K, \tag{2.11}$$

where K, as before, is the camera calibration matrix [10]. Since the essential matrix takes into account the intrinsic parameters of the camera it reduces the degrees of freedom to five.

This freedom implies that only five point correspondences is required to obtain the essential matrix. It was already shown in 1905 by [16] that this problem has at most 11 solutions. Later on, several proofs showing that the problem has 10 solutions have been presented by [12], [4] and [21]. Even though the so-called *five-point algorithm* is a more reliable algorithm, requiring only five point correspondences, an easier algorithm is the *eight-point algorithm* for estimating the essential matrix. The reason is that the five-point algorithm includes solving various non-linear equations, which the eight-point algorithm avoids. This thesis nonetheless uses a numerical solver to the five-point algorithm, as proposed and developed by [26].

2.1.6 From Essential Matrix to Camera Matrix

Once we have obtained the essential matrix we can extract the camera matrices, P_1 and P_2, from stereo image point correspondences [23]. As a first step we assume the first camera to be on the form

$$P_1 = [I \ 0]. \tag{2.12}$$

The second camera have to be related to the first. This is done by the use of singular value decomposition where we factorize the essential matrix, E into three 3 by 3 matrices U, S and V under the relationship $E = USV^T$. The camera matrix, P, is then formed in four different ways by using U and V where only one of the four P-matrices is the true one. To obtain the correct one, triangulation is often used. The four ways of forming P is

$$P_2 = [UWV^T \quad -u_3] \tag{2.13}$$

or

$$P_2 = [UW^TV^T \quad -u_3] \tag{2.14}$$

or

$$P_2 = [UWV^T \quad u_3] \tag{2.15}$$

or

$$P_2 = [UW^TV^T \quad u_3] \tag{2.16}$$

where u_3 is the third column in U and

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.17}$$

as is shown by [10]. The correct P-matrix is the one where all reconstructed points are placed in front of the camera, see Figure 2.7.

This concludes the part of the epipolar geometry where we have stated the mathematical model of describing cameras and how to obtain these cameras from stereo image point correspondences.

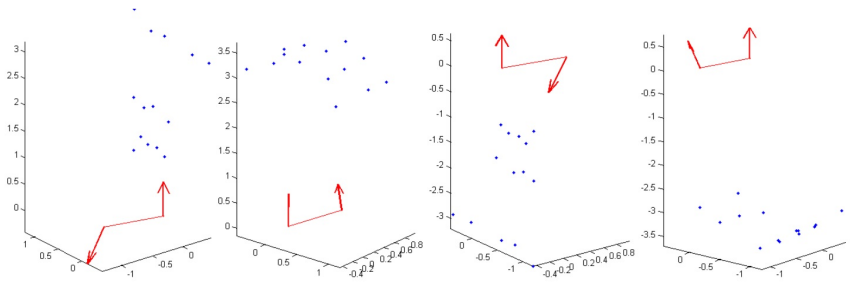

Figure 2.7: When trying to estimate the camera matrix in stereo image and assuming $P_1 = [I\ 0]$, there is four possible ways forming P_2 shown above. The correct camera matrix is the one where all reconstructed points are placed in front of the camera, in this case the second image from the left. Image courtesy of [23].

2.2 Coordinate System Transformation

Central to the implementation of our camera stabilization algorithm is to relate different pairwise camera matchings to each other. One way to relate is by transforming the different camera matrices so that they refer to the same global coordinate system. If pairwise matching for the image pair, i, has been done so that the following holds

$$P_i U = u, \tag{2.18}$$

$$P_i' U = u', \tag{2.19}$$

and in a sequence so that P_i' and P_{i+1} is describing the same image but in different coordinate system one can easily transform the different camera matrices to the same global coordinate system. Assuming that $P_i = K\,[I\ 0]$ and $P_i' = K\,[R_i'\ t_i']$ for all i we get P_i expressed in global coordinates, P_i^{Global}, by

$$P_i^{Global} = P_i * T_{i-1} * T_{i-2} ... T_3 * T_2 * T_1, \tag{2.20}$$

where T_i is a 4 by 4 matrix on the form

$$T_i = \begin{bmatrix} R_i' & t_i' \\ 0 & 1 \end{bmatrix}. \tag{2.21}$$

Notice that the length of t_i is unknown due to the fact that the relation in Equation 2.3 only can be determined up to scale. We have however assumed that the velocity is constant and therefore that $|t_i| = 1$.

2.3 Projective Transformations

A projective transformation is a transformation that maps lines to lines, not necessarily preserving parallelism. It is possible to do a projective transformation to simulate a rotation of a camera around its own center since the camera

Figure 2.8: To the left the original image is shown straight from the camera. To the right a projective transformed version of the original image is shown. This transformations corresponds to a rotation of the camera $\frac{\pi}{8}$ radians around the y-axis as defined in camera space. Note the black edges in the right image due to the lack of information outside the edges of the original image.

matrix operation itself operates by a projective transformation. For an example regarding a simulated rotation see figure 2.8.

Since this is done in each image camera space separately, one can set up the following system of equations,

$$PU = u, \tag{2.22}$$

$$P'U = u', \tag{2.23}$$

where u are the old pixel coordinates and u' are the new pixel coordinates in the projective transformed image. Furthermore the camera matrices are

$$P = K\left[I\ 0\right], \tag{2.24}$$

$$P' = K\left[R\ 0\right]. \tag{2.25}$$

Assuming that the point U is projected to u' in camera two we get

$$\lambda' u' = K\left[R\ 0\right]U,$$

this gives

$$U = \begin{bmatrix} \rceil & \lambda' R^{-1}K^{-1}u' \\ & 1 \end{bmatrix}. \tag{2.26}$$

But all of these points project to

$$\lambda u = K\left[I\ 0\right]U = \lambda' K R^{-1}K^{-1}u'$$

which, since it is only up to scale, gives

$$u \sim K R^{-1}K^{-1}u'. \tag{2.27}$$

In other words by inserting the new pixel coordinates corresponding to the projectively transformed image one gets the old image pixel coordinates and in turn one can get the color of the transformed pixel.

When doing a projective transformation there will be cases of trying to reach pixel coordinates outside the original image pixel coordinate space. There are different ways of handling this, one of the most common is simply setting the pixel value to 0 zero. This results in some black regions in the new transformed image due to the obvious lack of information outside the original image pixel coordinate space.

2.4 Feature Detection

A feature is a very imprecise concept, it varies with its application. A feature can be seen as a specific property of an object and is used as the name suggest, to describe objects and more importantly to classify them. In image processing and computer vision a feature is an interesting part of the image and many algorithms in these fields start with feature detection. As the feature detection often is the starting point and the rest of the algorithm relies on its result the importance of a good feature detector is high. You can design your own features for a specified problem. For example when dealing with images of letters you could count the height and width of the letter and the number of black pixels to determine which letter it is. There are a number of feature detectors that are well-known and some of these are implemented in Matlab. For example, features from accelerated segment test (FAST) [25] which is a corner-detection method, maximally stable extremal regions (MSER) [20] which is a method used to find blobs in an image and the speeded up robust features (SURF) [2] which is based on Haar wavelet responses and partly inspired by the scale-invariant feature transform (SIFT) [18] [19]. There are other feature detectors but not all of them are implemented in Matlab, such as SIFT.

2.5 RANSAC

The random sample consensus, RANSAC, algorithm is an algorithm with the aim to successfully build a mathematical model from a set of observed data which contains outliers, see Figure 2.9. In short the algorithm iterates random data point selections in order to form a hypothesis and then test this on the data set as a whole, noting the number of outliers. RANSAC is said to perform badly when the number of outliers is more then half of the total number of data points [17]. By doing this a number of times the solution theoretically converges to the 'true' solution very fast. By imposing some criteria regarding for example how many outliers should be tolerated or how many iterations should be allowed one can make a choice between a more accurate or a faster algorithm. For more on the RANSAC algorithm read [6] and [10].

In this thesis RANSAC is used to find the best essential matrix with the help of a five-point solver, see Section 2.1.5. The five-point solver is used to get a solution in every RANSAC iteration which determine the number of inliers of that solution. This is then done a number of times and the solution that gives the highest number of inliers is chosen to be the essential matrix we use for that

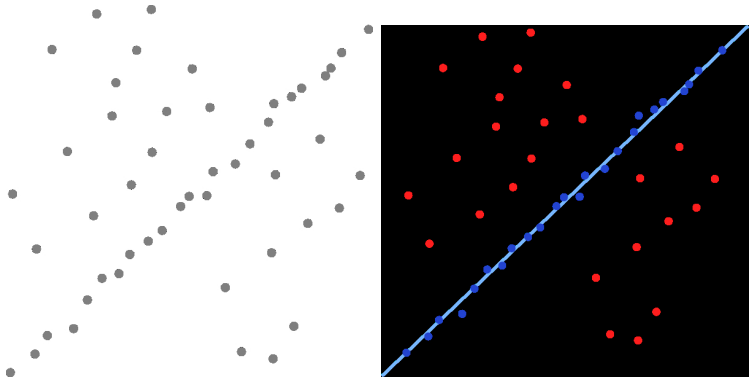

Figure 2.9: To the left is a data set with a lot of outliers to which we want to fit a linear curve. By using RANSAC we get the result as seen in the right where the outliers has been sorted out and therefore not contributing to the fitting of the curve. Image courtesy of [1].

image pair.

Chapter 3

Method & Delimitations

Over time we have made some method choices and also some decisions regarding delimitations that we intend to describe more extensively in this chapter.

3.1 Narrative Clip

The Narrative Clip is a small wearable camera with, for this thesis relevant, specifications according to Table 3.1.

Sensors	GPS
	Accelerometer
	Magnetometer
	Image sensor
Camera resolution	5 megapixel
Aspect ratio	4:3
Output format	jpeg
Frame rate	1 frame per 30 seconds

Table 3.1: Table showing specifications of the Narrative Clip, according to the Narrative website [22]. Note that the camera has some sensors available to further improve a video stabilizing algorithm in the future. Also, one of the more defining features of the camera is the low frame rate, when compared to video cameras.

Basically the camera takes an image every 30 seconds in order to create a narrative over your life. A user primarily attach their Narrative Clip to their clothes, see Figure 3.1, resulting in images facing forward, almost like a so called first person perspective.

Narrative in turn offers some ways to present these images in a compelling way through cloud services. Since a new image every 30 seconds adds up to more than potentially 2800 images a day, the high amount of images opens up the possibility to find new ways of presenting these images. One potential way is to present suitable image sequences in form of a hyperlapse. Because of the low frame rate, one of the first questions that arise is if it's even possible to present a good looking hyperlapse from the clip, and if not, what is the least possible frame rate that can do?

Figure 3.1: An image showing a typical way of using your Narrative Clip, even though other creative ways of using the clip have been seen. This results in images primarily facing forward in relation to how a person would move.

3.2 Programming Environment

There is various environments capable of handling image processing and computer vision applications. A very much used library is OpenCV (Open source Computer Vision) that has interfaces dedicated to both C++, C, Python and Java (for more information turn to http://opencv.org/).

Another, to engineering students, well known environment is Matlab, that we chose as our primary environment. Matlab has a computer vision toolbox that bundles with the main application. The computer vision toolbox includes built in methods such as feature detection, feature matching and reading and writing images as well as movies. Matlab is pretty much optimized for matrix and array operations which made us use Matlab's possibilities to program MEX-files. MEX-files contain one function programmed in C, C++ or Fortran using the Matlab api for MEX-files and then eventually compile in Matlab. We used the possibility to make MEX-files when we could not optimize the code in a way that makes Matlab work effectively. The reason we wanted to optimize the code, just a little bit, was that otherwise the algorithm easily becomes very time consuming.

3.3 Data Gathering

To test our video-stabilizing algorithm we gathered a couple of different data sets. Basically the data sets consists of a series of images taken with an iPhone camera. In the process we did a camera calibration using Matlab's built-in camera calibration tool and a printed checkerboard.

Furthermore we decided upon a couple of constraints regarding the form of the data sets. All images are taken with an upright position (see Section 4.1.1) because we did not see an obvious reason to orient the images in an horizontal position. The images also simulates a person wearing the Narrative Clip front facing when taking a stroll, because this is a common use case. A couple of different lengths of image sequences are used to test basic stability of

the stabilizing algorithm. Since the first versions of our algorithm were a lot more time consuming some shorter sequences was used enabling a "have a look at the result and then tweak the code"-approach. Even though these sequences are short they served well for studying the functionality of our code.

The basic data gathering through the test phase was aiming to have a great variety based on for example length of image sequence, physical length between frames and environments. We also wanted to test the limits when the algorithm totally failed by doing some image sequences with serious shakes.

In the final test we have chosen to make a movie recording from our iPhone in order to simulate different input frame rates by removing frames, see Section 3.6.

3.4 Stabilizing Algorithm

There has been a lot of work done in the field of video-stabilization algorithms and many different approaches. Among theses approaches we decided to limit ourselves to a few mainly because of the time available to this thesis. Firstly after having done some initial testing we chose to stick with SURF as our feature detector because SURF is implemented as a built in feature detector in Matlab and because it performed best out of the built in feature detectors in an initial test.

Secondly we only stabilize in xz-space ("seen from above") and zy-space ("seen from the side") according to the definitions of camera space in Section 2.1.1. This delimitation is chosen due to the estimation that rotation around the z-axis effects are less probable than pure tilt effects.

Thirdly, we do a stabilizer according to the principles in Figure 3.2. The stabilization method does not handle translation differences from a new calculated camera path. To handle something other than projective transformations (pure rotations) demands a very different approach which could not be included in the scope of this thesis.

3.5 Final Test Cases

As stated in Section 3.3, we did a movie recording with an input frame rate of approximately 30 frames per second. Accordingly we could simulate different input frame rates by removing various amounts of frames, see table 3.2.

The final decisions regarding which simulated input frame rates to use was done so as to get an understanding of what will be an hyperlapse that is nice to watch. Also the final test sequence tries to simulate a rather typical situation that would be nice to summarize using a hyperlapse.

Regarding the stabilization we decided to use system version 3, see Chapter 5, to stabilize the videos. Based on the results of the test sequences this seemed to be the best choice.

3.6 Evaluation

During development the evaluation was chosen to focus around how the matched points are placed in the images, how many matched points there are, how many

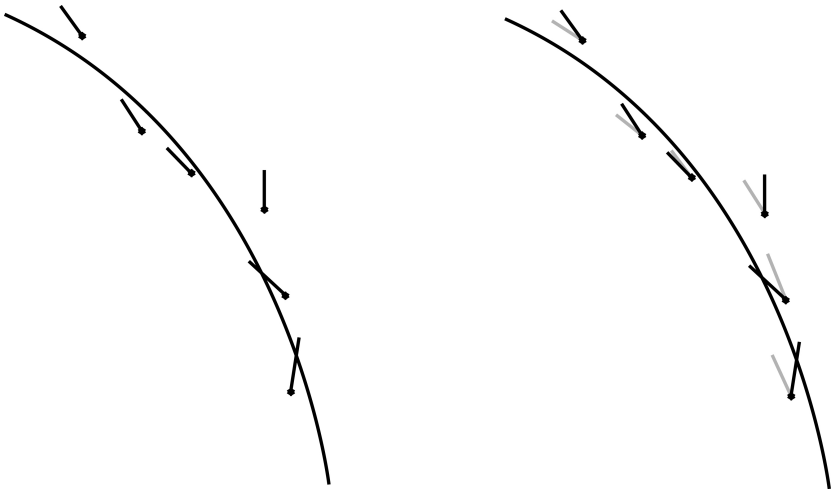

Figure 3.2: This figure shows an overview of the algorithm principles. To the left there are a couple of camera centers and corresponding viewing orientation. There is also a curve fitted to the camera centers serving as a new smooth camera path. To the left is the same figure but with the addition of new viewing orientations in light gray that will be the basis for the projective transformations.

of the matched points that can be classed as "inliers" and how the plots of how the relative camera movements seems to match our intuitive feeling of how the actual movement was. We also looked at the total number of features that were detected and the spreading of these as well as the execution time. A brief discussion regarding these factors follows here below.

Execution time: A lower execution time is to prefer since it indicates how optimized the code is. Also a shorter execution time is nicer to work with.

Number of features detected: A high number of features detected is desirable since the chance increases of having a higher number of matches increases with the number of features detected. To get an idea of how well the feature detection went we used the mean value of features detected in the images for each sequence.

Place of feature points in image: This is interesting since it will have a impact on how great the spread is among the matches. The standard deviation in both the vertical- and horizontal-axis was used to get an overview of the spreading of features.

Number of matched points: A higher number of matched points will result in higher certainty for the estimations of the camera matrices. To evaluate this the mean number of matches for each sequence has been used.

Place of matched points in image: To minimize the risk of numeric failures and other error sources it is better the more spreading of the matched points there is. Especially regarding points in the image that corresponds to real-world points in line with the viewing direction of the camera is a big error source when moving towards the viewing direction because of, in worst case, little or no movement from image to image. To get an idea of how big the spread is for each sequence the standard deviation in both the vertical- and

Desired simulated input (frames/second)	**Frames kept** (every k:th frame)
5	6
3	10
1	30
0.5	60
0.2	150

Table 3.2: Table describing the simulated input frame rate and which frame are selected with an original input frame rate of 30 frames per second. Basically the formula to get a new simulated input frame rate is to divide the original input frame rate by the desired simulated input frame rate to get which frame that should be removed and which ones to keep. For example to get a simulated input frame rate of 10 frames per second from an original frame rate of 30 frames per seconds you divide 30 by 10 to get that you should keep every 3rd frame.

horizontal-axis has been used.

Number of inliers: Since we want to have an estimation as good as possible for the camera matrices we want the number of inlier to be as high as possible in relation to the number of matches since the ratio between these is an indicator of how well the RANSAC algorithm has performed. To get an idea of how many inliers we had we used the mean value of inliers in the sequence.

Place of inlier points in image: As in the case with the matches and features one wants as big spread as possible even for the inliers. As before the standard deviation in both the vertical- and horizontal-axis has been used for each sequence.

Ratio of matched point inliers to outliers: Because of the criteria of the RANSAC algorithm, as mentioned in Section 2.5, we want a relatively high number of inliers compared to outliers in the process of estimating the essential matrix. A ratio of 1 between the inliers and the matches indicates that all the matches are inliers and a ratio of 2 indicates that only half of the matches are inliers. We have considered a ratio of 2 to be acceptable but the closer to 1 the better. To compare this for the sequences the mean value of the ratios in each sequence is used.

We have not evaluated how well our systems finds a new camera path mathematically, we have only looked at the plots and made evaluations based on what we see. This is due to the fact that it is very difficult to decide beforehand on what would be an optimal camera path and because, as discussed in Chapter 6, Section 6.2.1, it is not necessarily the curve closest to the camera centers that would be classified as the optimal one by the human eyes. We have, in other words, not found a suitable measure of how well the systems fit a curve to the given data points. What we have looked for when we have done an estimation of how well the system versions managed to find a new camera path is how straight the new path looks, how well it follows the camera centers and how good the new path represents how we have actually walked when the images were taken.

Chapter 4

The Algorithm

In order to get get a clear view of the progression the code was run for the ten data sequences collected and the result was saved before changing the code. Above all the view of our improvements and/or deterioration in results and execution time became very apparent. These trials are referred to as system versions and there are three of them. Below is a description of the general procedure and the differences between the system versions are described in Chapter 5.

4.1 The Data

Ten different sequences of images were collected. These sequences were taken as to vary in: scenery, length of sequence, footsteps between images (in average), view orientation and complexity of path walked when taking the sequence (in other words if the path was straight, curved, zigzagged and so on). A short information of the sequences are presented below.

Sequence 1: This sequence is taken in Stadsparken in Lund. The sequence consists of 48 images and has three steps between every images. The images are taken when walking straight on the path.

Sequence 2: This sequence is taken on Lilla Fiskaregatan in Lund and contains 54 images. We have walked along the street from Akademibokhandeln to the Narrative office and the images has been taken with three steps in between.

Sequence 3: This sequence is taken when walking from Norra Fäladstorget towards Delphi in Lund. There are three steps between every image and the sequence is 77 images long.

Sequence 4: This sequence is taken along the way from IKDC towards Delphi in Lund and is taken with different view orientations and when zigzagging along the way. This sequence is to really test our algorithm and consists of 35 images taken with three steps between the images.

Sequence 5: This sequence is taken just below Botan in Lund and is 18 images

long. The images are taken with ten steps in between.

Sequence 6: This sequence is taken along the exact same way as sequence 5 but with three steps between each image. This is to check if the algorithm results in a similar camera path. The sequence contains 50 images.

Sequence 7: This sequence is only seven images long and is taken on Stortorget in Lund. This sequence is mainly used to test our functions separately since it is so short and therefore takes a short time to process.

Sequence 8: This sequence is very long, 182 images, to test if the algorithm can handle it. The pictures are taken at LTH with three steps in between the images.

Sequence 9: This sequence has very different view-orientations in each image but is taken when walking straight in Stadsparken in Lund. The sequence contains 42 images which has three steps between each image.

Sequence 10: This is taken on Löparbanan, the running track, in Lund to check how exact the camera centers are placed when using our algorithm. The sequence is 168 images long with four steps between each image.

4.1.1 Example Images of the Sequences


Sequence 1, Stadsparken


Sequence 2, Lilla Fiskaregatan


Sequence 3, Norra Fäladen


Sequence 4, LTH


Sequence 5, Botan


Sequence 6, Botan


Sequence 7, Stortorget


Sequence 8, LTH


Sequence 9, Stadsparken


Sequence 10, Löparbanan

4.2 The Code and What It Does

Our algorithm consists of various steps. These are summarized in pseudo-code seen in the box below and explained in more detail later in this section.

```
for all images in one sequence
   read the image and the consecutive one
   extract features and find matching points between the images
   calculate the essential matrix
   calculate the camera matrices
for all camera matrices in the sequence
   refer the camera matrix to the global coordinate system
   calculate the camera center and the view orientation
using all or some camera centers and view orientations
   calculate new camera path
   calculate the angles of rotation
for all calculated rotations
   rotate the images
```

4.2.1 Read the Image and the Consecutive One

Assume an image sequence consisting of images, $I_1, I_2, ..., I_{n-1}, I_n$. Using the fact that the order in which the images were taken is known, the algorithm reads the image pairs as defined by I_k, I_{k+1}, where $k = 1, 2, ..., n-2, n-1$. This way we match two consecutive images to each other in the algorithm.

4.2.2 Extracting the Features and Finding Matching Points Between the Images

As mentioned in Section 2.4 there are many feature detectors that can be used when dealing with images. We chose to use SURF as it is already implemented in Matlab and its performance is known to be good.

4.2.3 Calculate the Essential Matrix and the Camera Matrices

For each pair of matched points we calculate the essential matrix E using a five-point solver [26], see Section 2.1.5. From that we use RANSAC, see Section 2.5, to obtain the best E-matrix. From E we extract the camera matrices P_1 and P_2, see Section 2.1.6.

4.2.4 Refer the Camera Matrix to the Global Coordinate System and Calculate the Camera Center and the View Orientation

When we have determined a P-matrix for all the images we need to refer them to the same coordinate system, see Section 2.2. This is important since we want to do analysis over all images in the same sequence and therefore need them to

be in relation to each other. After we have done that we proceed to find the camera centers, C, and the view orientations, VO. Since we know the P-matrices and our cameras have been calibrated, see Section 2.1.4, it is straight forward to extract C and VO, see Section 2.1.2.

4.2.5 Calculate New Camera Path and the Angles of Rotation

To avoid that the resulting image is very black, see Section 2.3, we've chosen to set a max angle for rotation. This is done in every system version and the max angles corresponds to no more black pixels than $\frac{1}{4}$ of the original image. The calculation of the new camera path and the desired rotation is done differently in each system version, see Chapter 5.

4.2.6 Rotate the Images

Each image is rotated as calculated from the new camera path to obtain the desired view orientation. To rotate the images we use rotation matrices

$$R_\theta = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix} \tag{4.1}$$

and

$$R_\phi = \begin{bmatrix} cos(\phi) & 0 & sin(\phi) \\ 0 & 1 & 0 \\ -sin(\phi) & 0 & cos(\phi) \end{bmatrix}, \tag{4.2}$$

where θ is the angle in the xz-plane and ϕ is the angle in the zy-plane.

Chapter 5

System Versions

As stated in the beginning of Chapter 4 we ran the code in three different versions. Between every system version we have made some improvements and changed the way to decide on a new camera path. Everything before the calculation of the new camera path, see box in Section 4.2, is the same in every system version in theory even though we optimized the code after system version 1. The order of the polynomials used is chosen by us based on how we've walked.

5.1 System Version 1

In system version 1 we simply let Matlab fit a second order polynomial to our camera centers in the xz-plane and a first degree polynomial in the zy-plane. We then find the point on the curve that is closest to each camera center by using the L2-norm and then we calculate the derivative to the curve in that point. We then calculate the angle between the real view orientation and the one that has the same slope as the tangent in each image. This gives us the angles for our rotation in the xz-plane and in the zy-plane.

5.2 System Version 2

In system version 2 we decided to optimize the code to get lower running time. To start with we used the Matlab's inbuilt function to scale the images to $\frac{1}{4}$ of their original size, this results in a image size of approximately $800 * 600$ pixels. This enabled us to run all our sequences. We also used the profiling tool in Matlab to see which parts of the code that needed to be optimized. When we did this we saw that a very large time was spent on rotating the images which led us to implement some parts of our function that does that as a mex-method using C-programming.

For the camera path we used a slightly different approach than in system version 1. We introduced a fourth variable, t, which is a vector with integers from 1 to the length of the sequence, $t = [1...length(sequence)]$, where $t = 1$ corresponds to image number 1 and so on. We then divided the camera space-variables, x, y and z into three coordinate systems where the camera space-variable is on the vertical axis and t is on the horizontal axis. In each of these

coordinate systems we fitted a curve to the coordinates of the camera centers in the respective dimensions using the inbuilt functions in Matlab. In the xt-plane and zt-plane we fitted a second degree polynomial and in the yt-plane we fitted a first degree polynomial, see Figure 5.1. We then took the point on the curves in each t that corresponds to a camera center and combined them into one xz-plane with data points $(x(t), z(t))$ and one zy-plane with data points $(z(t), y(t))$. This gave us two curves that were better fitted to the camera centers than before. The combined curve of these can be seen in Figure 5.2. To calculate the desired view orientation and the angle for rotation we needed to find the derivative. To do this we went back to our image-variables-t coordinates and took the derivative in the points on the curves nearest the camera centers and then combined them to the two image-variables coordinate systems so that we had in the xz-plane data points $(x'(t), z'(t))$ and in the zy-plane $(z'(t), y'(t))$. We then proceeded as before and calculated the angles between the real view orientation and the one with the same slope as the tangent in both the xz-plane and in the zy-plane.

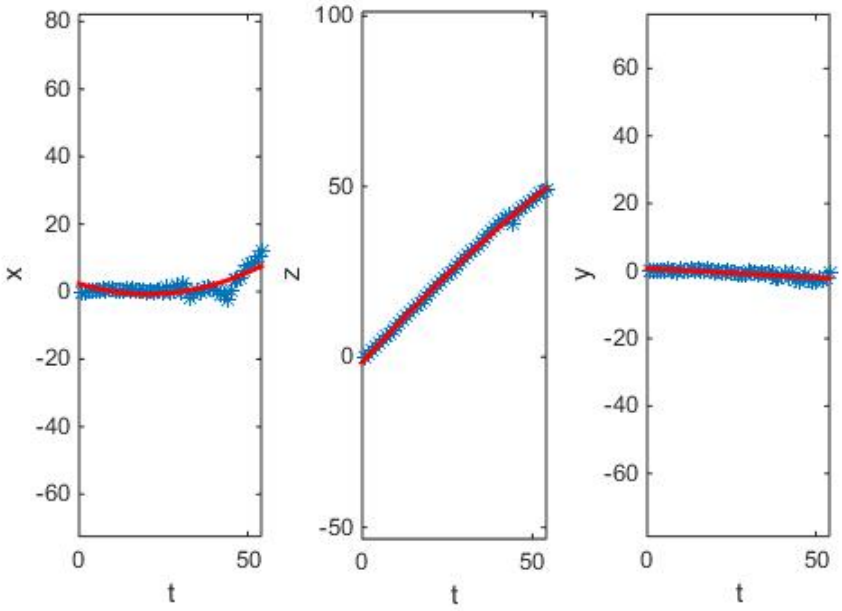

Figure 5.1: This figure shows the curves fitted for the camera centers in each camera space-variable against the variable t for data sequence 2 in system version 2.

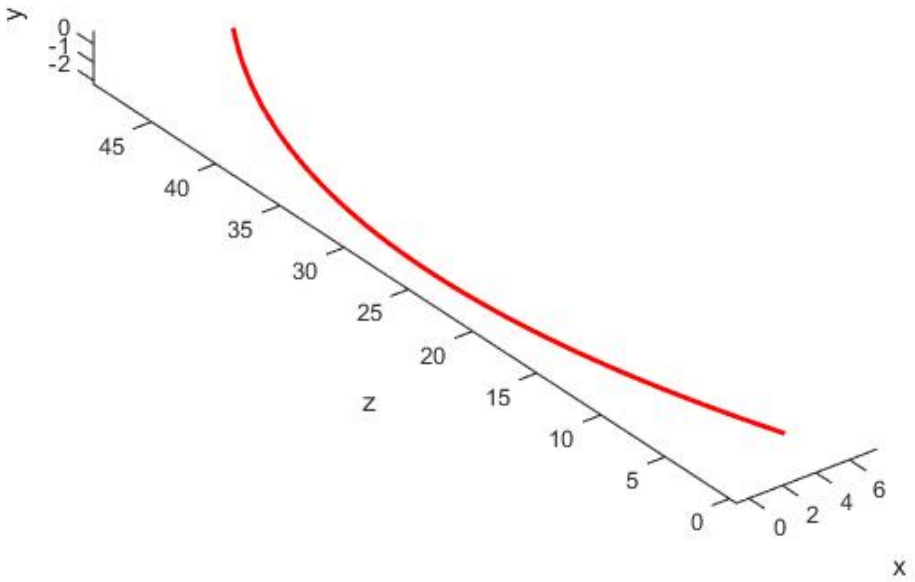

Figure 5.2: The curve above is the one we obtained when combining the points from the three camera space-variables that were closest to the curve fitted for each variable against t for data sequence 2 in system version 2.

5.3 System Version 3

In this system version we decided to try to get an even better fit, one that would smooth out the camera path but have the same general look as the camera centers were placed. What we did was that we used the same technique as in system version 2 but instead of fitting a curve too all camera centers for each image-variable against t we fitted a new curve to each camera center k and its closest neighbors, that is we fitted a curve in all image-variables for image k by using images $k-i$ to $k+i$ where i is implemented to be 3,5,7,9 or 11. For each of these curves we used the same procedure as in system version 2. This gave us curves that looks as if they more closely follows the camera centers but still smooths the original path.

Chapter 6

Results

6.1 Results of the General Code

To start with we looked at our matches to make sure that the algorithm performed as desired up til finding the new camera path, see Figure 6.1. We were pleased with the results since it seems to match the same points in the two images.

Figure 6.1: Plot of matches in a pairwise matching.

We also tuned the tolerance of our RANSAC by looking at the ratio between the number of matches and the number of inliers. Our aim was to have a ratio that was close to 1, see Section 3.6, without having a too low tolerance. This tolerance is set to $4 * 10^{-4}$ which corresponds to one pixel.

6.2 Results of the Differences in the System Versions

6.2.1 Summary of the Results for the Data Sequences

System versions 2 and 3 managed to run all the sequences, system version 1 only managed four of them and it seems that system version 1 is very bad at handling longer sequences because of limitations in RAM. The more difficult sequences, such as sequence 4 and sequence 9, did not get a nice representation in the way the camera centers were placed in any of the system versions. When the camera centers is placed in a fairly straight line system version 2 might do a better fitting of a new camera path than system version 3 although it is quite hard to tell. What is very interesting is that all of the sequences that did get a good representation in the xz-plane seems to have an error in the zy-plane that grows. If one looks at the plots, take Figure 6.3 for example, there seems to be an increasing degree of randomness towards the end of the sequences.

6.2.2 Comparison Between The System Versions

In Table 6.1 follows an example of the result of one of the sequences. This is to illustrate the differences in the system versions. Similar tables for each sequence can be found in Appendix A.

Sequence number and scenery	2, Lilla Fiskaregatan		
Number of pictures	54		
System version number	1	2	3
Execution time in seconds	2024	139	113
Mean of number of features	2551	336	336
Mean of number of matches	686	119	119
Mean of number of inliers	592	98	98
Mean of ratio of matches and inliers	1.2	1.23	1.24
Std of matches in x	363	106	106
Std of matches in y	406	98	98
Std of features in x	437	133	133
Std of features in y	435	109	109
Std of inliers in x	342	83	83
Std of inliers in y	389	97	97

Table 6.1: Table showing each evaluated feature, see section 3.6 for more information, for sequence 2 in the three system versions.

6.2.3 System Version 1

As one can see in Figure 6.2 the new camera path is not good for the xz-plane. It does not relate to the camera centers in a reasonable way and the new view orientations are not desirable at all. Both the camera centers and the view orientations are more or less on a line but the fitted curve is obviously not a line which makes the magenta dots placed in a very odd way. Since the rotations are

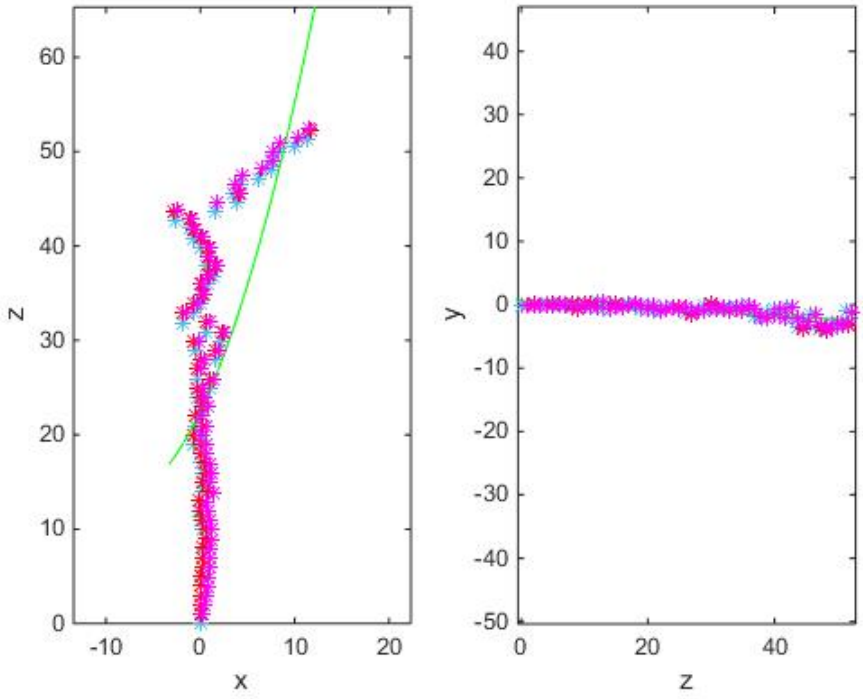

Figure 6.2: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 2, system version 1. The camera centers are nicely placed but the curve does not look like a good fit.

calculated with the use of the slope of the derivative to the fitted curve a wrong fit will give the wrong desired rotation. This seems to be the biggest problem we have encountered in system version 1 and will have to be taken care of. It seems that our code work fairly well except for the fitting of a new camera path, at least it does what we ask of it. There are however room for improvements and some optimization of the code to reduce the running time, see Table 6.1.

6.2.4 System Version 2

After optimization the running time for our algorithm was reduced by approximately $94,4\%$. Also the down-scaling of the images enabled us to run all our sequences. We were very pleased with these improvements. As one can see in Table 6.1 the number of inliers, matches and the standard deviation of the inliers, matches and features are lower than in system version 1 but since the ratio between matches and inliers still is near one this is fully acceptable and more over it is what is expected since the images contains fewer pixels.

As for the camera path it looks better in the xz-plane, see Figure 6.3 the plot to the left, and most importantly we got rid of the weird behavior that can be seen in the plot to the left in Figure 6.2. In the zy-plane the curves looks

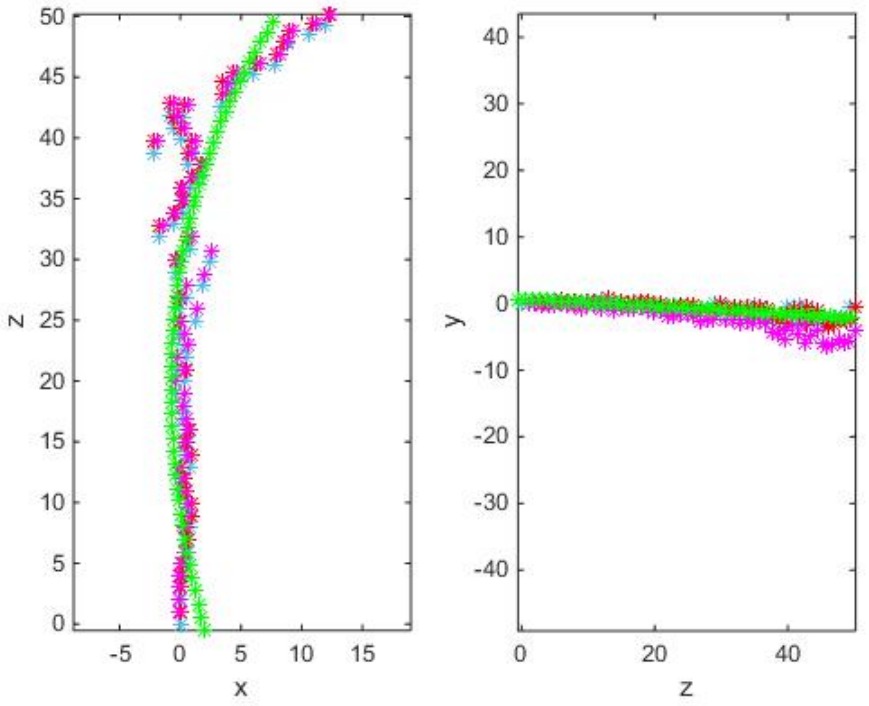

Figure 6.3: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 2, system version 2. The data points is places nicely and the curve is fitted better than in system version 1, it does leave room for improvements however.

very similar.

6.2.5 System Version 3

Since we haven't done anymore optimization or changed anything except the calculation of the new camera path the running time is expected to be the same as in system version 2.

For the camera path we see in Figure 6.4 to the left that the curve follows the camera centers better and only smooths it out. We found that $i = 9$ and $i = 11$ gave good results, better than $i \leq 7$, so we chose $i = 11$ to be the value for every sequence since many of them are quite long. For sequence 7 however we set $i = 7$ since the sequence only have seven images. In zy-plane the curve follows the camera centers better as well but it's not as different from the earlier system versions as it is in the xz-plane.

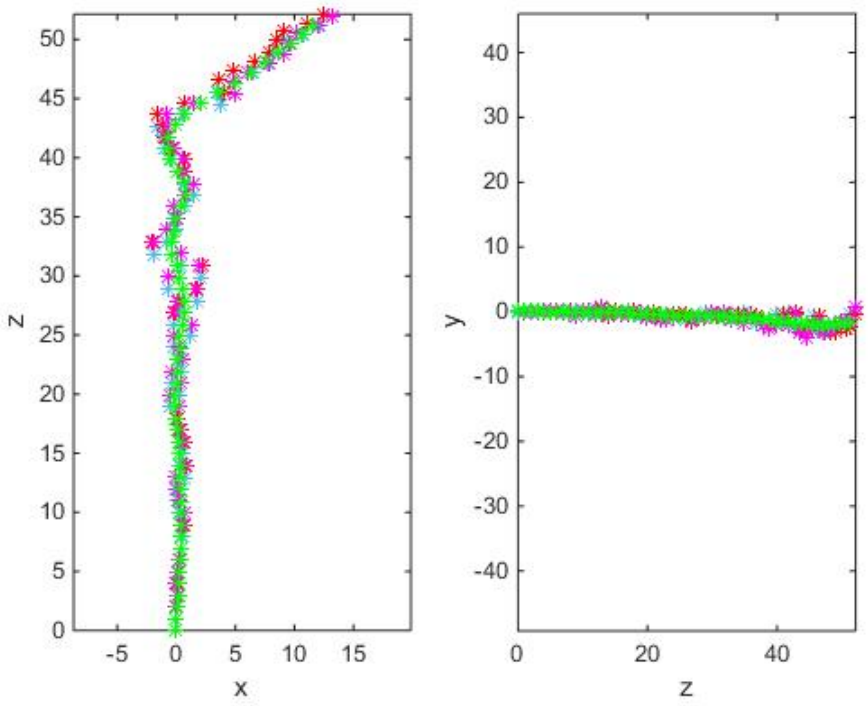

Figure 6.4: Plot over camera centers (blue), view-orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 2, system version 3. The camera centers represents how we have walked well and the curve fits much better now than in the two previous system versions.

6.3 The Final Test

As stated in Section 3.5 the final test consist of a movie recording from which different input frame rates have been simulated by removing frames. To stabilize the videos system version 3 has been used. We put the original video with removed framed to the left in a window and the stabilized version of that video to the left in the same window. We then let people look at these videos and grade them on a scale from 1 to 5 where 1 is very bad, 3 ok and 5 very good, the right and the left video, separately and we also asked them which one was best for each frame rate and at the end we asked them if they had a favorite among the videos and in that case which one. The results can be found in Tables 6.2, 6.3 and 6.4.

What we can see in these results are that a frame rate under 1 frame/second (fps) is too low. For both videos with a frame rate lower than 1 fps the viewers chose "none of them" when asked which of the two videos, original and stabilized, they liked best, see Table 6.3. We also note in the same table that the stabilized video did a tie at best in relation to the original video. What

Simulated input (frames/second)	**Grade of original video** (mean value)	**Grade of stabilized video** (mean value)
5	2.82	2.5
3	2.14	2.09
1	2.86	3.09
0.5	2.77	2.86
0.2	2.27	2.36

Table 6.2: Table showing the simulated input frame rate with an original input frame rate of 30 frames per second. In the two most right columns the mean value of the grades (1 to 5) for both the original video and the stabilized video for each simulated input frame rate are presented.

Simulated input (frames/second)	**Votes for the video original video** (percent)	**Votes for the stabilized video** (percent)	**Votes for non of them** (percent)
5	59.1	27.3	13.6
3	36.4	18.2	45.5
1	31.8	31.8	36.4
0.5	22.7	13.6	63.6
0.2	31.8	18.2	50

Table 6.3: Table showing the simulated input frame rate with an original input frame rate of 30 frames per second. The question asked was which of the videos, or non of them, did the viewer experience as less shaky.

also contributes to the conclusion that the videos with lowest frame rate are not pleasant to look at is that they got a low number of votes when the viewers were asked which video was their favorite, see Table 6.4. What is worth noticing is that only one video got a mean grade higher than 3, which was "ok", and that video was our stabilized version of an input of 1 fps, see Table 6.2, and our stabilized version got a higher mean grade for the three videos with the lowest frame rate.

The results seems to be a bit inconsistent, our stabilized video got a higher mean grade for three out of five videos but when asked which, if any, of the two videos were best the original video got more votes for every fps except in one case, 1 fps, were the two videos had a tie.

To conclude, it seems as if a frame rate lower than 1 fps is not appreciated by the viewers.

Simulated input (frames/second)	**Votes** (percent)
5	27.3
3	18.2
1	27.3
0.5	13.6
0.2	0
none	13.6

Table 6.4: Table showing the simulated input frame rate with an original input frame rate of 30 frames per second. The question asked was which of the frame rates, or none, was the viewers favorite.

Chapter 7

Conclusion

7.1 Conclusions of the Results

When we looked at all our plots we saw that some of our sequences didn't get a good representation. For example sequence 4, which is taken to be very difficult see Section 4.1, comes out like random dots in the plot, see Figure 7.1. To understand why, we looked at a histogram of the inliers of the feature matches in each image pair in sequence 2, which seems to work well, see for example Figure 6.2, and sequence 4, see Figure 7.2. As seen in the figure, sequence 4 has many image pairs which have a very low number of inliers compared to sequence 2. This is probably why it has such troubles to place the camera centers in a reasonable way, see Figure 7.1.

If one looks at the plots, take Figure 6.3 for example, there seems to be an increasing degree of randomness towards the end of the sequences. This might be due to the fact that we assume that the distance between every frame is equal but three steps is not a very precise measure and when we proceed to relate all the pictures to the first one this imprecise measure might get troublesome. What will happen when we assume the same distance between each frame is that we assume that between for example picture 1 and picture 3 there is twice the distance as between picture 1 and picture 2 which probably is not true.

As for the final test we have some suggestions as to how the evaluation could have been done in a more fool-proof way. When we talked to the people who participated in the evaluation the general impression we got was that the stabilized images should have been cropped. Apparently the movement of the black edges gave the impression of a shaky video. Also, since the lowest frame rate we simulated was 0.2 fps (every 150:th frame) the input video could have been longer for a longer output video. Another thing that we thought of was that maybe the stabilized videos would have been better if we had use system version 2. As stated in Section 6.2.1 system version 2 might be better when the images has been taken in a fairly straight line, as they were in the input video. We were however pleased that the only video with a mean grade over 3 was our stabilized version of the video with 1 fps as input.

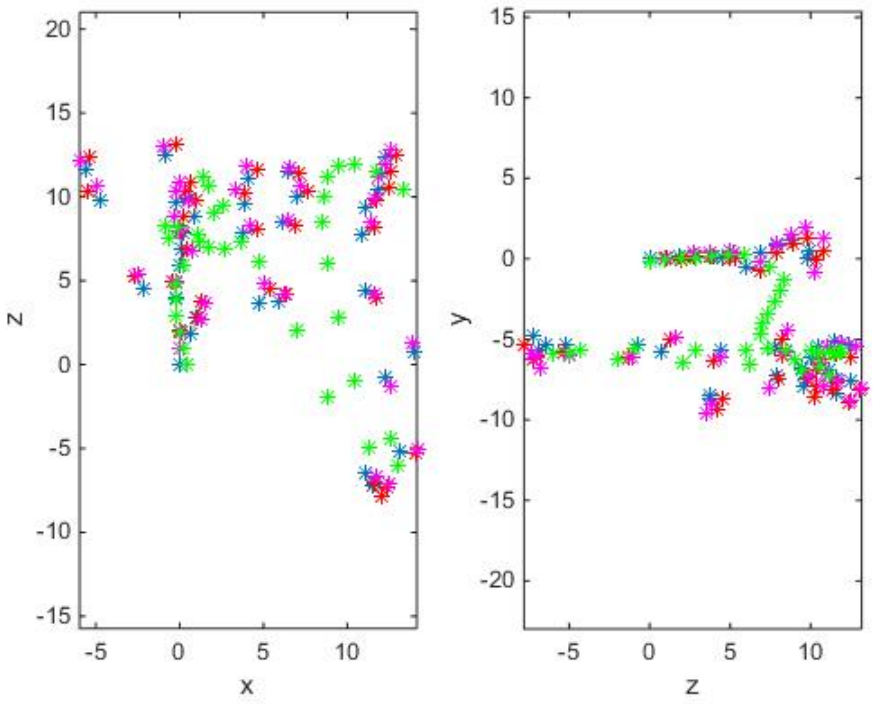

Figure 7.1: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 4, system version 3. As one can see the data points does not represent a zigzagging walk but look random.

7.2 Applications

If further improved the results of this thesis could have, as we see it, a number of different future applications. Above all it is an effort to make a nice hyper-lapse application from low input frame rate. This could be used in situations where one wants to present a longer sequence of events in a shorter video. This algorithm could also have applications in ordinary video stabilization.

7.3 Improvements and Further Work

Even though we see the results of our thesis as promising, there are a lot of possibilities of improvement and further work before getting a completely satisfactory result.

As seen in Figure 6.3 and briefly touched upon in Section 6.2.1, there seems to be some scattering of data points toward the end of longer sequences. One suggestions is that this depends on numerical problems integrated into the process of relating the camera centers to a common global coordinate system. To deal with this, ideally a method removing the amplification of error throughout a sequence will have to be found. Another idea is to somehow divide a longer

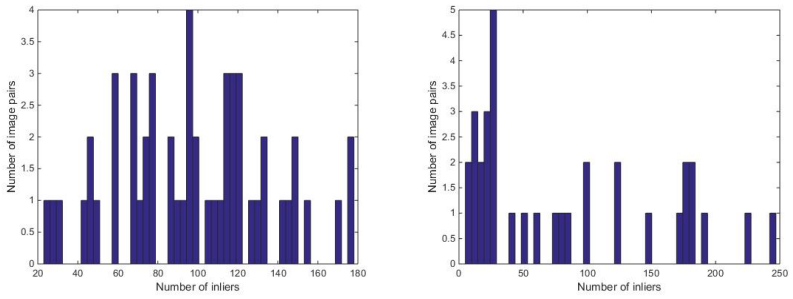

Figure 7.2: Histograms over the number of inliers between pictures in sequence 2 (left) and sequence 4 (right), system version 3. As can be seen sequence 4 (right) has many more image-pairs that have a low number of inliers compared to sequence 2 (left). This is probably why our algorithm has such trouble in placing the camera centers in a reasonable way.

sequence into shorter ones then somehow stitching them together. This way the error amplification may be reduced to acceptable levels. One obvious improvement is to not only do a pairwise matching, as proposed in this thesis, but to do matching over multiple images. A suggestion, according to [10], is to use a trifocal tensor, thus being able to match across three images in turn making the system more robust. Another suggestion is to use a multiple cross matching throughout a whole sequence and removing bad estimations of camera matrices as described by [5]. A last suggestion to get a more robust system is that of [8], where not fully reconstruct a scene but to only use fundamental matrices to determine how to transform images.

Assuming a system where full 3D reconstruction still is used there is a possibility of getting a better fit for the camera path doing a pre-processing of the camera centers. For example a filtering process of the camera centers could be used in order to get less noisy data. One suggestion is to design this filtering process as a fully 3D reconstructed equivalent to what is done by [8], with for example a moving average filter or a Gaussian kernel.

Nowadays, the style of hyperlapses seems getting more diverse from what was originally a pretty strict style using a landmark as a focus point that is placed in roughly the same coordinates in image space throughout an image sequence. Still this is something to consider, what is and what is not a good looking camera path in a hyperlapse. This in turn leads to that the fitting of a curve to the data sets might be more good looking for example using system version 2 rather than system version 3. Also, a sequence where the data points is placed in a straight line could benefit from using system version 2 as well. Optimal results of the fitting of a curve would perhaps be achieved if some information of the general structure of the data points could be retrieved before choosing how to do the fitting, together with a more strict definition of what a hyperlapse should look like. Another approach to achieve optimal results on the fitting of a curve could be to experiment with different parameters once an optimal path has been decided on, in other words to go backwards and try to find a general way of describing a optimal camera path.

To further improve the robustness of the system, more feature detectors

than just one could be introduced. There is even the possibility to write own features to, depending on the scenery involved, further optimize the feature detection. This could lead to that for example the scenery of a street view will use a completely different set of features than a scenery of a mountain. This will require a set of parameters weighing together to decide which features to use.

In the work leading to this thesis, only a projective transformation has been taken into account. Projective transformations is equivalent to a pure rotation of the camera, not caring for translations. To make further use of image warping techniques to improve the results is a suggestion for further work.

7.4 Summary of Thesis Achievements

Using state of the art methods to work with structure from motion technique we have developed a video stabilization algorithm. To do this we have used Matlab but some parts of the code is implemented in C. We have used ten image sequences to evaluate our algorithm and as a final test we have simulated different frame rates by removing frames from a movie recording and then asking a few people do grade the different outcomes. When looking at the plots of the results the algorithm seems to work fairly well on the easier sequences but when the stabilized images is put together and presented in a video the impression is that the algorithm have not been able to remove the camera shakes completely.

Appendices

Appendix A

Appendix

A.1 Detailed Results for Each Sequence

To get a short explanation of the values presented in the tables see Section 3.6.

To see more detailed information of the sequences see Section 4.1.

Std=standard deviation, x and y refers to the image space coordinates, see Figure 2.4. Error indicates that the computer did not manage to run the algorithm for that sequence without freezing or without taking unreasonably long time.

A.1.1 Sequence 1

Sequence number and scenery	1, Stadsparken		
Number of pictures	48		
System version number	1	2	3
Execution time in seconds	error	117	113
Mean of number of features		1796	1796
Mean of number of matches		230	230
Mean of number of inliers		182	182
Mean of ratio of matches and inliers		1.3	1.3
Std of matches in x		106	106
Std of matches in y		176	176
Std of features in x		161	161
Std of features in y		185	185
Std of inliers in x		82	83
Std of inliers in y		149	151

Table A.1: Table showing each evaluated feature, see section 3.6 for more information, for sequence 1 in the three system versions.

As seen in Figures A.1 and A.2 and also in Table A.1 the system versions 2 and 3 have performed very alike, both in the fitting of the curve and in the code in general. This is not surprising since the code up til fitting a new camera path is the same in these two system versions. The sequence was quite easy, that is the

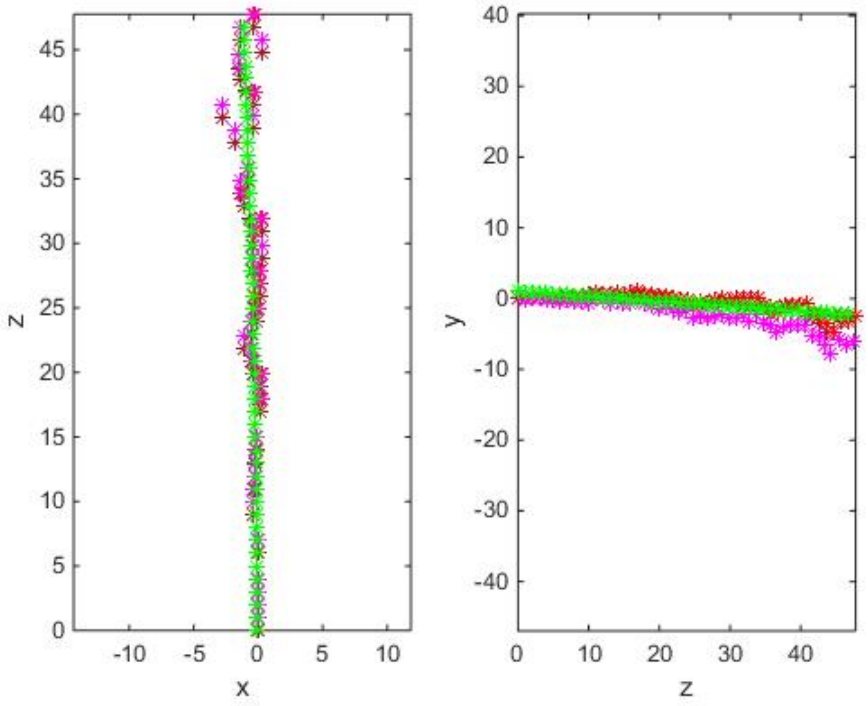

Figure A.1: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 1, system version 2.

photos has been taken in a straight line and with similar view orientations, but what is noticeable is that for this particular sequence system version 2 might be better since it is more straight and in this case it is probably a straight line one would like to adapt after. This sequence is, as stated above, not very difficult and yet system version 1 could not handle it.

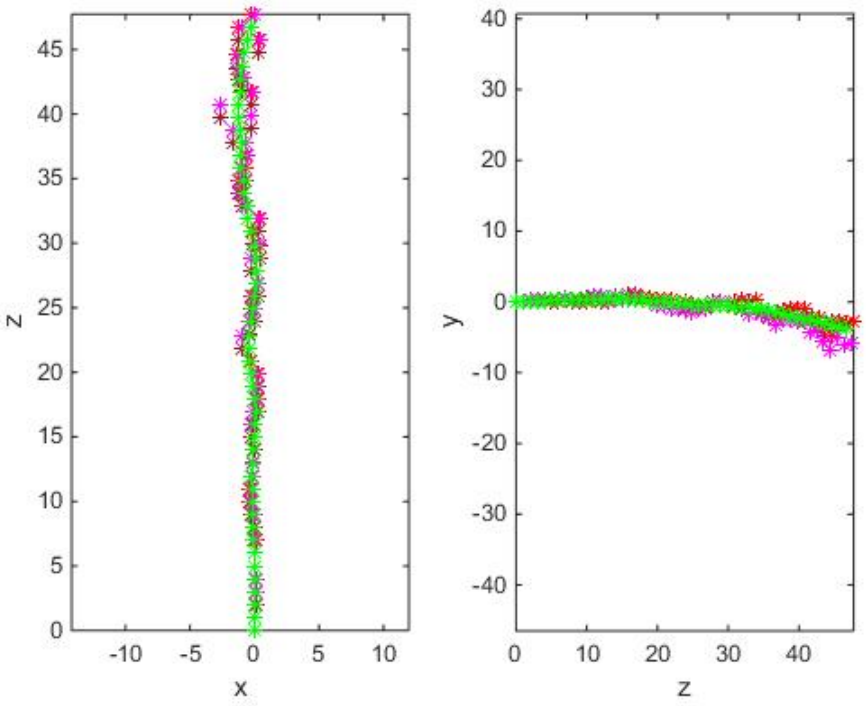

Figure A.2: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 1, system version 3.

A.1.2 Sequence 2

Sequence number and scenery	2, Lilla Fiskaregatan		
Number of pictures	54		
System version number	1	2	3
Execution time in seconds	2024	139	113
Mean of number of features	2551	336	336
Mean of number of matches	686	119	119
Mean of number of inliers	592	98	98
Mean of ratio of matches and inliers	1.2	1.23	1.24
Std of matches in x	363	106	106
Std of matches in y	406	98	98
Std of features in x	437	133	133
Std of features in y	435	109	109
Std of inliers in x	342	83	83
Std of inliers in y	389	97	97

Table A.2: Table showing each evaluated feature, see section 3.6 for more information, for sequence 2 in the three system versions.

The first thing to notice for this sequence is that system version 1 took a considerable longer time than the other two system versions to finish, see Table A.2. All of the system versions however seem to have managed to capture the way we actually walked and the camera centers looks to have been placed in a good way, see Figures A.3, A.4 and A.5. The curves that were fitted is very different, system version 1 is completely off compared to the others and system version 3 has a even better fit than system version 2.

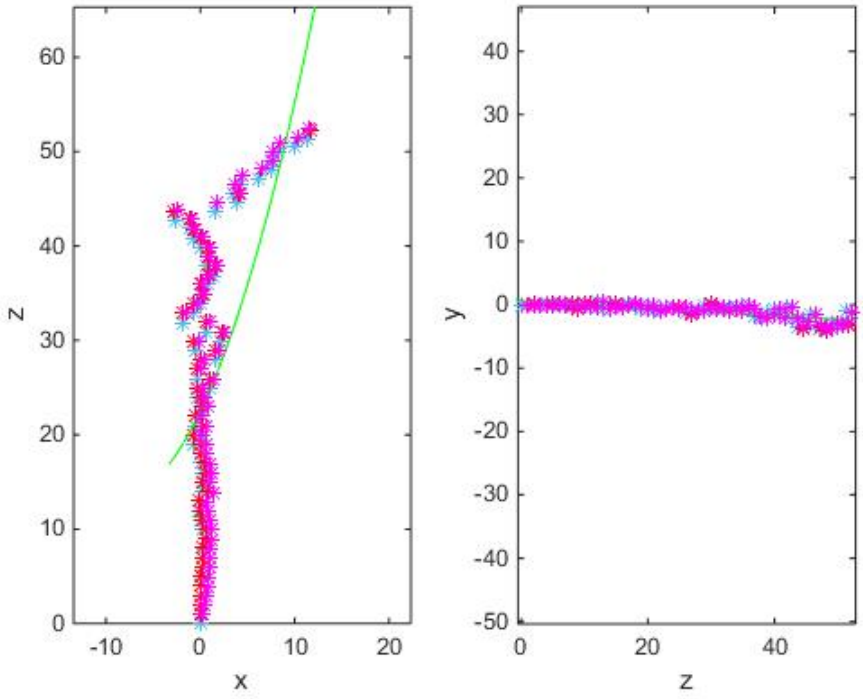

Figure A.3: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 2, system version 1.

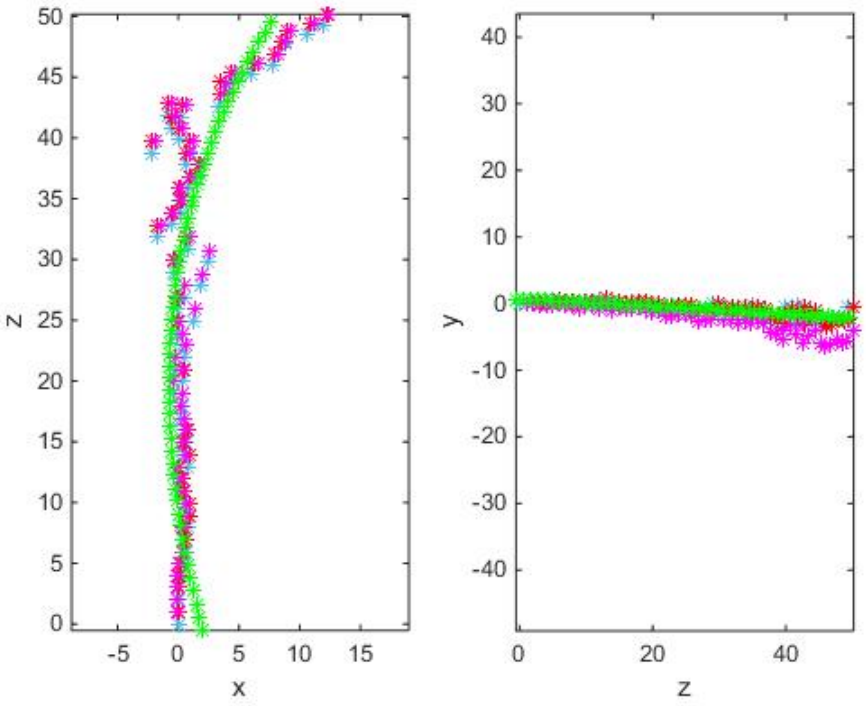

Figure A.4: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 2, system version 2.

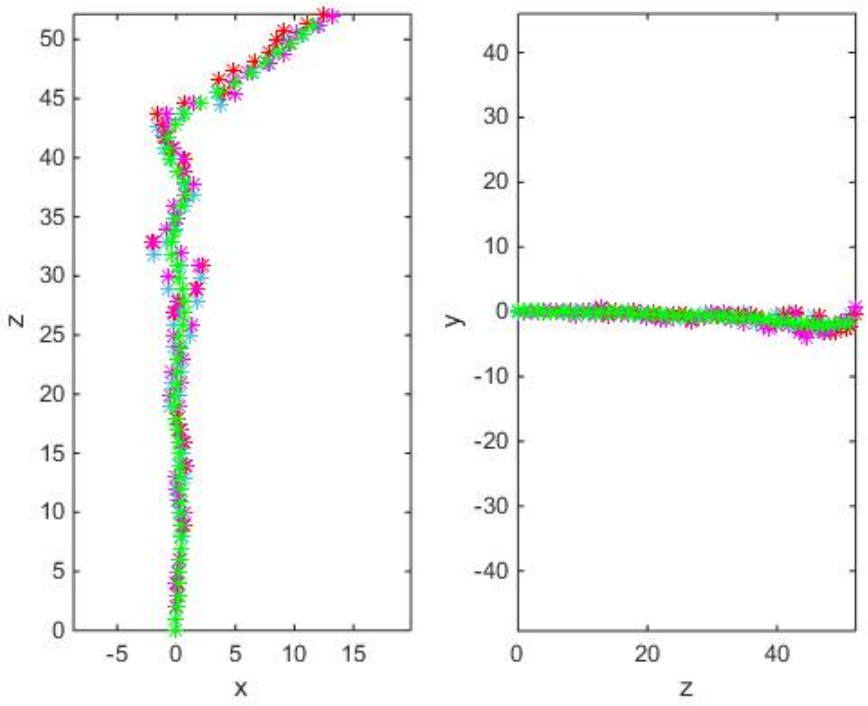

Figure A.5: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 2, system version 3.

A.1.3 Sequence 3

Sequence number and scenery	3, Cykelväg		
Number of pictures	77		
System version number	1	2	3
Execution time in seconds	error	172	171
Mean of number of features		1289	1289
Mean of number of matches		96	96
Mean of number of inliers		75	75
Mean of ratio of matches and inliers		1.3	1.3
Std of matches in x		114	114
Std of matches in y		114	114
Std of features in x		168	168
Std of features in y		112	112
Std of inliers in x		75	82
Std of inliers in y		66	60

Table A.3: Table showing each evaluated feature, see section 3.6 for more information, for sequence 3 in the three system versions.

For this sequence both the system version that worked, system version 1 did not, have managed to place the camera centers alright. Since there are three steps in between every image there should not be that much space between the data points as we can see in the middle of the plots of the xz-plane, see Figures A.6 and A.7, but otherwise the general appearance of the camera centers is accurate. For this sequence system version 2 did not managed to make a good fit, system version 3 however seems to have done better.

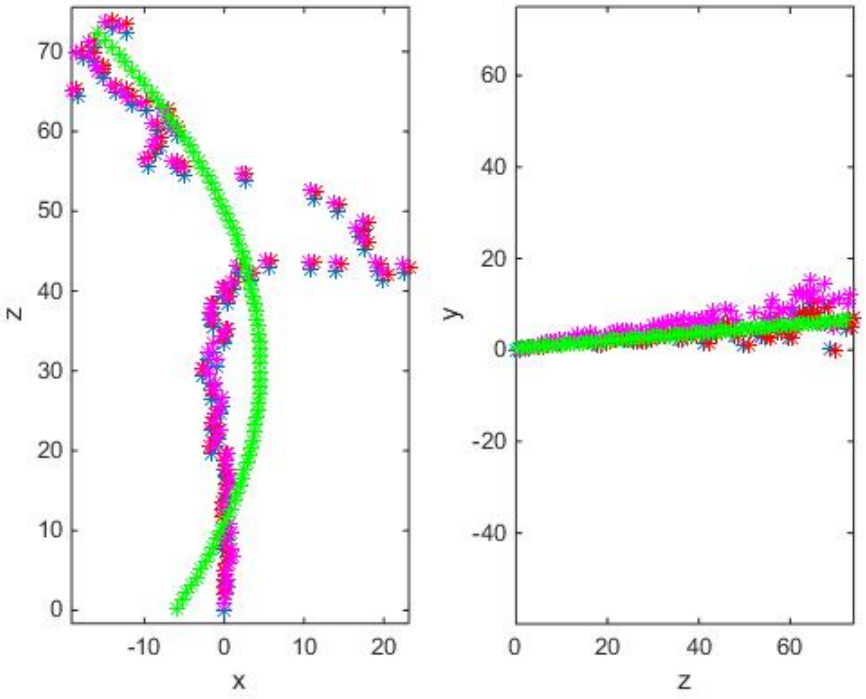

Figure A.6: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 3, system version 2.

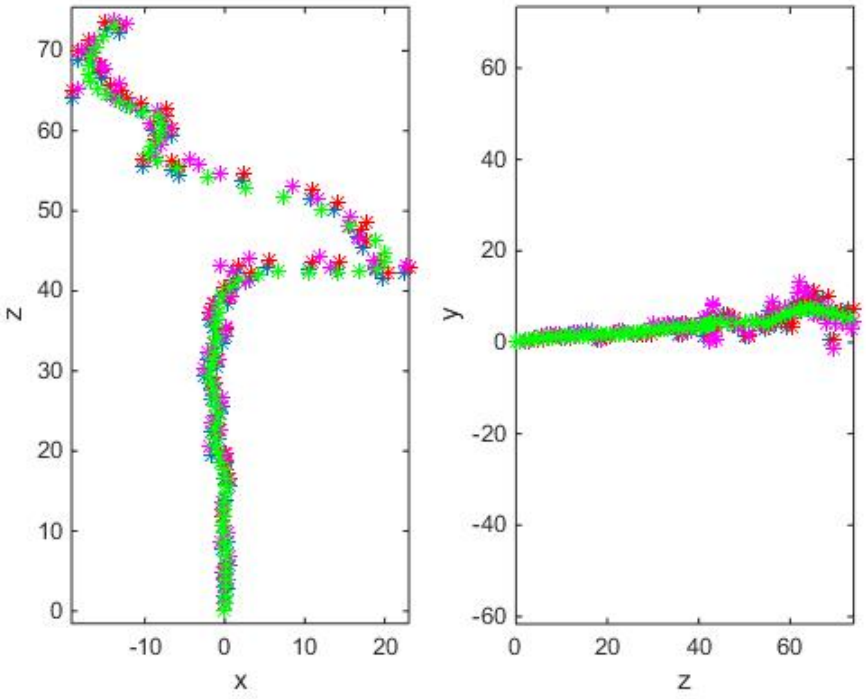

Figure A.7: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 3, system version 3.

A.1.4 Sequence 4

Sequence number and scenery	4, from IKDC		
Number of pictures	35		
System version number	1	2	3
Execution time in seconds	1375	74	75
Mean of number of features	1400	1422	1422
Mean of number of matches	705	109	109
Mean of number of inliers	509	87	87
Mean of ratio of matches and inliers	2	1.3	1.3
Std of matches in x	493	107	107
Std of matches in y	431	116	116
Std of features in x	826	181	181
Std of features in y	459	120	120
Std of inliers in x	157	66	71
Std of inliers in y	277	82	85

Table A.4: Table showing each evaluated feature, see section 3.6 for more information, for sequence 4 in the three system versions.

System version 1 took a very long time compared to the other two. Although non of them managed to place the camera centers in a reasonable way, see Figures A.8, A.9 and A.10. What does not show in Table A.4 is that this sequence have quite many image pairs with very low numbers of inliers in the matching between them, see Section 6.1. The fitted curves are in this case irrelevant, due to the randomness of the data points.

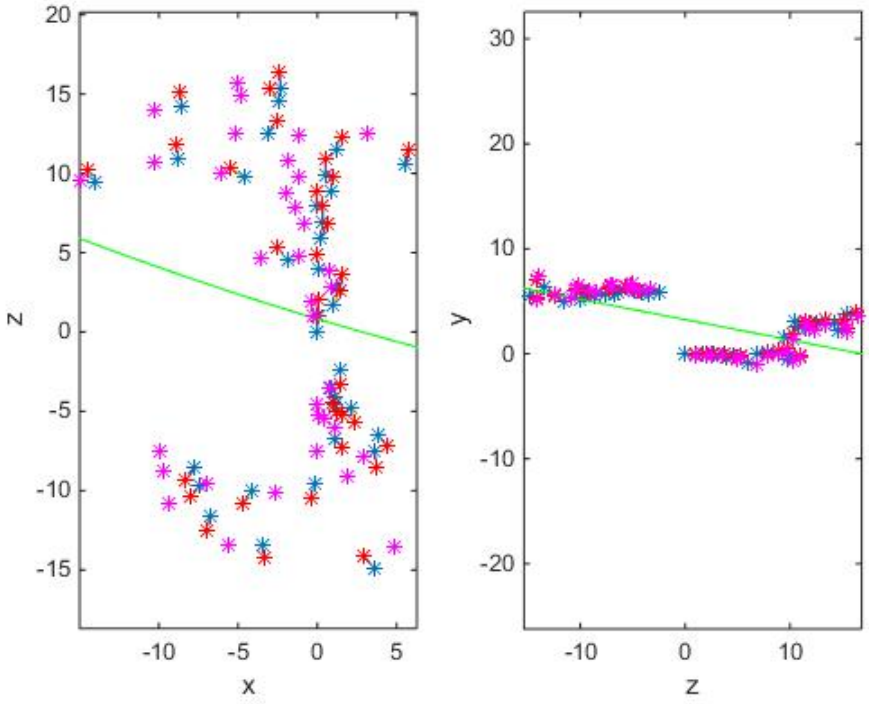

Figure A.8: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 4, system version 1.

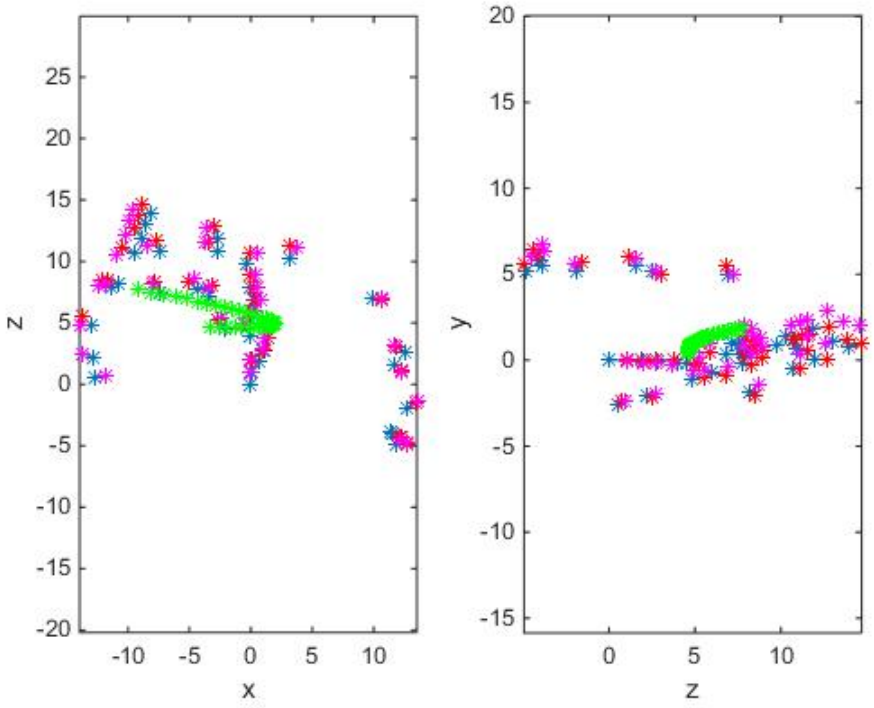

Figure A.9: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 4, system version 2.

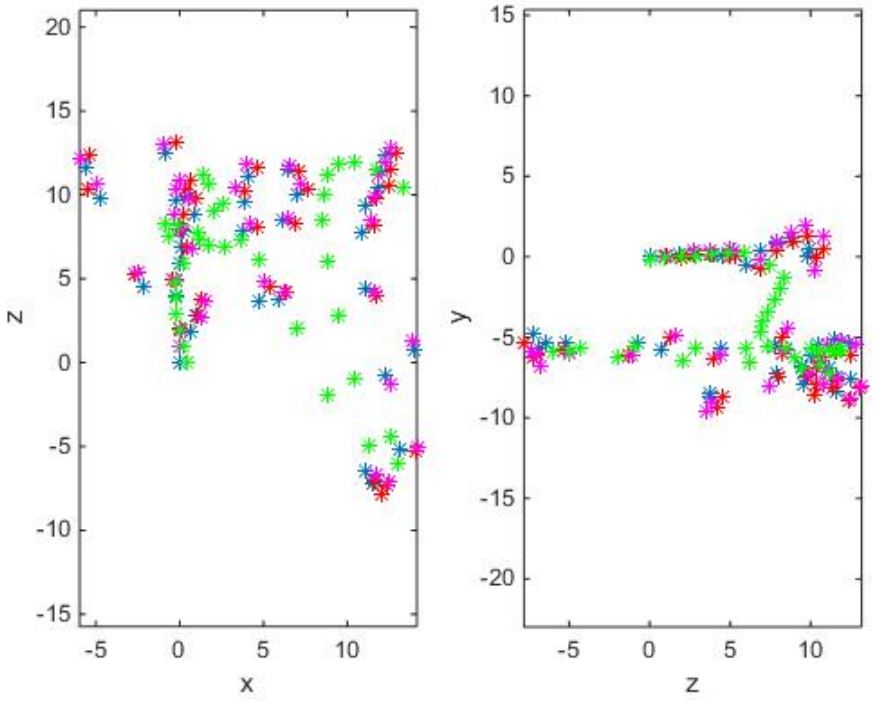

Figure A.10: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 4, system version 3.

A.1.5 Sequence 5

Sequence number and scenery	5, Botan		
Number of pictures	18		
System version number	1	2	3
Execution time in seconds	846	41	40
Mean of number of features	8900	1570	1570
Mean of number of matches	645	98	98
Mean of number of inliers	490	74	74
Mean of ratio of matches and inliers	1.8	1.4	1.4
Std of matches in x	547	137	137
Std of matches in y	401	100	100
Std of features in x	651	168	168
Std of features in y	395	106	106
Std of inliers in x	393	98	125
Std of inliers in y	309	90	99

Table A.5: Table showing each evaluated feature, see section 3.6 for more information, for sequence 5 in the three system versions.

This sequence worked for all of the system version, though system version 2 and 3 did it in a much shorter time than system version 1, see Table A.5. As to the fitted curve system version 2 and 3 performed a lot better than system version 1, see Figures A.11, A.12 and A.13. Since the camera centers is placed in a fairly straight line, which represents the actual walked way, it is not unreasonable to say that system version 2 made a better fit than system version 3 although it is quite hard to tell since they are very similar. What is noticeable about this sequence is that the images are taken with ten steps in between each image so our systems seems to be able to handle the larger gaps.

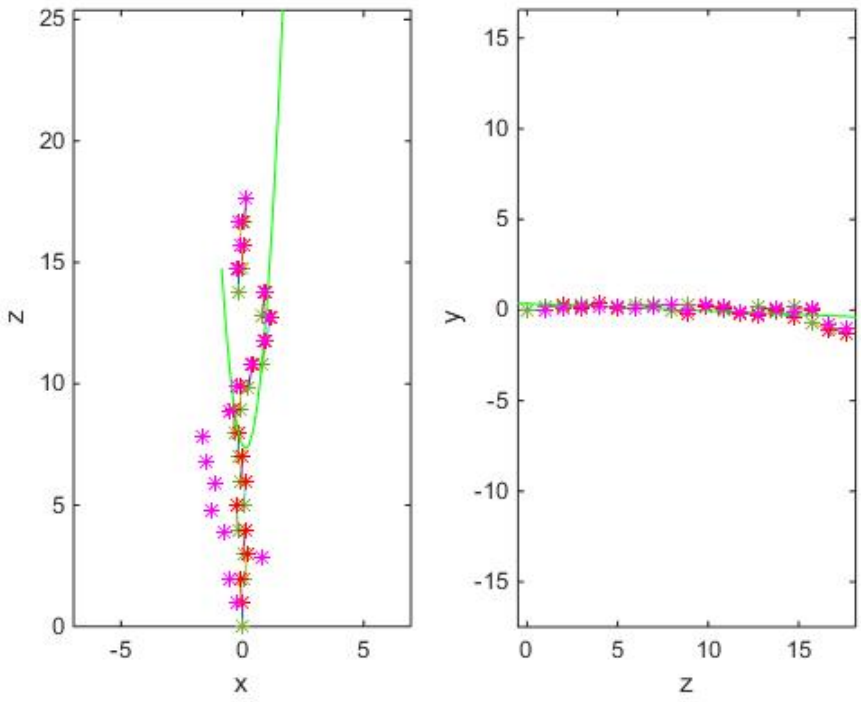

Figure A.11: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 5, system version 1.

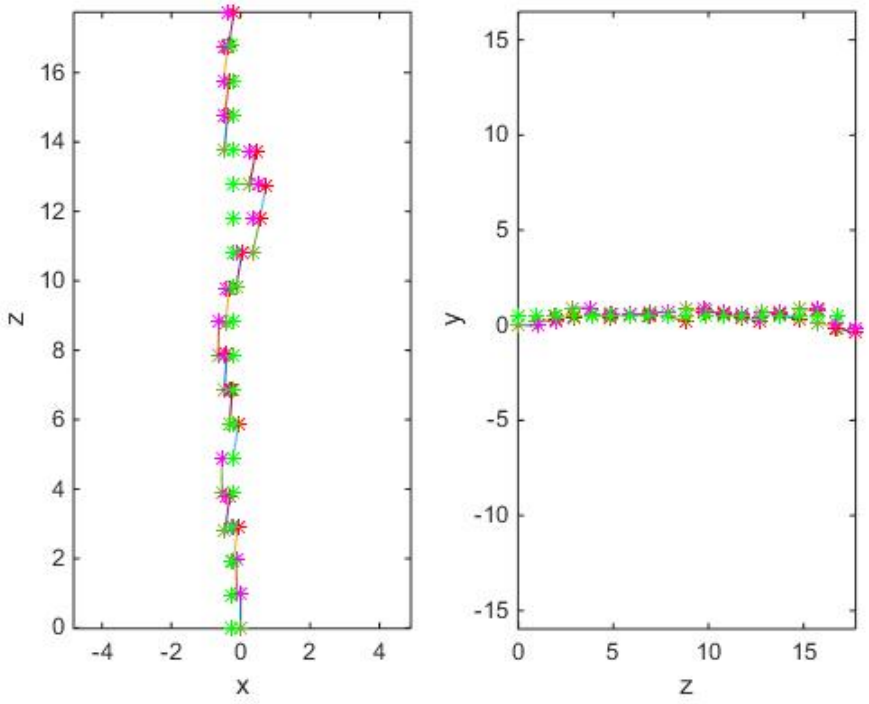

Figure A.12: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 5, system version 2.

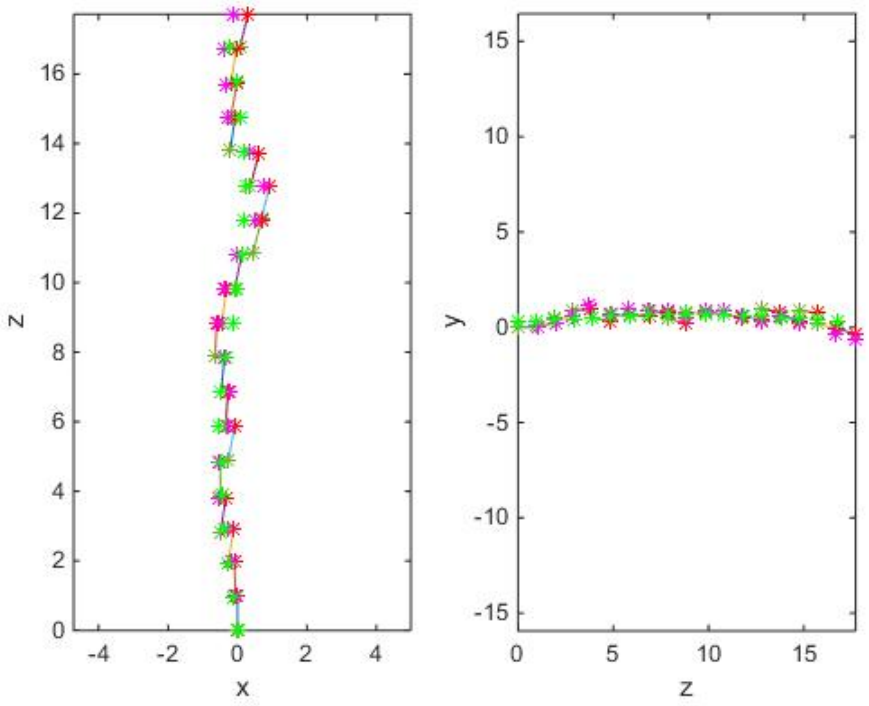

Figure A.13: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 5, system version 3.

A.1.6 Sequence 6

Sequence number and scenery	6, Botan		
Number of pictures	50		
System version number	1	2	3
Execution time in seconds	error	116	119
Mean of number of features		1719	1719
Mean of number of matches		223	223
Mean of number of inliers		180	180
Mean of ratio of matches and inliers		1.3	1.3
Std of matches in x		130	130
Std of matches in y		103	103
Std of features in x		190	190
Std of features in y		107	107
Std of inliers in x		111	110
Std of inliers in y		99	98

Table A.6: Table showing each evaluated feature, see section 3.6 for more information, for sequence 6 in the three system versions.

System version 1 could not handle this sequence, although it is taken at the exact same place and time as sequence 5. This leads to the conclusion that system version 1 can not process longer sequences. System version 2 and 3 handles it fine and the camera centers are placed in a way that represents the walked way. Since the camera centers is placed in a fairly straight line it may be that system version 2 has made a better fit than system version 3, see Figures A.14 and A.15.

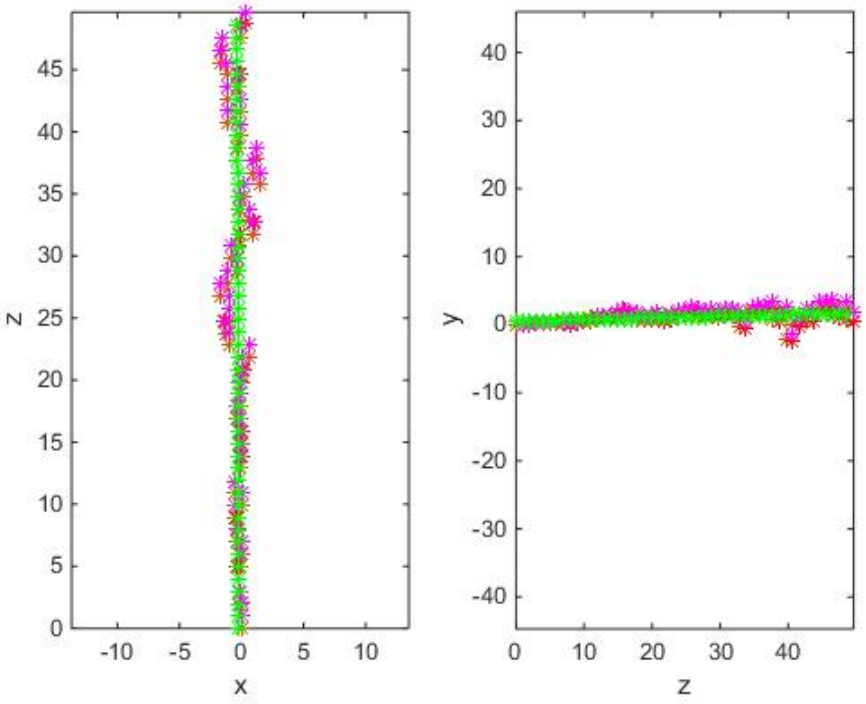

Figure A.14: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 6, system version 2.

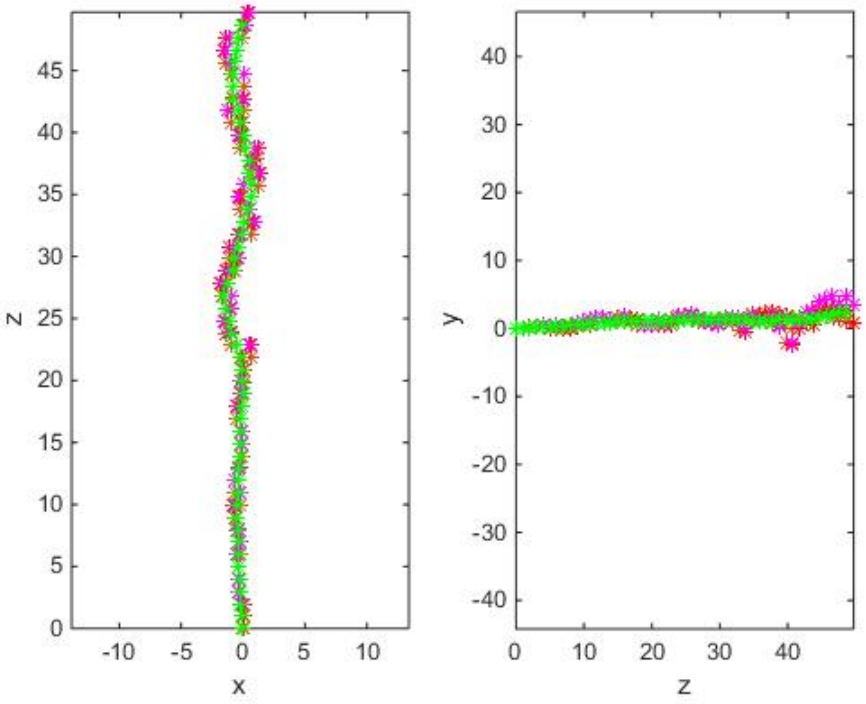

Figure A.15: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 6, system version 3.

A.1.7 Sequence 7

Sequence number and scenery	7, Stortorget		
Number of pictures	7		
System version number	1	2	3
Execution time in seconds	269	15	18
Mean of number of features	1137	1164	657
Mean of number of matches	558	145	172
Mean of number of inliers	364	120	153
Mean of ratio of matches and inliers	2	1.2	1.16
Std of matches in x	614	145	145
Std of matches in y	521	105	105
Std of features in x	596	152	152
Std of features in y	977	109	109
Std of inliers in x	571	133	133
Std of inliers in y	332	75	77

Table A.7: Table showing each evaluated feature, see section 3.6 for more information, for sequence 7 in the three system versions.

This sequence is the sequence we tested our code on in the beginning, since the execution time is relatively low due to the low number of images it consists of. For comparison of the different system versions it is not very interesting, all of them performed good on this sequence, but it is, as stated above, the sequence we have processed most and therefore we did not want to leave it out.

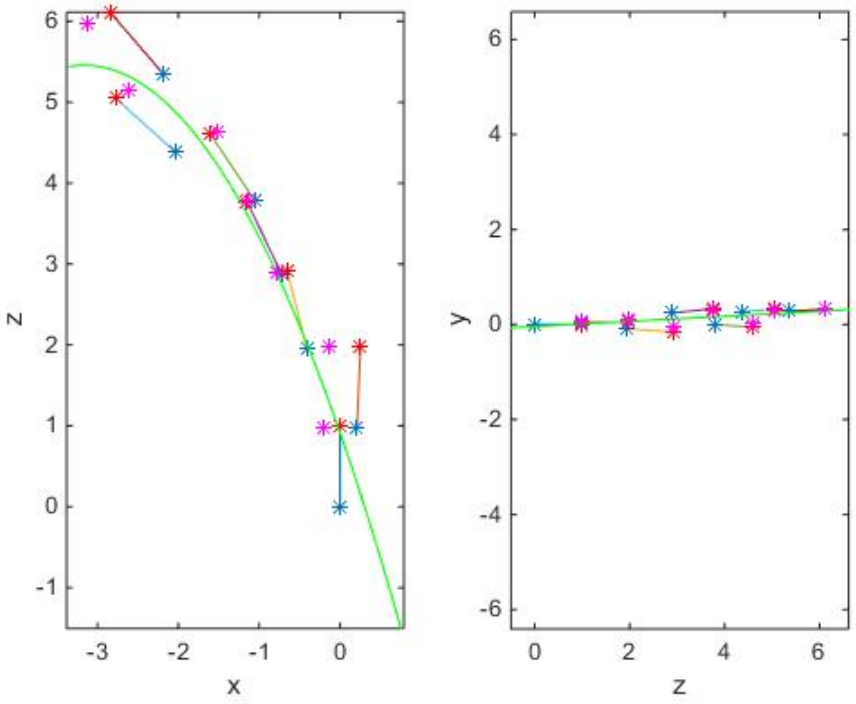

Figure A.16: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 7, system version 1.

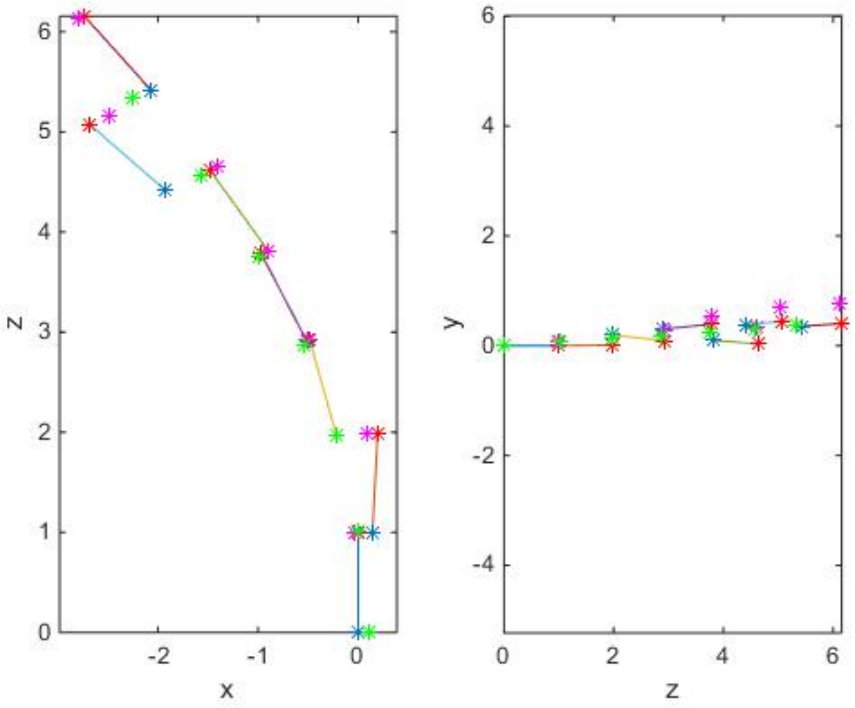

Figure A.17: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 7, system version 2.

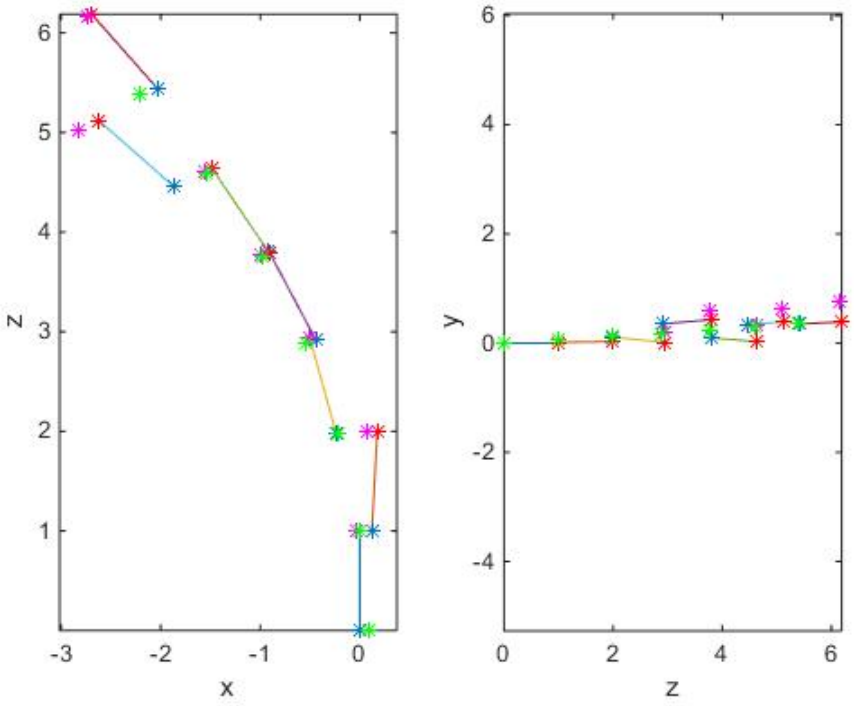

Figure A.18: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 7, system version 3.

A.1.8 Sequence 8

Sequence number and scenery	8, LTH		
Number of pictures	182		
System version number	1	2	3
Execution time in seconds	error	422	437
Mean of number of features		1075	1075
Mean of number of matches		138	138
Mean of number of inliers		117	117
Mean of ratio of matches and inliers		1.2	1.2
Std of matches in x		118	118
Std of matches in y		118	118
Std of features in x		164	164
Std of features in y		117	117
Std of inliers in x		94	99
Std of inliers in y		86	95

Table A.8: Table showing each evaluated feature, see section 3.6 for more information, for sequence 8 in the three system versions.

The system versions that did managed to run this sequence, system version 2 and 3, did not perform as well as we had hoped. The sequence is long, yes, but it is taken with three steps in between each image and the view orientations were kind so it should come out better than this. As seen in Figures A.19 and A.20 the camera centers is places very odd and does not represent the actual walked way.

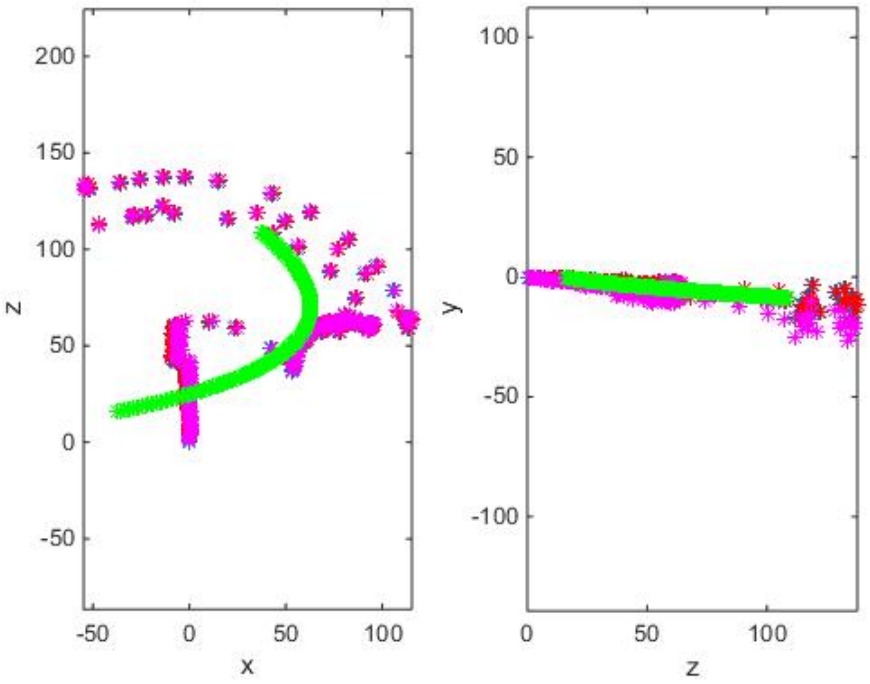

Figure A.19: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 8, system version 2.

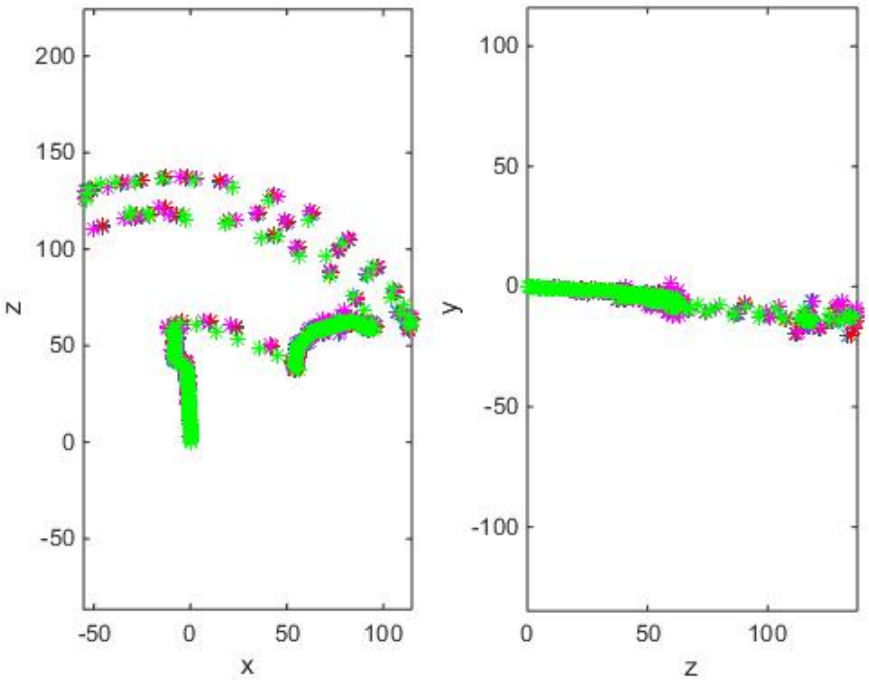

Figure A.20: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 8, system version 3.

A.1.9 Sequence 9

Sequence number and scenery	9, Stadsparken		
Number of pictures	42		
System version number	1	2	3
Execution time in seconds	error	105	117
Mean of number of features		1176	1176
Mean of number of matches		146	146
Mean of number of inliers		121	121
Mean of ratio of matches and inliers		1.2	1.2
Std of matches in x		110	110
Std of matches in y		181	181
Std of features in x		166	166
Std of features in y		192	192
Std of inliers in x		100	99
Std of inliers in y		157	150

Table A.9: Table showing each evaluated feature, see section 3.6 for more information, for sequence 9 in the three system versions.

That this sequence looks very bad, see Figures A.21 and A.22, is not very surprising since the images were taken with very different view orientations. The algorithm probably could not get enough feature matches between every image pair to relate them in a good way.

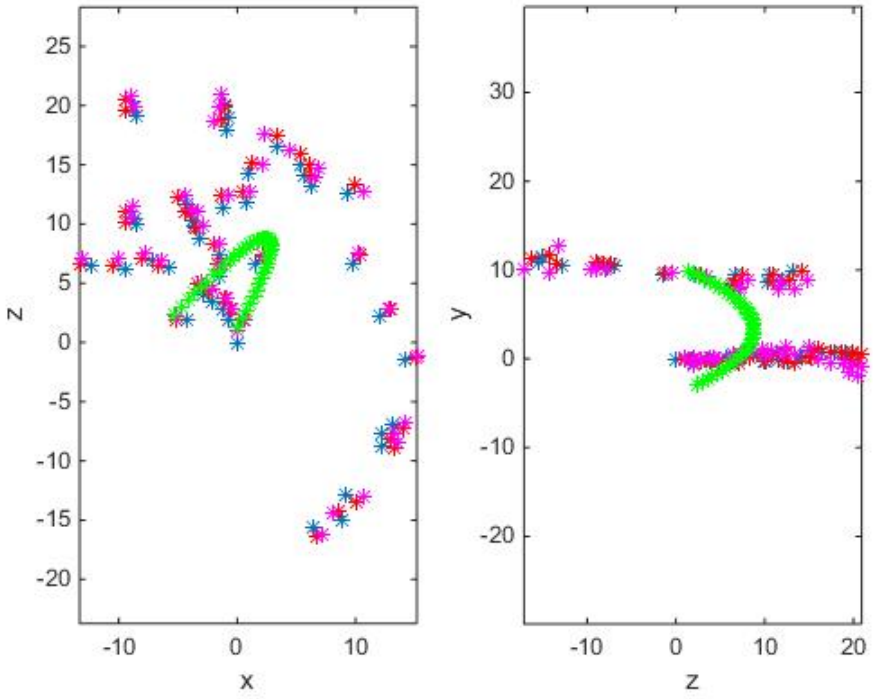

Figure A.21: Plot over camera centers (blue), view-orientations (red), new view orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 9, system version 2.

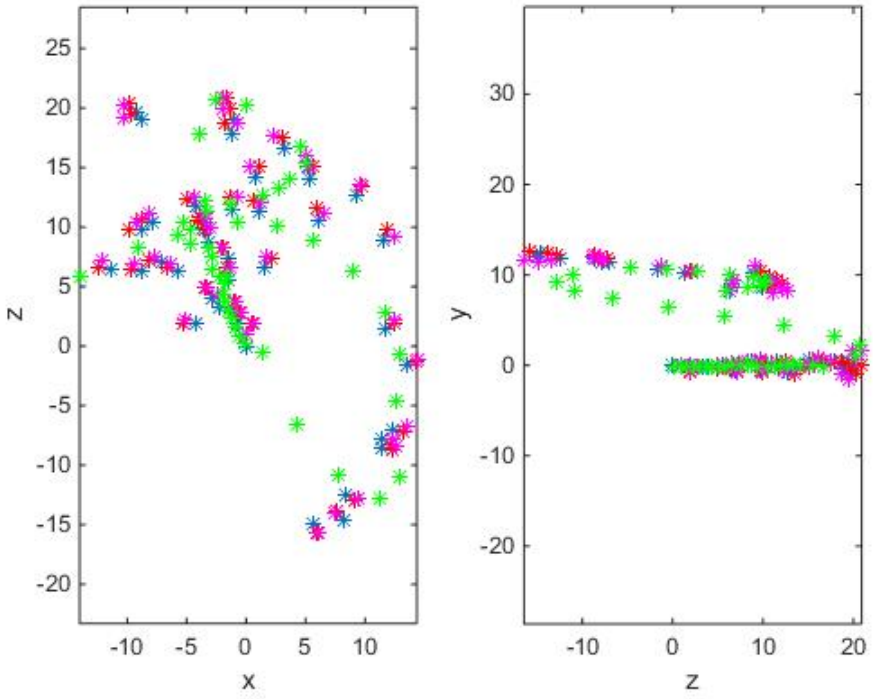

Figure A.22: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 9, system version 3.

A.1.10 Sequence 10

Sequence number and scenery	10, Löparbanan		
Number of pictures	168		
System version number	1	2	3
Execution time in seconds	error	372	393
Mean of number of features		621	647
Mean of number of matches		175	176
Mean of number of inliers		155	157
Mean of ratio of matches and inliers		1.16	1.15
Std of matches in x		136	136
Std of matches in y		62	62
Std of features in x		159	159
Std of features in y		147	147
Std of inliers in x		127	127
Std of inliers in y		62	62

Table A.10: Table showing each evaluated feature, see section 3.6 for more information, for sequence 10 in the three system versions.

Since this sequence is taken at the running tracks a second order polynomial never had a chance to work, see Figure A.23. System version 3 could have had a chance if the camera centers had been placed in a circle, as they should have been. System version 3 still does a good curve fit, see Figure A.24, considering the placement of the data points.

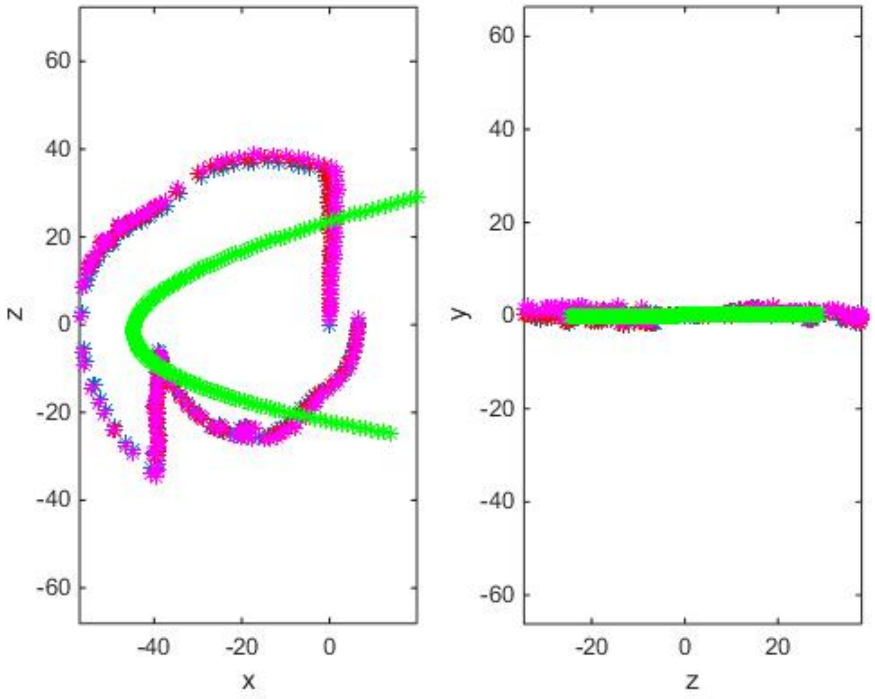

Figure A.23: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 10, system version 2.

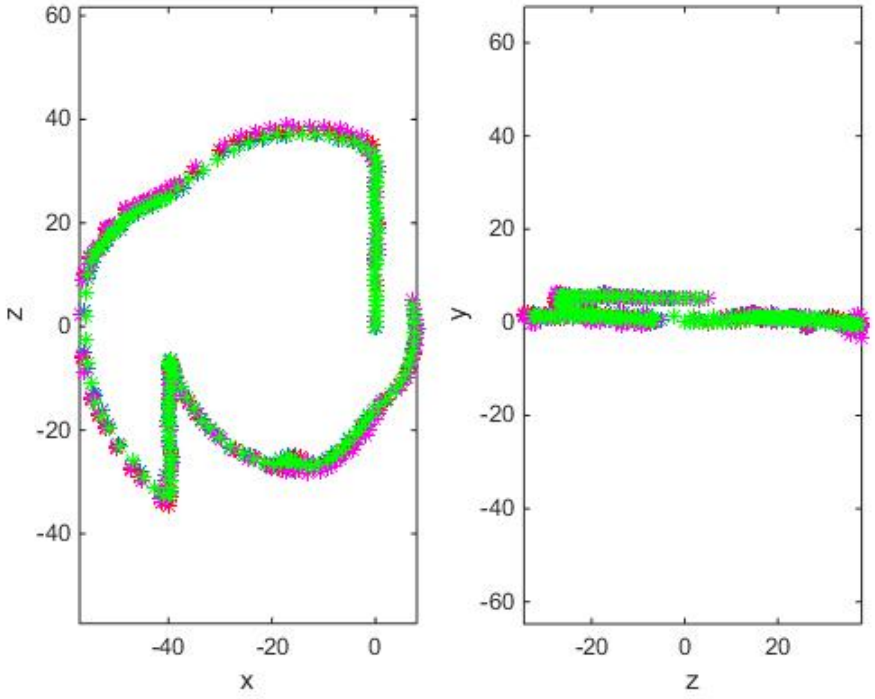

Figure A.24: Plot over camera centers (blue), view orientations (red), new view-orientations (magenta) and fitted curve (green) in xz-plane (left) and zy-plane (right) for data sequence 10, system version 3.

Chapter 8

Bibliography

[1] Wikipedia RANSAC article. http://en.wikipedia.org/wiki/RANSAC, May 2015.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision - ECCV 2006*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer Berlin Heidelberg, 2006.

[3] M. Chasles. Question 296. *Nouv. Ann. Math.*, 14(50), 1855.

[4] M. Demazure. Sur deux problemes de reconstruction. *Technical Report 882, INRIA*, 1988.

[5] Olof Enqvist, Fredrik Kahl, and Carl Olsson. Non-sequential structure from motion. In *Workshop on Omnidirectional Vision, Camera Networks and Non-Classical Cameras*, 2011.

[6] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications-of-the-ACM*, 24(6):381–95, 1981.

[7] David A. Forsyth and Jean Ponce. *Computer Vision A Modern Approach*. Pearson, ISBN-10: 013608592X, second edition, 2011.

[8] Amit Goldstein and Raanan Fattal. Video stabilization using epipolar geometry. *ACM Trans. Graph.*, 31(5):126:1–126:10, September 2012.

[9] M. Grundmann, V. Kwatra, and I. Essa. Auto-directed video stabilization with robust l1 optimal camera paths. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 225–232, Washington, DC, USA, 2011. IEEE Computer Society.

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

[11] A. Heyden and K. Åström. Euclidean reconstruction from constant intrinsic parameters. In *ICPR'96*, volume 1, pages 339–343. IEEE Computer Society Press, 1996.

[12] A. Heyden and G. Sparr. Reconstruction from calibrated cameras - a new proof of the kruppa demazure theorem. 1997. to appear.

[13] A. Karpenko. The technology behind hyperlapse from instagram. http://instagram-engineering.tumblr.com/post/95922900787/hyperlapse, May 2015.

[14] Johannes Kopf, Michael F. Cohen, and Richard Szeliski. First-person hyper-lapse videos. *ACM Trans. Graph.*, 33(4):78:1–78:10, July 2014.

[15] Johannes Kopf, Michael F. Cohen, and Richard Szeliski. The difference between microsoft's and instagram's hyperlapse algorithms. http://research.microsoft.com/en-us/um/redmond/projects/hyperlapse/igcomp/, May 2015.

[16] E. Kruppa. Zur Ermittlung eines Objektes Zwei Perspektiven mit innerer Orientierung. *Sitz-Ber. Akad. Wiss., Wien, math. naturw. Kl. Abt*, IIa(122):1939–1948, 1905.

[17] DavidG. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[18] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.

[19] D.G. Lowe. Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image, March 23 2004. US Patent 6,711,293.

[20] J Matas, O Chum, M Urban, and T Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761 – 767, 2004. British Machine Vision Computing 2002.

[21] S.J. Maybank. The projective geometry of ambigous surfaces. *Philosophical Transactions of the Royal Society*, 1990.

[22] Narrative. Narrative clip - specifications. http://getnarrative.com/narrative-clip-1#specs, May 2015.

[23] Carl Olsson. Computer vision, lecture notes. http://www.maths.lth.se/matematiklth/personal/calle/datorseende14/index.html, March 2014.

[24] Yair Poleg, Tavi Halperin, Chetan Arora, and Shmuel Peleg. Egosampling: Fast-forward and stereo for egocentric videos. *CoRR*, abs/1412.3596, 2014.

[25] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[26] Henrik Stewenius, Christopher Engels, and David Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[27] Wenping Wang, Helmut Pottmann, and Yang Liu. Fitting b-spline curves to point clouds by curvature-based squared distance minimization. *ACM Trans. Graph.*, 25(2):214–238, April 2006.

[28] Eric W. Weisstein. "B-Spline." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/B-Spline.html, May 2015.

Master's Theses in Mathematical Sciences 2015:E14
ISSN 1404-6342

LUTFMA-3274-2015

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/