


Real-time optical flow-based video stabilization for unmanned aerial vehicles

Anli Lim¹ · Bharath Ramesh¹  · Yue Yang¹ · Cheng Xiang¹ · Zhi Gao² · Feng Lin²

Received: 18 November 2016 / Accepted: 6 June 2017 / Published online: 14 June 2017
© Springer-Verlag GmbH Germany 2017

Abstract This paper describes the development of a novel algorithm to tackle the problem of real-time video stabilization for unmanned aerial vehicles (UAVs). There are two main components in the algorithm: (1) By designing a suitable model for the global motion of UAV, the proposed algorithm avoids the necessity of estimating the most general motion model, projective transformation, and considers simpler motion models, such as rigid transformation and similarity transformation; (2) to achieve a high processing speed, optical flow-based tracking is employed in lieu of conventional tracking and matching methods used by state-of-the-art algorithms. These two new ideas resulted in a real-time stabilization algorithm, developed over two phases. Stage I considers processing the whole sequence of frames in the video while achieving an average processing speed of 50 fps on several publicly available benchmark videos. Next, Stage II undertakes the task of real-time video stabilization using a multi-threading implementation of the algorithm designed in Stage I.

Keywords UAV video stabilization · Interest point matching · Affine transformation · Real-time processing · Multi-threading

1 Introduction

In recent times, unmanned aerial vehicles (UAVs) are often equipped with streaming video cameras that can be employed for immediate observation. They are popular for several applications such as rescue, surveillance, mapping. The main drawbacks of videos taken by UAV are the undesired shaky motion caused by atmospheric turbulence and jittery flight control of the platform. The shaky motion in the video hinders the fundamental intention of using UAV for vision-based tasks. In particular, the unstable motion in the video inhibits higher-level vision tasks, such as detecting and tracking of targets. While significant work has been done for handheld cameras and ground vehicles, there are limited works in the literature for stabilization of videos taken from UAVs. Moreover, the existing UAV video stabilization algorithms cannot be directly used for real-time tasks.

In general, UAV video stabilization algorithms [5, 11, 13, 15, 16] follow three main steps: (1) motion estimation, (2) motion compensation and (3) image composition, as indicated in Fig. 1. Most methods revolve around finding the 2D motion model (such as homography) to calculate the global motion trajectory. Then a low-pass filter is applied to the trajectory to sieve out the high-frequency jitters. Subsequently, the low-frequency parameters are applied onto the frames via warping. This framework is very effective for scenes with little dynamic movement, which is applicable to aerial videos taken by UAVs. However, estimating the global motion trajectory using a window of frames cannot achieve instantaneous stabilization, as required for real-time processing.

In Shen et al. [13], a video stabilization algorithm for UAV was proposed using a circular block to search and match key places. The estimated affine transform is then

✉ Bharath Ramesh
bharath.ramesh03@u.nus.edu

¹ Department of Electrical and Computer Engineering,
National University of Singapore, Singapore 117576,
Singapore

² Temasek Laboratories, National University of Singapore,
Singapore 117576, Singapore

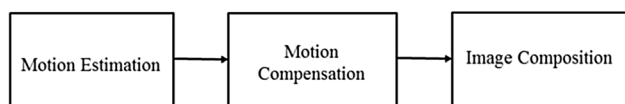


Fig. 1 Video stabilization framework

smoothed by the polynomial fitting and prediction method (PFPM). Nevertheless, this approach only achieves a speed of less than 10 fps (frames per second) on a desktop with 3.0 GHz processor and 1 GB RAM (random access memory) for images with resolution of 216×300 .

In Vazquez and Chang [15], a smoothing method utilizes Lucas–Kanade tracker [11] to detect interest points. The intended motion compensation is accomplished by adjusting for extra rotation and displacements that generate vibrations. This approach is able to achieve a stabilizing speed between 20 and 28 fps for images with resolution of 320×240 pixels on a laptop with 2.16 GHz Intel Core 2 Duo Processor with a three-frame delay.

Wang et al. [16] proposed a three-step video stabilization method for UAVs. Firstly, a features from accelerated segment test (FAST) corner detector is employed to locate the feature points in the frames. Secondly, the matched keypoints are used for estimation of affine transform to reduce false matches. Finally, motion estimation is performed based on the affine model and the compensation for vibration is conducted based on spline smoothing. It was reported that this algorithm can process up to 30 fps on a workstation with an Intel Xeon 2.26 GHz CPU and 6 GB RAM for images with resolution of 320×240 pixels.

A very recent work by Dong et al. [5] is able to process a 640×480 video at 40 fps on a notebook computer with a 2.5 GHz Intel Duo Core CPU. While it is able to achieve a high processing speed, the improvement comes at the price of unreliable motion estimation, which leads to failure in stabilizing the video ultimately. Moreover, Dong et al. [5] do not furnish evidence for processing on onboard systems in real time.

It is noted that the above methods work offline or with a delay of a few frames. For real-time UAV vision tasks, frames are required to be stabilized immediately and be

presented with other tasks, such as object tracking and detection.

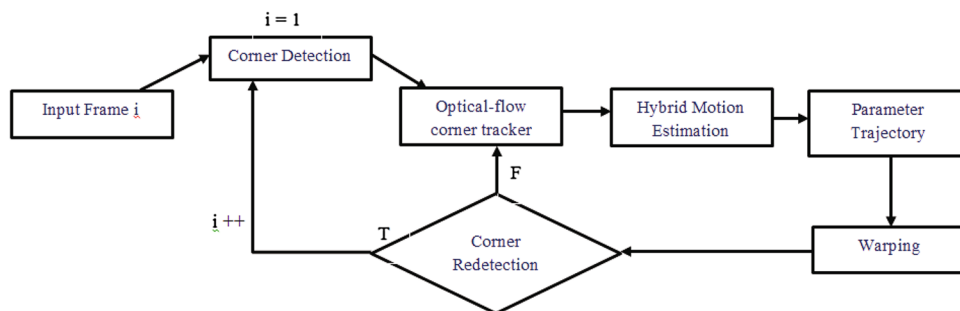
The proposed method in this paper is closely related to the method proposed by [9], which starts off by finding corners in frames, followed by estimating a 2D motion model between consecutive frames. Then the parameters of the motion model are treated as trajectory to be smoothed using an averaging window. This smoothing of the motion model parameters is different from the motion trajectory smoothing employed by previous works [5, 11, 13, 15, 16]. Our framework is similar to Ho [9], but has a significant difference in terms of implementation and performance. Most notably, a novel hybrid mechanism for motion estimation and an optical flow-based corner tracker has been proposed to overcome the challenges encountered by previous algorithms. In addition, the proposed stabilization algorithm performs in real time, whereas Ho [9] processes the whole video sequence before achieving stabilization.

The rest of this paper is structured as follows. In Sect. 2, we introduce the proposed framework for video stabilization. Then in Sect. 3, we discuss how the processing speed of motion estimation is expedited by an optical flow-based corner tracker and a reduced region of interest. In Sect. 4, we explain the improvements of motion estimation using a hybrid mechanism, following which, we outline the motion compensation and image warping step in Sect. 5. Next, in Sect. 6, we discuss the implementation of the real-time component of the algorithm using multi-thread processing. Then, we present the results on publicly available UAV videos and achieve a speed faster than the current state-of-the-art algorithm in Sect. 7. Finally, we conclude the paper in Sect. 8.

2 Proposed framework

Spurred by the challenge to meet the real-time requirements, we propose a novel stabilization framework, as shown in Fig. 2. The main idea is to build an appropriate model for the global motion trajectory of the UAV camera, estimate the motion parameters between two successive frames using an efficient keypoint detector and compensate

Fig. 2 Overview of the proposed video stabilization framework



the motion via an effective optical flow-based tracking of the keypoints.

In particular, the key novelty of the proposed algorithm lies in the high-speed estimation of the motion parameter trajectory between two consecutive frames, which other methods are less efficient in accomplishing. This is because the motion estimation phase chooses an appropriate lower-order homography, either rigid or similarity transformation between frames for a fixed dwell time. Thus, the algorithm is able to frequently use rigid transformation to estimate the motion parameters unless there is a need to use similarity transformation. Then, the motion compensation phase smooths the parameters via an averaging window. Lastly, the frames are warped according to the smoothed parameters.

For easy implementation on embedded platforms that have less memory and processing power, but multiprocessing capacity, the above-described video stabilization framework works in real time as follows. The three stages of video stabilization are separated into three different threads: motion estimation, motion compensation and image composition. Threads are basically independent processes that can run concurrently and share the same resources inside a program. The real-time algorithm is implemented such that the three threads are synchronized to stabilize incoming frames as and when available in a low-power embedded hardware. In other words, there is no need to process the full video to achieve stabilization, as done in previous works.

3 Optical flow-based tracking

Motion estimation is the very first step in video stabilization, and it is also the most time-consuming step. Consequently, our primary goal is to cut down the time required to compute the motion parameters trajectory between consecutive frames in the video. The motion parameter trajectory is estimated from how the feature points [8] move between consecutive frames. Therefore, to initialize motion estimation, we first efficiently detect feature points.

3.1 Feature point detection

Feature points are locations in the image with large variations in intensity in all directions. The number and quality of these feature points determine the quality of the motion estimation. One early attempt to find these corners was reported by [8], which now is called the Harris corner detector. It essentially finds the difference in intensity for a displacement of (u, v) in all directions. This can be easily expressed as:

$$E(u, v) = \sum_{x, y} w(x, y) [I(x + u, y + v) - I(x, y)]^2 \quad (1)$$

where $w(x, y)$ is either a rectangular or a Gaussian window function that gives weight to the surrounding pixels. Shi and Tomasi [14] made a small modification and showed better results compared to the Harris corner detector. The scoring function in Shi–Tomasi corner detector is given by:

$$R = \min(\lambda_1, \lambda_2) \quad (2)$$

- When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
- When R is negative, which happens when $\lambda_1 \gg \lambda_2$ and vice versa, the region is an edge.
- When R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \approx \lambda_2$, the region is a corner.

Since the proposed stabilization algorithm is to run in real time, the number of keypoints detected and matched from frame to frame is one of the main bottlenecks. To set a lower bound for the number of keypoints, we perform a panorama stitching experiment, which requires a homography estimation like the stabilization problem.

Three images of the same scene will be stitched together with a reference image (see Fig. 3). Firstly, keypoints will be detected in the left image and the center image. Next, keypoint matching will be performed between the corner points. Finally, a homography will be generated to stitch the left image to the perspective of the center image. The

Fig. 3 Mosaic images

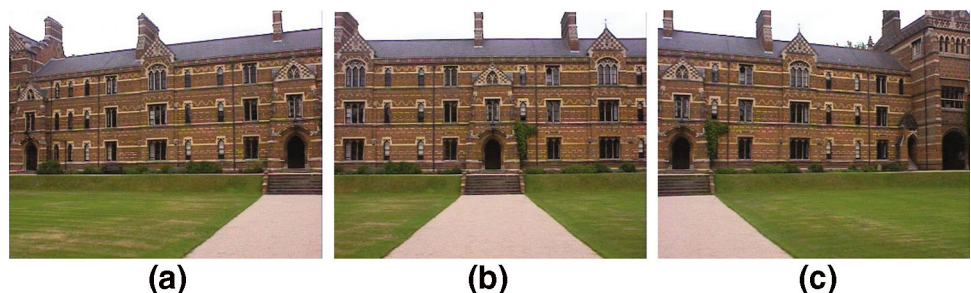




Fig. 4 Mosaic results. **a** 200 Keypoints, **b** 1000 keypoints

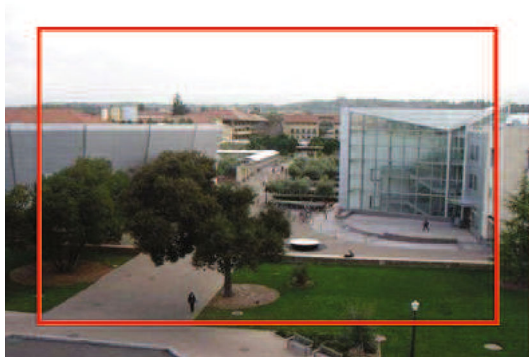


Fig. 5 Reduced region of interest

above procedure is repeated for the right image with the center image as reference.

The objective of the panorama experiment is to highlight the possibility of obtaining a good stitching using lower number of keypoints, thereby confirming a good estimation of homography between the mosaic images. As can be seen from the mosaic results in Fig. 4, the panorama stitch using 200 keypoints and 1000 keypoints is similar, but the latter comes at a heavier computational cost. Therefore, keeping in mind that our proposed idea is to run the algorithm in real time, the number of keypoints needed for a good estimate of the motion model is set to be less than 200. From stabilization experiments, we set a value of 50 which gave us visually pleasing results without compromising on stabilization quality for an in-house UAV video with severe shaky movements. This setting has been used throughout the paper for obtaining the stabilization results on public databases.

In addition to reducing the number of keypoints, the area for detection will be from a reduced region compared to that of the frame (see Fig. 5). The reason for this region limitation is because the pixels close to the edges of the frame have a high probability of not appearing in the next frame, especially when the UAV is in motion. In this manner, we can prevent the keypoints at the image boundaries from being detected while saving computational time needed to match those keypoints that have a high probability of vanishing in the next frame.

3.2 Feature point matching

To speed up the matching, a common way is to use approximate nearest neighbors [2]. This method is still slow for our real-time application. Furthermore, these techniques are only useful when the objective is just to characterize the neighborhood rather than the exact locations. In image stabilization, locations are important so that the homography generated is accurate. Therefore, we propose an optical flow-based matching of corners detected in one frame to the next frame.

Optical flow [6] is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of an object or camera. It is a 2D vector field that quantifies the motion of points from first frame to second. Optical flow assumes brightness constancy, as shown below.

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \quad (3)$$

Then taking Taylor series approximation of right-hand side, and removing common terms and dividing by dt , we get the following equation:

$$f_x u + f_y v + f_t = 0 \quad (4)$$

where

$$f_x = \frac{\partial f}{\partial x}; \quad f_y = \frac{\partial f}{\partial y} \quad (5)$$

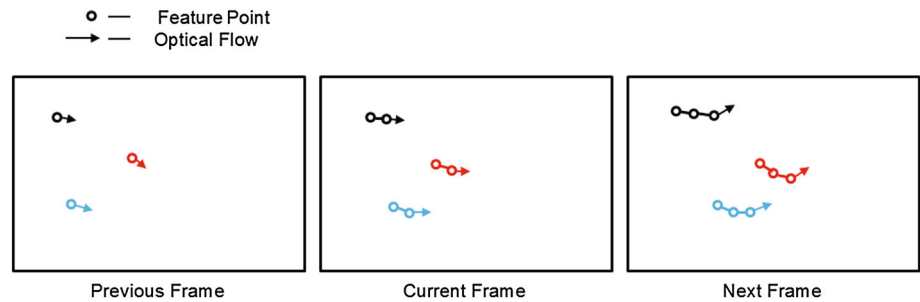
$$u = \frac{dx}{dt}; \quad v = \frac{dy}{dt} \quad (6)$$

The above equations describe the optical flow in terms of the spatial image gradients, f_x and f_y . Similarly, f_t is the gradient along time, but (u, v) is unknown. We cannot solve this one equation with two unknown variables. Several methods have been suggested to resolve this problem and we choose the gold standard, Lucas–Kanade algorithm [11].

By estimating the movement of the corner points, there is no need to detect corners in the next frame. And since the movement of the corners is known in the next frame, matching is already accomplished (see Fig. 6). Optical flow matching is repeated for five frames with corner detection performed only for the first frame, and then the corners are re-detected in the sixth frame. This is done to not violate the two main assumptions of optical flow, which are:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighboring pixels have similar motion.

Using the Lucas–Kanade method, the optical flow vectors of the corner points are obtained in the first frame and subsequently for the next four frames. However, the

Fig. 6 Optical flow to estimate motion

assumptions fail when there is large motion. Thus, we use pyramids [1, 4] to remove small motions and large motions becomes small motions as we go higher in the pyramid. Now applying Lucas–Kanade, the optical flow is obtained along with the scale.

In summary, the motion vectors are the coordinates of the keypoints that existed in the previous frame and have moved to a new location in the current frame. After retrieving the new keypoint locations, we know that there exists a set of corners that are both in the previous frame and in the current frame. This permits us to generate an inter-frame transformation matrix that is used for motion modeling and image composition.

4 Homography estimation

After acquiring the corners that exist in both current and previous frames, we can now estimate the motion model between these frames. The proposed motion estimation function allows the homography returned to be of similarity (also known as partial affine) or rigid (also known as Euclidean) transformation. If the homography returned is a rigid transformation, the transform parameters will store only the t_x , t_y , and angle of the transformation (scale factor is 1). If the homography matrix returned is of similarity transformation, the transform parameters will store t_x , t_y , angle and an arbitrary scale factor of the transformation. The deciding factor of the switching mechanism will be discussed in the next following subsection.

4.1 Hybrid mechanism for motion estimation

In previous works, motion estimation is carried out either using particle filters [17] or using variational methods [12], and mostly using scale invariant feature transform (SIFT)-based matching techniques [3], all of which are not suitable for the real-time implementation considered in this work. An extensive discussion of this point can be found in [5]. In fact, it was pointed out that the requirement of the long-range feature tracking makes many existing methods incompetent for challenging cases, since long feature

trajectories are difficult to obtain in sequences with rapid scene changes, texture less objects, severe occlusions or excessive motion blur. All these scenarios are common for UAV videos, and they are tackled in this work. Citing the above reasons, Dong et al. [5] made use of a fast KLT (Kanade–Lucas–Tracker) and achieved an average speed of 50 fps on publicly available UAV videos.

In general, motion estimation finds an affine transform [Alt] (a 2×3 floating-point matrix) that approximates best the affine transformation between two sets of points. In case of feature point sets, the problem is formulated as follows: find a 2×2 matrix A and 2×1 vector t for the source points (src) and the destination points (dst).

$$[A^* | t^*] = \arg \min \sum_i ||dst[i] - Asrc[i]^T - t||^2 \quad (7)$$

Solving for [Alt] requires a minimum of three pairing points that are not degenerate. This is straightforward to do. Let's denote the source point to be $X = [x \ y \ 1]$ and the destination to be $Y = [x' \ y' \ 1]$, giving:

$$TX = Y$$

$$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad (8)$$

The above equation can be rewritten as a typical $Az = b$ matrix and solved for z . Naturally, two extra pair of points are needed to solve for the motion parameters. The affine transform has a degree of freedom of six. There exist two lower degree of freedom transformations, namely similarity and rigid transformation, having four and three degrees of freedom, respectively. The four degrees of freedom include rotation angle, scaling, translation in x and translation in y . Rigid transformation assumes scaling to be 1.

Figure 7 illustrates the idea of the hybrid mechanism. E_r and E_s represent the total root-mean-square distance, (e_r , e_s), between the N corners of the previous frame and current frame, after applying the respective transformations. If the root-mean-square distance for rigid transformation is lower than similarity transformation, then the mechanism will select rigid transformation computation for a sequence of frames (i.e., dwell time). To achieve a balance between

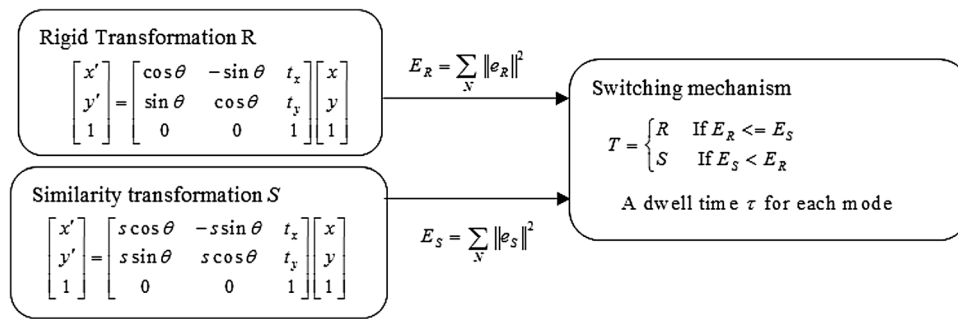


Fig. 7 Hybrid mechanism for motion estimation. The points in frame A are represented by (x, y) , and the points in frame B are represented by (x', y') . The homography parameters are scaling (s), angle (θ), translation (t_x and t_y). N is the number of keypoints. E_r and E_s represent the total root-mean-square distance, (e_r , e_s), between the

N corners of the previous frame and current frame, after applying the respective transformations. The switching mechanism chooses either the rigid or similarity transformation based on total root-mean-square distance between the transformed points and the source keypoints

speed and stability, a dwell time of 20 frames is used to switch between rigid and similarity transformation.

At the initial frame, both rigid and affine matrices are estimated from the set of keypoints that are detected and matched using optical flow. Then, two sets of motion vectors are estimated using the two transformation matrices. The average distance between the new points and the keypoint locations for both the transformations is calculated. If the average distance between the points generated by rigid transformation is smaller than that of the average distance between the points generated by similarity transformation, then the mechanism will remain to be rigid transformation or switch to similarity transformation. Therefore, the algorithm will return the transformation that gives less change in distance between the points.

The hybrid mechanism is switching between rigid and partial affine transformation. This means that we are stabilizing shaky motions using only up to four degrees of freedom, namely translation in X direction, translation in Y direction, the scale factor and the rotation factor.

Based on the video stabilization algorithm described so far, we have accomplished the implementation of the hybrid switching mechanism, which aims to achieve two main objectives:

1. By choosing the transformation (rigid or partial affine) with less overall root-mean-square distance between the corner points in successive frames, the switching mechanism stabilizes and reduces jitters dynamically.
2. By choosing the transformation with less parameters (rigid), the switching mechanism reduces computational time whenever possible.

Figure 8 gives an overview of the motion estimation workflow in the proposed algorithm. It starts by detecting corners in the first frame and then utilizes optical flow to detect and track corners in subsequent frames. On a rare basis, if the number of corners detected is too small, then stabilization is skipped for that frame. Additionally, keypoints are re-detected every five frames to maintain good quality of motion estimation. The optical flow tracking is followed by weeding step which checks for flow

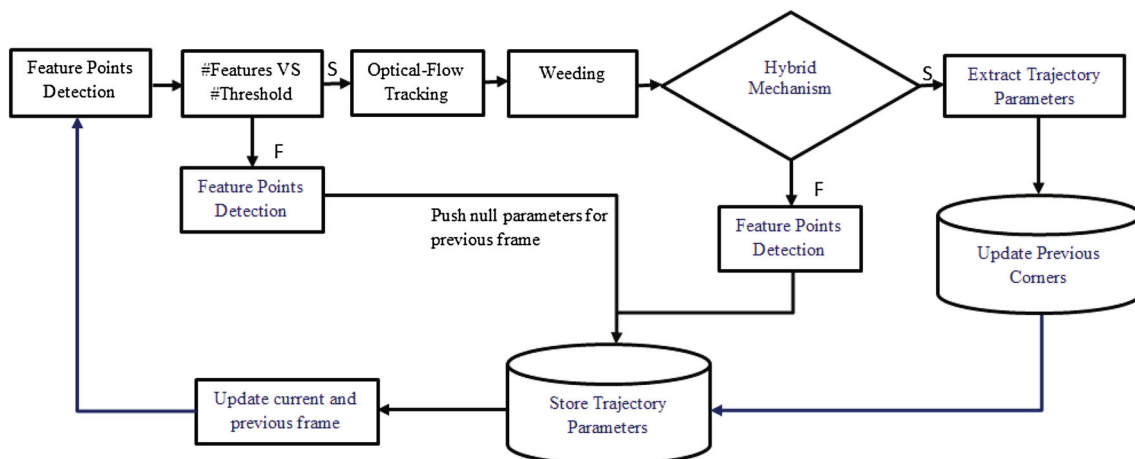


Fig. 8 Motion estimation

consistency in the backward direction, thereby eliminating bad matches. For every twenty frames, hybrid mechanism is called upon to determine the best transformation to be estimated. Then the parameter trajectories are extracted from the estimation and stored. The process is repeated for the total number of frames in the video.

5 Motion compensation and image composition

In the motion compensation stage, we employ a basic method of accumulating trajectory parameters and then smoothing them using an averaging window. The original trajectory of the parameters is accumulated within a window of frames, which is double the size of smoothing radius, and then the result of the addition of each parameter cumulatively is averaged out. Next, the difference between the averaged trajectory and the accumulated trajectory is added back to the original trajectory. The resulting quantity is called the smoothed trajectory. An example is shown below.

Figure 9a shows the original values of t_x . It can be seen that the values of t_x fluctuate a lot, which correlates with our observation of the video being shaky. The next step is to smooth the x values separately after accumulating the

frame-to-frame transformation using the approach described in the last paragraph (see Fig. 9b). Note that a large averaging window size is prone to the risk of including irrelevant frames compared to the target frame. Conversely, a small smoothing radius leads to an inadequate and ineffective smoothing of the higher-frequency parameter trajectories. Therefore, there is a need for finding a balanced setting, and this was obtained using the in-house UAV video. Subsequently, the same setting has been used for all other test videos.

It is worth mentioning that the parameter trajectory is a rather abstract quantity that doesn't necessarily have a direct relationship to the motion induced by the camera. For a simple panning scene with static objects, it has a direct relationship with the absolute position of the image. The important insight is that the trajectory can be smoothed, even if it does not have any physical interpretation, as shown in Fig. 9b.

In the final step of the video stabilization algorithm, a basic method is used to couple warping with cropping. First, warp the frame according to the new homography derived from the previous step, as shown in Fig. 10. Next, the cropping and resizing will reduce the unfilled area caused by the smoothed motion parameters.

Fig. 9 Motion compensation results

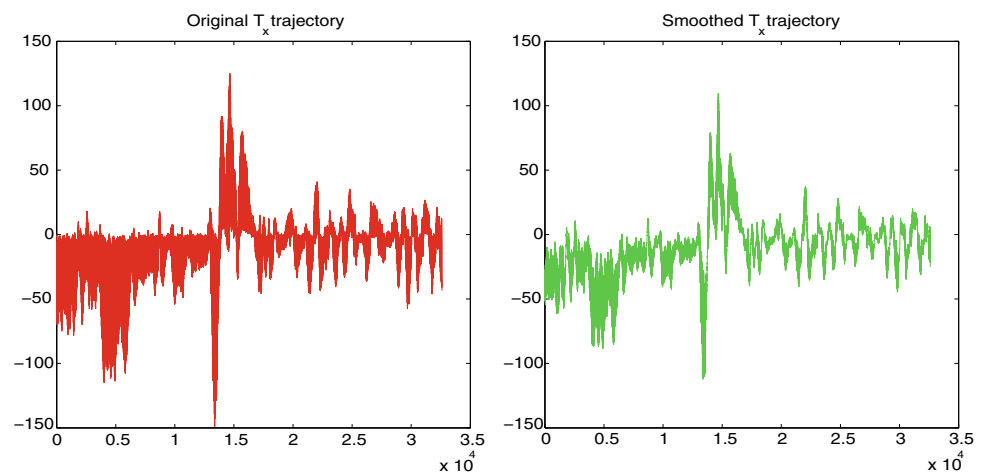


Fig. 10 Warping to image plane

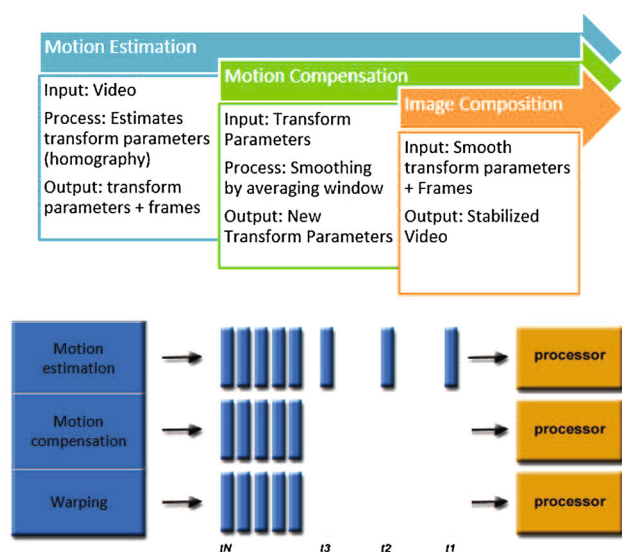


Fig. 11 Multi-threaded concurrent work flow

6 Real-time implementation

As discussed in the previous sections, the proposed stabilization algorithm has three steps, namely motion estimation, motion compensation and image composition. These three steps are executed one after another. Existing algorithms process the entire video as the input and then display the output, which is not real time at all.

This brings us to the second phase of development, which is to make the algorithm real time as shown in Fig. 11.

6.1 Multi-threaded approach

The real-time implementation consists of three threads as shown in Fig. 11, which are capable of running independently of each other. Parallel computing is used to separate the video stabilization algorithm into three parts: thread 1 (motion estimation), thread 2 (motion compensation) and thread 3 (warping).

Using the first thread, motion estimation is carried out for the first twenty incoming frames (smoothing radius for motion compensation is set to ten) while the other threads wait for this process to complete. This delay is because the motion compensation and warping require smoothed parameter trajectories that can be generated only after the first twenty frames are processed. From the 21st frame, all the three threads operate simultaneously. In other words, the motion estimation between the 20th and 21st frame is obtained using thread 1, while the other threads compute the smoothed trajectory using the information generated from the second frame to the 21st frame. The whole process continues indefinitely until the video stream ends.

Since the threads themselves process the frames much faster than the frame rate, the timings of the threads eventually catch up with the timings of the frame availability.

6.1.1 Motion estimation thread

Once the motion estimation (ME) thread is able to open a video stream, it will keep estimating the homography matrix between each frame until it reaches the end of the video stream. The motion compensation thread will be notified to run as soon as ME thread finishes estimating for twenty frames (dwell time).

6.1.2 Motion compensation thread

The motion compensation (MC) thread averages the transform parameters within the storage for the transform parameters. From then on, whenever a new homography is estimated by the ME thread, the MC thread will remove the oldest transform parameters and insert the latest parameters and then perform smoothing. It is to be reminded that the motion compensation stage has not changed from the previous implementation. However, the stage no longer has the whole videos parameters to work with anymore. Instead, it collects and manages the data storage of the transform parameters given by ME thread dynamically.

6.1.3 Image composition thread

Once motion compensation produces the smoothed parameters of the latest frame, it will insert the data into the storage so that the image composition thread is able to retrieve the data to display or store accordingly.

7 Results

7.1 Results of Stage I

We tested the enhanced corner tracking and hybrid motion estimation system on the video obtained from in-house UAV videos and on standard benchmark videos [5]. Figure 12 shows a screenshot of the comparison between the unstable and the stabilized video on the Forest video from the public database.

A web demo for the forest video is shared below:

<http://tinyurl.com/jv6fvvu>.

Compare the above result to the result obtained using rigid and similarity transformation alone:

<http://tinyurl.com/gnploac> (rigid).

<http://tinyurl.com/glxt65m> (similarity).



Fig. 12 Before and after stabilization

It is evident that the hybrid motion estimation is much better in comparison to the simple rigid or similarity transformation. This can be clearly witnessed after 20 s into the video when there are sudden camera movements. The web demos of the other videos in the public database can be found below.

1. Ground: <http://tinyurl.com/zfb4lo6>.
2. Bird: <http://tinyurl.com/jsn7g24>.
3. Girl: <http://tinyurl.com/hodl9ey>.
4. Hippo: <http://tinyurl.com/zrq8jna>.

The above results were compiled on a desktop with a CPU clock speed of 1.7 GHz using the same set of parameters obtained from tuning using the in-house UAV video. The videos tested are aerial view videos taken from a UAV to simulate the case when a real UAV is used for surveillance. The Stage I algorithm is able to estimate the motion matrix of adjacent frames at a maximum speed of 100 fps. The same algorithm is used to estimate the motion matrix for a frame size of 640×480 at a speed of 80 fps. This kind of result reassures us that this algorithm is very much suited for real-time processing.

Table 1 shows the results of how many times rigid and affine transformations have been utilized in different videos. It is clear that the hybrid motion estimation uses both rigid and similarity transformations in a manner dependent on the nature of the motion. For motion behavior that is not easy to be captured by the simple rigid mechanism, the similarity transformation offers a better model for motion behavior.

Using video frames obtained from in-house UAV videos, we tested the hybrid motion estimation-based

Table 1 Number of times different motion models are used by the hybrid motion model

Video	Number of rigid transformations (per 20 frames)	Number of similarity transformation (per 20 frames)
Forest	41	41
Ground	16	32
Bird	21	9
Girl	12	10

stabilization algorithm. The captured video frames were obtained by a UAV that shows the motion of another UAV. The video frames suffer from both object motion and scene motion (due to the onboard cam motion). Below is a demo of our system.

<http://tinyurl.com/z8muwab>.

From the above results, we can see that after the first few seconds, our method can better stabilize the unstable frames, especially when seen from the point of view of the UAV in the picture.

Furthermore, we observe that although scene change happens abruptly, our system can handle them without crashing. This is a common scenario when mechanical gimbals do not provide much stability to the captured UAV videos or when there are frame rate issues or when there is a sudden mid-flight recourse taken by the UAV. All these scenarios were handled efficiently by our system, as demonstrated in the comparison video above. Below is the link to our code that was used for generating the above results:

<http://tinyurl.com/jn7dw9g>.

7.2 Motion estimation timings

In order to ensure that our algorithm has the potential to execute in real time, we time the duration needed to finish estimating the motion model for the publicly available videos. Most of the algorithms reviewed in the literature are either not applicable for real-time processing or needs more processing time than the proposed algorithm in this paper.

Table 2 compares the processing speed of the proposed algorithm and the latest instantaneous video stabilization method in the literature. For a fair comparison, we use a notebook computer with a dual-core 1.70 GHz processor, as done in Dong et al. [5], using a notebook computer with a 2.5 GHz Intel dual-core CPU. In other words, our speed comparison with Dong et al. [5] is valid, since we show better or similar computational time using a notebook with lower processing speed. Motion estimation step is the most expensive step in video stabilization. We are able to utilize optical flow-based tracking to reduce the computational time.

Comparing Tables 1 and 2, we can see that the publicly available Bird video uses more rigid than partial affine (21–9), therefore has higher processing speed (44 fps) compared to the Ground video that uses much less rigid than partial affine (16–32) and has lower processing speed (34 fps). Both the videos are of the same resolution, and since we set the number of features to be around 50, the difference in processing speed is due to the Ground video's requirement of higher-order homography.

Table 2 Time taken to process entire video

Video name	Resolution	Frames	Dong et al. [5] (frames per second)	Offline method (frames per second)
Forest	320 × 240	1650	77.2	107.8
Ground	640 × 480	966	39.9	34.9
Bird	640 × 360	601	47.4	44.2
Girl	640 × 360	446	42.7	45.5

Moreover, from Table 2, we can infer that the lower the resolution of the video, the faster it is to calculate the motion model between the frames. Furthermore, we speed up the motion estimation by using tracking which effectively removes the need to do iterative detection of corners for each frame.

7.3 Results of real-time implementation

Multi-threaded implementation is possible due to the high speed of the proposed motion estimation compared to existing works. Experiments were conducted to estimate the speed at each stage. The result was that motion compensation and image composition step combined takes much shorter time than motion estimation (ME). Hence, it is concluded that ME is the bottleneck of the operation. Since through experimental results, ME can reach up to 80 fps on a 640 × 480p video, this implementation can allow the program to process frames in real time.

However, it has to be kept in mind that the threads execute under the same program explained in Sect. 6. This means that they are able to share resources or data with one another. This feature is perfect for the proposed algorithm because motion compensation needs to make use of the output (transform parameters) from the ME process and image composition needs to make use of the output from motion compensation as seen from Fig. 11.

This feature can serve as a convenient way to read and write to the same resource in the memory space. To avoid data corruption, the data accessibility need to be handled carefully. In this implementation, mutex is used to facilitate resource accessibility. A demo of the in-house UAV can be viewed using the link below.

<http://tinyurl.com/hel53sb>.

Furthermore, with the usage of multi-threading processing, the time involved to compute and smooth the

parameter trajectory is reduced significantly, as shown in Table 3. This is because once the parameters are calculated, the motion compensation thread is able to smooth the parameter values immediately without having to wait for the computation of the parameter trajectory of the entire video to be extracted.

The links below can be used to access the final video results obtained from our algorithm on publicly available videos.

1. Ground: <http://tinyurl.com/zfb4lo6>.
2. Bird: <http://tinyurl.com/jsn7g24>.
3. Girl: <http://tinyurl.com/hodl9ey>.
4. Hippo: <http://tinyurl.com/zrq8jna>.

7.4 Comparison to existing works

Since the main focus of the paper is real-time video stabilization for unmanned aerial vehicles, the comparison in Table 2 was done with only Dong et al. [5], which is the only related work in terms of the performance speed. Moreover, our stabilization quality is better or on par with Dong et al. [5] for all the reported videos. Note that other related methods mentioned in Sect. 1 have processing speeds less than 30 fps. Nevertheless, we still compare our method to some of the slower methods in terms of stabilization quality.

Wang et al. [16] can process UAV videos up to 30 fps. Their result is given below:

<https://goo.gl/NrvfKt>.

From the above result, we can notice obvious jitters in the stabilized video, whereas the proposed algorithm does a better job:

<http://tinyurl.com/zu6748e>.

Next, we compare our work to general video stabilization algorithms that are not limited to UAV videos.

Table 3 Comparison between offline and real time

Video name	Resolution	Frames	Offline method (frames per second)	Real-time version (frames per second)
Forest	320 × 240	1650	107.8	126.32
Ground	640 × 480	966	34.9	58.93
Bird	640 × 360	601	44.2	72.65
Girl	640 × 360	446	45.5	65.36

Grundmann et al. [7] performs video stabilization using L1-optimal camera paths with many optional visual constraints. Their results can be viewed at:

<https://goo.gl/5a3kk1>.

Our results can be viewed at:

<http://tinyurl.com/jmouy2l>.

With no additional constraints like face constraints, blur constraints or wobble removal, our method gives comparable results for most videos. Naturally, with additional constraints for very shaky videos, stabilization can be better using our algorithm.

Finally, we compare our work to Liu et al. [10] using their Supplemental Video Set 1, whose results can be viewed using the link below:

<https://goo.gl/haxC6N>.

Our result:

<http://tinyurl.com/jld8hlk>.

Generally, the proposed stabilization algorithm performs similar to state-of-the-art methods, but with a much higher processing speed.

8 Conclusion

This paper describes the algorithms and methods used to tackle the problem of real-time video stabilization for unmanned aerial vehicles (UAVs). Due to the size and structure limitations, UAVs are highly susceptible to atmospheric turbulence, which induces jitters and makes the video unstable. It is very difficult to detect and track targets of interest in such unstable videos. Therefore, it is necessary to design a real-time video stabilization algorithm for UAVs as a preprocessing module for higher-level vision tasks, such as object detection and tracking.

To achieve the above goal, two main components were designed as part of the proposed video stabilization algorithm: (1) by designing an appropriate motion model for the global motion of the UAVs, the proposed stabilization mechanism avoids the necessity of estimating the most general motion model, projective transformation, and considers simpler motion models like the similarity and rigid transformation; (2) in order to achieve high processing speeds, an optical flow-based motion estimation is proposed to replace the conventional tracking and matching algorithms used by state-of-the-art video stabilization methods. These novel ideas resulted in a real-time stabilization algorithm. A demo of the stabilized videos can be accessed using the web links provided in this paper.

Our method will not yield excellent results for videos that has large and sudden scene changes, which is problematic for all stabilization frameworks. This will cause the

frames to be warped in such a way to resist the changes resulting in large unfilled areas in the frames. Future work will be carried out to resolve this issue.

References

- Adelson, E.H., Anderson, C.H., Bergen, J.R., Burt, P.J., Ogden, J.M.: Pyramid methods in image processing. *RCA Eng.* **29**(6), 33–41 (1984)
- Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), IEEE, pp. 459–468 (2006)
- Battiatto, S., Gallo, G., Puglisi, G., Scellato, S.: Sift features tracking for video stabilization. In: *Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on*, IEEE, pp. 825–830 (2007)
- Bouguet, J.Y.: Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm. *Intel Corp.* **5**(1–10), 4 (2001)
- Dong, J., Xia, Y., Yu, Q., Su, A., Hou, W.: Instantaneous video stabilization for unmanned aerial vehicles. *J. Electron. Imaging* **23**(1), 013002 (2014)
- Fleet, D., Weiss, Y.: Optical flow estimation. In: Paragios, N., Chen, Y., Faugeras, O. (eds.) *Handbook of Mathematical Models in Computer Vision*, pp. 237–257. Springer, Berlin (2006)
- Grundmann, M., Kwatra, V., Essa, I.: Auto-directed video stabilization with robust l1 optimal camera paths. *CVPR* **2011**, 225–232 (2011)
- Harris, C., Stephens, M.: A combined corner and edge detector. In: *Alvey Vision Conference*, Citeseer, vol. 15, p. 50 (1988)
- Ho, N. (2014) Simple video stabilization using opencv. <http://nghiaho.com/?p=2093>
- Liu, F., Gleicher, M., Wang, J., Jin, H., Agarwala, A.: Subspace video stabilization. *ACM Trans. Graph.* **30**(1), 4:1–4:10 (2011)
- Lucas, B.D., Kanade, T., et al.: An iterative image registration technique with an application to stereo vision. *IJCAI* **81**, 674–679 (1981)
- Pilu, M.: Video stabilization as a variational problem and numerical solution with the Viterbi method. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2004)
- Shen, H., Pan, Q., Cheng, Y., Yu, Y.: Fast video stabilization algorithm for UAV. In: *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, IEEE, vol. 4, pp. 542–546 (2009)
- Shi, J., Tomasi, C.: Good features to track. In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, IEEE, pp. 593–600 (1994)
- Vazquez, M., Chang, C.: Real-time video smoothing for small rc helicopters. In: *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, IEEE, pp. 4019–4024 (2009)
- Wang, Y., Hou, Z., Leman, K., Chang, R.: Real-time video stabilization for unmanned aerial vehicles. In: *MVA*, pp. 336–339 (2011)
- Yang, J., Schonfeld, D., Chen, C., Mohamed, M.: Online video stabilization based on particle filters. In: *2006 International Conference on Image Processing*, IEEE, pp. 1545–1548 (2006)