Real-Time Video Stabilization for Unmanned Aerial Vehicles using Modified MeshFlow

Muhammad Khairul Makirin 13517088 (*Author*)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: khairul240999@gmail.com

***Abstract*—This paper propose a new framework for real-time video stabilization on-board of an unmanned aerial vehicle using a combination of method from MeshFlow motion model with an optical flow-based tracker for motion estimation. This framework resulted in a promising new method for hybrid video stabilization algorithm.**

Keywords—video stabilization, UAV, MeshFlow, real-time video stabilization.

I. Introduction

Video stabilization is a process to dampen or remove unwanted jitter and motion from a video. A common use for this is to lessen the effect of mechanical vibration or handshake from a video to produce a clearer output, this output can be then processed for video/image analysis. There are four main techniques of video stabilization, which is optical stabilization, electronic stabilization, mechanical stabilization, and digital stabilization [1]. Digital stabilization only uses information from the given video and does not need any additional sensors or hardware, therefore it is the cheapest option currently available. Commonly there are three main steps for video stabilization algorithm: (1) motion estimation, (2) motion smoothing/compensation, and (3) image synthesis/warping [1].

As the name implies, motion estimation is the process of estimating camera motion (global motion) through analysis of adjacent frames. There are two main approach of motion estimation, which is feature-based and direct pixel-based, feature-based are generally more computationally efficient. Motion smoothing or compensation are for removing high frequency jitters and motion that are typically caused by vibration or handshake. Most technique for motion smoothing usually involves a lowpass filter (e.g., kalman filter, gaussian filter, or median filter). The last step, image synthesis/warping, is the process of producing the stabilized frame using motion parameter of the previous step. This can be achieved through finding an interframe affine transformation from the motion parameter.

Currently most existing digital video stabilization are performed off-line or as a post-processing phase after the whole video has been recorded. This could hinder the performance for real-time on-board video processing in such environment. UAVs (Unmanned Aerial Vehicles) are a common example for using real-time application, such as rescue, surveillance, mapping, and object tracking. The main drawback of using UAVs for real-time image/video analysis is limited processing ability, often UAVs are not equipped with the same processing module found in consumer hardware. This will limit the efficiency of real-time application on-board the UAVs.

The proposed method in this paper is combining the on-line, but not actually real-time, method of MeshFlow [2] for motion compensation and real-time optical flow-based video stabilization method for UAVs [3] for motion estimation and image synthesis.

II. Feature Tracking

Feature is a region of special interest in an image that can be uniquely identified by an algorithm, feature could represent a corner, an edge, or a pixel that have contrasting intensity between its neighboring pixels. Whereas a feature tracker, is used to track the movement of features along the frames of a video and producing a feature trajectory along a given timeframe. The method used in MeshFlow [2] for motion estimation involves feature detection and tracking, specifically they used FAST (Features from Accelerated Segment Test) corner detection [4] for feature detection and KLT (Kanade-Lucas-Tomasi) tracker [5] for tracking previously detected features.

A. FAST Corner Detection

The FAST algorithm was first proposed by Edward Rostein and Tom Drummond to address computational efficiency for feature detection. The main idea is very simple, given a corner candidate p and its intensity I_p, check if there is n contiguous pixel along the Bresenham circle path centered at p that all have darker intensity than $I_p - t$ or all have brighter intensity than $I_p + t$, with t as a user defined threshold, a visualization can be seen in Figure 1. Later, a high-speed test was proposed to exclude a large number of pixels from n, the test examines only four pixels at 1, 5, 9, and 13 and check if at least three of them have brighter or darker intensity than $I_p + t$ and $I_p - t$ respectively. If neither of these is the case, then p is not considered as a corner.

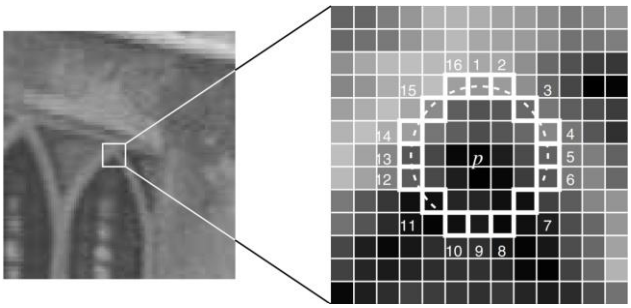

Fig. 1. A 16 contiguous pixels centered around *p* as the candidate corner. The arc, indicated by the dashed line, is the 12 contiguous pixels that all have intensity higher than $I_p + t$.

This method alone can produce high quality corners, but there are still some drawbacks:

1. High speed test does not generalize well for $n < 12$.
2. The choice of pixels is not optimal because it depends on an implicit assumption about the distribution of feature appearance.
3. Result of high-speed test is thrown away.
4. Multiple features are detected adjacent to one another.

In [4] point 1 – 3 are addressed using an ID3, a decision tree classifier, whereas point 4 is solved using non-maximal suppression.

B. KLT Tracker

As the name implies, the KLT tracker is used to track feature points resulted from FAST corner detection. [4] uses a tracking algorithm rather than detecting and matching over and over to limit computation cost of another feature matching algorithm. In general, the KLT tracker works as follow, given a feature point location *a* (in (x, y) format) at time *t* and its intensity I_a, search a displacement *d* that satisfy,

$$I(a + d, t + \tau) = I_a \tag{1}$$

and minimized the sum-of-square differences between adjacent frames [6]. A big difference of KLT to other tracking algorithm is that it is based on optical flow analysis. By utilizing optical flow vector, it can reduce propagated error in the long run, thus saving more computational cost of a re-detection.

III. Motion Estimation

Motion estimation is a process where we estimate a transformation matrix that is responsible for the movement of a feature point/plane. Many of the works on video stabilization for UAVs uses affine transformation, partial affine transformation [7], rigid (Euclidean) transformation, or a combination of them [3].

Affine transformation is a way to describe geometric transformation between two images. Suppose *P'(x', y')* is the point of *P(x, y)* in the next frame, we can compute the transformation from *P(x, y)* to *P'(x', y')* by solving for *A* in Equation 2.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = A \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

With full affine transformation, *A* has 6 degrees of freedom (rotation, translation, scaling, and sheering), whereas a partial affine only has 4 degrees of freedom (rotation, translation in x-axis, translation in y-axis, and uniform scaling). In full affine transformation, we define *A* as such,

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \tag{3}$$

expanding Equation 2 with Equation 3 gives us two new equations,

$$x' = ax + by + c \tag{4}$$
$$y' = dx + ey + f \tag{5}$$

equation 4 and 5 can be re-written in the form of $A\mathbf{z} = b$ with two additional points as such,

$$\begin{bmatrix} x1 & y1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x1 & y1 & 1 \\ x2 & y2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x2 & y2 & 1 \\ x3 & y3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x3 & y3 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x1' \\ y1' \\ x2' \\ y2' \\ x3' \\ y3' \end{bmatrix} \tag{6}$$

thus, solving for $\mathbf{z}$ with a linear solver will give us *A*, our affine transformation. For partial affine transformation, we replace *A* with a definition using rotation, scaling, and translation matrix as in Equation 7. Partial affine is preferred over full affine transformation in real-time video stabilization because of its lower computational cost relative to full affine transformation.

$$A = \begin{bmatrix} \cos(\theta)\, s & -\sin(\theta)\, s & tx \\ \sin(\theta)\, s & \cos(\theta)\, s & ty \end{bmatrix} \tag{7}$$

IV. MeshFlow

MeshFlow is a motion model proposed by S. Liu et al. that uses a grid mesh vertices layered over a frame to estimate global motion while smoothing it in the process, in [2] they fitted a uniform grid mesh of 16 × 16 onto each frame. There are four main process to calculate a global motion: (1) feature tracking, (2) motion propagation, (3) motion filtering, and (4) vertex profiles generation.

A. Feature Tracking

In this process, we detect and track each feature point motion frame-by-frame using the FAST and KLT tracker. We then compute the motion vector V_f by subtracting feature point position *p'* at time *t* with its previous position *p* at time *t* – 1.

$$V_f = p' - p \quad (8)$$

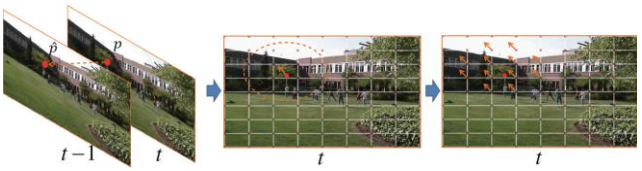

Fig. 2. Motion propagation of motion vector, indicated by the orange arrow in the middle image, to nearby mesh vertices, indicated by the orange arrow in the right image, covered by an eclipse of 3 × 3 cells centered at the feature point *p*.

B. Motion Propagation

After we have a motion vector for each feature point, we then propagate that motion vector to all nearby mesh vertices. Nearby mesh vertices are those that are covered by an eclipse of 3 × 3 cells centered at a feature point. A visualization of this can be seen in Figure 2.

C. Motion Filtering

After motion propagation, each mesh vertices could receive more than one or zero motion vector, therefore, in [2] they proposed to use a median filter f_1 to filter candidate motion vector and assigned the filter response to the vertex. A visualization of this can be seen in Figure 3.

After applying median filter f_1 to every mesh vertices, we then obtain a sparse motion field as illustrated on the left image in Figure 4. This sparse motion field could also retain some amount of noise, thus we apply a convolution using median filter f_2 for all the mesh vertices which then produce a smoother sparse motion field.

D. Vertex Profiles Generation

A vertex profile is a one-dimensional array consisting of all motion vector that has been assigned to a vertex over all the frames.

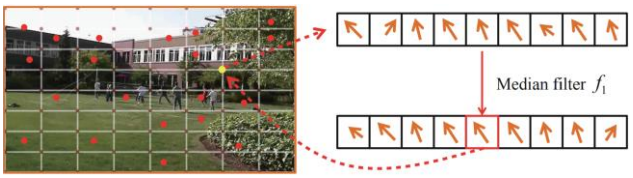

Fig. 3. Mesh vertex, indicated by the yellow dot, receive multiple motion vector from nearby feature point, indicated by orange dots. We then apply a median filter f_1 to the candidates and assigned the filter response as the mesh vertex motion vector.

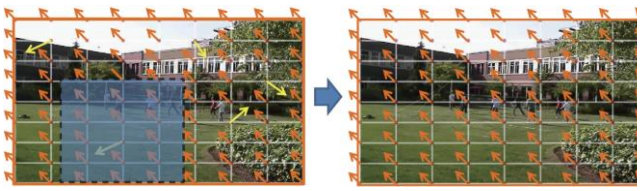

Fig. 4. The smoothing of sparse motion field on the left with median filter f_2 convolution to enforce spatial smoothness.

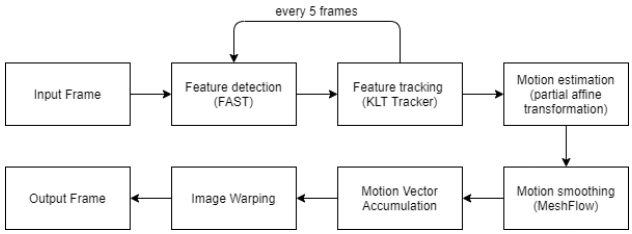

Fig. 5. Process diagram of the proposed framework.

V. Proposed Framework

By using partial affine transformation to estimate motion vector and MeshFlow model to generate a smooth sparse motion field, we propose a new real-time video stabilization framework as shown in Figure 5. We use a 4 × 4 mesh grid for vertex profile generation.

Firstly, the input frame is converted to grayscale and downscaled to 640 × 480 pixels, then it is processed by FAST corner detection to produce a set of feature points *F*, we use a threshold for FAST of 40, as without this, there will be too much feature point to compute. Then, in the next frame we tracked each feature point to obtain the latest position of *F* with KLT, to reduce accumulated error by KLT we re-detect a new set of feature point *F'* for every 5 frames. The re-detection also serves as a failsafe measure for very dynamic motions such as abrupt panning/rotating.

After tracking *F* in subsequent frames, we estimate its motion by using partial affine transformation, this process will produce a vector of *dx*, *dy*, and *da* for x-axis displacement, y-axis displacement, and angle displacement respectively. We then propagate this vector to nearby mesh vertices using MeshFlow, where in the process creates a smooth sparse motion field to be used as an estimate of global motion. Vertex profile will then be generated, we use the vertex profile to warp the current frame according to its motion vector, thus resulting in a stabilized output frame.

We implemented this framework using OpenCV 4.5.2 in a Python 3.9 and Windows 10 environment, the source code for this paper can be downloaded from here[1].

VI. Experiements and Conclusion

For experimenting we use a dataset created by Jing et al [8] of aerial videos. In those videos we analyzed how does our framework perform with real shaky movement of the aircraft, qualitatively. In total we processed 3 video and achieved promising result. In all of three videos, our framework manages to stabilize erratic movement of the aircraft and produce a clearer output than before. Video result of before and after can be seen in the next section.

[1]https://gitlab.informatika.org/tugas-makalah-if4073-pengolahan-citra/real-time-video-stabilization-for-unmanned-aerial-vehicles-using-modified-meshflow

VIDEO LINK AT GOOGLE DRIVE

1. https://drive.google.com/file/d/1UxN5BOXXKY8Y5EvW6En_7-omt4sDYsnm/view?usp=sharing
2. https://drive.google.com/file/d/1iZZMswXDBOviJtVBgN7Dt8ywefJan6I4/view?usp=sharing
3. https://drive.google.com/file/d/1pl7uHwEho0c9N9QclyfGt8hHt5eahnk6/view?usp=sharing

ACKNOWLEDGMENT

The author like to thank his family as without their support and motivation, this work will never be finished. The author would also like to thank Mr. Rinaldi Munir as the author's lecturer for IF4073 Pengolahan Citra, IF4073 class assistant, and all his friends for always bringing new challenges and help the author be better. Thank you.

REFERENCES

[1] P. Amisha and M.H. Vala, "A survey of video stabilization techniques," in Journal of IJESRT, February 2015.

[2] S. Liu, P. Tan, L. Yuan, J. Sun, and B. Zeng, "MeshFlow: minimum latency online video stabilization," in proc. European Conference on Compter Vision (ECCV), 2016.

[3] B. Ramesh, A. Lim, Y. Yang, C. Xiang, Z. Gao, F. Lin, "Real-time optical flow-based video stabilization for unmanned aerial vehicles," in Journal of Real-Time Image Processing, vol. 16, pp. 1975-1985, 2019.

[4] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in proc. ECCV, pp. 430-443, 2006.

[5] J. Shi and C. Tomasi, "Good features to track," in proc. CVPR, pp. 593-600, 1994.

[6] M.S. Sri, "Object detection and tracking using KLT algorithm," in Journal of IJEDR, vol. 7, no. 2, 2019.

[7] Y. Wang, Z. Hou, K. Leman, and R. Chang, "Real-Time Video Stabilization for Unmanned Aerial Vehicles," in proc. IAPR Conference on Machine Vision Applications, pp. 336-339, June 2011.

[8] J. Li, D.H. Ye, T. Chung, M. Kolsch, J. Wachs, and C. Bouman, "Multi-target detection and tracking from a single camera in unamanned aerial vehicles (uavs)," in proc. IROS, October 2016.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Depok, 25 Mei 2021

M. Khairul Makirin (13517088)