J. Vis. Commun. Image R. 96 (2023) 103922

Contents lists available at ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Full length article

Efficient online real-time video stabilization with a novel least squares formulation and parallel AC-RANSAC☆

Jianwei Ke *, Alex J Watras, Jae-Jun Kim, Hewei Liu, Hongrui Jiang, Yu Hen Hu

Department of Electrical and Computer Engineering at the UW-Madison, 1415 Engineering Drive, Madison, WI 53706, United States

ARTICLE INFO

Keywords:
Online real-time video stabilization
Least squares smoothing
Parallel AC-RANSAC

ABSTRACT

A novel online real-time video stabilization algorithm (LSstab) that suppresses unwanted motion jitters based on cinematography principles is presented. LSstab features a parallel realization of the *a-contrario* RANSAC (AC-RANSAC) algorithm to estimate the inter-frame camera motion parameters. A novel least squares based smoothing cost function is then proposed to mitigate undesirable camera jitters according to cinematography principles. A recursive least square solver is derived to minimize the smoothing cost function with a linear computation complexity. LSstab is evaluated using a suite of publicly available videos against state-of-the-art video stabilization methods. Results show that LSstab achieves comparable or better performance, which attains real-time processing speed when a GPU is used.

1. Introduction

With the rapid growth of digital camera technologies, the amount of video footage created by daily users has increased tremendously. A majority of these videos are created by amateurs using hand-held cell phone cameras. Hence, many video clips suffer motion jitters due to unwanted handshaking. The visual quality of these videos can be significantly enhanced with video stabilization.

A video stabilization algorithm can be helpful in many applications, such as visual tracking [1,2], video surveillance [3], and wearable cameras [4]. If the video stabilization algorithm is online and in real-time, it is also beneficial to minimally invasive surgery [5–7], unmanned aerial vehicles [8,9], etc.

Video stabilization is to reduce annoying jitter in the captured video due to unwanted or uncontrolled camera shake during video capturing [10–12]. Hardware video stabilizers have been developed to mitigate the physical shaking of the camera [11,13] while shooting the video. Video stabilization algorithms [10,11,13–17] may also be applied to post-process a captured video to produce a stabilized video that exhibits smoother global camera motion. Both the hardware and software solutions can be combined to ensure desired video quality.

Algorithmically the process of video stabilization consists of the following steps: (a) Estimate global camera motion trajectory in a given video clip; (b) Choose a targeted (usually smoothed) camera motion trajectory; (c) Modify individual video frames according to the targeted camera motion trajectory; (d) apply additional post-processing steps to mitigate potential motion blurs due to (original) camera jitters and irregular frame boundaries due to geometrical transformation applied to realize the desired camera motion trajectory.

Depending on the context of the video, the global camera motion may not be easily defined. If the video consists of a rapidly moving foreground object, one may want to track the foreground object to maintain its position at the center of the video frame. The camera trajectory should be estimated from the tracked foreground object in this case. On the other hand, for surveillance purposes, the desired camera motion may be stationary or smooth panning of the camera. In this case, the background may be used to determine the camera's global motion trajectory. Thus, an ideal video stabilization algorithm must allow human input to estimate the global camera motion and appropriately determine the targeted global camera motion trajectory. Currently, almost all video stabilization algorithms directly use static background objects to estimate the desired camera's global motion. [8–11,13–15,17] compute optical flow and feature points and then use RANSAC [18] to find a subset of feature points belonging to the background. Liu et al. [11,17] use Structure-from-Motion (SfM) to reconstruct the 3D camera trajectory and a sparse 3D point cloud, where RANSAC is used to select a subset of background feature points to derive the camera motions.

The targeted global motion trajectory is also affected by the length of the video clips. Video processing may be applied in a batch mode, where the entire video is to be processed at once, or in an online real-time mode, where the stabilization may be applied incrementally for

☆ This paper has been recommended for acceptance by Zicheng Liu.

* Corresponding author.
E-mail address: jke9@wisc.edu (J. Ke).

https://doi.org/10.1016/j.jvcir.2023.103922
Received 29 October 2021; Received in revised form 1 March 2023; Accepted 12 August 2023
Available online 19 August 2023

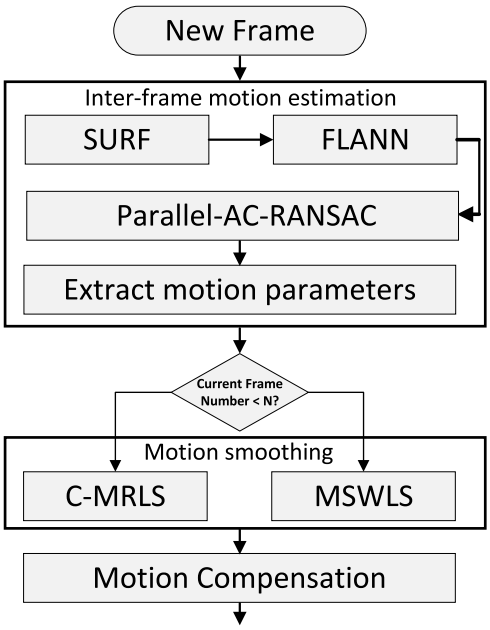

Fig. 1. Overall flowchart of the proposed algorithm, where N is specified by users.

each additional short video clip which may be a single frame. The targeted motion trajectory can be regarded as a smoothed trajectory of the original trajectory. How "smooth" the targeted trajectory is relative to the original motion trajectory is yet another hyper-parameter that may need to be fine-tuned based on the outcome of a chosen view quality metric. Grundmann et al. [13] and Liu et al. [11] state that a desirable stable camera path should follow the cinematography principles. In other words, the desired motion path should be composed of constant, linear, and parabolic segments as if the video is taken with professional stabilization tools.

Motion smoothing is particularly challenging in the online real-time mode if the unseen future motion trajectory cannot be reliably predicted based on prior knowledge about the global camera motion. Successful video stabilization algorithms [10,11,13–17] first compute the entire motion trajectory either in 2D or 3D then smooth it at once. Several real-time approaches [14,19] adapt Kalman filters for smoothing, but their performance is limited.

Given the current and desired global motion trajectories, a 2D or 3D geometrical transformation will be applied to each video frame to obtain the final stabilized frame. During this process, the quality of the output video frame may be impacted, and additional mitigating measures may be applied. These may include the correction of motion blurs due to rapid camera jitters and undefined frame boundaries due to the geometrical transformation of the frame images. Several post-processing techniques such as cropping [11,13,16] mosaicing [20], and inpainting [10] are proposed to mitigate this issue, which can be considered as a supplement to the standard video stabilization procedure.

This paper proposes an online real-time geometry transformation-based algorithm for video stabilization. Unlike other geometry transformation-based algorithms, we adapt and parallelize a new technique called *a-contrario* RANSAC (AC-RANSAC), which does not require any hard thresholds for inlier/outlier discrimination appearing in RANSAC. Hence, it can compute inter-frame global motions more robustly.

The proposed algorithm starts by estimating the inter-frame global motion between two consecutive frames except for the first one. First, features points for the current frame are extracted and matched over the previous frame. We choose SURF [21] and FLANN [22] as our feature extraction and matching algorithm because of their delicate balance of robustness and efficiency [23,24]. Then, the parallel AC-RANSAC is used to estimate the inter-frame geometry transformation, from which the motion parameters (translations, rotations, and scales) are derived. The inter-frame global motion estimation is performed for each incoming frame.

Then the algorithm enters the motion smoothing phase, which requires a user-specified parameter N. If the current frame number is less than N, the camera motion for the current frame will be smoothed by our cinematography principles guided modified recursive least squares algorithm (C-MRLS). After the Nth frame, we stabilize the current camera motion with our modified sliding window least squares algorithm (MSWLS). Finally, the current frame is warped to the previous stable space in the motion compensation stage to create a stabilized video sequence. The whole algorithm pipeline is shown in Fig. 1

The key contribution of the proposed algorithm is a novel least-squares-based smoothing cost for estimating the intentional motion and its associating solver that minimizes the cost in linear time, which is described in Section 4. Section 2 describes the Related Work, and parallel AC-RANSAC is detailed in 3. Experiment validation is described in Section 6. The conclusions and future work are presented in Section 7.

2. Related work

Video stabilization algorithms can generally be divided into three categories: (1) 2D methods [8–11,13–15,17], (2) 3D methods [11,17,25–29], and (3) learning-based methods [30–35]. The 2D and 3D methods are considered conventional but differ in the assumed global camera motion model. A 2D method assumes that the global camera motion between consecutive frames is an affine or homography transformation, whereas 3D methods try to reconstruct the relative 3D camera poses for each video frame.

2.1. 2D methods

2D algorithms start by estimating 2D global camera motion, such as affine transformation or Homography, between consecutive frames. Optical flow [10,14] and geometry transformation [8,9,11,13,15,17] are two conventional methods. Feature points are first extracted from both video frames. Then RANSAC [18] is used to select a subset of matched features to estimate the transformation parameters.

The next step is to derive a smooth motion path. The processed motion path should be sufficiently smooth so as not to cause discomfort during viewing. Several motion-smoothing methods have been proposed in the literature, including low pass filtering [11], Kalman filtering [20,36,37], Gaussian Filtering [10], Spline Smoothing [9], Motion Vector Integration [15], etc. Grundmann et al. [13] and Liu et al. [11] state that a desirable stable camera path should follow the cinematography principles. Grundmann et al. [13] also propose an offline algorithm based on Linear Program optimization with L1-smoothness constraints to find a camera path obeying such principles.

The 2D method achieves a great balance of robustness and efficiency and thus is a great choice for real-time development. Optical flow [10,14] and geometry transformation [8,9,11,13,15,17] are two conventional ways to estimate the 2D inter-frame motion. The latter is becoming more popular because of its efficiency and robustness. Geometry transformation-based methods directly estimate 2D transformation due to camera jitters between adjacent video frames. Feature points are first extracted from both video frames. Then, a subset of matched feature points is selected to estimate the transformation parameters using the RANSAC [18] algorithm, whose performance heavily depends on a user-specified parameter.

2.2. 3D methods

3D methods stabilize videos by reconstructing the camera poses in 3D space. Liu et al. [11] proposed to use Structure-from-Motion(SfM) to compute the relative camera and sparse feature trajectory in 3D space. Then each frame is wrapped to a user-specified path with the "content-preserving" principles to generate a stable virtual output. To avoid reconstructing long camera and feature trajectory, Liu et al. [17] smooth the basis trajectories of the subspace formed by the 3D feature track. Goldstein et al. [25] used epipolar constraints to estimate the fundamental matrices accounting for the stabilized 3D camera motion, reducing the dependency on long feature tracks. Methods [26,27] using gyroscope are also proposed to estimate and 3D rotation. Additional hardware, such as depth sensor [28] and light field cameras [29], are also used to estimate the 3D camera motion and synthesize the stable virtual video.

Generally speaking, 2D methods are more robust and faster than 3D methods, but the 2D motion is insufficient to deal with complex scenes with significant depth variations and severe parallax. On the other hand, 3D methods can handle depth variation and generate great stabilized results in principle. However, the 3D motion model estimation is fragile to various degeneration, such as feature tracking failure, motion blue, etc. Besides, 3D methods often have expensive computational costs or require additional hardware devices. Hence, it is much slower and less robust than 2D methods, which limits its usage in real-time applications.

2.3. Learning-based methods

Recently DNN based video stabilization has attracted more and more attention. According to [12], StabNet [12,30] is the first deep-learning approach for video stabilization, where an encoder and multi-grain transformation regressor are trained under a Siamese network. The authors of StabNet [30] also collect 61 pairs of training videos. Xu et al. [31] train a GAN network to extract the affine transformation for warping unsteady frames.

Instead of predicting transformations between images or in coarse grid level, PWStableNet [32] learns a pixel-wise warping map through a cascade encoder–decoder based on the siamese network. Yu et al. [33] first run a 2D method to stabilize the video. Then the optical flows are computed and fed into an encoder–decoder network to train a pixel-wise warp map. Choi et al. [34] propose an unsupervised deep approach that iteratively interpolates the input video to a stable video without cropping. Another unsupervised method is suggested by Shi et al. [35], where gyroscopes provide the actual camera poses.

It is noted that the performances of these learning-based methods highly depend on the training data [12] and can suffer from large motions [35]. Due to the lack of publicly available data sets, the conventional methods are more robust and perform better in a general setting than the learning-based methods [12]. Learning-based methods are typically computationally demanding and also unsuitable for real-time applications.

2.4. Real-time methods

Although most techniques are offline for post-processing, several real-time approaches [8,9,14,19,37,38] have been studied for video stabilization. Ratakonda et al. [38] uses Integration Projection Matching, which is very computationally efficient for computing the translation of consecutive frames. However, the method is limited as they assume camera motions are always translational. Most of the current real-time methods [8,19,37] are geometry-transformation-based, in which RANSAC is used to estimate the inter-frame transformation, and Modified Kalman filter [14,19,37], Spline smoothing [9], and low-pass filter [8] is employed to smooth the motion parameters on-the-fly.

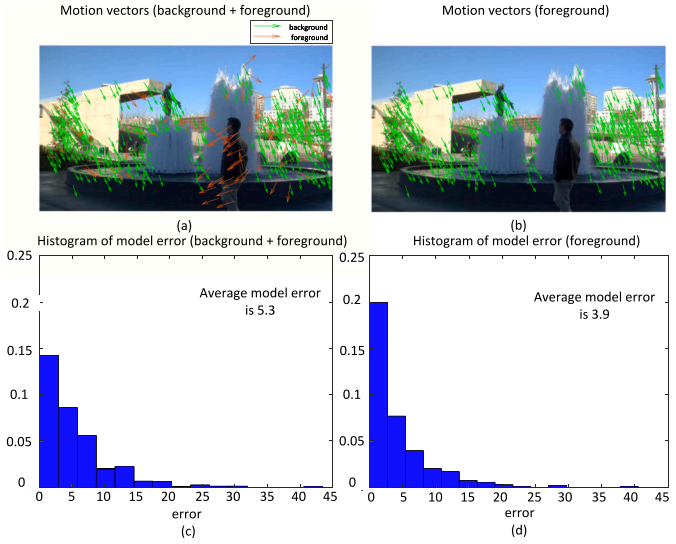

Fig. 2. Match selection. (a) Motion vectors of feature points belonging to foreground moving and background static feature points. (b) Motion vectors of background only. (c) Histogram of model error estimated with background and foreground feature points and (d) with background feature points only.

3. Inter-frame global motion estimation

3.1. Match selection

Matches returned by FLANN are outliers contaminated. There are three main types of outliers: (1) matches with low matching scores, (2) matches belonging to moving foreground objects which would disturb the estimation of the global camera motion, and (3) matches with high matching scores but do not correspond to the same 3D point. We could use a threshold for the first type of outliers to filter out the low-score matches. Although finding a generic threshold that works well for all images is nearly impossible, a popular alternative is the ratio test proposed in [18]. Therefore, we use the ratio test to filter the matches returned by FLANN and pick the 1024 matches with the highest matching scores.

Scenes with moving foreground objects are always challenging to a video stabilization algorithm because the algorithm cannot distinguish whether the displacements of image contents in consecutive frames are caused by foreground object movement or camera motion. For example, a camera can remain stationary while foreground objects actually move. Then, any non-stationary motion estimated by those moving feature matches would be erroneous. On the contrary, the movements of background objects in an image purely result from camera motion and are ideal for camera motion estimation. For a general scene where static background objects are relatively far away from the camera, background objects' 2D motions usually share a similar direction, as shown in Fig. 2(a). Based on this observation, we first compute the 2D motion direction for each match by first subtracting the corresponding feature point coordinates and then normalizing the difference:

$$\mathbf{v} = \frac{\mathbf{I}_{next} - \mathbf{I}_{cur}}{\|\mathbf{I}_{next} - \mathbf{I}_{cur}\|_2} \tag{1}$$

where $\mathbf{I}_{cur}$ is a feature point in the current frame, and $\mathbf{I}_{next}$ is the corresponding feature point in the next frame, $\mathbf{v}$ is the normalized motion vector.

We filter out those matches whose directions are one standard deviation away from the average direction. As shown in Fig. 2(b)(c)(d), this simple pre-processing step effectively eliminates most matches from foreground objects and results in a more accurate global camera model, where the model error is defined in Eq. (32).

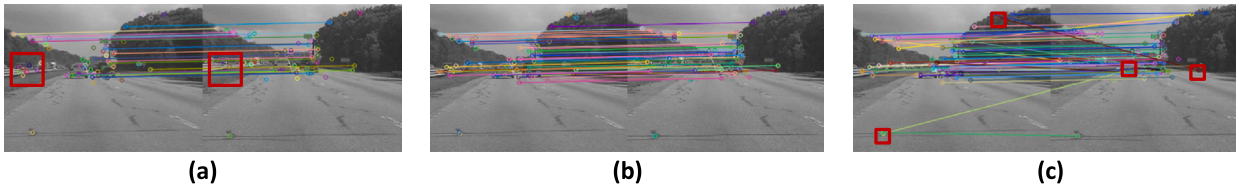

Fig. 3. Robust Affine estimation. (a) If RANSAC error threshold ($\sigma = 0.05$) is small, good matches are filtered out. (b) AC-RANSAC: The error threshold is statistically computed, which provides a well balanced between the number of matches and the correctness. (c) Error threshold $\sigma = 0.3$. All the matches (including false matches) are returned.

In the next section, we will discuss that the parallel implementation of AC-RANSAC can only handle less than 1024 matches due to GPU's inherent hardware limitation, which makes match selection a necessary step. The third type of outliers can be effectively removed by robust estimation techniques such as RANSAC or AC-RANSAC.

3.2. Parallel AC-RANSAC

We choose our camera motion model to be the simplified affine transformation as it provides an excellent trade-off between effectiveness and complexity [15]:

$$A = \begin{bmatrix} s \cdot \cos(\theta) & -s \cdot \sin(\theta) & t_x \\ s \cdot \sin(\theta) & s \cdot \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where s is the scale, θ is the rotation, and t_x and t_y are translations in x, y axis.

Feature matching plays a vital role in estimating A, but matches returned by matching algorithms are often outlier-contaminated. Therefore robust estimation techniques such as RANSAC [18] are required.

RANSAC randomly draws N_{sample} samples to estimate a temporary model A_{temp}. Then it counts the number of inliers according to a threshold σ(specified by the user) on the residual error. After N_{iter} iterations, the model with the largest number of inliers is returned. Finally, all the inliers are used to generate a final robust model. In our case, we need $N_{sample} = 2$ samples to estimate A_{temp}.

The correct choice of σ is critical but depends profoundly on the underlying data and model. If σ is too small, numerous true inliers are classified as outliers and eliminated. If it is too large, the algorithm treats many outliers as inliers, and an inaccurate model will be generated, as shown in Fig. 3.

AC-RANSAC [39,40] avoids the problem of manually setting σ arising in RANSAC by leveraging an *a contrario* criterion, which has been successfully applied to estimating Homography [41], Fundamental matrix [39], and Structure-from-Motion [42]. Here, we extend it to estimating affine model and apply it to affine model estimation and video stabilization.

The essence of AC-RANSAC is that an observed geometric event is significant if the expectation of its occurrences is minimal in a random image [40]. In our case, a subset of feature matches is significant to the model A_{temp} if the occurrence of this subset's matches compatible with A_{temp} is minimal, assuming all features and matches are independent and uniformly distributed. Hence, we can use the most meaningful subset of matches that complies with A_{temp} as the proxy of the largest consensus support subset of A_{temp}.

The meaningfulness of a set of feature matches is quantified by the expected number of false alarms (NFA), defined as [39–41]

$$NFA(k) = N_{outcome}(n - N_{sample})\binom{n}{k}\binom{k}{N_{sample}}\alpha^{k-N_{sample}} \tag{3}$$

- $N_{outcome}$ is the number of possible models (In the case of affine model, $N_{outcome} = 1$).
- n is the total number of feature matches.
- k is the number of inliers.
- α is the probability of feature matches being an inlier assuming they follow a uniform distribution

Indeed, if there are k inliers, a total of $\binom{k}{N_{sample}}$ inliers would generate A_{temp}. Under the null hypothesis that all data are independent and uniformly distributed, there are $\binom{n}{k}$ number of k tuples out of the total n feature matches. The number k of inliers is usually unknown, so all values of k (from $N_{sample} + 1$ to n) are tested, which gives rise to the factor $(n - N_{sample})$. The probability of all k matches are inliers is $\alpha^{k-N_{sample}}$ because the N_{sample} ones used to yield A_{tamp} have zero errors by default. For a rigorous proof of (2), we refer the reader to [39,40].

Similar to [41], let α_0 be the ratio of the area a disk with radius 1 and the area of the image. Then α can be defined as

$$\alpha = \epsilon_k^2 \alpha_0 = \frac{\epsilon_k^2 \pi}{\text{image size}} \tag{4}$$

where ϵ_k is the kth least error among all feature matches.

Therefore, the formula of the NFA for our simplified affine transformation is

$$NFA(k) = (n-2)\binom{n}{k}\binom{k}{2}\left(\frac{\epsilon_k^2 \pi}{\text{image size}}\right)^{k-2} \tag{5}$$

To find the largest consensus support subset of A_{temp} generated by $N_{sample} = 2$ random matches, we can first sort all the matches ascendingly by the residual error, then compute the NFA(k) according to (4) for k from 3 to n (since the selected 2 random matches to generate A_{temp} have 0 error). Finally, the first k matches where k minimizes $NFA(k)$ are chosen to be the consensus support set of A_{temp}. The above procedure is summarized in Algorithm 1.

Algorithm 1 Parallel AC-RANSAC for affine model

```
procedure PARALLEL AC-RANSAC(M, N_iter)
    ▷ M is the list of matches
    ▷ N_iter is the number of iterations
    L_ran = Generate a list of 2N_iter random numbers
    minNFA = ∞
    nBlocks = dim3(N_iter, 1, 1)
    nThreads = dim3(32, 32, 1)
    run AC-RANSAC-KERNEL with nBlocks and nThreads
    model = element in models with smallest NFA
end procedure
procedure AC-RANSAC-KERNEL(models)
    Thread0 estimate affine model A_temp
    Each thread compute model error for each match
    Block-wise Radixsort for model errors
    Each thread computes NFA for each match
    Use reduction to find the model with min NFA in the block
end procedure
```

As opposed to RANSAC, without any user-specified thresholds, AC-RANSAC adaptively chooses the most meaningful set in the sense of generating the model as the largest consensus support set. However, AC-RANSAC requires sorting all data samples at each iteration. With a large number of iterations, AC-RANSAC becomes impractical. Due to

this high demand for computational power and opaqueness in interpreting the *a-contrario* principle, AC-RANSAC has yet to become pervasive in many Computer Vision tasks. To our best knowledge, we are the first to apply AC-RANSAC to video stabilization.

Taking advantage of the parallel nature of AC-RANSAC, we propose a parallel optimization and real-time CUDA implementation of AC-RANSAC. The key idea is to process each AC-RANSAC iteration in parallel and combine the results at the last step. In the CUDA programming model, threads are organized into thread blocks, and grids hold thread blocks. On current GPUs, a thread block can contain no more than 1204 threads. Each thread has its private local memory, and each thread block has shared memory accessible to all threads of the block. The global, constant, and texture memory optimized for different memory usages are visible to all threads [43].

Specifically, each CUDA thread block is responsible for one AC-RANSAC iteration, and each thread in a thread block is used to estimate the error for each feature meach. The main steps are as follows:

1. Generate $2N_{iter}$ of random integers ranging from 1 to M in CPU host, where N_{iter} is the number of iteration, and M is the total number of feature matches.
2. Thread 0 of each thread block retrieves two random numbers and the corresponding feature matches from the constant memory in GPU, and then uses the two matches to estimate a simplified affine model A_{temp}.
3. Each thread retrieves a feature match according to its thread ID and computes the error to the model A_{temp} associated with the block.
4. The M number of errors computed in the previous step for each block is sorted ascendingly by blockwise RadixSort.
5. Each thread in a block computes $NFA(k)$, where k is the same as the thread ID. Reduction [44] is used to find the minimal NFA within a block. Thread 0 stores the minimal NFA for the block and A_{temp}.
6. Finally, the NFAs and their models are sent back to the CPU host, and CPU finds the minimal NFA and its corresponding model A_{temp} which is the final model.

The whole parallel AC-RANSAC is also shown in Algorithm 1, and the specific configuration of CUDA are detailed in the experiment section.

4. Least squares-based motion stabilization

After estimating a robust affine model A, we can extract the motion parameters s, θ, t_x, t_y, from which we need to obtain a stable camera motion without annoying shakes and perturbation. There are multiple ways to approach this problem, such as low pass filter [11], Kalman filter [20,36,37], Gaussian Filter [10], Spline Smoothing [9], Motion Vector Integration [15] and so on. Here, we propose a novel least squares-based formulation for finding smooth camera motions obeying cinematography principles.

For ease of notation, from now on, we denote the motion parameter as m, which can be s, θ, t_x or t_y. We start by considering the problem of finding the stable camera motions as a least squares optimization:

$$\arg\min_{\overline{m}_1,\dots\overline{m}_n} \sum_{i=1}^{n}(m_i-\overline{m}_i)^2 + \lambda \sum_{i=2}^{n}(\overline{m}_i-\overline{m}_{i-1})^2 \tag{6}$$

where

- $m_i(\overline{m}_i)$ is the original(stabilized) camera motion for the ith frame.
- n is the current frame number.

The data term $\sum_{i=1}^{n}(m_i-\overline{m}_i)^2$ ensures that the difference between the original and stabilized motions is small in order to reduce the distortion introduced in warping. The regularization term $\sum_{i=2}^{n}(\overline{m}_i-\overline{m}_{i-1})^2$ guarantees that the estimated camera motions $(\overline{m}_1,\dots,\overline{m}_n)$ are stable. The regularization parameter λ controls the degree of motion smoothness and the tracking ability of the stable camera motions $(\overline{m}_1,\dots,\overline{m}_n)$

This formulation and its variants are used in offline video stabilization [14,16] and yield good results. However, (6) in [14,16] operates in a batch fashion that computes all the stable motions at once offline. Next, we propose three new formulations based on (6) that robustly estimate the current stabilized motion online in real-time.

4.1. A direct approach

For (6) to be applicable to an online and real-time manner, a direct modification is

$$\overline{m}_n = \arg\min_{\overline{m}_n} \sum_{i=1}^{n-1}(m_i-\hat{m}_i)^2 + (m_n-\overline{m}_n)^2 + \lambda \sum_{i=2}^{n}(\hat{m}_i-\hat{m}_{i-1})^2 + \lambda(\overline{m}_n-\hat{m}_{n-1})^2 \tag{7}$$

where we try to find the current estimate $\overline{m}_n$ using all initial estimates $m_1,\dots,m_n$ and all previous stabilized results $\hat{m}_1,\dots,\hat{m}_{n-1}$. We can further simplify (7) to:

$$\overline{m}_n = \arg\min_{\overline{m}_n}(m_n-\overline{m}_n)^2 + \lambda(\overline{m}_n-\hat{m}_{n-1})^2 \tag{8}$$

Since (8) is quadratic, the minimum occurs when the derivative is zero. The solution to (8) is

$$\overline{m}_n = \frac{m_n+\lambda\hat{m}_{n-1}}{1+\lambda} \tag{9}$$

However, (9) is simply the weighted average of the last stabilized motion $\hat{m}_{n-1}$ and the current motion m_n, where the weight is controlled by λ. Hence, (9) has a limited capability of stabilizing the current frame, as shown in Fig. 4.

4.2. Modified Recursive Least Squares Stabilization (MRLS)

The direct approach provides a formulation that runs online and in real-time but with a limited stabilization ability. Although (6) operates in a batch fashion, it is still suitable for real-time stabilization if we can efficiently and robustly estimate the current stabilized motion.

At the nth frame, suppose we compute $(\overline{m}_1,\dots,\overline{m}_n)$ in (6) in real-time, then we can use $\overline{m}_n$ as the stabilized motion for current frame, i.e. $\hat{m}_n = \overline{m}_n$. However, (6) does not take into account the past stabilized motions $(\hat{m}_1,\dots,\hat{m}_{n-1})$ when computing $\overline{m}_n$, hence there is no guarantee that $(\hat{m}_1,\dots,\hat{m}_{n-1}, \hat{m}_n = \overline{m}_n)$ will form a smooth path. To remedy this, we further propose the following cost function:

$$\arg\min_{\overline{m}_1,\dots\overline{m}_n} \sum_{i=1}^{n}(m_i-\overline{m}_i)^2 + \lambda_1 \sum_{i=2}^{n}(\overline{m}_i-\overline{m}_{i-1})^2 + \lambda_2 \sum_{i=1}^{n-1}(\overline{m}_i-\hat{m}_i)^2 \tag{10}$$

Similar to (6), here (10) tries to estimate an augmented path $(\overline{m}_1,\dots,\overline{m}_n)$ that starts at the first frame and ends at the current frame, and $\hat{m}_n$ is set to $\overline{m}_n$ for the next round of estimation.

The data term $\sum_{i=1}^{n}(m_i-\overline{m}_i)^2$ ensures that the distortion between the original path $(m_1,\dots,m_n)$ and the augmented path is small. The augmented path is guaranteed to be smooth by the minimization of the first regularization term $\lambda_1\sum_{i=2}^{n}(\overline{m}_i-\overline{m}_{i-1})^2$. Furthermore, the second regularization term $\lambda_2\sum_{i=1}^{n-1}(\overline{m}_i-\hat{m}_{i-1})^2$ secures that the augmented path is similar to the previous smoothed path $(\hat{m}_1,\dots,\hat{m}_{n-1})$, making sure that $(\hat{m}_1,\dots,\hat{m}_{n-1},\hat{m}_n=\overline{m}_n)$ will form a smooth curve, as shown in Fig. 5.

Similarly, (10) is quadratic, so we can find the optimal augmented path $(\overline{m}_1,\dots,\overline{m}_n)$ to achieve the minimum of (10) by setting all partial derivatives to zero, which is equivalent to solve the system of linear equations (11), given in Box I. The derivation for (11) is detailed in Appendix A. Note that (11) can be written compactly in matrix form $\mathbf{S}_n\overline{\mathbf{m}}_n = \mathbf{Y}_n$.

For small $\mathbf{S}_n$, we could just invert it to find $\hat{m}_n$, i.e. $\hat{m}_n = \overline{m}_n = \left[\mathbf{S}_n^{-1}\mathbf{Y}_n\right]_n$, where $[\cdot]_i$ denotes ith element of a vector. However, the size

$$\underbrace{\begin{bmatrix} (1+\lambda_1+\lambda_2) & -\lambda_1 & 0 & \cdots & \cdots & 0 \\ -\lambda_1 & (1+2\lambda_1+\lambda_2) & -\lambda_1 & \ddots & \ddots & \vdots \\ 0 & -\lambda_1 & (1+2\lambda_1+\lambda_2) & -\lambda_1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -\lambda_1 & (1+2\lambda_1+\lambda_2) & -\lambda_1 \\ 0 & \cdots & \cdots & 0 & -\lambda_1 & (1+\lambda_1) \end{bmatrix}}_{\mathbf{S}_n} \underbrace{\begin{bmatrix} \overline{m}_1 \\ \overline{m}_2 \\ \overline{m}_3 \\ \vdots \\ \overline{m}_{n-1} \\ \overline{m}_n \end{bmatrix}}_{\overline{\mathbf{m}}_n} = \underbrace{\begin{bmatrix} m_1 + \lambda_2 \hat{m}_1 \\ m_2 + \lambda_2 \hat{m}_2 \\ m_3 + \lambda_2 \hat{m}_3 \\ \vdots \\ m_{n-1} + \lambda_2 \hat{m}_{n-1} \\ m_n \end{bmatrix}}_{\mathbf{Y}_n} \tag{11}$$

Box I.

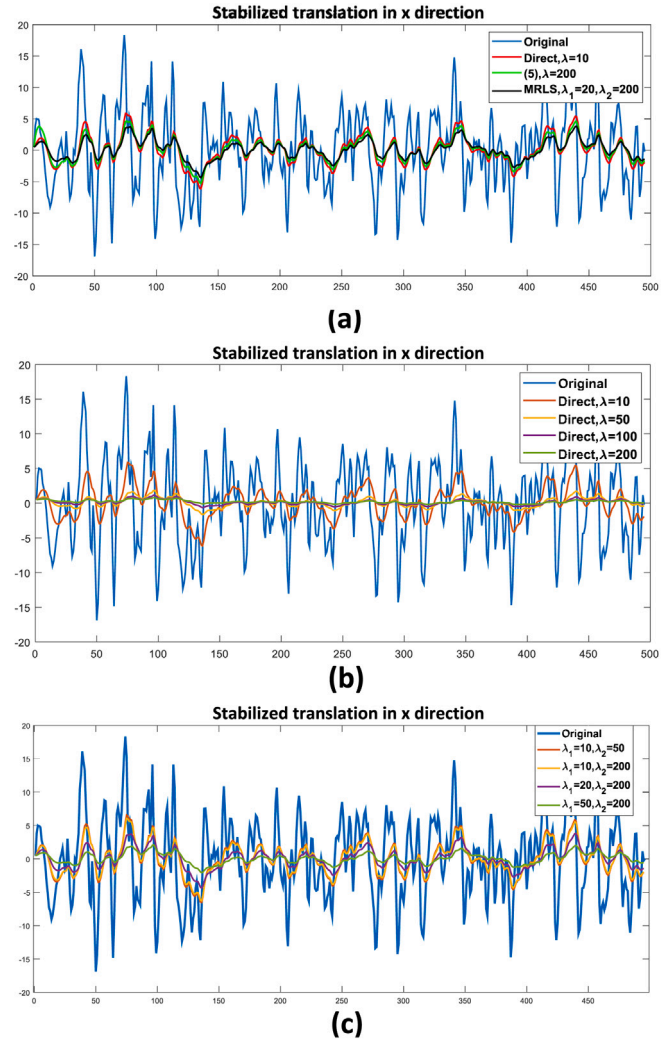

Fig. 4. Stabilization of translation in x direction, i.e. $m = t_x$, via the original approach (6), the direct approach (7) and MRLS (10). (a) Comparison of stabilization effects of the three formulae. (b) The direct approach has limited freedom to stabilize the motions. As λ increases, it can over-stabilize the original motions resulting in a static, instead of stabilized, video. (c) MRLS makes use of all the previous estimates for stabilization and avoids over-stabilizing the motions. The stabilized motions can still reflect the intentional motion of the camera.

of $\mathbf{S}_n$ increases with n, which makes computing $\mathbf{S}^{-1}$ not practical to a real-time application for large n. Since $\hat{m}_n$ is the quantity that we try to estimate, we do not have to compute the complete inverse of $\mathbf{S}_n$. Instead, knowing the last row of $\mathbf{S}_n^{-1}$ is enough for computing $\hat{m}_n$.

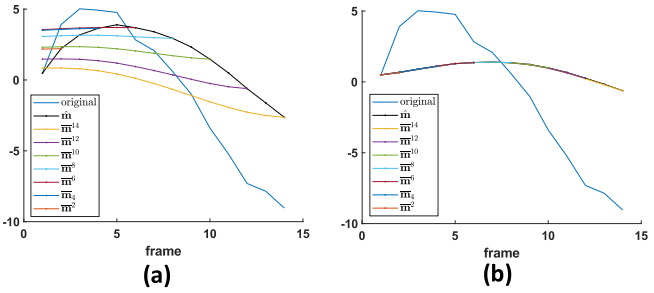

Fig. 5. Stabilization of translation in x direction, i.e. $m = t_x$. $\hat{\mathbf{m}}$ denotes final stabilized estimates. $\overline{\mathbf{m}}^k$ denotes the optimal solution $(\overline{m}_1, \ldots, \overline{m}_k)$ to (6) for (a) or (10) at frame k for (b). (a) At each frame, (6) tries to estimate an augmented path $(\overline{m}_1, \ldots, \overline{m}_k)$ without considering the previous estimate $(\hat{m}_1, \ldots, \hat{m}_{k-1})$. (b) (10) assures that $\hat{m}_i$ and $\overline{m}_i, i = 1, \ldots, n-1$ are similar, so $(\hat{m}_1, \ldots, \hat{m}_{k-1}, \hat{m}_k = \overline{m}_k)$ will be smooth.

Thanks to the unique structure of $\mathbf{S}_n$, we derive a recursive formula for computing the last row of $\mathbf{S}_n^{-1}$ by recursively constructing the row echelon form of $\mathbf{S}_n$ from $\mathbf{S}_{n-1}$ in $O(n)$ operations.

The key observations are (1) that the $(n-2)$th row of $\mathbf{S}_{n-1}$'s echelon form is the same as $\mathbf{S}_n$'s echelon form and (2) that the operations to derive the last row of $\mathbf{S}_n$'s echelon form from the $(n-2)$th row is straightforward.

Theorem 1. *Let $\mathcal{S}_{n-1} = [S_{n-1}|I]$ be an augmented matrix, where I is the identity matrix. $\mathcal{S}^e_{n-1} = [S^e_{n-1}|A_{n-1}]$ is the echelon form of $\mathcal{S}_{n-1}$ with $b = -\lambda_1$ as leading term for each row. Then $\{S_{n-1}\}_{n-2} \in \mathbb{R}^{2(n-1)}$ and $\{S_n\}_{n-2} \in \mathbb{R}^{2n}$ have the form:*

$$\{S_{n-1}\}_{n-2} = \begin{bmatrix} 0 & \cdots & b & x_1^{(n-1)} & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 \end{bmatrix} \tag{12}$$

$$\{S_n\}_{n-2} = \begin{bmatrix} 0 & \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \end{bmatrix} \tag{13}$$

where $\{\cdot\}_i$ denotes the ith row of a matrix, and

$$x_1^{(n-1)} = \frac{b^2}{c - x_1^{(n-2)}} \tag{14}$$

$$y_i^{(n-1)} = \frac{-y_i^{(n-2)} b}{c - x_1^{(n-2)}}, \quad i = 1 \ldots n-3 \tag{15}$$

$$y_{n-2}^{(n-1)} = \frac{b}{c - x_1^{(n-2)}} \tag{16}$$

where

$$a = (1 + \lambda_1 + \lambda_2) \tag{17}$$

$$b = -\lambda_1 \tag{18}$$

$$c = (1 + 2\lambda_1 + \lambda_2) \tag{19}$$

$$d = (\lambda_1 + 1) \tag{20}$$

Proof. See Appendix B □

Once we have $\{S_n\}_{n-2}$, we can obtain the $\{S_n\}_n$ by two more row operations:

$$\frac{b^2}{c - x_1^{(n-1)}} \left(\{S_n\}_{n-1} - \{S_n\}_{n-2}\right) \to \{S_n\}_{n-1} \tag{21}$$

$$\frac{c - x_1^{(n-1)}}{q_n} \left(\{S_n\}_n - \{S_n\}_{n-1}\right) \to \{S_n\}_n \tag{22}$$

where $q_n = cd - x_1^{(n-1)}d - b^2$. Note that the first of half of $\{S_n\}_n$, denoted as $\{S_n\}_n^{1st}$, is a unity row vector with the last element being 1, and its second half, denoted as $\{S_n\}_n^{2nd}$, is $\{\mathbf{S}_n^{-1}\}_n$ (the last row of $\mathbf{S}_n^{-1}$), which can be used to find $\hat{m}_n$:

$$\hat{m}_n = \overline{m}_n = \{\mathbf{S}_n^{-1}\}_n \cdot \mathbf{Y}_n = \{S_n\}_n^{2nd} \cdot \mathbf{Y}_n \tag{23}$$

To estimate $\hat{m}_n$ recursively, we will first need to store $\{S_{n-1}\}_{n-2}$ and $\mathbf{Y}_n$ which takes $O(n)$ storage, then use (21), (22) to compute $\hat{m}_n$ which takes $O(n)$ operations, and finally store $\{S_n\}_{n-1}$ for estimating $\hat{m}_{n+1}$ which also takes $O(n)$ storage. Therefore, solving (11) has $O(n)$ time and space complexity in total.

Note that (11) differs from a typical recursive least squares [45] in that $\mathbf{S}_n$, $\overline{\mathbf{m}}_n$ and $\mathbf{Y}_n$ change at each time step. Hence we refer (11) as the modified recursive least square(MRLS) in our algorithm pipeline.

4.3. Cinematography Principles guided Modified Recursive Least Squares Stabilization (C-MRLS)

MRLS assures that the estimated path is smooth and close to the original path by adding the two regularization terms. However, according to cinematography principles [11,13], the desired stabilized path should have constant velocity and constant accelerations, i.e., the second and third derivatives for the path are zero. For our stabilized path to have these cinematographic characteristics, we first define two proxies as the variations of velocity and acceleration.

From finite difference methods [46], the backward difference approximation for the second and the third derivatives are [46]

$$(dv)_i = \overline{m}_i - 2\overline{m}_{i-1} + \overline{m}_{i-2} \tag{24}$$

$$(dc)_i = \overline{m}_i - 3\overline{m}_{i-1} + 3\overline{m}_{i-2} - \overline{m}_{i-3} \tag{25}$$

Note that (24) and (25) have first-order truncation error [46]. We could also choose other more complex forms for approximating dv and dc with smaller truncation errors in principle.

Hence, the desired cost function following the cinematography principle is

$$\begin{aligned}\underset{\overline{m}_1,\dots\overline{m}_n}{\arg\min} &\sum_{i=1}^{n}(m_i - \overline{m}_i)^2 + \lambda_1 \sum_{i=2}^{n}(\overline{m}_i - \overline{m}_{i-1})^2 + \lambda_2 \sum_{i=1}^{n-1}(\overline{m}_i - \hat{m}_i)^2 \\ &+ \lambda_3 \sum_{i=3}^{n}(dv)_i^2 + \lambda_4 \sum_{i=4}^{n}(dc)_i^2\end{aligned} \tag{26}$$

The path that minimizes (26) will be smooth and have near-constant velocity and near-constant acceleration, which obeys the cinematography principles.

Since (26) is also quadratic, we can find the solution by solving the corresponding system of linear equations whose data matrix is (27), given in Box II. The derivation for (27) and its linear solver are the same as (11). Hence we refer (27) as the cinematography principles guided modified recursive least square(C-MRLS). As shown in Fig. 6, C-MRLS avoids rapid changes in the resulting path, and thus the result is smoother than that of MRLS but well still preserves the intentional camera global motion.

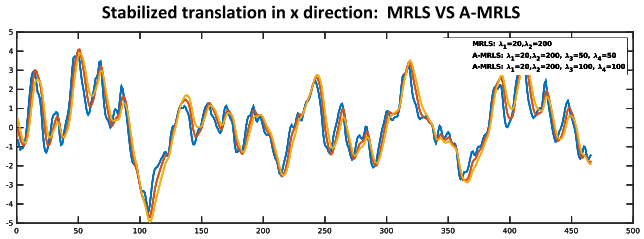

Fig. 6. Comparison of MRLS and C-MRLS in translation in x direction, i.e. $m = t_x$. The result for C-MRLS is smoother than MRLS but well preserves the intentional camera global motion.

4.4. Modified Sliding Window Least Squares Stabilization (MSWLS)

In MRLS and C-MRLS, all previous frames are used to estimate the stable motion, which takes $O(n)$ time and space complexity. As n increases, MRLS and C-MRLS will eventually become impractical. In practice, the frames far before the current frame contain little information for the inference of the current frame's stable motion. Hence, we propose to use only the latest N frames to estimate the motion of the current frame, where N is specified by users:

$$\begin{aligned}\underset{\overline{m}_{n-N+1},\dots\overline{m}_n}{\arg\min} &\sum_{i=n-N+1}^{n}(m_i - \overline{m}_i)^2 + \lambda_1 \sum_{i=n-N+2}^{n}(\overline{m}_i - \overline{m}_{i-1})^2 \\ &+ \lambda_2 \sum_{i=n-N+1}^{n-1}(\overline{m}_i - \hat{m}_i)^2 + \lambda_3 \sum_{i=n-N+1}^{n}(dv)_i^2 + \lambda_4 \sum_{i=n-N+1}^{n}(dc)_i^2\end{aligned} \tag{28}$$

(28) not only emphasizes the most relevant information in the time domain but also can be regarded as a failsafe for MRLS. Similarly, to find the $\overline{\mathbf{m}}$ that minimizes (10), we can set all the partial derivatives to zero, which is equivalent to solving the linear system:

$$\mathbf{S}_N \cdot \underbrace{\begin{bmatrix}\overline{m}_{n-N+1}\\ \overline{m}_2\\ \vdots\\ \overline{m}_{n-1}\\ \overline{m}_n\end{bmatrix}}_{\overline{\mathbf{m}}'_N} = \underbrace{\begin{bmatrix}m_{n-N+1} + \lambda_2\hat{m}_{n-N+1}\\ m_{n-N+2} + \lambda_2\hat{m}_{n-N+2}\\ \vdots\\ m_{n-1} + \lambda_2\hat{m}_{n-1}\\ m_n\end{bmatrix}}_{\mathbf{Y}_N} \tag{29}$$

$$\hat{m}_n = \overline{m}_n = \{\mathbf{S}_N^{-1}\}_N \mathbf{Y}_N \tag{30}$$

where $\{\mathbf{S}_N^{-1}\}_N$ can be precomputed or set to $\{S_n\}_n^{2nd}$ when $n = N$ in MRLS, and N is specified by users.

We see that saving $\mathbf{Y}_N$ requires $O(N)$ storage and computing $\hat{m}_n$ by (30) takes $O(N)$ operation, so MSWLS also has $O(N)$ time and space complexity. For similar reason, we refer (29) as the modified sliding window least square(MSWLS) in our algorithm pipeline. Our least squares motion stabilization is summarized in Algorithm 2.

5. Motion compensation

After the stabilized motions are estimated, we need to warp the current frame to generate the final stabilized frame. Let $F_i, \overline{F}_i$ denote the ith original frame and stabilized frame, A_i denote the simplified affine between F_i and, F_{i+1}, $\overline{A}_i$ denote the stabilized affine transformation between $\overline{F}_i$ and $\overline{F}_{i+1}$. From Fig. 7, we see that F_n relates F_1 with

$$F_n = \left(\prod_{i=1}^{n} A_i\right) F_1 = \left(A_n \cdot \ldots \cdot A_1\right) F_1$$

and $\overline{F}_n$ relates $\overline{F}_1$ with

$$\overline{F}_n = \left(\prod_{i=1}^{n} \overline{A}_i\right) \overline{F}_1 = \left(\overline{A}_n \cdot \ldots \cdot \overline{A}_1\right) \overline{F}_1$$

$$
\underbrace{\begin{bmatrix}
(1+\lambda_1+\lambda_2+\lambda_3+\lambda_4) & -(\lambda_1+\lambda_3+3\lambda_4) & \lambda_3+3\lambda_4 & \lambda_4 & \cdots & \cdots & \cdots & 0 \\
-(\lambda_1+\lambda_3+\lambda_4) & (1+2\lambda_1+\lambda_2+3\lambda_3+4\lambda_4) & -(\lambda_1+3\lambda_3+6\lambda_4) & \lambda_3+4\lambda_4 & -\lambda_4 & \cdots & \cdots & 0 \\
(\lambda_3+\lambda_4) & -(\lambda_1+3\lambda_3+4\lambda_4) & (1+2\lambda_1+\lambda_2+4\lambda_3+7\lambda_4) & -(\lambda_1+3\lambda_3+7\lambda_4) & (\lambda_3+4\lambda_4) & -\lambda_4 & \cdots & 0 \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\
0 & \cdots & (\lambda_3+\lambda_4) & -(\lambda_1+3\lambda_3+4\lambda_4) & (1+2\lambda_1+\lambda_2+4\lambda_3+7\lambda_4) & -(\lambda_1+3\lambda_3+7\lambda_4) & (\lambda_3+4\lambda_4) & -\lambda_4 \\
0 & \cdots & \cdots & -\lambda_4 & \lambda_3+4\lambda_4 & -(\lambda_1+3\lambda_3+6\lambda_4) & (1+2\lambda_1+\lambda_2+3\lambda_3+4\lambda_4) & -(\lambda_1+\lambda_3+\lambda_4) \\
0 & \cdots & \cdots & \cdots & \lambda_4 & \lambda_3+3\lambda_4 & -(\lambda_1+\lambda_3+3\lambda_4) & (1+\lambda_1+\lambda_2+\lambda_3+\lambda_4)
\end{bmatrix}}_{\mathbf{S}_n}
\times \underbrace{\begin{bmatrix} \overline{m}_1 \\ \overline{m}_2 \\ \overline{m}_3 \\ \vdots \\ \overline{m}_{n-1} \\ \overline{m}_n \end{bmatrix}}_{\overline{\mathbf{m}}_n} = \mathbf{Y}_n \tag{27}
$$

Box II.

Algorithm 2 Least Squares-based Motion Stabilization

procedure C-MRLS($m, n, \mathbf{Y}, S$)
  append m_n to $\mathbf{Y}$
  construct S in (13) from (12) ▷ differs from MSWLS
  estimate $\hat{m}_n$
  set the last element of $(m_n + \lambda_2 \hat{m}_n)$
  return $\hat{m}, \mathbf{Y}, S$
end procedure
procedure MSWLS($m, n, \mathbf{Y}$)
  append m_n to $\mathbf{Y}$
  estimate $\hat{m}_n$ using (30) ▷ differs from C-MRLS
  remove $m_{n-N+1} + \lambda_2 \hat{m}_{n-N+1}$ from $\mathbf{Y}$
  set the last element of $(m_n + \lambda_2 \hat{m}_n)$
  return $\hat{m}, \mathbf{Y}$
end procedure
procedure LS-Stabilize($m, n, \mathbf{Y}, S$)
  if $n < 1$ **then**
    $\hat{m} = m_n$
    append $m_1 + \lambda_2 \hat{m}_1$ to $\mathbf{Y}$
  else if $n < N$ **then**
    $[\hat{m}, \mathbf{Y}, S] =$ C-MRLS$(m, n, \mathbf{Y}, S)$
  else
    $[\hat{m}, \mathbf{Y}] =$ MSWLS$(m, n, \mathbf{Y})$
  end if
  return $\hat{m}, \mathbf{Y}, S$
end procedure

Therefore, the nth stabilized frame $\overline{F}_n$ can be obtained from the current input F_n by

$$
\begin{aligned}
\overline{F}_n &= \left(\prod_{i=1}^{n} \overline{A}_i\right) \cdot \left(\prod_{i=1}^{n} A_i\right)^{-1} F_n \\
&= \left(\overline{A}_n \cdot \ldots \cdot \overline{A}_1\right) \cdot \left(A_1^{-1} \cdot \ldots \cdot A_n^{-1}\right) F_n
\end{aligned} \tag{31}
$$

6. Experiment and result

We implement our algorithm on a PC with a Core i7-6770 4.0 GHz CPU and a GeForce GTX2080 Ti GPU. SURF and FLANN are based on the GPU implementation of OpenCV. We implement both Parallel AC-RANSAC and RANSAC on GPU with CUDA, and the rest is run on CPU. Table 1 presents the timing profile for the proposed algorithm. For videos with resolutions of 1920 × 1080 or higher, we downsample them to 1280 × 720 before further processing.

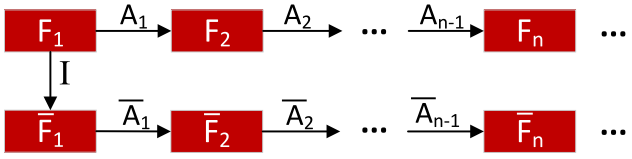

Fig. 7. Global transformation relationship between original F_i and stable $\overline{F}_i$ frames.

Table 1
Timing Performance of the proposed algorithm in milliseconds (Total time is computed with parallel AC-RANSAC).

Resolution	SURF and FLANN	Parallel AC-RANSAC	AC-RANSAC	Motion Smoothing	Total
320 × 240	6.83	3.19	79.75	0.14	12.59
640 × 360	8.48	5.10	128.52	0.56	16.21
1280 × 720	17.00	7.28	181.27	1.04	31.67
1920 × 1080	17.10	7.30	183.23	1.25	32.80

6.1. Inter-frame global motion estimation via parallel AC-RANSAC

6.1.1. Feature detection, matching, and match selection

We use SURF and FLANN as our feature detection and matching method, as they provide a fair tradeoff of robustness and efficiency [23,24]. We use OpenCV's parallel implementation for SURF and FLANN on GPU. As we can see from Table 1, SURF and FLANN are the most time-consuming compared to other parts because of the computation of SURF features and the descriptors. Since OpenCV's GPU implementation of SURF is at the pixel level, the number of pixels easily dominates the number of CUDA cores in a GPU, which diminishes the gain of parallelism when the image resolution is high.

6.1.2. RANSAC vs. AC-RANSAC

Compared to RANSAC, AC-RANSAC automatically selects thresholds for inlier/outlier discrimination, but the extra computation of AC-RANSAC prevents it from being used in real-time applications. We hence propose the parallel AC-RANSAC implemented with CUDA on GPU. To further reduce the processing time, we massively utilize the on-chip shared memory. In our implementation, each thread inside a thread block independently estimates the model error for a match. Since a maximum of 1024 threads is allowed in a thread block in GPU, only consecutive frames that generate less than 1024 matches can be handled inherently, which is usually valid for videos with less than 1920 × 1080 pixels. Otherwise, as described in 3.1, the matches

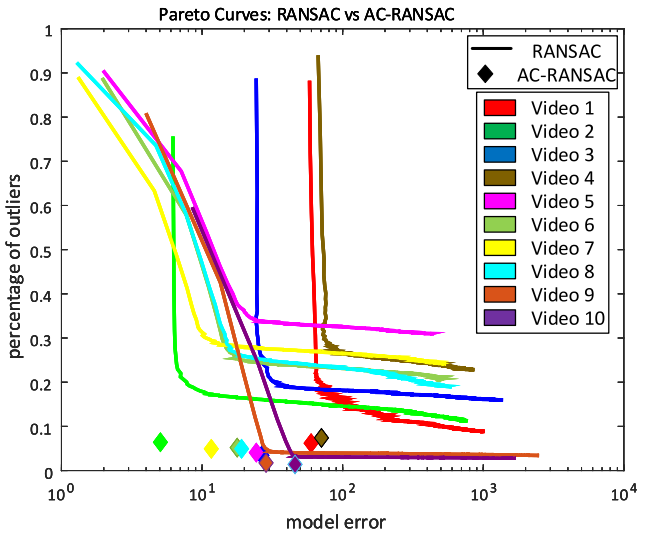

Fig. 8. The threshold for RANSAC ranges from 0.001 to 0.3. Each data point in RANSAC is the result of a run with a threshold value. For all the 10 testing data, AC-RANSAC computes a model with more inliers (less outliers in the plot) and smaller model error.

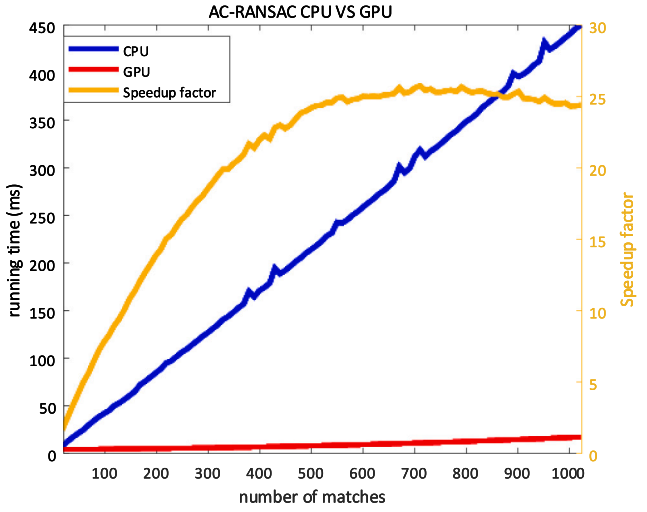

Fig. 9. Timing performance between GPU and CPU implementation of AC-RANSAC as number of matches increases. The result is the average of three trials.

selection method is applied to choose the most 1024 robust matches for AC-RANSAC.

For challenging frames that do not have enough feature points or undergo significant non-ridge motion, etc., AC-RANSAC (or RANSAC) will not be able to estimate a robust affine/homography model that well explains the geometric transformation between the frames. In such cases, the model returned by AC-RANSAC (or RANSAC) is ill-conditioned. Hence, we discard the estimated simplified affine transformation and set it to the identity matrix if the smallest singular value of it is less than 0.001.

As discussed in 3.2, the drawback of AC-RANSAC is the high computational complexity compared to RANSAC. As shown in Fig. 9, our parallel implementation achieves 25X speed up and around 20 ms for the 1024 matches. We use the number of inliers and model errors to assess AC-RANSAC's and RANSAC's effectiveness. We define the model error as

$$e = \max\left(A\mathbf{I}_l - \mathbf{I}_r, \mathbf{I}_l - A^{-1}\mathbf{I}_r\right) \tag{32}$$

where A is the estimated affine model, $\mathbf{I}_l$ is the feature point in the current frame, and $\mathbf{I}_r$ is the matched feature point in the next frame.

For RANSAC, we range the discrimination threshold from 0.001 to 0.5 and plot the number of inliers and model errors for each run. In AC-RANSAC, the inliers are the matches whose evaluated errors are smaller than the match that achieves minNFA. Since AC-RANSAC does not require a threshold, there is only one data point for each video for AC-RANSAC. As shown in Fig. 8, AC-RANSAC results in a model that has more inliers and a smaller model error than RANSAC.

6.2. Stabilization comparison

We compare three real-time [19,37,47] and three offline video stabilization algorithms [10,13,48] with the proposed algorithm, and further quantify two essential aspects (distortion and smoothness) of stabilization quality with two objectives metrics.

We use Inter-frame Transformation Fidelity (ITF) to compute the distortion introduced by stabilization algorithms. ITF is a popular evaluation metric of stabilization quality, which is defined as

$$ITF = \frac{1}{N-1}\sum_{k=1}^{N-1} PSNR(F_{k+1}, F_k) \tag{33}$$

where N is the total number of frames and

$$PSNR(F_{k+1}, F_k) = 10\log_{10}\left(\frac{peakval^2}{MSE(F_{k+1}, F_k)}\right) \tag{34}$$

is the peak signal-to-noise ration between the two consecutive frames with $peakval$ being the maximum pixel value and $MSE(F_{k+1}, F_k)$ the mean square error between consecutive frames F_k and F_{k+1}.

A higher value of ITF indicates smaller average inter-frame distortion across the video and thus better stabilization quality. We compare the proposed algorithm with other state-of-the-art stabilization algorithms in terms of ITF. Since [10,37] use inpainting to fill the missing borders of the stabilized frame, for a fair comparison, we crop all the stabilized video by 20 pixels before computing ITF for the stabilized videos produced by other algorithms.

We use the normalized decrease in feature point acceleration to measure thesmoothness of a stabilized video, as feature points in a smooth video should have zero or constant acceleration [13]. [37] adopts such an idea to assess the smoothness of a stabilized video. For each output video, we first extract SURF features for each frame and perform matching to build feature trajectories. Then we compute the sum of acceleration for each trajectory by

$$V(I) = \sum\sqrt{\left(I_{i+1}^x - 2I_i^x + I_{i-1}^x\right)^2 + \left(I_{i+1}^y - 2I_i^y - I_{i-1}^y\right)^2} \tag{35}$$

where (I_i^x, I_i^y) is the image coordinate of feature point I at the ith frame. Finally, the smoothness can be calculated as the average normalized decrease in trajectory acceleration [37]:

$$smoothness = \frac{1}{N}\sum_{k=1}\frac{|V_k^o - V_k^s|}{V_k^o} \tag{36}$$

where V_k^o is the sum of acceleration of the trajectory in the original video, and V_k^s is the sum of acceleration in the stabilized video. An output video with higher smoothness is superior.

We use the OpenCV implementations for [10,13], the official implementation for [48] provided by *Adobe After Effect*, and the original implementation or software available in GitHub for [19,37,47] provided by the authors. All algorithms are run on public data set available in [13,47,48] and compared in terms of distortion and smoothness. We compared LSstab with state-of-the-art real-time algorithms [19,37,47] and offline algorithms [10,13,48] for videos with different resolutions. Because the undefined borders of the stabilized outputs for the algorithms above [10,13,19,37,47,48] are removed or inpainted, we remove 10% border, i.e., 0.9 cropping ratio, before comparing the results.

Table 2
Comparison with real-time video stabilization algorithms. The average time per frame is reported in table. The best results in ITF and Smoothness are bolded. Processing times below 33 ms are also highlighted.

Methods	320 × 240			640 × 360			1280 × 720			1920 × 1080		
	ITF(dB)	Smooth	Time	ITF (dB)	Smooth	Time	ITF (dB)	Smooth	Time	ITF (dB)	Smooth	Time
Meshflow [47]	**31.10**	**0.88**	483.9 ms	26.48	**0.75**	612.5 ms	23.33	0.75	3.421 s	22.57	**0.22**	7.562 s
LP-Kalman-LP [19]	24.54	0.38	**20.08** ms	22.15	0.33	**21.58** ms	20.35	0.23	53.01 ms	21.44	0.12	94.91 ms
Frame-Orbit [37]	27.87	0.75	**4.000** ms	24.25	0.71	**7.300** ms	22.55	0.75	**16.20** ms	21.97	0.13	**27.70** ms
LSstab (ours)	28.68	0.82	**12.59** ms	**26.69**	0.64	**16.21** ms	**24.41**	**0.79**	**31.67** ms	**22.62**	**0.22**	**32.80** ms

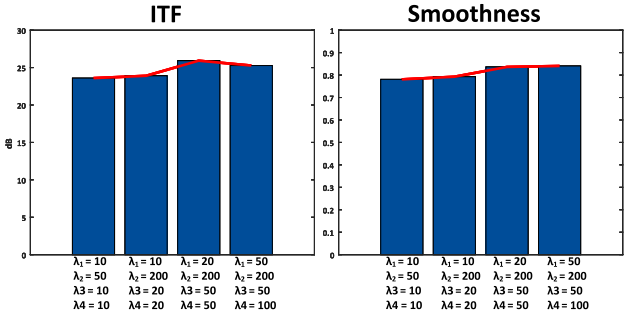

Fig. 10. Comparison with different combination of $\lambda_1, \lambda_2, \lambda_3, \lambda_4$.

As shown in Table 2, the proposed method does the best in minimizing distortion, and [47] does the best in encouraging smoothness. However, the processing time for [47] grows rapidly as video resolution increases, even after we decrease the grid resolution to 0.8 of the default value. As shown in Fig. 11, [37,47] and the proposed algorithm are competitive to [10,13] compared to offline algorithms. [11,47] have the best performance on average in terms of distortion and smoothness.

To investigate the impact of the four regularization parameters $\lambda_1, \lambda_2, \lambda_3$, and λ_4, we run the proposed algorithm with different values on the first video in Fig. 11.

As Fig. 10 shows, larger regularization parameters will encourage smoothness. However, as they get larger and larger, the algorithm will try to compensate for inter-frame global motions as much as possible, resulting in a static video instead of a stable video. Moreover, overcompensating inter-frame global motions will also introduce an undefined area in the result frames, which makes the distortion more significant.

6.3. Limitations

The choice of parameters has a direct impact on the quality of stabilization. If λ_1, λ_2, λ_3, and λ_4 are set too large, the proposed algorithm will produce an over-stabilized video and introduce a large cropping area. On the other hand, if they are too small, the algorithm cannot remove the jitters in the input video. We found that $\lambda_1 = 2, \lambda_2 = 200, \lambda_3 = 50, \lambda_4 = 50$ produce good empirical results. The other limitation is the accumulated error in the proposed algorithm. The stabilized frame is obtained through the accumulating products of $\hat{m}$ and m_i based on (31) in the motion compensation stage so that any wrong estimates will affect the subsequent frames.

7. Conclusions and future work

While most video stabilization algorithms are offline and require future frames for smoothing, we propose an online real-time video stabilization algorithm, LSstab, which stabilizes the incoming video frame in real-time using only past frames. LSstab features a parallel realization of the *a-contrario* RANSAC (AC-RANSAC) algorithm to estimate the inter-frame camera motion parameters and a novel fast recursive least squares algorithm to find the stable camera motion obeying cinematography principles. Techniques such as inpainting can be readily included in our algorithm. Currently, the cost function (26) does not include the cropping area yet. In the future, one possible direction is to incorporate a proxy of cropping area that is a function of λ_1 and λ_2 to the cost function and further develop an adaptive approach for setting λ_1 and λ_2. The keyframe selection technique proposed in [13] can smoothly be adapted to mitigate the accumulated error.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Acknowledgments

This work was supported by the National Institute of Biomedical Imaging and Bioengineering (NIBIB) of the US National Institutes of Health (NIH) under award number R01EB019460.

Appendix A. Derivation of (10) in matrix form

$$E = P + \lambda_1 Q + \lambda_2 U$$
$$= \sum_{i=1}^{n}(m_i - \overline{m}_i)^2 + \lambda_1 \sum_{i=2}^{n}(\overline{m}_i - \overline{m}_{i-1})^2 + \lambda_2 \sum_{i=1}^{n-1}(\overline{m}_i - \hat{m}_i)^2$$

Taking all partial derivatives of E w.r.t $\overline{m}_1, \ldots, \overline{m}_n$ and setting them to zero, we have

$$\frac{\partial E}{\partial \overline{m}_1} = \frac{\partial}{\partial \overline{m}_1}[(m_1 - \overline{m}_1)^2 + \lambda_1(\overline{m}_2 - \overline{m}_1)^2 + \lambda_2(\overline{m}_1 - \hat{m}_1)^2] = 0$$
$$\Rightarrow (1 + \lambda_1 + \lambda_2)\overline{m}_1 - \lambda_1 \overline{m}_2 = m_1 + \lambda_2 \hat{m}_1$$

For $i = 2, \ldots, n-1$

$$\frac{\partial E}{\partial \overline{m}_i} = \frac{\partial}{\partial \overline{m}_i}[(m_i - \overline{m}_i)^2 + \lambda_1(\overline{m}_i - \overline{m}_{i-1})^2 + \lambda_1(\overline{m}_{i+1} - \overline{m}_i)^2 +$$
$$\lambda_2(\overline{m}_i - \hat{m}_i)^2] = 0$$
$$\Rightarrow -\lambda_1 \overline{m}_{i-1} + (1 + 2\lambda_1 + \lambda_2)\overline{m}_i - \lambda_1 \overline{m}_{i+1} = m_i + \lambda_2 \hat{m}_i$$

Finally,

$$\frac{\partial E}{\partial \overline{m}_n} = \frac{\partial}{\partial \overline{m}_n}[(m_n - \overline{m}_n)^2 + \lambda_1(\overline{m}_n - \overline{m}_{n-1})^2] = 0$$
$$\Rightarrow -\lambda_1 \overline{m}_{n-1} + (1 + \lambda_1)\overline{m}_n = m_n$$

Therefore, solving (10) is equivalent to solve the system (11) of linear equations.

Fig. 11. Comparison with state-of-the-art video stabilization algorithms. Higher ITF and Smoothness means better stabilization result.

Appendix B. Recursive solution for finding $\overline{m}_n$ in (11)

Let $a = (1+\lambda_1+\lambda_2), b = -\lambda_1, c = (1+2\lambda_1+\lambda_2), d = (1+\lambda_1)$

For $n = 2$, We have $\mathbf{S}_2 = \begin{bmatrix} a & b \\ b & d \end{bmatrix}$, $\mathbf{S}_2^{-1} = \frac{1}{ad-b^2}\begin{bmatrix} d & -b \\ -b & a \end{bmatrix}$, so

$$\hat{m}_2 = \frac{-b(1+\lambda_2)m_1 + am_2}{ad-b^2} \tag{B.1}$$

For $n = 3$,

We perform the Gauss-Jordan elimination on $\mathbf{S}_3$:

$$\left[\begin{array}{ccc|ccc} a & b & 0 & 1 & 0 & 0 \\ b & c & b & 0 & 1 & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(1)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ b & c & b & 0 & 1 & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(2)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & \frac{ac-b^2}{a} & b & -\frac{b}{a} & 1 & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(3)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & b & \frac{ab^2}{ac-b^2} & -\frac{b^2}{ac-b^2} & \frac{ab}{ac-b^2} & 0 \\ 0 & b & d & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(4)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & b & \frac{ab^2}{ac-b^2} & -\frac{b^2}{ac-b^2} & \frac{ab}{ac-b^2} & 0 \\ 0 & 0 & d-\frac{ab^2}{ac-b^2} & \frac{b^2}{ac-b^2} & -\frac{ab}{ac-b^2} & 1 \end{array}\right]$$

$$\xrightarrow{(5)} \left[\begin{array}{ccc|ccc} b & \frac{b^2}{a} & 0 & \frac{b}{a} & 0 & 0 \\ 0 & b & \frac{ab^2}{ac-b^2} & -\frac{b^2}{ac-b^2} & \frac{ab}{ac-b^2} & 0 \\ 0 & 0 & 1 & \frac{b^2}{q_3} & -\frac{ab}{q_3} & \frac{ac-b^2}{q_3} \end{array}\right]$$

Then, $\hat{m}_3 = \begin{bmatrix} \frac{b^2}{q_3} & -\frac{ab}{q_3} & \frac{ac-b^2}{q_3} \end{bmatrix} \cdot \mathbf{Y}_3$ where $q_3 = acd - b^2d - ab^2d$

For $n = 4$,

Note that first 3 row operations on $\mathbf{S}_3$ will yield the same result for $\mathbf{S}_4$. Hence, we can reuse the partial result of Gauss-Jordan elimination of $\mathbf{S}_3$ for $\mathbf{S}_4$. We first record the 2nd row of the above echelon form of

$\mathbf{S}_3$, $x_1^{(3)} = \frac{ab^2}{ac-b^2}, y_1^{(3)} = -\frac{b^2}{ac-b^2}, y_2^{(3)} = \frac{ab}{ac-b^2}$, then we have

$$\left[\begin{array}{cccc|cccc} a & b & 0 & 0 & 1 & 0 & 0 & 0 \\ b & c & b & 0 & 0 & 1 & 0 & 0 \\ 0 & b & c & b & 0 & 0 & 1 & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(3)} \left[\begin{array}{cccc|cccc} b & \frac{b^2}{a} & 0 & 0 & \frac{b}{a} & 0 & 0 & 0 \\ 0 & b & x_1^{(3)} & 0 & y_1^{(3)} & y_2^{(3)} & 0 & 0 \\ 0 & b & c & b & 0 & 0 & 1 & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(4)} \left[\begin{array}{cccc|cccc} b & \frac{b^2}{a} & 0 & 0 & \frac{b}{a} & 0 & 0 & 0 \\ 0 & b & x_1^{(3)} & 0 & y_1^{(3)} & y_2^{(3)} & 0 & 0 \\ 0 & 0 & b & \frac{b^2}{c-x_1^{(3)}} & \frac{-y_1^{(3)}b}{c-x_1^{(3)}} & \frac{-y_2^{(3)}b}{c-x_1^{(3)}} & \frac{b}{c-x_1^{(3)}} & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\xrightarrow{(5)} \left[\begin{array}{cccc|cccc} b & \frac{b^2}{a} & 0 & 0 & \frac{b}{a} & 0 & 0 & 0 \\ 0 & b & x_1^{(3)} & 0 & y_1^{(3)} & y_2^{(3)} & 0 & 0 \\ 0 & 0 & b & \frac{b^2}{c-x_1^{(3)}} & \frac{-y_1^{(3)}b}{c-x_1^{(3)}} & \frac{-y_2^{(3)}b}{c-x_1^{(3)}} & \frac{b}{c-x_1^{(3)}} & 0 \\ 0 & 0 & 0 & 1 & \frac{y_1 b}{q_4} & \frac{y_2 b}{q_4} & \frac{-b}{q_4} & \frac{c-x_1}{q_4} \end{array}\right]$$

Then, $\widehat{m}_4 = \left[\frac{y_1 b}{q_4} \quad \frac{y_2 b}{q_4} \quad \frac{-b}{q_4} \quad \frac{c-x_1}{q_4}\right] \cdot \mathbf{Y}_4$ where $q_4 = cd - x_1^{(3)}d - b^2$. Note that we directly use the result from the previous iteration in (3), and the row operations (4) (5) are the same for each iteration.

For $n = 5$, we would need to store 3rd row of the above echelon form of $\mathbf{S}_4$, i.e. $x_1^{(4)} = \frac{b^2}{c-x_1^{(3)}}, y_1^{(4)} = \frac{-y_1^{(3)}b}{c-x_1^{(3)}}, y_2^{(4)} = \frac{-y_2^{(3)}b}{c-x_1^{(3)}}, y_3^{(4)} = \frac{b}{c-x_1^{(3)}}$.

Suppose we have compute the echelon form for $\mathbf{S}_{n-1}$ and store the $(n-1)$th row as $x_1^{(n-1)}, y_1^{(n-1)}, \ldots, y_{n-3}^{(n-1)}$, Then we have the Gauss-Jordan elimination for $\mathbf{S}_n$ as:

$$\left[\begin{array}{cccc|cccc} a & b & \ldots & 0 & 1 & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & b & c & b & 0 & 0 & 1 & 0 \\ 0 & 0 & b & d & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{cccc|cccccc} \vdots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & y_2^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \\ \cdots & b & c & b & 0 & 0 & 0 & 0 & 1 & 0 \\ \cdots & 0 & b & d & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{cccc|ccccc} \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \\ \cdots & 0 & b & \frac{b^2}{c-x_1^{(n-1)}} & \frac{-y_1^{(n-3)}b}{c-x_1^{(n-1)}} & \cdots & \frac{-y_{n-3}^{(n-1)}b}{c-x_1^{(n-1)}} & \frac{b}{c-x_1^{(n-1)}} & 0 \\ \cdots & 0 & b & d & 0 & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow$$

$$\left[\begin{array}{cccc|ccccc} \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \cdots & b & x_1^{(n-1)} & 0 & y_1^{(n-1)} & \cdots & y_{n-3}^{(n-1)} & 0 & 0 \\ \cdots & 0 & b & \frac{b^2}{c-x_1^{(n-1)}} & \frac{-y_1^{(n-3)}b}{c-x_1^{(n-1)}} & \cdots & \frac{-y_{n-3}^{(n-1)}b}{c-x_1^{(n-1)}} & \frac{b}{c-x_1^{(n-1)}} & 0 \\ \cdots & 0 & 0 & 1 & \frac{y_1^{(n-1)}b}{q_n} & \cdots & \frac{y_{n-3}^{(n-1)}b}{q_n} & \frac{-b}{q_n} & \frac{c-x_1^{(n-1)}}{q_n} \end{array}\right]$$

Therefore,

$$\widehat{m}_n = \left[\frac{y_2^{(n-1)}b}{q_n} \quad \ldots \quad \frac{y_{n-3}^{(n-1)}b}{q_n} \quad \frac{-b}{q_n} \quad \frac{c-x_1^{(n-1)}}{q_n}\right] \cdot \mathbf{Y}_n \tag{B.2}$$

where $q_n = cd - x_1^{(n-1)}d - b^2$. Then we store the $(n-1)$th row of the above echelon form of $\mathbf{S}_n$ for the next iteration.

References

[1] P.L.M. Bouttefroy, A. Bouzerdoum, S.L. Phung, A. Beghdadi, Integrating the projective transform with particle filtering for visual tracking, EURASIP J. Image Video Process. 2011 (1) (2010) 839412.

[2] J. Sánchez, Comparison of motion smoothing strategies for video stabilization using parametric models, Image Process. Online 7 (2017) 309–346, http://dx.doi.org/10.5201/ipol.2017.209.

[3] J. Sánchez, Comparison of motion smoothing strategies for video stabilization using parametric models, Image Process. Online 7 (2017) 309–346.

[4] J. Chen, J. Ke, F. Lu, J. Fernandes, B. King, Y.H. Hu, H. Jiang, Gait monitoring using an ankle-worn stereo camera system, in: 2022 IEEE Sensors, 2022, pp. 1–4, http://dx.doi.org/10.1109/SENSORS52175.2022.9967079.

[5] J. Ke, A.J. Watras, J.-J. Kim, H. Liu, H. Jiang, Y.H. Hu, Towards real-time 3D visualization with multiview RGB camera array, J. Signal Process. Syst. 94 (3) (2022) 329–343.

[6] A.J. Watras, J.-J. Kim, J. Ke, H. Liu, J.A. Greenberg, C.P. Heise, Y.H. Hu, H. Jiang, Large-field-of-view visualization with small blind spots utilizing tilted micro-camera array for laparoscopic surgery, Micromachines 11 (5) (2020).

[7] J. Ke, A.J. Watras, J.-J. Kim, H. Liu, H. Jiang, Y.H. Hu, Towards real-time, multi-view video stereopsis, in: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2020, pp. 1638–1642, http://dx.doi.org/10.1109/ICASSP40776.2020.9054391.

[8] W.G. Aguilar, C. Angulo, Real-time model-based video stabilization for microaerial vehicles, Neural Process. Lett. 43 (2) (2016) 459–477.

[9] Y. Wang, Z.J. Hou, K. Leman, R. Chang, Real-time video stabilization for unmanned aerial vehicles, in: Proceedings of the 12th IAPR Conference on Machine Vision Applications, MVA 2011, 2011, pp. 336–339.

[10] Y. Matsushita, W. Ge, X. Tang, H.Y. Shum, Full-frame video stabilization with motion inpainting, IEEE Trans. Pattern Anal. Mach. Intell. 28 (7) (2006) 1150–1163.

[11] F. Liu, M. Gleicher, H. Jin, A. Agarwala, Content-preserving warps for 3D video stabilization, ACM Trans. Graph. 28 (3) (2009).

[12] W. Guilluy, L. Oudre, A. Beghdadi, Video stabilization: Overview, challenges and perspectives, Signal Process., Image Commun. 90 (2021) 116015.

[13] M. Grundmann, V. Kwatra, I. Essa, Auto-directed video stabilization with robust L1 optimal camera paths, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2011, pp. 225–232.

[14] H.C. Chang, S.H. Lai, K.R. Lu, A robust real-time video stabilization algorithm, J. Vis. Commun. Image Represent. 17 (3) (2006) 659–673.

[15] S. Battiato, G. Gallo, G. Puglisi, S. Scellato, SIFT features tracking for video stabilization, in: 14th International Conference on Image Analysis and Processing (ICIAP 2007), 2007, pp. 825–830, http://dx.doi.org/10.1109/ICIAP.2007.4362878.

[16] S. Liu, L. Yuan, P. Tan, J. Sun, Bundled camera paths for video stabilization, in: ACM Transactions on Graphics (SIGGRAPH 2013), Vol. 32, ACM, 2013, URL: https://www.microsoft.com/en-us/research/publication/bundled-camera-paths-video-stabilization/.

[17] F. Liu, M. Gleicher, J. Wang, H. Jin, A. Agarwala, Subspace video stabilization, ACM Trans. Graph. 30 (1) (2011).

[18] M.A. Fischler, R.C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACM 24 (6) (1981) 381–395.

[19] L. Kejriwal, I. Singh, A hybrid filtering approach of digital video stabilization for uav using Kalman and low pass filter, Procedia Comput. Sci. 93 (2016) 359–366, Proceedings of the 6th International Conference on Advances in Computing and Communications.

[20] A. Litvin, J. Konrad, W.C. Karl, Probabilistic video stabilization using Kalman filtering and mosaicing, in: Image and Video Communications and Processing 2003, Vol. 5022, SPIE, International Society for Optics and Photonics, 2003, pp. 663–674, http://dx.doi.org/10.1117/12.476436.

[21] H. Bay, A. Ess, T. Tuytelaars, L.V. Gool, Speeded-up robust features (SURF), Comput. Vis. Image Underst. 110 (3) (2008) 346–359, Similarity Matching in Computer Vision and Multimedia.

[22] M. Muja, D.G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: VISAPP International Conference on Computer Vision Theory and Applications, 2009, pp. 331–340.

[23] S.A.K. Tareen, Z. Saleem, A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK, in: 2018 International Conference on Computing, Mathematics and Engineering Technologies (ICoMET), 2018, pp. 1–10.

[24] F.K. Noble, Comparison of OpenCV's feature detectors and feature matchers, in: 2016 23rd International Conference on Mechatronics and Machine Vision in Practice (M2VIP), 2016, pp. 1–6.
[25] A. Goldstein, R. Fattal, Video stabilization using epipolar geometry, ACM Trans. Graph. 32 (5) (2012).
[26] S. Bell, A. Troccoli, K. Pulli, A non-linear filter for gyroscope-based video stabilization, in: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (Eds.), Computer Vision – ECCV 2014, Springer International Publishing, Cham, 2014, pp. 294–308.
[27] H. Ovrén, P.-E. Forssén, Gyroscope-based video stabilisation with auto-calibration, in: 2015 IEEE International Conference on Robotics and Automation, ICRA, 2015, pp. 2090–2097, http://dx.doi.org/10.1109/ICRA.2015.7139474.
[28] S. Liu, Y. Wang, L. Yuan, J. Bu, P. Tan, J. Sun, Video stabilization with a depth camera, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 89–95, http://dx.doi.org/10.1109/CVPR.2012.6247662.
[29] B.M. Smith, L. Zhang, H. Jin, A. Agarwala, Light field video stabilization, in: 2009 IEEE 12th International Conference on Computer Vision, 2009, pp. 341–348, http://dx.doi.org/10.1109/ICCV.2009.5459270.
[30] M. Wang, G.-Y. Yang, J.-K. Lin, S.-H. Zhang, A. Shamir, S.-P. Lu, S.-M. Hu, Deep online video stabilization with multi-grid warping transformation learning, IEEE Trans. Image Process. 28 (5) (2019) 2283–2292.
[31] S.-Z. Xu, J. Hu, M. Wang, T.-J. Mu, S.-M. Hu, Deep video stabilization using adversarial networks, Comput. Graph. Forum 37 (7) (2018) 267–276.
[32] M. Zhao, Q. Ling, PwStableNet: Learning pixel-wise warping maps for video stabilization, IEEE Trans. Image Process. 29 (2020) 3582–3595.
[33] J. Yu, R. Ramamoorthi, Learning video stabilization using optical flow, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2020.
[34] J. Choi, I.S. Kweon, Deep iterative frame interpolation for full-frame video stabilization, ACM Trans. Graph. 39 (1) (2020).
[35] Z. Shi, F. Shi, W.-S. Lai, C.-K. Liang, Y. Liang, Deep online fused video stabilization, in: 2022 IEEE/CVF Winter Conference on Applications of Computer Vision, WACV, 2022, pp. 865–873, http://dx.doi.org/10.1109/WACV51458.2022.00094.
[36] S. Ertürk, Real-time digital image stabilization using Kalman filters, Real-Time Imaging 8 (4) (2002) 317–328.
[37] J. Dong, H. Liu, Video stabilization for strict real-time applications, IEEE Trans. Circuits Syst. Video Technol. 27 (4) (2017) 716–724.
[38] K. Ratakonda, Real-time digital video stabilization for multi-media applications, in: Proceedings - IEEE International Symposium on Circuits and Systems, Vol. 4, 1998, pp. 69–72.
[39] L. Moisan, B. Stival, A probabilistic criterion to detect rigid point matches between two images and estimate the fundamental matrix, Int. J. Comput. Vis. 57 (3) (2004) 201–218.
[40] A. Desolneux, L. Moisan, J.-M. Morel, Meaningful alignments, Int. J. Comput. Vis. 40 (1) (2000) 7–23.
[41] L. Moisan, P. Moulon, P. Monasse, Automatic homographic registration of a pair of images, with a contrario elimination of outliers, Image Process. Online 2 (2012) 56–73.
[42] P. Moulon, P. Monasse, R. Marlet, Adaptive structure from motion with a contrario model estimation, in: Proceedings of the Asian Computer Vision Conference (ACCV 2012), Springer Berlin Heidelberg, 2012, pp. 257–270, http://dx.doi.org/10.1007/978-3-642-37447-0_20.
[43] NVIDIA, Programming guide :: CUDA toolkit documentation, 2020, URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.
[44] M. Harris, Optimizing parallel reduction in CUDA, 2007, URL: https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf.
[45] Q. Zhang, Some implementation aspects of sliding window least squares algorithms, IFAC Proc. Vol. 33 (15) (2000) 763–768.
[46] R.J. LeVeque, Finite Difference Methods for Ordinary and Partial Differential Equations, Society for Industrial and Applied Mathematics, 2007, http://dx.doi.org/10.1137/1.9780898717839, URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898717839. arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9780898717839.
[47] S. Liu, P. Tan, L. Yuan, J. Sun, B. Zeng, MeshFlow: Minimum latency online video stabilization, in: B. Leibe, J. Matas, N. Sebe, M. Welling (Eds.), Computer Vision – ECCV 2016, Springer International Publishing, Cham, 2016, pp. 800–815.
[48] F. Liu, M. Gleicher, J. Wang, H. Jin, A. Agarwala, Subspace video stabilization, ACM Trans. Graph. 30 (1) (2011).