Bachelor Informatica

University of Amsterdam

Video stitching for virtual reality environments

Kristian Korrel

June 8, 2016

Supervisors: Rein van den Boomgaard | UvA

Rowan de Graaf | ILLuSKY

Signed:

Abstract

With the upcoming of virtual reality, there is an apparent need for practical and matured applications that utilize the full potential of headsets made for experiencing virtual reality. In this thesis we will focus on the process of creating videos for virtual reality by combining the recordings of multiple cameras, to create a fully immersive, $360^\circ \times 180^\circ$ spherical video. We will explore the complete process of video stitching and look into possible improvements on this process to create a more natural-looking and seamless panorama. The method we used for stitching videos uses areas of overlap between the individual frames to recover the geometric properties of each camera. We discovered that the main reasons for bad stitches are radial distortions of the lenses, bad relative positions and rotations of the cameras, and frames with too little overlap or too little detail in the captured scene.

Contents

CHAPTER 1

Introduction

With the introduction of multiple commercially available virtual headsets, the need for matured virtual reality applications has risen. Virtual reality headsets can provide the user with a greater feeling of immersion and interactivity in entertainment and educational applications than traditional media like televisions and personal computers. Although the most used application for virtual reality are video games, virtual reality films could also be an interesting application.

A virtual reality film could be in the form of a video projected onto a two-dimensional plane in the 3D virtual environment. Although this does create the ability to watch videos in any room without a traditional monitor, and the ability to create a virtual environment in which the video is watched, this does not fully utilize the immersive possibilities of virtual reality. A technique that does utilize these possibilities is one where the video is projected on a sphere surrounding the viewer in all directions. With this technique, the viewer is given the illusion of being located in the scene where the action and acting takes place, and is given the freedom to observe this scene in every possible direction. To improve on this feeling of immersion, techniques like dynamic spatial sound and stereoscopy may be used.

To create a virtual reality film, we need a projection of a real world scene. The most ideal solution to this would be a camera with a field of view of 360 degrees horizontally and 180 degrees vertically that would cover the entire visual sphere with one lens. Although there are cameras available that can record 360 degrees horizontally with one lens, no camera is able to cover all viewing angles on one lens. There is therefore a need to approximate this behavior by combining the recordings of multiple cameras into one video.

Image stitching is a well studied problem in computer vision. Lowe [1] described a method on how to find local features in images and use these to match images containing the same objects. In collaboration with Brown [2], he described in more detail how above mentioned algorithm could be used to stitch images. These techniques have been used in various image and video stitching software.

Using some of the available software products, you could come to the conclusion that it is quite hard to acquire a seamlessly stitched video without the user manually refining the found parameters, like focal lengths and the rotation of each camera. This is a task that typically requires a profound knowledge about the stitching process and the cameras used, to do correctly. With virtual reality arguably becoming as commonplace as personal computers and smartphones, there is an apparent need to simplify the process of creating virtual reality films.

For this reason, we will focus on how existing image stitching techniques can be applied to the problem of stitching videos in order to create a $360^\circ \times 180^\circ$ fully spherical video to be used in virtual reality environments. We will further focus on possible reasons why, in some cases, this does not result in seamless panoramas, and whether we can make adjustments to this process to improve on the stitching of said cases.

The remaining part of this thesis is structured as follows: Section 2 describes the theory behind every step necessary in the process of video stitching. In Section 3 we will highlight how we have implemented this. What follows are the results of this implementation performing on different datasets in Section 4 and a discussion of these results in Section 5.

CHAPTER 2

Stitching pipeline

In this chapter, we will describe all the steps that we performed to create a spherical video by stitching multiple regular videos. First, we have to prepare the footage for stitching. This includes synchronizing them and correcting their possible distortions. What follows are several steps in which we identify overlapping areas between frames and use these to estimate the relative rotations of the cameras. This starts with finding and matching feature points between frames. These matches are used to estimate the pair-wise homography matrices, which define the geometric relationship between the frames of two different cameras. To avoid propagating errors by stitching with those pair-wise defined homographies, we use bundle adjustment to refine the relative rotations of all cameras jointly. When the correct rotations are calculated, we can project the frames onto a sphere and blend them together to create a single spherical frame.

Figure 2.1: An overview of the stitching pipeline.

2.1 Frame synchronization

When dealing with multiple-camera setups, there is no good solution to start the recording of every device at the same time. Therefore, we wish to temporally synchronize the frames of every camera before we begin with the stitching process.

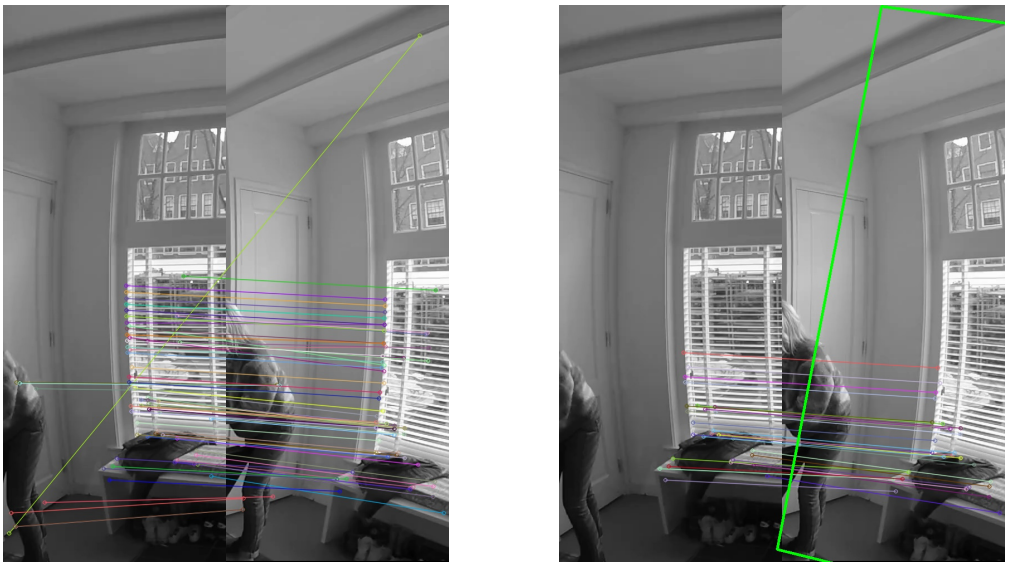
This is a task that can be accomplished in multiple ways. One method involves synchronizing the audio tracks using dynamic time warping [3]. By comparing detected features in the audio waveforms, we can determine the temporal shift between every audio track. For every video we should select the frame that best matches the audio offset as the first frame for that video.

Similarly, we could synchronize the videos by analyzing motion between consecutive frames [4]. This does require a distinguishable, equal motion between frames in each camera. This can most easily be accomplished by moving the camera setup itself, thereby creating an equal motion in all videos at the same time. When this motion can reliably be detected, synchronization of frames is analogous to audio-based synchronization.

Both methods do impose some conditions on the recorded footage, but if those conditions are met, both methods can provide a reliable synchronization. We believe that frame synchronization is beyond the scope of this project, in which we focus on video stitching. Therefore, we used third-party software to find the temporal shift between each camera. This software is capable of applying both methods of synchronization.

2.2 Radial distortion correction

Most camera lenses come with some form of distortion. Although the distortion could theoretically come in all sorts of patterns, in approximation most lenses cause a radially symmetric distortion. Whether this effect is desirable or not depends on the use case. However, when these frames have to be stitched together, we need to correct the distortion first, as the geometrical models that will later be estimated, assume distortion free images. Since the distortion is radially symmetric, the



(a) (b)

Figure 2.2: Figure 2.2a shows some of the best matching feature points of two details of distorted frames. Figure 2.2b shows the found homography from the left frame to the right frame and its inliers. We can clearly see that the straight, vertical lines of the window are curved in opposite directions in each frame. Theoretically, this makes it impossible to find a perfect homography.

distortion is similar for all pixels with the same radius from the center of distortion c_d. The distortion for a certain radius is generally described with the polynomial model [5]

$$r_u = r_d * L(r_d, k) \tag{2.1}$$

where r_u and r_d are the pixel radii from c_d in the undistorted and distorted frames respectively, and $L(r_d, k)$ is the polynomial expression

$$L(r_d, k) = 1 + k_1 r_d^2 + k_2 r_d^4 + \cdots + k_n r_d^{2n} \tag{2.2}$$

where $k_1, \ldots, k_n$ are the distortion coefficient we want to estimate.

Closely related to this model is the division model [6], which typically requires less coefficients to describe the distortion.

$$r_u = \frac{r_d}{L(r_d, k)} \tag{2.3}$$

We define the amount of distortion for a frame as

$$\frac{\left| r_u^{max} - r_d^{max} \right|}{r_u^{max}} * 100\% \tag{2.4}$$

where r_u^{max} and r_d^{max} are the maximum values for the radii in the undistorted and distorted frame respectively. They do not necessarily have to be related as described in Equation 2.3.

Various methods have been proposed to find these distortion parameters. Several are based on finding straight lines in the real world and adjusting the distortion parameters by means of an iterative process to straighten out the corresponding curves in the distorted image, like [7]. Non-iterative methods have also been proposed, like the proposed method of Li and Hartley [5]. Correcting the distortion in such ways has the benefit that distortion does not have to be dealt with within the video stitching process itself. It does however, require the user to either calibrate each camera themselves or to use a lookup-table to find the distortion parameters for every camera.

Figure 2.3 shows a distorted video frame and its correction. It shows that the corrected version of a frame is not rectangular anymore. Two things should be taken into consideration to ensure a good stitch of the corrected frames. To maintain a large enough area of overlap, the corrected frames should not be cropped to the point where we get rectangular frames again. Furthermore, in the feature extraction stage (see Section 2.3), we must prevent feature points outside of the actual frame and features that are caused by the edges of the undistorted frame to be detected.

To accomplish this, we need a way to distinguish the pixels which contain information about the captured scene from the pixels that do not contain any useful

information. We did this by adding an alpha channel to the undistorted frame. The alpha component of each pixel is only filled for pixels that come from the captured scene. We use the alpha channel of the frame as a region of interest in the feature extraction stage. This filters out feature point detection outside of the captured scene, but still allows the feature detection algorithm to detect feature point right on the edges and in the corners of the scene that would not be detected in a regular, rectangular frame. We have found that applying a small erosion function on this alpha channel to slightly decrease the size, yields better results in our experiments. We are aware that this method could possibly also filter out legitimate feature points. Another way to tackle this problem would be not to use erosion on the alpha channel, but filter out feature points after they have been detected in the region of interest. The feature points we would have to filter out are the ones of which the circular neighborhood, in which the feature point is found, intersects the edge of the captured scene.

Ju and Kang [8] proposed a method in which the distortion parameters are estimated alongside the homography estimation in the RANSAC stage (see Section 2.5). This homography estimation depends on the repeatability property of the feature extraction algorithm, meaning that the same feature points are found in different images of the same scene. Experiments by Lourenço et al. [9] have shown that the repeatability properties of SIFT quickly degrades with increasing distortions. This can be explained by the fact that radial distortion causes parts of the captured scene to compress or expand on the frame in a non-linear way, thereby changing the spatial frequencies in the frame. They propose an adjustment to the

Figure 2.3: A distorted frame and its correction. The used distortion parameters were found by trial-and-error.

traditional SIFT algorithm, called sRD-SIFT, in which the shape of the Gaussian convolution kernels used in SIFT follows the same pattern as the distortion. Their experiments show that with distortions of up to around 35%, using sRD-SIFT on distorted images yields better results than using SIFT on either distorted or corrected images. With larger distortions, using SIFT on corrected images yields the best results. Their sRD-SIFT algorithm however, requires the distortion parameters to be known beforehand to determine the shape of the kernel, making the proposed method by Ju and Kang [8] of determining these parameters in the RANSAC stage obsolete in this scenario.

We have chosen to correct the distortions of every frame before performing feature extraction. With distortions of less than 35%, a better result could possibly be achieved by using sRD-SIFT on the distorted images with the already retrieved distortion parameters, and correcting them after the feature points are detected with those same parameters. The remaining part of the stitching process will then be performed on the corrected images. A downside of this strategy is that it requires the user to either calibrate their cameras or use an exhaustive database in which the distortion parameters for their lenses are documented.

2.3 Feature extraction

The basis for many computer vision applications is detecting and describing local features in the images. The most notable feature extraction algorithm is the Scale-Invariant Feature Transform algorithm (SIFT) proposed by Lowe [1], but many others are designed to accomplish the same task, like SURF [10], ORB [11] and KAZE [12]. This paper is not an survey on feature extraction algorithms, but we imagine several algorithms could be used depending on the needs of the user, like performance and robustness, and whether or not the algorithm is patented.

A feature point is generally described as a point in the image that stands out based on its surroundings, and that can robustly be detected in several images of the same scene. For each of the extracted feature points, an accompanying vector is calculated, called its descriptor. A feature extraction algorithm is generally considered to be robust when these feature points and descriptors are extracted similarly under certain conditions. We say that the algorithm is invariant to conditions such as image rotation, translation and scaling. Properties that could all well be present in the set of frames that we want to stitch. These invariances allow us to determine whether and where objects in the real world are present in the frames of different cameras, which could provide information about the mutual relationship between cameras.

2.4 Feature matching

The feature points and corresponding descriptors found in the previous stage will be used to pairwise match frames with overlapping areas to each other. For each descriptor of a frame, we want to find the best matching descriptor of every other frame. Instead of a linear nearest neighbor search, we use the Fast Library for Approximate Nearest Neighbors (FLANN) [13]. This is a library with multiple approximate nearest neighbor search algorithms that trade some accuracy for potential enormous performance improvements. In optimal conditions it is found to be 3 orders of magnitude faster than exhaustive nearest neighbor search.

Muja and Lowe [13] conducted experiments to find out which of these algorithms are most appropriate for different kinds of datasets and requirements by the user, like search speed and memory usage. They implemented an algorithm that chooses the most appropriate approximate nearest neighbor algorithm and its parameters based on a subset of the dataset and the users preferences. As can be read in Section 3 however, our implementation is based on OpenCV, whose wrapper for FLANN currently does not support automatic algorithm selection. Based on the findings of Muja and Lowe, we have chosen to use one randomized KD-tree.

A classical KD-tree is a binary tree in which every internal node splits a subset of the data in two smaller subsets by a hyperplane orthogonal to the dimension that shows the greatest variance. Typically, the hyperplane is chosen to intersect at the median value of that dimension to generate two equal-sized subsets. Silpa-Anan and Hartley [14] have proposed an adjustment to the classical KD-tree which performs better on larger datasets. This algorithm generates one or more randomized KD-trees. A randomized KD-tree splits the subset of each internal node at a randomly chosen dimension of the D dimensions with the greatest variance, where D is empirically chosen to be 5. So in our case, we generate one randomized KD-tree for every frame. This is searched for an approximate nearest neighbor by all feature descriptors in all other frames.

Searching the tree to find an approximate nearest neighbor for query vector q is done with priority search. Starting at the root, we travel down the tree by continuously selecting the child node to which q should belong if we look at the dimension on which the node is split. When a leaf node is reached, it is selected as the best candidate so far. There is a good chance however, that this is not actually the nearest neighbor. For this reason, we construct a hypersphere with q as the origin and the euclidean distance to the best candidate as its radius. We travel back up the tree again and check for every sibling we have not visited whether the hypersphere intersects this cell. If it does, the node is added to the priority queue with its distance to q as the priority. We now repeat this process for the node in the priority queue with the smallest distance to q until we have emptied the priority queue. By setting a maximum for the amount of leaf nodes that may be visited, we can prioritize between accuracy and performance.

2.5 Image matching

Now that we have determined the approximate best match for every descriptor between frames, we will use this to estimate the homography matrix between each pair of frames. We use RANdom SAmple Consensus (RANSAC) and the Direct Linear Transformation (DLT) algorithm in order to calculate the homographies between frames. We then use a probabilistic model to determine whether the area of overlap between frames is large enough, if there is any overlap at all. In this section, we will first shortly explain what a homography is. After that, we will digress about RANSAC, DLT and the used probabilistic model.

If we imagine a world coordinate system in which a camera is centered in the origin, 3D points in coordinate system can be projected to image coordinates by

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \cong KRX = \begin{bmatrix} \frac{f_x}{a_x} & s & c_x \\ 0 & \frac{f_y}{a_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.5}$$

K is the camera's intrinsic matrix and R its rotation or extrinsic matrix. Since the camera is located at the origin, the extrinsic matrix does not contain any translations. f_x and f_y are the cameras focal lengths on both axes. These are normalized by a_x and a_y to get the focal lengths measured in pixel distance. Since pixel coordinates are normally counted from $(0, 0)$ in the top left corner of the image, K contains a translation to the images principal point c. For simplicity, and because this is approximately true in most cases, we assume the skew s to be zero and c to be located in the center of the image.

We can apply the inverse of equation 2.5 to recover the direction of X from its frame coordinate.

$$X \cong R^T K^{-1} \tilde{x} \tag{2.6}$$

Theoretically, we can thus define the relationship between points in two frames i and j shot by cameras that have only been rotated around a single, common center of projection with the 3×3 homography matrix H_{ij}

$$\tilde{x}_j \cong K_j R_j R_i^T K_i^{-1} \tilde{x}_i = H_{ij} \tilde{x}_i \tag{2.7}$$

This is a special kind of homography for cameras that have not been translated, but only rotated in the origin.

RANSAC is an iterative method to estimate a model that best fits a certain dataset. At the same time, RANSAC determines the inliers and outliers of this model. The set of outliers are datapoints that do not fit in the estimated model. In the case of video stitching, these are, for example, feature points caused by extreme noise or parallax effects, or feature points outside of the area of overlap between frames.

In each iteration, a random subset of the dataset is chosen. This subset is used to estimate a model to fit the complete dataset. For this hypothetical model, the set of inliers and outliers are determined. After N iterations, the model with the largest set of inliers is chosen to be the best fitting model for the dataset. This model is then improved upon by calculating a model based on its set of inliers.

In image stitching, the model we want to estimate in the RANSAC phase is the homography between two images. The data we need for this are the matching feature points determined in section 2.4. If we have i matching feature points x_i and x_i' in two different images, we want to know the 3×3 homography matrix H that projects the positions of feature points x_i in homogeneous coordinates to x_i'

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2.8}$$

If we write this multiplication out and express it in Cartesian coordinates, we get

$$x_i' = \frac{x_i h_{11} + y_i h_{12} + h_{13}}{x_i h_{31} + y_i h_{32} + h_{33}} \tag{2.9}$$

$$y_i' = \frac{x_i h_{21} + y_i h_{22} + h_{23}}{x_i h_{31} + y_i h_{32} + h_{33}}, \tag{2.10}$$

We can rewrite equation 2.9 as

$$-x_i h_{11} - y_i h_{12} - h_{13} + x_i'(x_i h_{31} + y_i h_{32} + h_{33}) = 0 \tag{2.11}$$

which can be represented in vector notation as

$$\begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x_i' & y_i x_i' & x_i' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{bmatrix} = 0 \tag{2.12}$$

If we do the same for equation 2.10, we get

$$\begin{bmatrix} 0 & 0 & 0 & -x_i & -y_i & -1 & x_i y_i' & y_i y_i' & y_i' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{bmatrix} = 0 \tag{2.13}$$

Each pair of matching feature points results in two equations (2.12 and 2.13). A homography is defined up to scale, and thus has 8 degrees of freedom to solve

for. To calculate $h = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{33} \end{bmatrix}^T$, we thus need 4 non-collinear correspondences. If we stack these 8 equations on top of each other, we get

$$Ah = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' & y_2' \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' & x_3' \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y_3' & y_3y_3' & y_3' \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' & x_4' \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y_4' & y_4y_4' & y_4' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \vec{0} \quad (2.14)$$

To determine the values for H, we must find the one-dimensional right null space of A. We can accomplish this by applying a singular value decomposition on A

$$A = U\Sigma V^T \quad (2.15)$$

where Σ is a diagonal matrix containing $\sigma_1, \sigma_2, \dots, \sigma_n$ in the diagonal and V^T an orthogonal matrix. If the feature points x_i and x_i' are related by a homography, there should be one $\sigma_j = 0$. The null space for A must thus by the j'th column of V. Because the point correspondences are not exact, we will select the column V_j that corresponds to the smallest σ_j in Σ as h, containing the values for H.

For every pair of frames, we have now estimated a homography which describes the relationship between the two frames. We should also take into account that some frames might not have any direct relationship at all. Some frames might not have any area of overlap, or this could be too small to get a reliable estimation of the homography. Therefore, we calculate the probability of the estimated homography actually coming from two matching frames.

Brown and Lowe [2] have proposed a probabilistic verification model to determine whether the found model comes from two images that actually have an overlap. We denote the number of matching feature points in the found area of overlap with n_f. The number of inliers is n_i. Furthermore, we define the event that the images are matching with $m \in \{0, 1\}$ and the event that a matching feature point is an inlier or outlier with $f^{(i)} \in \{0, 1\}$.

They assume that the following posterior probabilities follow a Binomial distribution

$$p(f^{(1:n_f)}|m = 1) = \binom{n_f}{n_i} p_1^{n_i}(1 - p_1)^{n_f - n_i} \quad (2.16)$$

$$p(f^{(1:n_f)}|m = 0) = \binom{n_f}{n_i} p_0^{n_i}(1 - p_0)^{n_f - n_i} \quad (2.17)$$

where $f^{(1:n_f)}$ is the set $\{f^{(1)}, \dots, f^{(f_n)}\}$, and p_1 and p_0 are the probabilities of a matching feature being an inlier given a correct image match or a false image

match respectively. By Bayes' theorem, we get that

$$\begin{aligned} p(m=1|f^{(1:n_f)}) &= \frac{p(f^{(1:n_f)}|m=1)p(m=1)}{p(f^{(1:n_f)}|m=1)p(m=1)+p(f^{(1:n_f)}|m=0)p(m=0)} \\ &= \frac{1}{1+\frac{p(f^{(1:n_f)}|m=0)p(m=0)}{p(f^{(1:n_f)}|m=1)p(m=1)}} \end{aligned} \tag{2.18}$$

We say that the images match when this probability is larger than a certain p_{min}. We can thus derive the following condition.

$$\begin{aligned} \frac{1}{1+\frac{p(f^{(1:n_f)}|m=0)p(m=0)}{p(f^{(1:n_f)}|m=1)p(m=1)}} &> p_{min} \\ 1+\frac{p(f^{(1:n_f)}|m=0)p(m=0)}{p(f^{(1:n_f)}|m=1)p(m=1)} &> \frac{1}{p_{min}} \\ \frac{p(f^{(1:n_f)}|m=0)p(m=0)}{p(f^{(1:n_f)}|m=1)p(m=1)} &> \frac{1}{p_{min}}-1 \\ \frac{p(f^{(1:n_f)}|m=1)p(m=1)}{p(f^{(1:n_f)}|m=0)p(m=0)} &> \frac{1}{\frac{1}{p_{min}-1}-1} \\ \frac{\binom{n_f}{n_i}p_1^{n_i}(1-p_1)^{n_f-n_i}p(m=1)}{\binom{n_f}{n_i}p_0^{n_i}(1-p_0)^{n_f-n_i}p(m=0)} &> \frac{1}{\frac{1}{p_{min}-1}-1} \end{aligned} \tag{2.19}$$

Brown and Lowe have chosen to use $p_0 = 0.1$, $p_1 = 0.6$, $p(m = 1) = 10^{-6}$ and $p_{min} = 0.999$. With these parameters, we get the condition

$$n_i > 8 + 0.3n_f \tag{2.20}$$

indicating a correct image match.

2.6 Bundle adjustment

If we would only have two frames to stitch, the estimated homography H_{ij} could be used to transform frame i to overlay frame j. If we have more than two frames to stitch, pairwise transforming the frames with corresponding homographies would result in a bad stitch. Since the estimated homographies are not exact, an accumulated error would result in visible seams and undesirable black areas when frames match to multiple other frames. Better results can be achieved by estimating each camera's intrinsic and extrinsic parameters with respect to all other cameras with matching frames. These parameters can be estimated for each camera jointly with bundle adjustment [15].

As we saw in Section 2.5, we can decompose the homography for frames of cameras that have only been rotated in the origin as

$$H_{ij} = K_j R_j R_i^T K_i^{-1} \tag{2.21}$$

Where K contains the focal lengths of a camera, and R its rotation matrix. With bundle adjustment, we use the homographies estimated in Section 2.5 and refine the individual parameters by solving a non-linear least squares problem with a minimization algorithm.

We initialize the bundle adjuster with parameters K_i and R_i of each camera by adding the frames one-by-one and decomposing the homography to one of the frames already in the bundle adjuster.

We then use the already calculated matching feature points of each frame and all its other matching frames to define a reprojection error

$$r = \sum_{i=1}^{n} \sum_{j \in M(i)} \sum_{(f_i, f_j) \in F(i,j)} h(f_j - H_{ij} f_i) \tag{2.22}$$

where n is the total number of frames, $M(i)$ is the set of matching frames for frame i and $F(i, j)$ is the set of matching feature points between frames i and j. Instead of a squared loss function, we use a Huber loss function h. this leads to an estimation which is less sensitive for outliers than a quadratic loss function.

We try to find a minimum value of this reprojection error by iteratively updating K_i and R_i of each camera with a minimization algorithm. In bundle adjustment, typically the Levenberg-Marquardt algorithm (LMA) is chosen because of its robustness.

The gradient of the reprojection error in terms of parameters K_i and R_i can be derived analytically. In each iteration of the LMA, we calculate this gradient with the current K_i and R_i of every camera as parameters and update the parameters in a similar manner as in the gradient descent algorithm. The size of the step we take in this direction is based on how the reprojection error has changed in the last iteration. If the error has only decreased slightly, this step will be larger in the next iteration to speed up the process. If either the step size becomes too large or the gradient gets below a certain threshold, we accept the found parameters.

This algorithm is only able to find a local minimum of the reprojection error. For this reason, we need to find already somewhat correct estimates of the homographies, as described in Section 2.5, before we use bundle adjustment to refine the individual parameters K and R of each camera. Since the reprojection error estimates these parameters for each camera jointly, we should have resolved the problems that could arise if we would pairwise stitch the frames with the homographies found in Section 2.5.

2.7 Blending

When the intrinsic and extrinsic parameters for every camera are known, a panorama can be constructed. In areas where two or more frames are overlapping, the result should be a composite of these frames. This allows to straighten out some minor errors in the estimation of radial distortion and other parameters. Since the

different cameras are not physically at the exact same location, some inevitable parallax effects that occur for nearby objects could also be masked to some degree.

We have chosen to apply multi-band blending [2]. The idea behind this method is that high spatial frequency objects in the frames need to blended over a smaller region, whereas lower frequency objects need to be blended over a larger region. We will use the term *levels* to denote the different ranges of frequencies with which we blend. The first step in this process is to create a weight map for every frame which is 1 at the center of the frame and gradually decreases to 0 at each border. This weight map for a frame i is defined as

$$W_i(x, y) = w_i(x)w_i(y) = \frac{|x - c_{i,x}|}{\frac{1}{2}\text{width}_i} \frac{|y - c_{i,y}|}{\frac{1}{2}\text{height}_i} \tag{2.23}$$

where c_i is the center of frame i. For distorted frames which have been corrected, width_i and height_i indicate the maximum width and height of the corrected frame.

Next, we calculate the so-called max-weight map W_i^{max} for every frame. This map will hold information about the input pixel of which of the n frames is most responsible for any output pixel. Therefore, we define this map in the spherical coordinate system of the to be constructed panorama (see Section 2.8). Input pixels (x, y) from different frames may be mapped to the same output pixel (θ, ϕ). Only the one with the highest value $W_i(x, y)$ will cause the corresponding $W_{max}^i(\theta, \phi)$

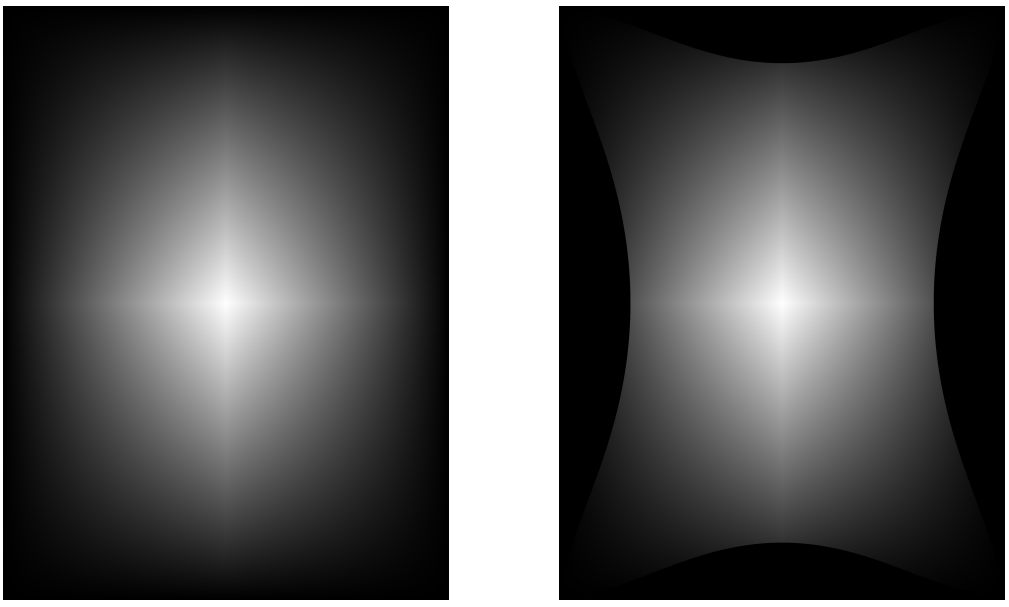


Figure 2.4: Illustrations of weight maps $W_i(x, y)$ for a frame without radial distortion and a frame with radial distortion which has been corrected. It has a maximum value at the center and linearly goes to 0 in both the x and y direction.

to hold a weight of 1.

$$W^i_{max}(\theta,\phi) = \begin{cases} 1 & \text{if } W_i(x,y) = \max\limits_{j\in[0,n]} W_j(x,y) \\ 0 & \text{else} \end{cases} \tag{2.24}$$

where the (x, y) positions for every frame are backwards-mapped from the corresponding (θ, ϕ) position.

We create a blurred version of these max-weight maps for every level by convoluting them with Gaussians of increasing σ. When we want to blend with l levels, we define the blurred versions of the max-weight maps $W^i_{\sigma_k}$ for $k = 0, \ldots, l-1$ recursively as

$$W^i_{\sigma_k} = \begin{cases} W^i_{max} * g_\sigma & \text{if } k = 0 \\ W^i_{\sigma_{k-1}} * g_{\sqrt{2k+1}\sigma} & \text{if } 0 < k < l \end{cases} \tag{2.25}$$

The convolution of two Gaussians with variances s_1^2 and s_2^2 is also a Gaussian with variance $s^2 = s_1^2 + s_2^2$. Above definition thus results in Gaussian blurs of W^i_{max} with standard deviations $\sigma, 2\sigma, \ldots, l\sigma$.

Convolution by a Gaussian function acts as a low-pass filter, filtering out high spatial frequencies. If we have a Gaussian blurred image and subtract a Gaussian blurred image with a larger standard deviation, this difference of Gaussians (DoG) acts as a band-pass filter, leaving only frequencies in a certain range.

For every frame, we create $l-1$ differences of Gaussians, which will be used to blend in levels 0 to $l-2$. These differences of Gaussians act as band-pass filters in the ranges $[0, \sigma], [\sigma, 2\sigma], \ldots, [(l-2)\sigma, (l-1)\sigma]$. Level $l-1$ will be blended with a standard Gaussian blurred version of the frame. This is illustrated in Figure 2.5. If we would also subsample the constructed frame at each level, this would be called a Laplacian pyramid.

For every frame I^i we define the l images in the Laplacian pyramid used for blending recursively as

$$B^i_{\sigma_k} = \begin{cases} I^i_{\sigma_k} - I^i_{\sigma_{k+1}} & \text{if } 0 \le k < l-1 \\ I^i_{\sigma_k} & \text{if } k = l-1 \end{cases} \tag{2.26}$$

where

$$I^i_{\sigma_k} = \begin{cases} I^i & \text{if } k = 0 \\ I^i_{\sigma_{k-1}} * g_{\sqrt{2k+1}\sigma} & \text{if } 0 < k < l \end{cases} \tag{2.27}$$

For every frame I^i, we have now defined l blurred max-weight maps and a Laplacian pyramid with l levels. We will first blend the n frames together per level. For level k, we take for every frame the particular version in its Laplacian pyramid and

multiply it with its respective blurred max-weight map. We add these weighted version of the images up and normalize it by the combined weights in level k to get the output panorama O in level k.

$$O_{\sigma_k} = \frac{\sum_{i=1}^{n} B^i_{\sigma_k} W^i_{\sigma_k}}{\sum_{i=1}^{n} W^i_{\sigma_k}} \tag{2.28}$$

Similar to how an image is normally reconstructed from its Laplacian pyramid, we add each of the weighted versions up to produce a blended composite of the individual frames.

$$O = \sum_{k=0}^{l-1} O_{\sigma_k} \tag{2.29}$$

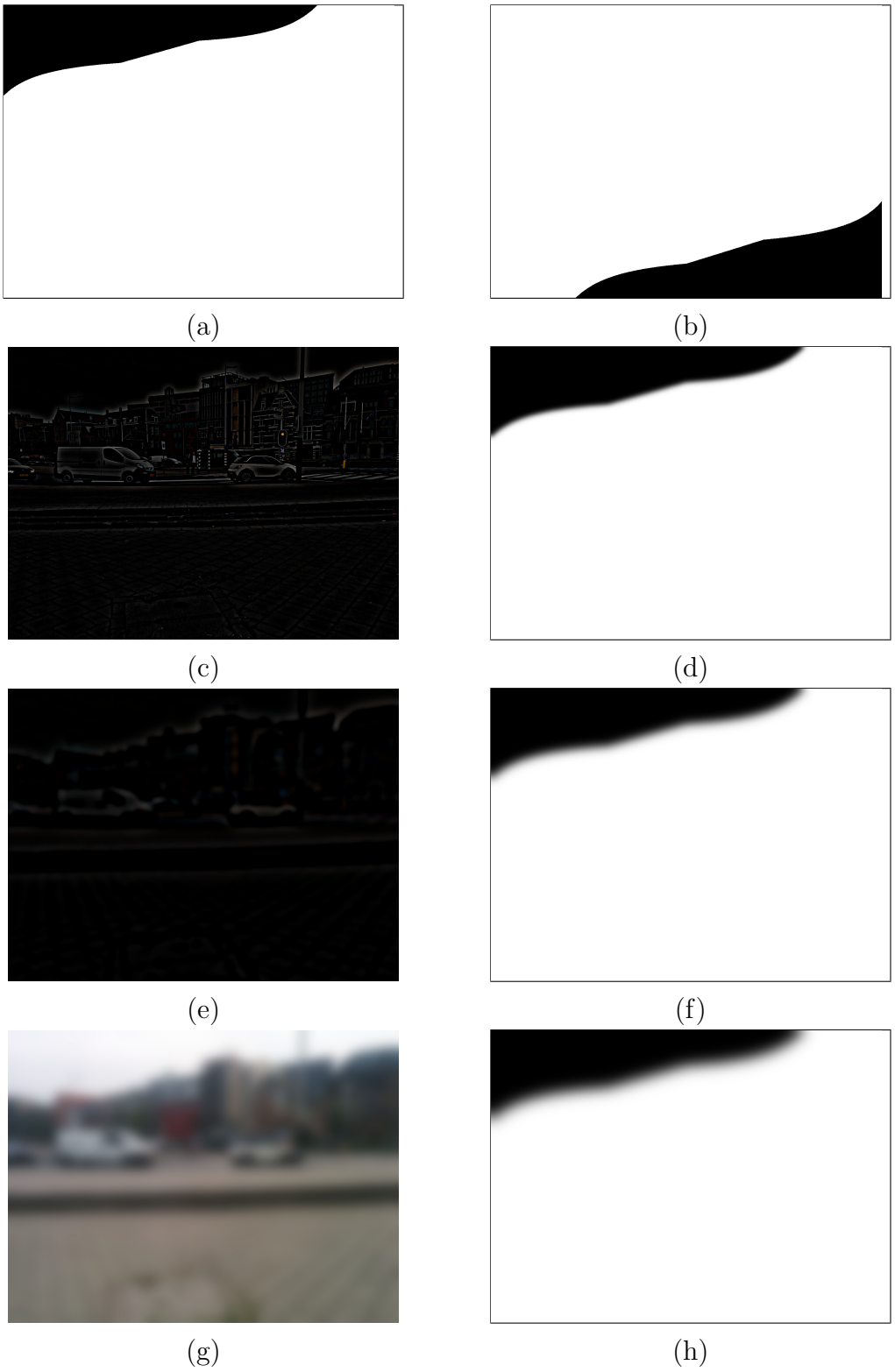


Figure 2.5: Figures 2.5a and 2.5b show the max-weight maps for two images that are blended with multi-band blending. The results of this panorama is shown in Figure 2.6. Figures 2.5c, 2.5e and 2.5g show $B^i_{\sigma_0}$ to $B^i_{\sigma_2}$. Figures 2.5d, 2.5f and 2.5h show $W^i_{\sigma_0}$ to $W^i_{\sigma_2}$. We used 3 levels with initial $\sigma = 10$

(a)

(b)

(c)

Figure 2.6: Figures 2.6a and 2.6b show two images that were shot hand-held. Figure 2.6c shows the result of stitching these images and blending them with 3 levels and an initial $\sigma = 10$.

2.8 Representation

The final process of blending and composing the panoramas can be done on different surfaces, depending both on how the footage is shot and what representation is most useful for the viewer. When blending just a couple of videos, of which the relative camera rotations are relatively small, a regular flat surface might be the right choice since this does not require specially designed software or hardware to view. When the footage is shot in a full circle of 360° horizontally, a cylindrical representation could be appropriate. Whereas for 360° × 180° panoramas, to be viewed in virtual reality environments, a spherical representation would be a more logical choice.

We have chosen to project the video on the inside of a unit sphere around the origin. The software for viewing this video would then place the viewer in the origin, allowing to view the inside of the sphere on which the video is projected. These frames are digitally stored as typical 2D frames by an equirectangular projection. This projection simply takes the (θ, ϕ) positions and normalizes them to the dimensions of the frame to get a pixel coordinate. Since θ and ϕ are in the range $[0, 2\pi]$ and $[0, \pi]$ respectively, this representation typically has a 2 by 1 ratio.

This projection causes a lot of distortion nearing both poles, with the extreme case of every pixel of the first and last row representing the same spherical coordinate of each pole respectively. Its ease of implementing however, makes that it is considered to be the standard for texture mapping spheres. For the same reason, it is the main method used in software for viewing spherical images and videos.

In Section 2.7 we described that we need a mapping from spherical coordinates to the frame coordinates and vice versa. For a spherical coordinate (θ, ϕ), we can compute the 3D position with

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} cos(\phi)sin(\theta) \\ sin(\phi) \\ cos(\phi)cos(\theta) \end{bmatrix} \tag{2.30}$$

This 3D point can be projected to frame coordinates as described in Equation 2.5.

CHAPTER 3

Implementation

We have implemented the steps described in sections 2.2 to 2.8 in C++ using the computer vision library OpenCV 3.1.0. Determining each videos delay for synchronization was determined with an external tool, called Kolor Autopano.

Video can be thought of as a series of images that are shown in rapid succession. The process of video stitching is therefore not much different from that of image stitching. However, if we assume that the cameras are steadily mounted on a rigid body, we can take advantage of the fact that the relative geometric properties of each camera does not change during the filming of a scene. Feature extraction, feature matching, image matching and bundle adjustment are all performed on a single set of frames to gain a tremendous improvement in performance. With the estimated camera parameters, we can construct a data object which holds information about which input pixels from each frame should be blended together with which weighting to compose every output pixel.

After this data object is obtained, we can compose a panorama of each set of frames with the following three steps. First, we correct the distortions of each frame with the previously estimated distortion parameters. Next, we construct a Laplacian pyramid with l levels for each frame, as described in Section 2.7. Finally, these different version of the frame are combined as described in the blending data object to compose a single spherical panorama.

Video stitching is quite a computationally expensive procedure. The performance of our method depends mostly on the amount of input images or videos, their respective resolution, the desired resolution of the constructed panorama and the desired level of blending. Especially blurring frames by convolution with Gaussians in both the feature extraction and blending stages take their toll on processing power and memory usage. The performance of bundle adjustment can vary wildly, depending on the initializations of its parameters.

Luckily, our implemented method of video stitching lends itself greatly for parallelization in some steps. While calibrating the cameras, for example, feature point extraction can be performed for every frame separately. Every pair of frames can

be processed separately in the feature matching and image matching stages. When composing the panoramas, we can think of two obvious ways of partitioning the data to allow easy parallelization. We can let each thread compose a panorama from its own set of frames, or we can process one set of frames at a time and let each thread produce a different set of output pixels. The user must be wary however, that when processing more sets of frames at the same time, keeping all those frames and their Laplacian pyramid in memory might not be feasible on present-day personal computers. We were able to successfully implement parallelization in some steps, but because of these limitations in memory usage, we decided not to use it in our experiments.

CHAPTER 4

Results

We have tested our method on videos as well as images. With most of our datasets, panoramas can be constructed with minimal or no visible seams at all (Figure 4.1).

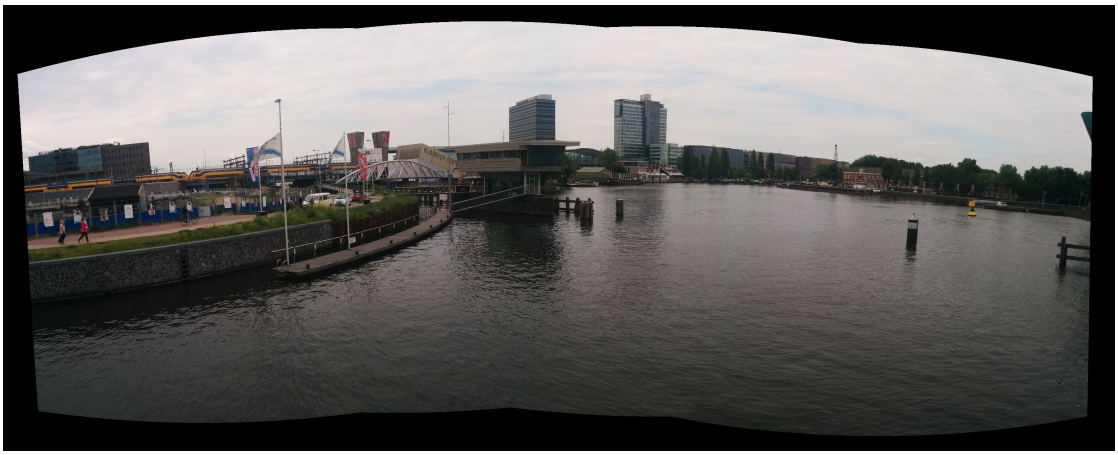


Figure 4.1: The result of stitching 4 images that were taken hand-held. The used camera did not seem to show any significant radial distortion, so it was not necessary to correct distortions in this scenario.

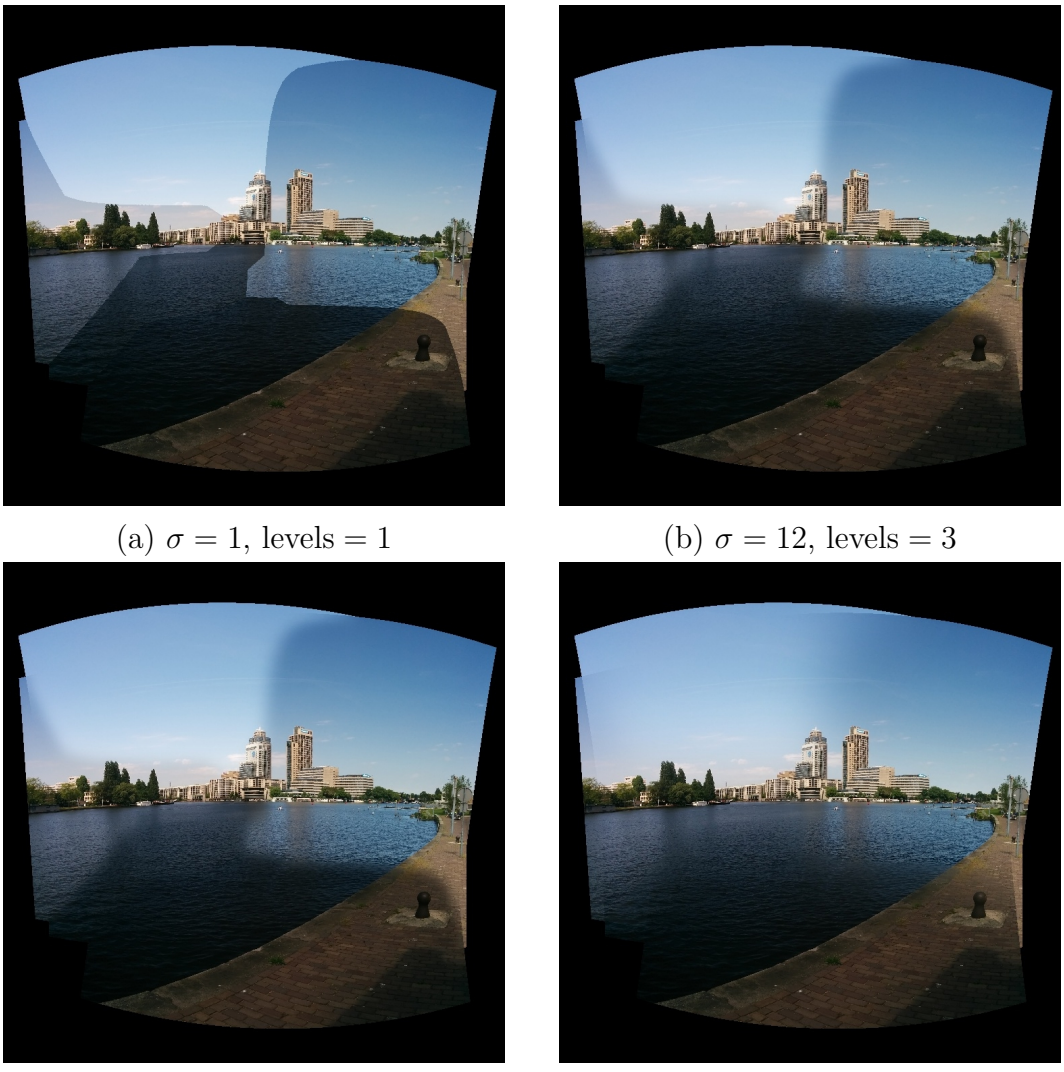
Figure 4.2 shows the positive effects of multi-band blending. Not only can minor misalignment be masked, with optimal parameters, we are even able to produce a somewhat smooth transition between images with different lighting conditions.

As pointed out in Section 3, the process of video stitching is an expensive operation, but is a great candidate for parallelization. High memory usage however, is a factor that should be taken into account. We also think that single core performance is an easier to compare measurement. This is why we present the performance without parallelization.

We measured the execution time of each stage in the stitching process. We used the footage of 7 different cameras with a resolution of 1920×1440 pixels and a

significant radial distortion. The distortion parameters were estimated beforehand by trial-and-error, but the correction had to be applied to every individual frame. This experiment was performed multiple times on an Intel Core i7-3610QM processor, after which we took the average execution time for every stage. The results are shown in Table 4.1.

Thus far, we have presented how well our implemented system works for certain datasets. This is certainly not the case for every dataset however. The original motive for this thesis was a request for the company ILLuSky, which specializes in virtual reality, to aid in the process of creating a virtual reality film. The editors of this film found that the stitching process was difficult and time-consuming. We were not able to achieve the results we wanted for this particular dataset. We



(a) $\sigma = 1$, levels $= 1$

(b) $\sigma = 12$, levels $= 3$

(c) $\sigma = 3$, levels $= 12$

(d) $\sigma = 12$, levels $= 12$

Figure 4.2: The result of stitching 6 images that were shot hand-held. We manually adjusted the lighting for each image. Different parameters were used for blending.

will try to analyze the causes for these disappointing results and produce a list of conditions the recorded footage must comply to, to achieve better results with our method.

4.3 shows the result of stitching the 7 videos of this dataset together. The panorama shows almost no resemblance to the actual recorded footage. We believe that this has several causes. First of all, the footage with which we created the panorama contained significant radial distortion. Since we did not have access to the particular cameras used, we were unable to find the distortion parameters with any of the methods described in Section 2.2. We estimated these parameters ourselves by trial-and-error, which produced a slightly better result, see Figure 4.4. In this figure we can see some interesting phenomena. Our method was unable to correctly calculate the rotation of the camera that was aimed towards the ceiling. We suspect that this comes from the fact that this part of the scene contains too little objects with enough detail, which negatively impacts the repeatability property of SIFT (Figure 4.5).

The fact that none of the frames are perfectly aligned to each other, especially at the wall nearest to the camera setup, could possibly be contributed to the fact that the cameras are not physically at the exact same location, causing parallax effects.

	Execution time (ms)	**% of total execution time**
Camera calibration	220313	100
Extracting features	21117	9.58
Matching features	9269	4.21
Matching images	2975	1.35
Bundle adjustment	63322	28.74
Calculating blend weights	123630	56.12
Panorama composition per frame	26408	100
Correcting distortion	19917	75.42
Creating Laplacian pyramids	3931	14.89
Blending and projecting onto sphere	2560	9.69

Table 4.1: Average execution time of individual stages and contribution to the total execution time of both the calibration phase in which we find the geometric properties of each camera and the composition phase in which we compose a panorama of each set of frames. The execution time of the composition phase is averaged over the total amount of panoramas we compose. Since we are mostly interested in the relative execution time of the stages described in Chapter 2, the total execution time of both phases were obtained by adding the individual components up. We did not include execution time of distortion correction in the camera calibration stage and trivial parts like reading the videos from memory.

Another explanation would be that the frames had too little area of overlap, which results in too little feature points to be matched between frames. A solution would be to use more cameras to enlarge the areas of overlap.

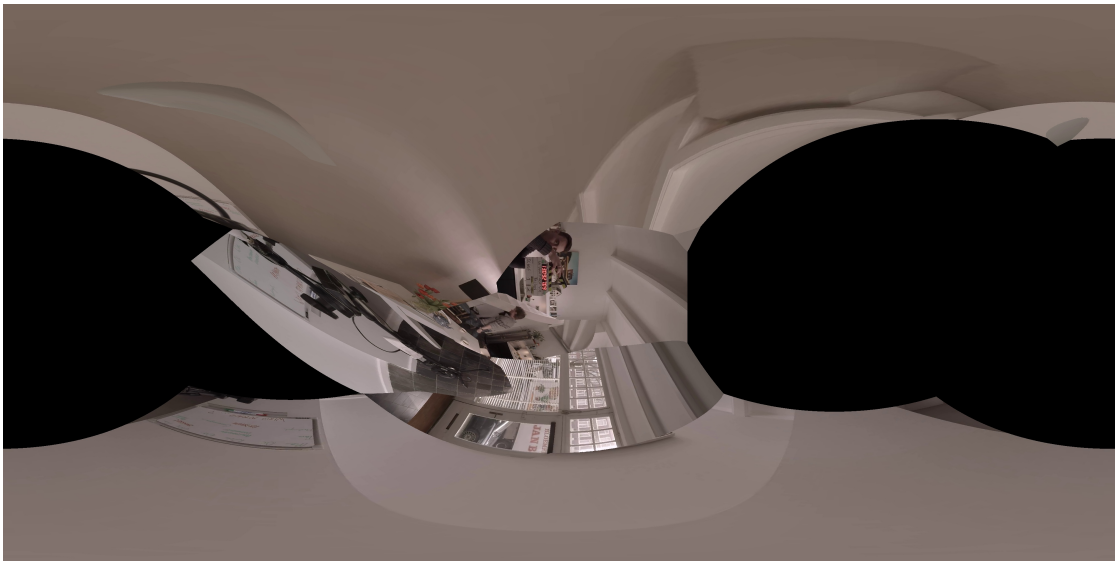


Figure 4.3: A video still of the result of stitching 7 videos. The cameras contain significant radial distortion, which was not corrected.

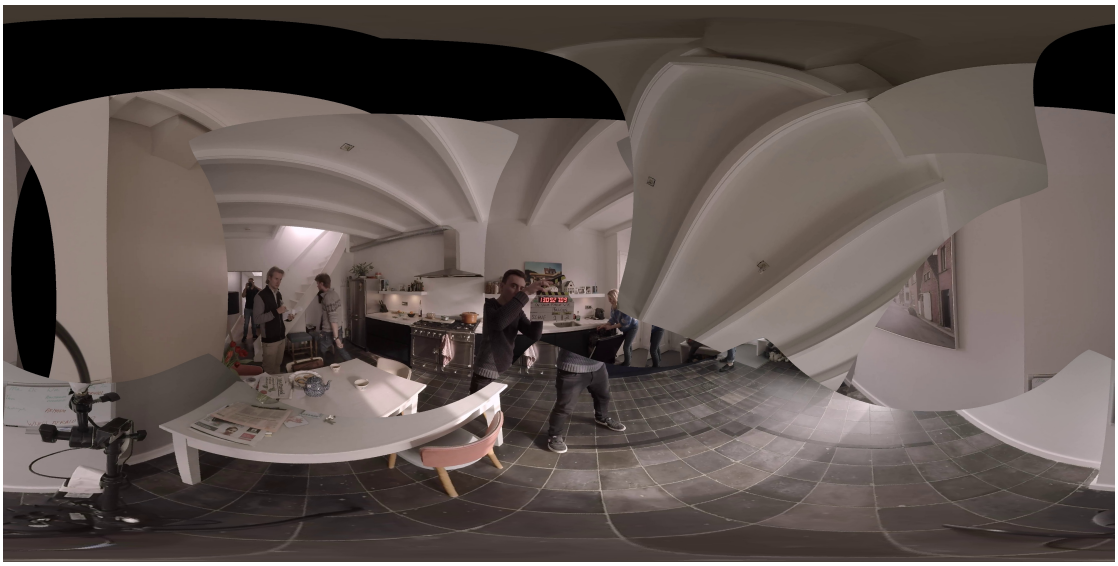


Figure 4.4: The result of stitching the same footage used in Figure 4.3, but with correcting the radial distortion first. The distortion parameters were found by trial-and-error.

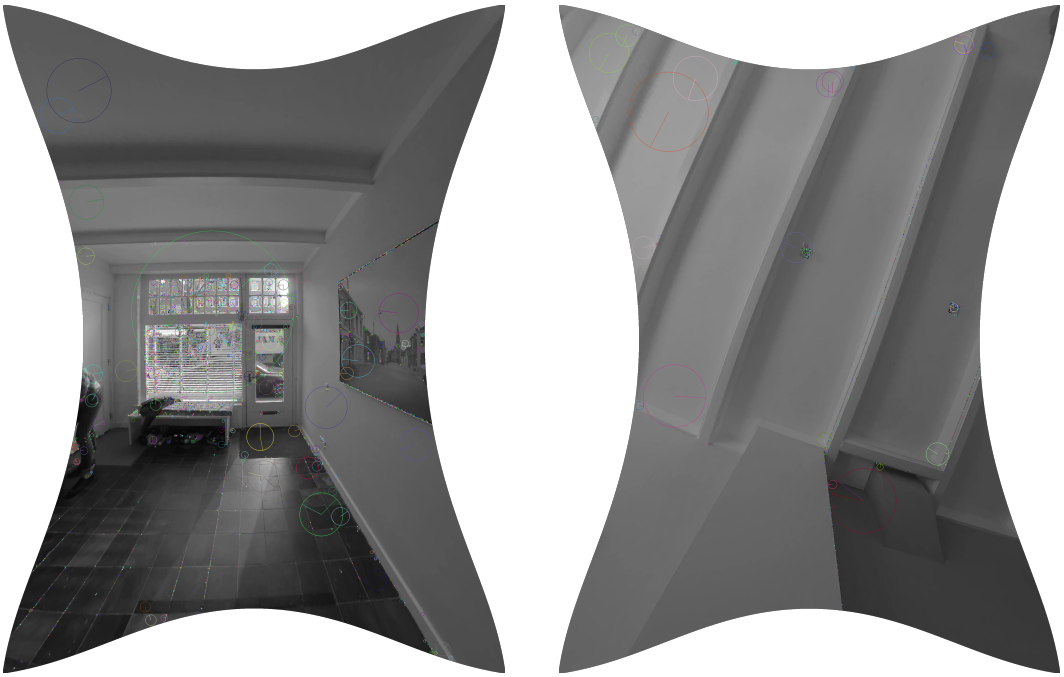


Figure 4.5: We show the feature points there were found on two different frames. The image of the ceiling does not contain much feature points. This makes it hard to find the correct homography.

CHAPTER 5

Conclusions

In this thesis, we described the process of stitching multiple videos into a $360° \times 180°$ spherical video that is suitable for viewing in a virtual reality environment. In summary, this process involves:

1. Preparing the footage by synchronizing them and correcting their distortions.
2. Detecting feature points and their respective descriptors in every frame.
3. Matching these feature points between frames.
4. Using the found matches to estimate pairwise homographies.
5. Refining the individual components of these homographies with the use of a non-linear minimization algorithm.
6. Calculating the blending weights for every level of each frame.
7. Blending the Laplacian pyramid of each frame with the calculated weights and projecting the panoramas onto a sphere.

This method is applicable to both video and image stitching. In the case of video stitching, if the cameras are mounted firmly and their relative rotation to each other does not change during the filming of a scene, steps 1 to 6 can be performed on a single set of frames, after which step 7 has to be performed for all other sets of frames.

We implemented this method in C++ with the use of the OpenCV library. Our experiments show that the construction of a seamless panorama greatly depends on how and where the footage was shot. This is why we make the following recommendations concerning the filming of a scene for those who want to create seamless spherical videos.

To reduce parallax effects, the cameras must be placed as close to each other as possible. This requirements does depend on the distance to the captured objects.

The closer the objects are, the closer the cameras must be to each other. The cameras must also be located on an imaginary sphere in such a way that lines orthogonal to the center of each lens intersect each-other at the center of the camera setup. By adding more cameras, we get a better approximation of a sphere and more overlapping area between frames, which can improve the robustness of image matching. If we want to speed up the process of image stitching by performing above mentioned steps 1 to 6 on only one set of frames, the cameras must also be firmly mounted onto a rigid body.

Our results also show that the calculated camera parameters in steps 1 to 6 depend on the objects in the captured scene. If these contain too little detail, the relative rotations of the cameras can not be reliably estimated. This can happen, for example, when the camera is directed at a blue sky, plain ceiling, grass or water. Therefore, we recommend to do this calibration phase on a set of frames that was recorded in a scene with enough high-detail objects. If finding such a place is infeasible, it might make sense to design a non-uniform, detailed pattern that is printed on the inside of a construction which can be placed around the camera setup.

Radial distortion is another factor that requires a good calibration. The distortion parameters can be retrieved by filming a specially designed pattern board. These parameters need to be retrieved once for every lens. When the distortion parameters are known, we can correct the distortions beforehand and treat the videos as undistorted videos in the stitching process. If the distortion are relatively low, we might get better results by first performing sRD-SIFT on the distorted frames first and correcting them after.

Bibliography

[1] David G Lowe. "Object recognition from local scale-invariant features". In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[2] Matthew Brown and David G Lowe. "Automatic panoramic image stitching using invariant features". In: *International journal of computer vision* 74.1 (2007), pp. 59–73.

[3] Meinard Müller. "Dynamic time warping". In: *Information retrieval for music and motion* (2007), pp. 69–84.

[4] Yaron Caspi and Michal Irani. "Aligning non-overlapping sequences". In: *International Journal of Computer Vision* 48.1 (2002), pp. 39–51.

[5] Hongdong Li and Richard Hartley. "A non-iterative method for correcting lens distortion from nine point correspondences". In: *OMNIVIS 2005* 2 (2005), p. 7.

[6] Andrew W Fitzgibbon. "Simultaneous linear estimation of multiple view geometry and lens distortion". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–125.

[7] Zhengyou Zhang. "A flexible new technique for camera calibration". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.11 (2000), pp. 1330–1334.

[8] Myung-Ho Ju and Hang-Bong Kang. "Stitching Images with Arbitrary Lens Distortions". In: *International Journal of Advanced Robotic Systems* 11 (2014).

[9] Miguel Lourenço, Joao P Barreto, and Abed Malti. "Feature detection and matching in images with radial distortion". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 1028–1034.

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[11] Ethan Rublee et al. “ORB: an efficient alternative to SIFT or SURF”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2564–2571.

[12] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. “KAZE features”. In: *Computer Vision–ECCV 2012*. Springer, 2012, pp. 214–227.

[13] Marius Muja and David G Lowe. “Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration.” In: ().

[14] Chanop Silpa-Anan and Richard Hartley. “Optimised KD-trees for fast image descriptor matching”. In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[15] Bill Triggs et al. “Bundle adjustmenta modern synthesis”. In: *Vision algorithms: theory and practice*. Springer, 1999, pp. 298–372.