22^{nd} *Computer Vision Winter Workshop*
Nicole M. Artner, Ines Janusch, Walter G. Kropatsch (eds.)
Retz, Austria, February 6–8, 2017

Automatic Stabilization of Image Sequences

Bertram Sabrowsky-Hirsch
University of Applied Sciences Upper Austria,
School of Informatics, Communications and Media,
Department of Digital Media, 4232 Hagenberg, Austria
b.sabrowsky@gmx.at

Abstract. *This paper presents a technique to automatically process image sequences taken with handheld cameras into stabilized animations. This involves the alignment of images of different viewpoints and the detection and stabilization of the scenes static background while preserving the motion of moving objects. For this we present two different approaches that each solve subproblems. The first approach aligns the images by tracking SIFT features through the sequence and calculating linear mappings for each image. The second approach refines the results of the first by using block-motion analysis to divide the images into foreground and background segments, correcting the motion vectors of each segment and rendering a final sequence using the corrected motion vectors. The effectiveness of both approaches is demonstrated on several sample sequences.*

1. Introduction

The problem this paper approaches is the stabilization of image sequences—especially sequences with a low and irregular frame rate which implies a large and varying time offset between subsequent frames. The typical use case would be a sequence of pictures taken from a handheld camera.

Given an image sequence an algorithm should automatically process the sequence and create a stabilized animation. The images can contain noise and significant warps of the camera viewpoint as they would occur in a sequence of photos taken with a conventional camera without using a tripod or other tools to stabilize the device. The task is to create a smooth animation with a minimum of jitter. An assumption is that parts of the scene contain a static background without significant motion and that the sequence contains no large camera motions like a camera pan. However, the scene may contain moving objects and motion and the camera viewpoint may be warped irregularly between subsequent frames as long as the images still intersect significantly. A further assumption is that the camera does not move into the scene. However, the camera zoom may vary between images.

The resulting animation should be stabilized in a way that the motion of the static background of the scene is minimized. This can be measured by comparing the amount of motion before and after stabilization. The amount of motion can be measured based on the optic flow or motion detection techniques like frame differencing.

In this paper we present two different approaches to process image sequences into stabilized animations. Both stabilize the image sequence by reducing the amount of motion. However, they use different techniques and solve different subproblems. The approaches will be referred to as *feature-based stabilization* and *block-motion stabilization*.

The *feature-based stabilization* first detects SIFT features in each image and tracks them through the sequence. The feature tracks are used to estimate linear mappings to align the images. The tracks are in turn filtered using the calculated mappings and again used to calculate better estimates for the mappings. The process is repeated until the quality converges. The final mappings are used to transform the sequence into a stabilized animation.

The *block-motion stabilization* divides the images into a block grid and estimates the motion vectors using the temporal median of the sequence as a reference. The motion vectors are used to classify the blocks into background, dynamic and foreground blocks. Segments of blocks of the same type are

identified and tracked through the sequence. The motion vectors of the blocks of each segment are corrected based on their type and a stabilized animation is rendered using the corrected motion vectors.

While both approaches can be used independently, it is recommended to employ them sub-sequentially with the *feature-based stabilization* first and the *block-motion stabilization* second since the effectiveness of the latter depends on the temporal median which is in turn relying on a certain degree of stability in the background and also because the run-time performance of the approach suffers immensely if its configured to tolerate large motions.

1.1. Related Work

The main challenge in the stabilization task is finding out how the images are related to each other and to draw conclusions about their viewpoints in the global scene. The problem is known as image registration/alignment in terms of panorama stitching. Brown [2] gives an overview of different approaches for image registration. Approaches divide into intensity- and feature-based approaches. Feature-based approaches are recommended by Zitová and Flusser [13] for images with distinctive and easily detectable objects. This is usually the case in photographs. Zhang describes in [11] how the projective transformation parameters can be calculated from a number of point correspondences. However, not all feature matches are correct and features detected in dynamic parts of the scene have to be excluded. A RanSaC approach is described by Dung et al. [3] to find the best projective mapping given a set of feature matches.

For the block-motion stabilization the basic problem remains unchanged, but we assume that the background of the image sequence is already stabilized up to a certain degree. The goal is to further increase the visual stability in the scene. Beauchemin et al. [1] give an overview of different approaches to calculate the optical flow. The approaches split in block-based, differential (e.g., the Horn–Schunck method [6]), phase correlation and discrete optimization methods (Glocker et al. [5]). Given its promising performance the block-based method was chosen for determining the optical flow. The approach then analyses and corrects the optical flow using ideas from Huang et al. [7] and Vella et al. [10].

2. Feature-based Stabilization

This section is about an algorithm using the *feature-based stabilization* approach. The idea is to track SIFT features through the animation to determine the relative transform between the images. Since tracking is not reliable and the scene can contain dynamic elements (moving objects, motion, noise) the algorithm tries to filter those features that were tracked correctly and probably do not belong to dynamic parts of the scene. Using the filtered features the algorithm calculates a linear mapping for each image that describes how the image has to be transformed to fit in the global scene. As a final step the sequence is cropped to an intersecting area between all mapped images to avoid empty regions in the animation. The result is a stabilized animation.

In the first step we calculate a set of SIFT features for each image. For each Image I_i in the input image sequence $\boldsymbol{\mathcal{I}}_{\text{in}}$ we calculate the associated feature set S_i and append it to the vector $\mathcal{S}$.

In the next step the features $s_{j,i}$ are matched subsequently through all S_i to form a feature track t_j. Therefore, for each $s_{j,i} \in S_i$, the algorithm finds a $s_{j,i+1} \in S_{i+1}$. In case the algorithm fails to match a $s_{j,i}$, $s_{j,i+1}$ gets assigned nil and $s_{j,i}$ is used to find a match in S_{i+2} and so forth until a new match is found. This way the algorithm can compensate for temporary obscured features. However, the algorithm requires a feature to be present in S_0 to form a feature track. A *track* t_j can be represented as a vector of features with one entry for each S_i, i.e.,

$$t_j = (s_{j,0}, \ldots, s_{j,N-1} \mid s_{j,k} \in S_k). \qquad (1)$$

All tracks t_j are stored in a set of feature tracks $\mathcal{T}$.

What follows is the estimation of the mapping between the images. All images $\boldsymbol{I_i} \in \boldsymbol{\mathcal{I}}_{\text{in}}$ are mapped to the first image I_0 using the information in $\mathcal{T}$. The core of the algorithm is an iterative loop that calculates the mappings and keeps track of the quality of the current batch. If the quality decreases the loop ends.

For the calculation of the quality measure Φ of a vector of mappings $\mathcal{M}$ we define an error $e(t_j)$ for a track t_j that is the sum of the squared distances between the mapped ($s_{j,0}$ mapped by $M_i \in \mathcal{M}$) and observed ($s_{j,i} \in t_j$) features that belong to the track. Further we declare the reliability $r(t_j)$ of a track that is defined as the ratio between the size of the track (all features belonging to the track, not counting nil entries) and the product of the error $e(t_j)$ of the track

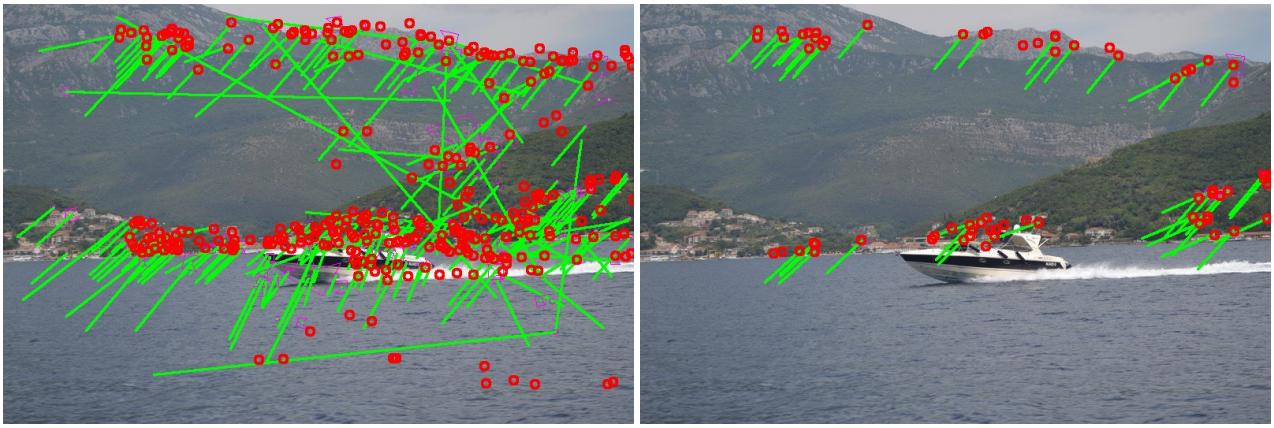


Figure 1: Feature tracks before (left) and after (right) filtering: The current feature locations are marked in red, the previous locations are indicated by green lines.

and the size N of the image vector. The quality of a vector of mappings Φ is calculated as the sum of the reliabilities $r(t_j)$ of all $t_j \in \mathcal{T}$.

The tracks used for the calculation of the mappings are filtered in every iteration. The idea is to only take reliable tracks into account for the generation of new mappings (e.g., Figure 1). First we define a vector $\mathcal{T}_{sort}$ that contains all tracks $t_j \in \mathcal{T}$ sorted by their reliability $r(t_j)$ in descending order. The filtered Tracks $\mathcal{T}'$ are then selected using a threshold τ_{Filter} where $0 < \tau_{\text{Filter}} < 1$. The threshold determines how many percent of the original tracks are filtered. Since the vector is sorted the algorithm just copies the n first elements.

The mappings $\mathcal{M}$ are calculated using a RanSaC approach. For a mapping M_i the algorithm extracts a set of pairs of features $a = \{\langle s_{j,0}, s_{j,i}\rangle, \dots\}$ where each pair consists of a $s_{j,0} \in S_0$ and a $s_{j,i} \in S_i$ of the same track t_j. The RanSaC algorithm loops for K times and searches for the mapping with the best quality ω. The set s_a is calculated by randomly selecting c pairs from the set a. Depending on the configuration affine or projective mappings are calculated from s_a. The quality ω of the mapping is the ratio between the size of a and the summed absolute distances between mapped ($s_{j,0}$ mapped by M_i) and observed ($s_{j,i} \in p[1]$) feature coordinates for all pairs $p \in a$.

The final step is to warp the images using the estimated mappings. We assume that a library function handles the actual mapping and creates a mapped image I'_i for every I_i. The resulting image vector $\mathcal{I}_{\text{out}}$ is the stabilized animation. However, since the mapped images have different boundaries it is important to crop the images to the biggest intersecting rectangle to avoid empty spaces. The biggest intersecting rectangle can be found by mapping the corner coordinates of the image bounds. This assumes that all images have the same height h and width w.

3. Block-motion Stabilization

This section is about an algorithm developed using the *block-motion stabilization* approach. The idea is to calculate the motion of individual blocks of every image and analyze it to split the scene into background, dynamic and foreground segments. The segments are processed differently depending on the type and the motion is corrected to reduce jitters and distortions.

The algorithm starts by processing the input image sequence $\mathcal{I}_{\text{in}}$ to generate a smoothed image sequence $\mathcal{I}_{\text{sm}}$ and a reference image I_{ref}. This information is necessary for the motion estimation. $\mathcal{I}_{\text{sm}}$ is generated by calculating the average for every pixel in its 3×3 neighbourhood. This way high frequency noise is attenuated which proved beneficial for the motion estimation step.

The reference image I_{ref} is generated by calculating a temporal median over the entire image sequence. The temporal median is an easy way to stabilize the background and eliminate the foreground in a scene with motion. In general the temporal median works well as long as the moving objects occlude the background only for a fraction of the time. Foreground objects that remain still for a long time may get part of the reference image. However artifacts in the reference image not necessarily affect the final result as it is used for motion estimation only, specif-

ically to determine stable background segments. I_{ref} is used to calculate the motion vectors for the blocks. After calculating the temporal median the image I_{ref} is also smoothed with a 3×3 average filter.

In the next step block-motion estimation is used on the preprocessed image sequence. First each image of $\mathcal{I}_{\text{sm}}$ is divided in a grid of blocks B_i with P columns and Q rows where each block $b_{i,p,q}$ is associated with $n \times n$ pixels at pixel position $(u, v) = (n \cdot p, n \cdot q)$ in the source image $I_{\text{sm, i}}$.

The motion vector $\mathbf{x}_{i,p,q}$ for each block $b_{i,p,q}$ is then estimated by finding the maximum normalized cross-correlation value v_1 in the $w \times w$ neighbourhood of position (u, v) in the reference image I_{ref}. Also the second highest value v_2 is determined to check whether the maximum value is a unique local maximum. Based on this information a first classification is applied to the blocks.

What follows is the determination of anchor points based on the correlation between motion vectors of neighbouring blocks. The correlation is calculated pairwise between the source block $b_{p,q}$ and its adjacent neighbours $\mathcal{N}_{p,q}$. The correlation value is calculated as the normalized cross-correlation of the motion vectors of two blocks over all images of the sequence. The final neighbourhood correlation value $c_{p,q}$ is the mean value of the correlation values calculated for all neighbouring blocks. A high value for $c_{p,q}$ indicates a stable background block in respect of the entire sequence. We further compare the motion vector $\mathbf{x}_{i,p,q}$ with the mean motion vector $\bar{\mathbf{x}}_{i,p,q}$ of the neighbouring blocks. Based on this information we determine anchor blocks in specific images.

Having detected stable anchor blocks for every image a first detection run processes the sequence in order to identify background blocks. Starting with the anchor points, a region growing approach recursively processes all neighbours $b_{i,p',q'} \in \mathcal{N}_{i,p,q}$ of the current block $b_{i,p,q}$ that have a similar motion vector and classifies them as background blocks. In a follow-up step the foreground segments are extended in order to avoid visual artifacts caused by motion at the block borders.

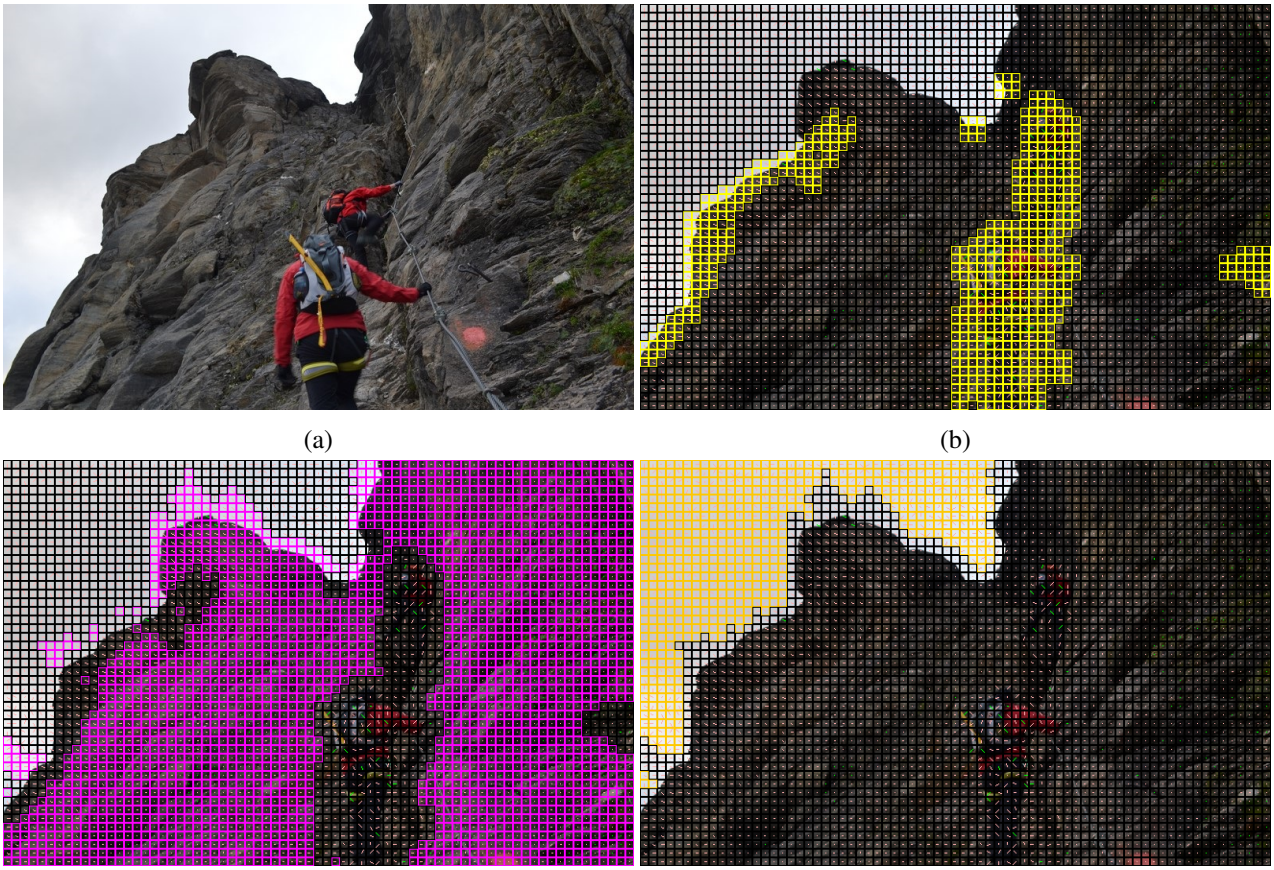
The blocks that remain at this point might be gaps in the background segments with incorrect motion vectors or belong to dynamic segments. Dynamic segments contain stochastic motion or lack the structure necessary for reliably determining the motion vectors. For example the sky or the surface of the sea would be classified as dynamic segments. Every remaining block is classified based on the classification of its neighbours.

Each image are then divided into segments of blocks of the same type (e.g., Figure 2). The result is a set of spatially separate segments for every block type and for every image. We define three sets $\mathcal{S}_{\text{Fg}}$ (foreground), $\mathcal{S}_{\text{Dy}}$ (dynamic) and $\mathcal{S}_{\text{Bg}}$ (background) where each element $S_{\text{i, j}}$ is a segment of spatially connected blocks in a specific image I_i of the sequence.

With the images of the sequence analyzed and split into three sets of segments, tracks of segments through subsequent images are calculated. This is done by intersecting the segment of subsequent images in each set and calculating a directed graph that represents the connections between the segments. This graph is filtered to eliminate all foreground and dynamic segments belonging to tracks that span over less than t_{min} tracks. The filtered segments are converted to background segments. This is done to remove short motion sequences that are most likely incorrectly classified and would disturb the visual flow. In the implementation the default value for t_{min} is 4.

The segments are then analyzed to correct the motion vectors. This is most important for dynamic and foreground segments where the motion vectors are very likely to be incorrect, but also for background segments to smoothen the motion and filter outliers. While the background segments are corrected by simply setting the motion vector of each block to the mean motion vector in its block neighbourhood the foreground and dynamic segments are corrected by spreading the motion vectors of the surrounding background blocks into these segments. Both is done at once by an iterative algorithm that corrects all remaining blocks that have one or more neighbouring blocks that are classified as background blocks or have been corrected in a previous iteration (using the mean motion vector of mentioned blocks). This way the surrounding motion vectors smoothly spread into foreground segments.

Having classified the blocks and corrected their motion vectors in every image of the sequence the output sequence $\mathcal{I}_{\text{out}}$ is rendered using this information. First a stable background image I_{bg} is extracted from the sequence. This is done by finding the first anchor block in the sequence for every block $b_{p,q}$ or when inapplicable the first background block and copy its pixel content to I_{bg}. The corrected motion vector $\mathbf{x}_{i,p,q}$ is used to translate translate the blocks source pixel coordinates. Blocks that never



(a) (b) (c) (d)

Figure 2: The original sequence (a), foreground (b), background (c) and dynamic (d) segments.

are part of a background segment are not rendered and appear black in the background image. With the background image I_{bg} extracted it is copied to all $I_{\text{out},i}$. What is left is to render the foreground and dynamic segments to the output sequence. At some point there was the decision whether to keep the motion of dynamic segments or to eliminate it. A future implementation could feature the option to eliminate the dynamic segments and only show the stable background for those segments. This would require an adaption of the background extraction part. However, for the project the motion of dynamic segments is preserved and thus the dynamic segments are treated like foreground segments. For every output image $I_{\text{out},i}$ the dynamic segments and foreground segments are rendered using their corrected motion vectors. However, visible artifacts are caused by different lighting conditions throughout the source image sequence. To remove these artifacts photometric stabilization and blending techniques could be applied. The effect of the stabilization algorithm becomes obvious when comparing the results of a frame differencing operation (which compares the difference in pixel intensity over subsequent frames) on the sequence before and after the stabilization.

4. Evaluation

Both approaches have been implemented as *ImageJ* plugins and tested on a test set of ten sample sequences. The two plugins have been executed subsequently in two steps with the *feature-based stabilization* first and the *block-motion stabilization* second. Therefore, the following figures will refer to three data sets: *Original*, *Stabilized* and *Final*. *Original* stands for the original image sequences before any stabilization has been applied. *Stabilized* refers to the stabilized image sequences generated by the first (feature-based) stabilization plugin. *Final* are the image sequences generated by the second (block-motion) stabilization plugin using the sequences from *Stabilized* as input. Three metrics (see

4.1) are used to measure the stability of a sequence. The metrics are calculated before and after each stabilization step. By relating the metrics to the initial value the improvement in stability (i.e. decrease of motion) is evaluated as a percentage number. This way the effectiveness of both algorithms is measured.

4.1. Metrics

The following three metrics have been used to measure the motion in a sequence:

Difference metric: The average intensity difference calculated as the sum of the absolute intensity differences of every pixel pair at the same position (u, v) and of two subsequent images I_i and I_{i-1} divided by the number of pixel pairs $(N-1) \cdot w \cdot h$.

Threshold metric: The percentage of pixels affected by motion. Like the difference metric, but the difference value of each pixel pair is thresholded by a value τ_{m} and every pair exceeding the threshold value is counted. The metric is defined as the ratio between the count of masked pixel-pairs and the total count of pairs.

Motion vector metric: The sum of the magnitudes of all motion vectors of every image in the sequence. Evaluated only on the sequences of the *Stabilized* and *Final* data sets.

4.2. Results

Each of the three metrics has been evaluated for all sample image sequences in the data sets *Original*, *Stabilized* and *Final*. The charts in Figure 3, 4 and 5 visualize the improvement of each sample sequence over the processing steps. Note that the values in the charts are not absolute but relative to the result of the leftmost data set (which is in turn 100%). Since all metrics measure the amount of motion a decrease of the value marks improved stability.

Figure 3 shows the chart for the difference metric. The plugins always achieve an improvement with shaky sample sequences faring better in the first stabilisation step compared to still sample sequences and sequences with small moving objects yielding better results than sequences with widespread and complex motion in the second stabilization step.

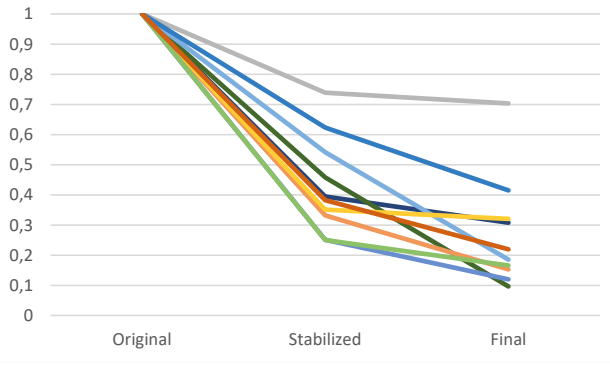


Figure 3: Results for the difference metric. The average for the stabilized sequences is 43.2% (σ of 16%). The average for the final sequences is 26.8% (σ of 18.2%).

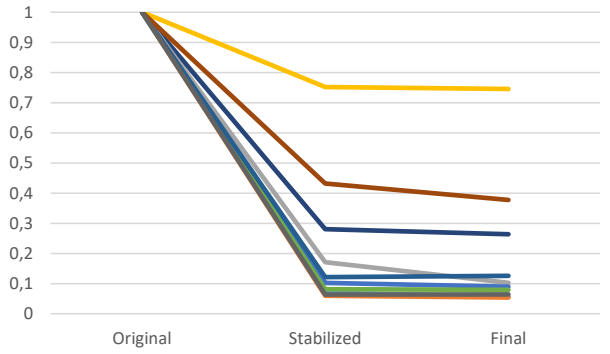


Figure 4: Results for the threshold metric. The average for the stabilized sequences is 21.3% (σ of 22.3%). The average for the final sequences is 19.6% (σ of 21.9%).

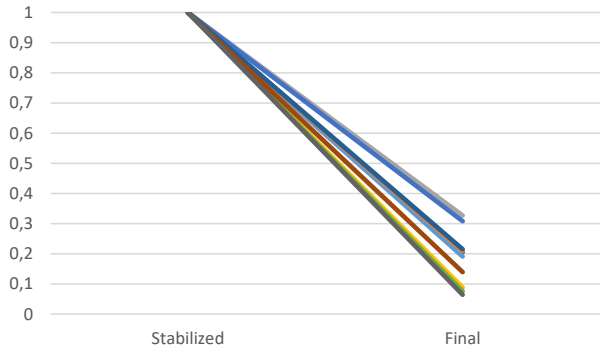
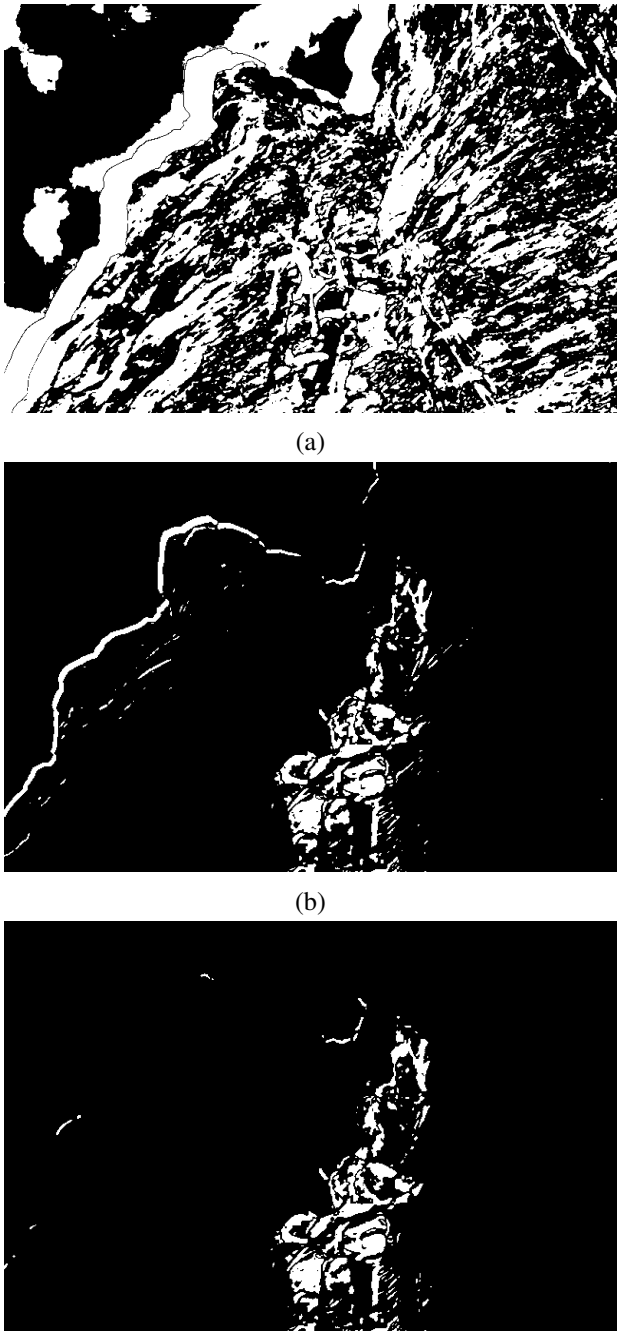


Figure 5: Results for the Motion vector metric. The average for the final sequences is 18.2% (σ of 9.1%).

Figure 4 shows the chart for the threshold metric. The plugins achieve improvements with one exception at the second stabilization step. While the first stabilization step always achieves significant im-



(a) (b) (c)

Figure 6: Visualizations of the threshold metric for the original (a), stabilized (b) and final (c) sequence.

provements with better results for sequences with small areas of motion the second stabilization step seems to have no consistent effect in terms of this metric.

Figure 5 shows the chart for the motion vector metric with all sequences yielding significant improvements. Based on these results it is reasonable to state that the algorithms are successful in stabilizing image sequences with the feature-based stabilization achieving reliable and significant improvements in stability and the block-motion stabilization adding additional refinement.

5. Conclusions

The aim of the project was to design and implement algorithms to stabilize image sequences. While the first one is based on SIFT features the second utilizes block-motion analysis to stabilize image sequences.

The stabilization results of the first algorithm are comparable to the solution Google offers for its *Google Photo* auto-animation stabilization. The implementation even succeeds for sequences that *Google Photo* fails to create an animation for (state of October 2016). However the method is limited by the projective mapping used to align the images. This is most obvious in scenes with high depth variation i.e. with objects close to the camera. Simple projective mapping proves insufficient to accurately align images of a three dimensional scene. The SIFT features would allow for sub-pixel accuracy, but the projective mapping calculated from the feature tracks ignores the depth of the scene. This is why it can never fully align images of scenes with a high amount of depth disparity.

While the second algorithm is not as tolerant as the first one in terms of the input data it is capable of dramatically reducing the amount of motion and noise. The algorithm assumes that the background of the input image sequence is already stabilized up to a certain degree. The implementation allows to adjust parameters to deal with different scenarios. However there is still much potential for improvements. In terms of quality the algorithm lacks a strategy to deal with artifacts caused by different lighting through the images. Also the accuracy of the motion detection using normalized cross-correlation only is not optimal, especially at the blocks borders. Alternative means to create a reference image other than the temporal median could yield better results (e.g., [4]). Also blending techniques could reduce the artifacts at segment borders (e.g., [8]). Another important point is the background extraction part where a more sophisticated approach could yield more reliable background images. A possible improvement would be to take the correlation between background blocks into account to select the best block.

In terms of performance the implementation

would benefit from faster techniques for the block-motion estimation part (e.g., [12]). The implementations of both algorithms still have a lot of potential for optimization. However, as a proof-of-concept the implementations succeeded and are able to generate quite impressive results (e.g., Figure 6). The evaluation confirms this statement as all three metrics show significant improvements in terms of stability after applying the algorithms.

The conclusion of this paper is that feature-based and block-based approaches can be very effectively used for stabilization and can be applied to automatically process image sequences. The algorithms presented in this paper could be used for practical applications, however additional work in automatic parametrization would be necessary to achieve optimal results. More information on the project can be found in the corresponding master thesis [9].

References

[1] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Computing Surveys*, 27(3):433–466, 1995. 2

[2] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys (CSUR)*, 24(4):325–376, 1992. 2

[3] L.-R. Dung, C.-M. Huang, and Y.-Y. Wu. Implementation of RANSAC algorithm for feature-based image registration. *Journal of Computer and Communications*, 1(06):46–50, 2013. 2

[4] A. Elgammal, D. Harwood, and L. Davis. Non-parametric model for background subtraction. In *Proceedings of the European Conference on Computer Vision*, pages 751–767. Springer, 2000. 7

[5] B. Glocker, N. Komodakis, G. Tziritas, N. Navab, and N. Paragios. Dense image registration through mrfs and efficient linear programming. *Medical Image Analysis*, 12(6):731–741, 2008. 2

[6] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. 2

[7] A.-M. Huang and T. Nguyen. Correlation-based motion vector processing with adaptive interpolation scheme for motion-compensated frame interpolation. *IEEE Transactions on Image Processing*, 18(4):740–752, 2009. 2

[8] A. Levin, A. Zomet, S. Peleg, and Y. Weiss. Seamless image stitching in the gradient domain. In *Proceedings of the European Conference on Computer Vision*, pages 377–389. Springer, 2004. 7

[9] B. Sabrowsky-Hirsch. Automatic stabilization of image sequences. Master's thesis, University of Applied Sciences Upper Austria, Dept. of Digital Media, Hagenberg, 2016. 8

[10] F. Vella, A. Castorina, M. Mancuso, and G. Messina. Digital image stabilization by adaptive block motion vectors filtering. *IEEE Transactions on Consumer Electronics*, 48(3):796–801, 2002. 2

[11] Z. Zhang. Estimating projective transformation matrix (collineation, homography). *Microsoft Research, Redmond, WA, Technical Report MSR-TR-2010-63*, 2010. 2

[12] S. Zhu and K.-K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2):287–290, 2000. 8

[13] B. Zitová and J. Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977 – 1000, 2003. 2