# Video Stabilization for Camera Shoot in Mobile Devices via Inertial-Visual State Tracking

Fei Han, *Student Member, IEEE,* Lei Xie, *Member, IEEE,* Yafeng Yin, *Member, IEEE,* Hao Zhang, *Student Member, IEEE,* Guihai Chen, *Member, IEEE,* and Sanglu Lu, *Member, IEEE*

**Abstract**—Due to the sudden movement during the camera shoot, the videos retrieved from the hand-held mobile devices often suffer from undesired frame jitters, leading to the loss of video quality. In this paper, we present a video stabilization solution in mobile devices via inertial-visual state tracking. Specifically, during the video shoot, we use the gyroscope to estimate the *rotation* of camera, and use the structure-from-motion among the image frames to estimate the *translation* of camera. We build a camera projection model by considering the rotation and translation of the camera, and the camera motion model to depict the relationship between the inertial-visual state and the camera's 3D motion. By fusing the inertial measurement (IMU)-based method and the computer vision (CV)-based method, our solution is robust to the fast movement and violent jitters, moreover, it greatly reduces the computation overhead in video stabilization. In comparison to the IMU-based solution, our solution can estimate the translation in a more accurate manner, since we use the feature point pairs in adjacent image frames, rather than the error-prone accelerometers, to estimate the translation. In comparison to the CV-based solution, our solution can estimate the translation with less number of feature point pairs, since the number of undetermined degrees of freedom in the 3D motion directly reduces from 6 to 3. We implemented a prototype system on smart glasses and smart phones, and evaluated the performance under real scenarios, i.e., the human subjects used mobile devices to shoot videos while they were walking, climbing or riding. The experiment results show that our solution achieves 32% better performance than the state-of-art solutions in regard to video stabilization. Moreover, the average processing time latency is 32.6ms, which is lower than the conventional inter-frame time interval, i.e., 33ms, and thus meets the real-time requirement for online processing.

**Index Terms**—Video Stabilization, Mobile Device, 3D Motion Sensing, Inertial-Visual State Tracking

## 1 Introduction

Due to the proliferation of mobile devices, nowadays more and more people tend to use their mobile devices to take videos. Such devices can be smart phones and smart glasses. However, due to the sudden movement from the users during the camera shoot, the videos retrieved from such mobile devices often suffer from undesired frame jitters. This usually leads to the loss of video quality. Therefore, a number of video stabilization techniques are proposed to remove the undesired jitters and obtain stable videos [1], [2], [3], [4], [5], [6], [7]. Recently, by leveraging the embedded sensors, new opportunities have been raised to perform video stabilization in the mobile devices. For the mobile devices, conventional video stabilization schemes involves estimating the motion of the camera, smoothing the camera's motion to remove the undesired jitters, and warping the frames to stabilize the videos. Among these procedures, it is especially important to accurately estimate the camera's motion during the camera shoot, since it is a key precondition for the following jitters removal and frame warping.

Conventionally, the motion estimation of the camera in 3D space is either based on the inertial measurement-based techniques [8], [9], [10] or the computer vision-based techniques [3], [4]. The inertial measurement-based approaches mainly use the built-in inertial measurement unit (IMU) to continuously track the 3D motion of the mobile device. However, they mainly focus on the *rotation* while ignoring the *translation* of the camera. The reason is two folds: First, the gyroscope in the IMU is usually able to accurately track the rotation, whereas the accelerometer in the IMU usually fails to accurately track the translation due to the large cumulative tracking errors. The computer vision (CV)-based approaches mainly use the structure-from-motion [11] among the image frames to estimate both the rotation and translation of the camera. Although they achieve enough accuracy for the camera motion estimation, they require plenty of feature point pairs and long feature point tracks. The requirement of massive feature points for motion estimation increases the computational overhead in the resource-constrained mobile devices. This makes the real-time processing impractical in the mobile devices. Hence, to achieve a tradeoff between performance and computation overhead, only rotation estimation is considered for the state-of-the-art solutions. Second, according to our empirical studies, when the target is at a distance greater than 100cm, the rotation usually brings greater pixel jitters than the translation, hence, most previous work consider the rotation has a greater impact on performance than the translation. However, when the target is within a close range, e.g., at the distance less than 100cm, the translation usually brings greater pixel jitters than the rotation, thus the translation tracking is also very essential for real applications of camera shooting. Therefore, to efficiently perform video

- *Fei Han, Lei Xie, Yafeng Yin, Hao Zhang, Guihai Chen and Sanglu Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, China.*
  *E-mail: feihan@smail.nju.edu.cn, {lxie,yafeng}@nju.edu.cn, H.Zhang@smail.nju.edu.cn, {gchen,sanglu}@nju.edu.cn.*
- *Lei Xie is the corresponding author.*

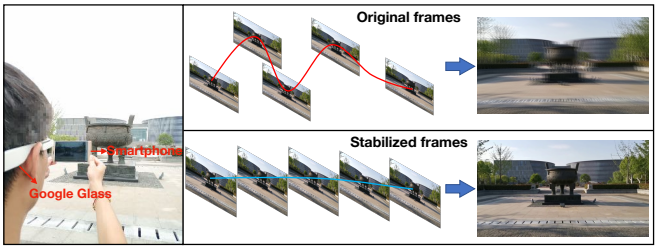



Fig. 1. Video Stabilization in Mobile Devices. Videos captured with mobile devices often suffer from undesired frame jitters due to the sudden movement from the users. We first estimate the original camera path (red) via inertial-visual state tracking, then smooth the original camera path to obtain the smoothed camera path (blue), and finally obtain the stabilized frames by warping the original frames.

stabilization in mobile devices, it is essential to fuse the CV-based and IMU-based approaches to accurately estimate the camera's 3D motion, including the *rotation* and *translation*.

In this paper, we propose a video stabilization scheme for camera shoot in mobile devices, based on the visual and inertial state tracking. Our approach is able to accurately estimate the camera's 3D motion by sufficiently fusing both the CV-based and IMU-based methods. Specifically, during the process of video shoot, we use the gyroscope to estimate the *rotation* of camera, and use the structure-from-motion among the image frames to estimate the *translation* of the camera. Different from the pure CV-based approaches, which estimate the *rotation* and *translation* simultaneously according to the camera projection model, our solution first estimates the *rotation* based on the gyroscope measurement, and plugs the estimated *rotation* into the camera projection model, then we estimate the *translation* according to the camera projection model. In comparison to the CV-based solution, our solution can estimate the *translation* in a more accurate manner with less number of feature point pairs, since the number of undetermined degrees of freedom in the 3D motion directly reduces from 6 to 3. After that, we further smooth the camera's motion to remove the undesired jitters during the moving process. As shown in Fig.1, according to the mapping relationship between the original moving path and the smoothed moving path, we warp each pixel from the original frame into a corresponding pixel in the stabilized frame. In this way, the stabilized video appears to have been captured along the smoothed moving path of the camera. In the context of recent visual-inertial based video stabilization methods [12], [13], our solution is able to estimate the translation and rotation in a more accurate manner, and meets the real time requirement for online processing, by directly reducing the number of undetermined degrees of freedom from 6 to 3 for CV-based processing.

There are two key challenges to address in this paper. *The first challenge is to accurately estimate and effectively smooth the camera's 3D motion in the situation of fast movement and violent jitters, due to the sudden movement during the video shoot*. To address this challenge, firstly, we use the gyroscope to perform the rotation estimation to figure out a $3 \times 3$ rotation matrix, since it can accurately estimate the rotation even if the fast movement and violent jitters occur. Then, to smooth the rotation, instead of smoothing the 9 dependent parameters separately, we further transform the $3 \times 3$ rotation matrix into the $1 \times 3$ Euler angles, and apply the low pass filter over the 3 independent Euler angles separately. In this way, we are able to effectively smooth the rotation while maintaining the consistency among multiple parameters. Secondly, we build a camera projection model by considering the rotation and translation of the camera. Then, by substituting the estimated rotation into the camera projection model, we directly estimate the translation according to the matched feature point pairs in adjacent image frames. For the situation of fast movement and violent jitters, it is usually difficult to find enough feature point pairs between adjacent image frames to estimate the camera's 3D motion. In comparison to the traditional CV-based approaches, our solution requires less number of feature point pairs, as we directly reduce the number of undetermined degrees of freedom in the 3D motion from 6 to 3. *The second challenge is to sufficiently reduce the computation overhead of video stabilization, so as to make the real-time processing practical in the resource-constrained mobile devices*. For traditional CV-based approaches, they usually require at least 5$\sim$8 pairs of feature points to estimate the rotation and translation. They involve 6 degrees of freedom, thus they usually incur large computation overhead, failing to perform the video stabilization in a real-time manner. To address this challenge, our solution reduces the computation overhead by directly reducing the undetermined degrees of freedom from 6 to 3. Specifically, we use the inertial measurements to estimate the rotation. Our solution only requires at least 3 pairs of feature points to estimate the translation, which reduces over 50% of the burden in the CV-based processing. This makes the real-time processing possible in the mobile devices.

We make three key contributions in this paper. 1) We investigate video stabilization for camera shoot in mobile devices. By fusing the IMU-based method and the CV-based method, our solution is robust to the fast movement and violent jitters, and greatly reduces the computation overhead in video stabilization. 2) We conduct empirical studies to investigate the impact of movement jitters, and the measurement errors in IMU-based approaches. We build a camera projection model by considering the rotation and translation of the camera. We further build the camera motion model to depict the relationship between the inertial-visual state and the camera's 3D motion. 3) We implemented a prototype system on smart glasses and smart phones, and evaluated the performance under real scenarios, i.e., the human subjects used mobile devices to shoot videos while they were walking, climbing or riding. The experiment results show that our solution achieves 32% better performance than the state-of-art solutions in regard to video stabilization. Moreover, the average processing time latency is 32.6ms, which is lower than the conventional inter-frame time interval, i.e., 33ms, and thus meets the real-time requirement for online processing.

## 2 Related Work

**CV-based Solution**: Traditional CV-based solutions for video stabilization can be roughly divided into 2D stabilization and 3D stabilization. 2D video stabilization solutions use a series of 2D transformations between adjacent frames to represent the camera motion, and smooth these transformations to stabilize the video [1], [2], [14]. However, these methods cannot figure out the camera's 3D motion, thus

they usually fail to compute the changes of projection for the target scene when there exist significant depth changes. Recent 3D video stabilization solutions [3], [4], [15] all seek to stabilize the videos based on the 3D camera motion model. They use the structure-from-motion among the image frames to estimate the 3D camera motion, thus they can deal with parallax distortions caused by depth variations. Hence, they are usually more effective and robust in video stabilization, at the cost of large computation overhead. Therefore, they are usually performed in an offline manner for video stabilization. Moreover, when the camera moves fast or experiences violent jitters, they may not find enough amount of feature points to estimate the motion.

**IMU-based Solution**: For mobile devices, since the built-in gyroscopes and accelerometers can be directly used to estimate the camera's motion, the IMU-based solutions [8], [9], [16], [17] are proposed for video stabilization recently. For video stabilization, Karpenko et al. calculate the camera's rotation by integrating the gyroscope readings directly [8], whereas Hanning et al. take into account the noise of the gyroscope readings, they estimate the camera's rotation with an extended Kalman filter to fuse the readings of gyroscope and accelerometer [9]. These IMU-based solutions are much faster than the CV-based solutions, but they only consider the rotation in modeling the camera motion without the translation, since the gyroscope can accurately track the rotation, whereas the accelerometer usually fail to accurately track the translation due to large cumulative tracking errors.

**Hybrid Solution**: Recent work seek to fuse the inertial and visual-based methods to track the camera's motion [18], [19], [20]. Yang et al. fuse the visual and inertial measurements to track the camera state for augmented reality [19]. In video stabilization, Jia et al. propose an EKF-based method to estimate the 3D camera rotation by using both the video and inertial measurements [20]. Still, they only use the pure rotation to depict the camera motion and ignore the camera's translation. In this paper, we investigate video stabilization in mobile devices, by accurately estimating and smoothing the camera's 3D motion, i.e., camera rotation and translation. By fusing the IMU-based method and the CV-based method, our solution is robust to the fast movement and violent jitters, moreover, it greatly reduces the computation overhead in video stabilization. In the context of recent visual-inertial based video stabilization methods [12], [13], our solution is able to estimate the translation and rotation in a more accurate manner, and meets the real time requirement for online processing, by directly reducing the number of undetermined degrees of freedom from 6 to 3 for CV-based processing.

## 3 Preliminary

To illustrate the principle of camera shoot in the mobile devices, we can use the *pinhole camera model* [11] to depict the camera projection. As illustrated in Fig. 2, for an arbitrary point **P** from the specified object in the scene, a ray from this 3D point **P** to the camera optical center $\mathbf{O}_c$ intersects the image plane at a point $\mathbf{P}'$. Then, the relationship between the point $\mathbf{P} = [X, Y, Z]^T$ in the 3D camera coordinate and its image projection pixel $\mathbf{P}' = [u, v]^T$ in the 2D image plane can be represented as:

$$Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha f & 0 & c_x \\ 0 & \beta f & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{KP}, \tag{1}$$

where **K** is the camera intrinsic matrix [11], which contains the camera's intrinsic parameters $[c_x, c_y]^T$, $\alpha$, $\beta$ and $f$. Here, $[c_x, c_y]^T$ is the pixel coordinate of the principal point **C** in the image plane. $f$ is the camera focal length, which is represented in physical measurements, i.e., meters, and is equal to the distance from the camera center $\mathbf{O}_c$ to the image plane, i.e., $\mathbf{O}_c\mathbf{C}$. Considering that the projected points in the image plane are described in pixels, while 3D points in the camera coordinate system are represented in physical measurements, i.e., meters, we introduce the parameters $\alpha$ and $\beta$ to correlate the same points in different coordinate systems using different units. Thus the parameters $\alpha$ and $\beta$ are the number of pixels per meter(i.e., unit distance in physical measurements) along $x_i$-axis and $y_i$-axis, as shown in Fig.2. Note that $\alpha$ and $\beta$ may be different because the aspect ratio of the unit pixel is not guaranteed to be one. We can obtain these camera's intrinsic parameters in advance from prior calibration [21]. Then, the coordinate of the projection $\mathbf{P}'$ in the 2D image plane, i.e., $[u, v]^T$, can be computed according to Eq.(1).

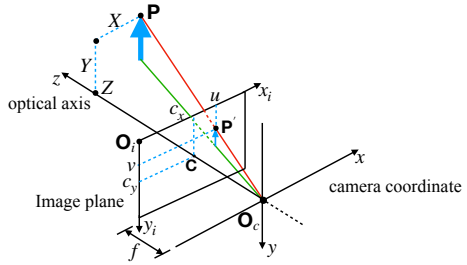
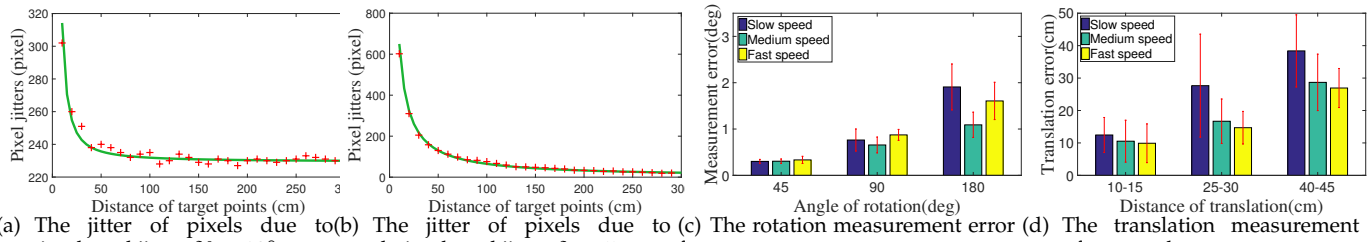



Fig. 2. Pinhole camera model.

## 4 Empirical Study

During the camera shoot, it is known that the camera is usually experiencing back-and-forth movement jitters of fairly high frequency, fast speed and small rotations and translations. In this section, we perform empirical studies on the real-world testbed in regard to the movement jitters and measurement errors, so as to investigate the following issues: 1) In what level do the movement jitters in the 3D space affect the pixel jitters in the image plane of the camera? 2) What are the average measurement errors in measuring the rotation and translation of the camera with the inertial sensors?

Without loss of generality, we use the smart phone Lenovo PHAB2 Pro as the testing platform. This platform has a 16-megapixel camera, we use it to capture the 1080p videos at 30 frames per second. Moreover, this platform has an inertial measurement unit (BOSCH BMI160) consisting of a 3-axis accelerometer and a 3-axis gyroscope, we use them to capture the linear acceleration and the angular rate of the body frame at a frequency of 200Hz, respectively. To capture the ground-truth of the 3D motion for the mobile device, including the rotation and translation, we use the OptiTrack system [22] to collect the experiment data.




(a) The jitter of pixels due to rotation-based jitter, $\delta\theta = 10°$ (b) The jitter of pixels due to translation-based jitter, $\delta t = 5$cm (c) The rotation measurement error from gyroscope (d) The translation measurement error from accelerometer

Fig. 3. The experiment results of the empirical study.

### 4.1 Observations

*Observation 1. When the camera is subject to the same rotation-based jitters, the stationary target points with closer distance to the image plane are suffering from stronger pixel jitters in the image plane*. To evaluate how the rotation-based jitters affect the pixel jitters in the image plane, we deployed the stationary target points in a line parallel to the optical axis of camera and with different distances to the image plane. Without loss of generality, we performed the rotation-based jitters around the $y$-axis of the camera coordinate system, which leads to the coordinate change of the projection in $x$-axis. The maximum rotation angle $\delta\theta$ is set to 10 degrees by default. Then, we measured the *pixel jitter* for the target points with different depths, i.e., coordinate difference in pixels between the projections before and after the rotation-based jitter. As shown in Fig.3(a), we use the pinhole camera model to predict the pixel jitter of an object at a given distance, and plot it as the curve shown in green color, then we plot the corresponding experiment results for pixel jitter of an object at a given distance. The comparison between the theoretical results and the experiment results shows that the observations from the experiments are consistent with the theoretical hypothesis from the pinhole camera model. According to the experiment results, we found that as the depth of the target point increases from 10cm to 50cm, the pixel jitter decreases rapidly from 314 pixels to 235 pixels. Then, as the depth further increases from 50cm to 150cm, the pixel jitter decreases very slowly from 235 pixels to 230 pixels.

*Observation 2. When the camera is subject to the same translation-based jitters, the stationary target points with closer distance to the image plane are suffering from stronger pixel jitters in the image plane*. To evaluate how the translation-based jitters affect the pixel jitters in the image plane of the camera, we deployed the target points in the optical axis of the camera and with different distances to the image plane. Without loss of generality, we performed the translation-based jitters along the $x$-axis of the camera coordinate system, the maximum displacement $\delta t$ is set to 5cm by default. Then, we also measured the *pixel jitter* for the target points with different depths. As shown in Fig.3(b), we use the pinhole camera model to predict the pixel jitter of an object at a given distance, and plot it as the curve shown in green color, then we plot the corresponding experiment results for pixel jitter of an object at a given distance. We found that as the depth of the target point increases from 10cm to 50cm, the pixel jitter decreases rapidly from 650 pixels to 130 pixels. Then, as the depth further increases from 50cm to 150cm, the pixel jitter decreases very slowly from 130 pixels to 43 pixels.

*Observation 3. When the mobile device is rotating, the gyroscope is able to accurately measure the rotation in low, medium and high speed mode*. To evaluate the average measurement errors in measuring the rotation with the gyroscope, without loss of generality, we rotated the mobile device around the $z$-axis of the local coordinate system with the angle of 45°, 90° and 180°, respectively. Besides, for each rotation angle, we evaluate the measurement errors with the low speed (10°/s), medium speed (40°/s) and high speed (100°/s) mode, respectively. Specifically, the measurement errors are calculated by comparing the gyroscope measurement with the ground truth. According to the experiment results in Fig.3(c), we found that, as the rotation angle increases from 45° to 180°, the measurement error is slightly increasing, which is always less than 2° in all cases.

*Observation 4. When the mobile device is moving back and forth, the accelerometer usually fails to accurately measure the translation in low, medium and high speed mode*. To evaluate the average measurement errors in measuring the translation with the accelerometer, without loss of generality, we move the mobile device back and forth in the range of [-5cm, +5cm] along the $z$-axis of the local coordinate system, by varying the overall distance from 10∼15cm to 40∼45cm, respectively. Besides, for each moving distance, we evaluate the measurement errors with the low speed (3cm/s), medium speed (30cm/s) and high speed (100cm/s) mode, respectively. Specifically, the measurement errors are calculated by comparing the accelerometer measurement with the ground truth. As shown in Fig.3(d), we found that, for all three speed modes, as the moving distance increases from 10∼15cm to 40∼45cm, the corresponding measurement errors are linearly increasing. Nevertheless, the measurement errors of all speed modes with all moving distances are all greater than 10cm. Since the actual translation ranges in [-5cm, +5cm], and the maximum moving distance is less than 45cm, thus the average measurement error (whether displacement error or distance error) greater than 10cm is not acceptable at all.

### 4.2 Summary

Both the rotation-based jitters and the translation-based jitters cause non-negligible pixel jitters in the image plane during video shoot. With the inertial measurement units, usually the *rotation* can be accurately measured by the gyroscope, whereas the translation fails to be accurately measured by the accelerometer. Therefore, it is essential to estimate the translation in an accurate and lightweight manner, such that the video stabilization can be effectively performed.

# 5 Problem Formulation and Modeling

## 5.1 Problem Formulation

According to the observations in the empirical study, in order to achieve video stabilization, we need to *accurately track the rotation and translation* during the video shot, so as to effectively remove the jitters from the rotation and translation. Meanwhile, we need to perform the rotation/translation estimation in a lightweight manner, so as to *make the computation overhead suitable for real-time processing*. Therefore, based on the above understanding, it is essential to statistically minimize the expectation of both rotation estimation error and translation estimation error during the process of video shot. Meanwhile, we need to effectively limit the expected computation overhead within a certain threshold, say $\tau$. Specifically, let the rotation estimation error and translation estimation error be $\delta_r$ and $\delta_t$, respectively, and let the computation overhead for rotation estimation and translation estimation be $c_r$ and $c_t$, respectively. We use the function $\exp()$ to denote the expectation. Then, the objective of our solution is to

$$\min \exp(\delta_r) + \exp(\delta_t), \tag{2}$$

subject to:

$$\exp(c_r) + \exp(c_t) \leq \tau.$$

To achieve the above objective, we first analyze the pros and cons for the IMU-based and CV-based approaches, as shown in Table 1. To track the translation, considering that only the CV-based approach is able to track the translation with high accuracy, we thus use the CV-based approach to estimate the translation. Moreover, to track the rotation, on one hand, both the IMU-based and CV-based approaches are able to track the rotation with high accuracy, on the other hand, the compute complexity of the CV-based approach is relatively high, especially when the 6 degrees of freedom (DoF) are undetermined. Hence, we use the IMU-based approach to estimate the rotation, due to its low compute complexity. In this way, the compute overhead of the CV-based approach is greatly reduced, since the undetermined DoF for CV-based processing is greatly reduced from 6 to 3.

| | Rotation Tracking | Translation Tracking | Compute Complexity |
|---|---|---|---|
| **IMU-based** | High Accuracy (3 DoF) | Low Accuracy (3 DoF) | Low |
| **CV-based** | High Accuracy (3 DoF) | High Accuracy (3 DoF) | High |

TABLE 1
Pros and cons of IMU and CV-based approaches for video stabilization.

Therefore, after formulating the video stabilization problem in an expectation-minimization framework, we can decompose and solve this complex optimization problem by breaking it down into two subproblems, i.e., using the IMU-based approach to estimate the rotation and using the CV-based approach to estimate the translation.

## 5.2 Camera Projection Model

According to the pinhole camera model, for any arbitrary 3D point $\mathbf{P}$ from the stationary object in the scene, the corresponding 2D projection $\mathbf{P}'$ in the image plane always keeps unchanged. However, when the body frame of the camera is dynamically moving in the 3D space, the camera coordinate system as well as the image plane is also continuously moving, which involves *rotation* and *translation*. In this way, even if the point $\mathbf{P}$ keeps still in the 3D space, the corresponding projection $\mathbf{P}'$ is dynamically changing in the 2D plane, thus further leading to video shaking in the image plane.

As any 3D motion can be decomposed into the combination of rotation and translation, we can use the rotation matrix $\mathbf{R}_{t_0,t}$ and a vector $\mathbf{T}_{t_0,t}$ to represent the rotation and translation of the camera coordinate system, respectively, from the time $t_0$ to the time $t$. Then, for a target point $\mathbf{P}_i$ in the camera coordinate system, if its coordinate at time $t_0$ is denoted as $\mathbf{P}_{i,t_0}$, then, after the rotation and translation of the camera coordinate system, its coordinate $\mathbf{P}_{i,t}$ at time $t$ can be computed by

$$\mathbf{P}_{i,t} = \mathbf{R}_{t_0,t}\mathbf{P}_{i,t_0} + \mathbf{T}_{t_0,t}. \tag{3}$$

Therefore, according to Eq. (1), for the point $\mathbf{P}_{i,t}$ at time $t$, the corresponding projection in the image plane, i.e., $\mathbf{P}'_{i,t} = [u_{i,t}, v_{i,t}]^T$, can be computed by

$$Z_{i,t} \cdot [u_{i,t}, v_{i,t}, 1]^T = \mathbf{K}\mathbf{P}_{i,t} = \mathbf{K}(\mathbf{R}_{t_0,t}\mathbf{P}_{i,t_0} + \mathbf{T}_{t_0,t}), \tag{4}$$

where $Z_{i,t}$ is the coordinate of $\mathbf{P}_{i,t}$ in the $z-$axis of the camera coordinate at time $t$, $\mathbf{K}$ is the camera intrinsic matrix.

## 5.3 Camera Motion Model

### 5.3.1 Coordinate Transformation

As the mobile devices are usually equipped with Inertial Measurement Units (IMU), thus the motion of the camera can be measured by IMU, in the *local coordinate system* of the body frame, as shown in Fig. 4. As aforementioned in Section 5.2, the camera projection is measured in the *camera coordinate system*, once we figure out the camera's motion from the inertial measurements in the *local coordinate system*, it is essential to transform the camera's motion into the *camera coordinate system*.

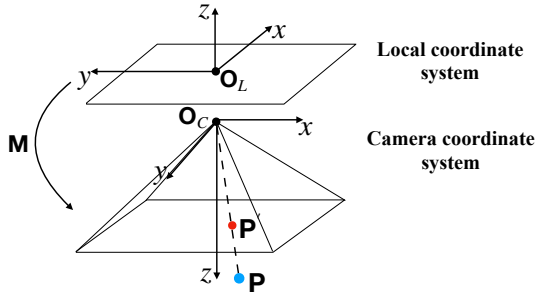



Fig. 4. The local coordinate system and the camera coordinate system of the rear camera.

For the embedded camera of the mobile device, we take the mostly used rear camera as an example. As shown in Fig. 4, we show the *camera coordinate system* and the *local coordinate system*, respectively. According to the relationship between *camera coordinate system* and the *local coordinate system*, we can use a $3\times3$ rotation matrix $\mathbf{M} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$ to denote the coordinate transformation between the two coordinate systems. For any other camera, we can also use a similar rotation matrix $\mathbf{M}'$ to denote the corresponding coordinate transformation.

#### 5.3.2 Rotation Estimation

According to Eq. (4), it is essential to accurately estimate the rotation matrix $\mathbf{R}_{t_0,t}$ and the translation vector $\mathbf{T}_{t_0,t}$, such that the projection of $\mathbf{P}_i$ at time $t$ in the image plane, i.e., $\mathbf{P}'_{i,t} = [u_{i,t}, v_{i,t}]^T$, can be figured out. To estimate the camera's rotation $\mathbf{R}_{t_0,t}$ from the time $t_0$ to time $t$, we first use the gyroscope to measure the angular speed in each axis of the local coordinate system. Then, according to the small angle approximation [23], we can compute the rotation matrix $\mathbf{A}_{t,t+\delta t}$ relating the local coordinate at time $t$ to the one at time $t+\delta t$. After obtaining $\mathbf{A}_{t,t+\delta t}$, we can further update the rotation matrix $\mathbf{R'}_{t_0,t}$ for the local coordinate system as follows:

$$\mathbf{R'}_{t_0,t+\delta t} = \mathbf{A}_{t,t+\delta t}\mathbf{R'}_{t_0,t}. \tag{5}$$

Hence, considering the coordinate transformation between the local coordinate system and the camera coordinate system, we further compute the camera's rotation in camera coordinate system as follows:

$$\mathbf{R}_{t_0,t} = \mathbf{M}\mathbf{R'}_{t_0,t}\mathbf{M}^{-1}. \tag{6}$$

#### 5.3.3 Translation Estimation

Considering the nonnegligible error accumulation of using linear acceleration to calculate the translation, we introduce the computer vision (CV)-based method, which utilizes the feature point pairs to estimate the motion between two frames. However, different from traditional CV-based methods which calculate both rotation and translation in each axis, i.e., 6 degrees of freedom (DOF), we have calculated rotation from gyroscope and only need to calculate the unknown translation, i.e., 3 DOFs. Therefore, we reduce the DOFs in the 3d motion from 6 to 3. Specifically, we first detect the feature point pairs to estimate the 3D motion of camera. After that, we subtract the rotation measured by IMU from the estimated 3D motion to obtain the translation. However, due to the continuous change of camera coordinate system and the non-unified unit for the estimated translation, we introduce the initialization to define the unified *translation unit* and represent the fixed 3D points in the unified unit to estimate the following translation in a unified unit.

**Feature Point Extraction**. According to each image frame of the video, we first utilize the FAST (Features from Accelerated Segment Test) keypoint detector to detect the feature point, and then calculate the binary BRIEF (Binary Robust Independent Elementary Features) descriptor [24] of the feature point. Both the feature point and the descriptor form an ORB feature [25]. Specifically, for the feature point $\mathbf{P}'_{i,t_0}$ in the image frame $I_{t_0}$ and the feature point $\mathbf{P}'_{j,t_1}$ in image frame $I_{t_1}$, we use $\mathbf{D}_i$ and $\mathbf{D}_j$ to represent their descriptors, respectively. Then, the similarity between $\mathbf{P}'_{i,t_0}$ and $\mathbf{P}'_{j,t_1}$ can be measured by the *hamming distance* [26] between their descriptors $\mathbf{D}_i$ and $\mathbf{D}_j$. Given $\mathbf{P}'_{i,t_0}$, we choose the feature point with the nearest hamming distance in the image $I_{t_1}$ to be the matching feature point for $\mathbf{P}'_{i,t_0}$, and they form a feature point pair. The coordinate difference between the feature point pair can be used to estimate the camera's motion.

**Initialization for Translation Unit**. As shown in Fig. 5, for each feature point pair $(\mathbf{P}'_{i,t_0}, \mathbf{P}'_{i,t_1})$, given the projection

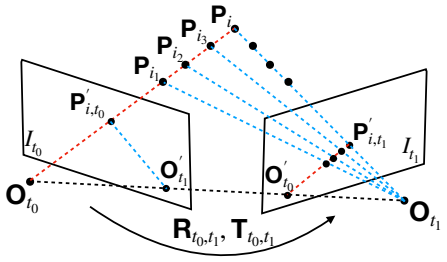



Fig. 5. Epipolar geometry.

point $\mathbf{P}'_{i,t_0}$ and the camera optical center $\mathbf{O}_{t_0}$, the target 3D point $\mathbf{P}_i$ must locate in the ray $\mathbf{O}_{t_0}\mathbf{P}'_{i,t_0}$. Similarly, the target point $\mathbf{P}_i$ should also locate in the ray $\mathbf{O}_{t_1}\mathbf{P}'_{i,t_1}$. Thus $\mathbf{P}_i$ is the intersection point of $\mathbf{O}_{t_0}\mathbf{P}'_{i,t_0}$ and $\mathbf{O}_{t_1}\mathbf{P}'_{i,t_1}$. In computer vision, this is referred to the epipolar constraint [11]. Then, we can use the fundamental matrix $\mathbf{F}_{t_0,t_1}$ [11] to describe the epipolar constraint, i.e.,

$$\mathbf{P}'^{\,T}_{i,t_1}\mathbf{F}_{t_0,t_1}\mathbf{P}'_{i,t_0} = 0. \tag{7}$$

Here, the fundamental matrix $\mathbf{F}_{t_0,t_1}$ can be generated by the relative rotation $\mathbf{R}_{t_0,t_1}$ and translation $\mathbf{T}_{t_0,t_1}$ from the image $I_{t_0}$ to image $I_{t_1}$, i.e.,

$$\mathbf{F}_{t_0,t_1} = \mathbf{K}^{-T}[\mathbf{T}_{t_0,t_1}]_\times \mathbf{R}_{t_0,t_1}\mathbf{K}^{-1}, \tag{8}$$

where $\mathbf{K}$ is the camera intrinsic matrix, $[\mathbf{T}_{t_0,t_1}]_\times$ is a $3\times 3$ matrix. Specifically, let $\mathbf{T}_{t_0,t_1} = [T^x_{t_0,t_1}, T^y_{t_0,t_1}, T^z_{t_0,t_1}]^T$, then $[\mathbf{T}_{t_0,t_1}]_\times = \begin{bmatrix} 0 & -T^z_{t_0,t_1} & T^y_{t_0,t_1} \\ T^z_{t_0,t_1} & 0 & -T^x_{t_0,t_1} \\ -T^y_{t_0,t_1} & T^x_{t_0,t_1} & 0 \end{bmatrix}$. Therefore, by substituting Eq. (8) to Eq. (7), we have

$$(\mathbf{P}'^{\,T}_{i,t_1}\mathbf{K}^{-T})[\mathbf{T}_{t_0,t_1}]_\times(\mathbf{R}_{t_0,t_1}\mathbf{K}^{-1}\mathbf{P}'_{i,t_0}) = 0. \tag{9}$$

Here, the rotation matrix $\mathbf{R}_{t_0,t_1}$ can be estimated from the gyroscope measurements. Then, the only unknown factor in Eq. (9) is $[\mathbf{T}_{t_0,t_1}]_\times$, which has three unknown parameters $T^x_{t_0,t_1}, T^y_{t_0,t_1}, T^z_{t_0,t_1}$. Therefore, as long as we can obtain more than three pairs of matching feature points, we can solve $[\mathbf{T}_{t_0,t_1}]_\times$ based on the *Least Square Error* (LSE) method.

However, there could be multiple solutions for $[\mathbf{T}_{t_0,t_1}]_\times$, as we can multiply a nonzero coefficient on both sides of Eq. (9), whose right side is 0. For convenience, we figure out one of the solutions for $T^x_{t_0,t_1}, T^y_{t_0,t_1}$, and $T^z_{t_0,t_1}$ in *camera coordinate system*, based on Eq.(9). It is noteworthy that the calculated translation $\mathbf{T}_{t_0,t_1}$ from Eq. (9) is represented in a relative manner, instead of in a absolute unit. Therefore, there is a scale factor $\alpha$ between the calculated translation and the actual translation $\mathbf{T}^*_{t_0,t_1}$ in the absolute unit, as shown in Eq. (10), where $T^{x*}_{t_0,t_1}$, $T^{y*}_{t_0,t_1}$ and $T^{z*}_{t_0,t_1}$ mean the actual translation along $x$-axis, $y$-axis, $z$-axis.

$$T^x_{t_0,t_1} = T^{x*}_{t_0,t_1}\cdot\alpha, T^y_{t_0,t_1} = T^{y*}_{t_0,t_1}\cdot\alpha, T^z_{t_0,t_1} = T^{z*}_{t_0,t_1}\cdot\alpha. \tag{10}$$

However, it is actually difficult to transform the calculated translation $\mathbf{T}_{t_0,t_1}$ in the absolute unit, since the values of $\alpha$, $T^{x*}_{t_0,t_1}$, $T^{y*}_{t_0,t_1}$, and $T^{z*}_{t_0,t_1}$ are all unknown. To tackle the above issue, we define

$$|\mathbf{T}_{t_0,t_1}| = \sqrt{(T^x_{t_0,t_1})^2 + (T^y_{t_0,t_1})^2 + (T^z_{t_0,t_1})^2} \tag{11}$$

as the *translation unit*. In the following frames, when we calculate a new translation, we represent it in the above translation unit. Consequently, we can represent the calculated translation over all frames with a unified unit.

**Compute Coordinates of Fixed 3D Points in Unified Unit**. According to Eq. (4), representing the translation $\mathbf{T}_{t_0,t}$ in the unified unit also means representing the coordinate of 3D point $\mathbf{P}_{i,t}$ in the unified unit. Since the 3D points are stationary, we can use the coordinate of 3D point $\mathbf{P}_{i,t_0}$ at time $t_0$ to represent the coordinate of $\mathbf{P}_i$ at any time. In regard to the coordinate of any fixed 3D point at time $t_0$, according to Eq. (4), we can use the rotation $\mathbf{R}_{t_0,t_1}$ and the translation $\mathbf{T}_{t_0,t_1}$ from $t_0$ to $t_1$ to calculate it in the unified unit, since $\mathbf{T}_{t_0,t_1}$ is a unified unit. Specifically, for an arbitrary target point $\mathbf{P}_i$, suppose the 3D coordinates of $\mathbf{P}_i$ in the camera coordinate system are $\mathbf{P}_{i,t_0} = [X_{i,t_0}, Y_{i,t_0}, Z_{i,t_0}]$ and $\mathbf{P}_{i,t_1} = [X_{i,t_1}, Y_{i,t_1}, Z_{i,t_1}]$, respectively, at the time $t_0$ and $t_1$. Then, the corresponding 2D projections in the image plane are $\mathbf{P}'_{i,t_0}$ and $\mathbf{P}'_{i,t_1}$, respectively. Hence, based on the camera projection model in Eq.(4), we have

$$\begin{cases} Z_{i,t_0}\mathbf{P}'_{i,t_0} &= \mathbf{K}\mathbf{P}_{i,t_0} \Rightarrow Z_{i,t_0}\mathbf{K}^{-1}\mathbf{P}'_{i,t_0} = \mathbf{P}_{i,t_0}, \\ Z_{i,t_1}\mathbf{P}'_{i,t_1} &= \mathbf{K}\mathbf{P}_{i,t_1} \Rightarrow Z_{i,t_1}\mathbf{K}^{-1}\mathbf{P}'_{i,t_1} = \mathbf{P}_{i,t_1}. \end{cases} \tag{12}$$

After we calculate the rotation $\mathbf{R}_{t_0,t_1}$ and translation $\mathbf{T}_{t_0,t_1}$ for the camera coordinate system between the two time points $t_0$ and $t_1$, we have $\mathbf{P}_{i,t_1} = \mathbf{R}_{t_0,t_1}\mathbf{P}_{i,t_0} + \mathbf{T}_{t_0,t_1}$, based on Eq. (3). Thus according to Eq.(12), we further have

$$\mathbf{P}_{i,t_1} = Z_{i,t_0}\mathbf{R}_{t_0,t_1}\mathbf{K}^{-1}\mathbf{P}'_{i,t_0} + \mathbf{T}_{t_0,t_1}. \tag{13}$$

If we let $\mathbf{X}_{i,t_0} = \mathbf{K}^{-1}\mathbf{P}'_{i,t_0}$ and $\mathbf{X}_{i,t_1} = \mathbf{K}^{-1}\mathbf{P}'_{i,t_1}$, then, according to Eq.(12) and Eq.(13), we have

$$\mathbf{P}_{i,t_1} = Z_{i,t_1}\mathbf{X}_{i,t_1} = Z_{i,t_0}\mathbf{R}_{t_0,t_1}\mathbf{X}_{i,t_0} + \mathbf{T}_{t_0,t_1}. \tag{14}$$

Thus, to compute the coordinate of $\mathbf{P}_{i,t_1}$, we only need to solve $Z_{i,t_0}$ or $Z_{i,t_1}$. By multiplying both sides of Eq.(14) with the vector $\mathbf{X}_{i,t_1}$, we can eliminate the unknown parameter $Z_{i,t_1}$ and then calculate the unknown parameter $Z_{i,t_0}$. Specifically, the left side of Eq.(14), i.e., $Z_{i,t_1}(\mathbf{X}_{i,t_1} \times \mathbf{X}_{i,t_1})$ should be equal to 0, since the cross product of any vector itself should be equal to 0, then the right side is

$$Z_{i,t_0}(\mathbf{R}_{t_0,t_1}\mathbf{X}_{i,t_0}) \times \mathbf{X}_{i,t_1} + (\mathbf{T}_{t_0,t_1}) \times \mathbf{X}_{i,t_1} = 0. \tag{15}$$

According to Eq. (15), we are able to solve $Z_{i,t_0}$. Then, based on Eq. (14), we can further calculate $\mathbf{P}_{i,t_1}$. Similarly, we can also calculate $\mathbf{P}_{i,t_0}$ as well as the 3D coordinates of other target points.

**Translation Estimation in Unified Unit**. According to the projection model in Eq. (4), at any time $t$ during the camera shoot, we can depict the relationship between the 3D point and its corresponding projection in the image plane. Here, $\mathbf{K}$ is a known parameter, as aforementioned, the rotation $\mathbf{R}_{t_0,t}$ can be calculated with the gyroscope-based method, and the 3D coordinates of $\mathbf{P}_{i,t_0}$ can be calculated with the CV-based method. Thus, the only unknown parameters are $\mathbf{T}_{t_0,t} = [T^x_{t_0,t}, T^y_{t_0,t}, T^z_{t_0,t}]$ and $Z_{i,t}$. To solve the above four parameters, we need at least two pairs of feature points to set up four equations. We can use the *Least Square Error* (LSE) method to solve the overdetermined equation system. After that, we are able to depict the translation with the unified *translation unit*. Specifically, let $u = |\mathbf{T}_{t_0,t_1}|$, then we can denote $T^x_{t_0,t} = \gamma_x \cdot u, T^y_{t_0,t} = \gamma_y \cdot u, T^z_{t_0,t} = \gamma_z \cdot u$. In this way, we can estimate the translation of the camera with the unified unit.

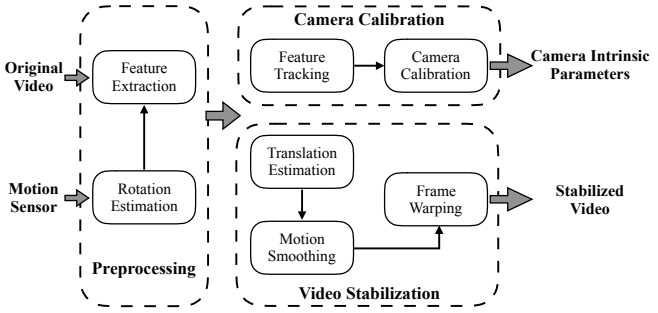



Fig. 6. System Framework

## 6 System Design

### 6.1 System Overview

The system architecture is shown in Fig.6. We take as input frames from original video and sensor readings from motion sensor. We first perform *Preprocessing* to estimate the 3D rotation of the camera based on the solution aforementioned in Section 5.3.2, and extract features for video frames. The estimated rotation and the video frames with feature points will be served for two tasks, *Camera Calibration* and *Video Stabilization*. The Camera Calibration performs feature tracking between consecutive video frames to obtain feature point pairs, and then uses feature point pairs to calculate camera intrinsic parameters. The Video Stabilization performs video stabilization in three major steps. First, the 3D translation of the camera is estimated based on the solution aforementioned in Section 5.3.3. Second, the 3D motion of the camera is sufficiently smoothed to remove the undesired jitters, thus a smoothed moving path of the camera is generated. Finally, given a smoothed moving path of the camera in the 3D space, the stabilized video is created by the frame warping, i.e., warping each pixel in the original frame to the corresponding stabilized frame, according to the mapping relationship between the original moving path and the smoothed moving path. After that, each frame of the stabilized video appears to be captured along the smoothed moving path.

### 6.2 Camera Calibration

According to the pinhole camera model aforementioned in Section 3, in order to depict the camera projection, we need to know the camera's intrinsic parameters, i.e., $[c_x, c_y]^T$, $\alpha$, $\beta$ and $f$. Here, $[c_x, c_y]^T$ is the pixel coordinate of the principal point in the image plane. Without loss of generality, if we set the image size to $(w, h)$ in pixels, then $[c_x, c_y]^T$ is ideally equal to $[\frac{w}{2}, \frac{h}{2}]^T$. However, due to the sensor manufacturing errors, the principal point, which is the intersection point of the optical axis and the image plane, will be slightly offest from the center of the image, i.e., $[c_x, c_y]^T$ will not be equal to $[\frac{w}{2}, \frac{h}{2}]^T$ and needs us to estimate. $f$ is the camera focal length, which is represented in physical measurements, i.e., meters. $\alpha$ and $\beta$ are the number of pixels per unit distance in physical measurements (i.e. meter) along the $x_i$-axis and $y_i$-axis of the image plane, and they are used to correlate the image plane using pixels and the camera coordinate system using meters. If given an arbitrary camera, we may not have access to these parameters. However, we can access to the images the camera takes. Thus we can find a way to deduce these parameters from images, which is referred as camera

calibration. There are many different approaches to calculate the intrinsic parameters for a camera, which can be divided into two main categories, i.e., traditional camera calibration which uses reference objects with known geometry (e.g. spin table and checkerboard) [21], and automatic-calibration which does not use any known pattern [27]. Taking into account the convenience of operations for everyday use, we propose *motion-assisted calibration* to calculate the intrinsic parameters of camera, by performing a structured movement.

We design a simple camera calibration process, i.e., the user only needs to use the camera to shoot a video for 5∼10 seconds. In the process of shooting video, the user keeps the camera's position unchanged, and just changes the camera's orientation by rotation. Then we perform feature tracking between consecutive frames to obtain feature point pairs, and use these feature point pairs to calculate the camera intrinsic parameters. Specifically, during the camera calibration, the motion of camera only involves rotation. For a target 3D point $\mathbf{P}_i$, if its coordinate at time $t_0$ is denoted as $\mathbf{P}_{i,t_0}$, then at time $t$, after the camera rotation $\mathbf{R}_{t_0,t}$ from time $t_0$ to time $t$, its corresponding projection in the image plane, i.e., $\mathbf{P}'_{i,t}$, can be computed by

$$\mathbf{P}'_{i,t} = \mathbf{K}\mathbf{R}_{t_0,t}\mathbf{P}_{i,t_0}, \tag{16}$$

where $\mathbf{K}$ is the camera intrinsic matrix, which contains the camera's intrinsic parameters. In order to calculate $\mathbf{K}$, we

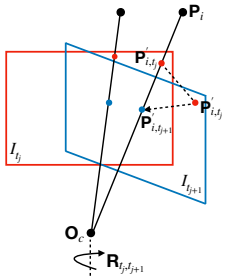



Fig. 7. The pure rotation motion model

use feature point pairs of consecutive frames. As shown in Fig.7, for each feature point pair $(\mathbf{P}'_{i,t_j}, \mathbf{P}'_{i,t_{j+1}})$ in the consecutive frames $(I_{t_j}, I_{t_{j+1}})$, we have $\mathbf{P}'_{i,t_j} = \mathbf{K}\mathbf{R}_{t_0,t_j}\mathbf{P}_{i,t_0}$ and $\mathbf{P}'_{i,t_{j+1}} = \mathbf{K}\mathbf{R}_{t_0,t_{j+1}}\mathbf{P}_{i,t_0}$, based on Eq.(16). Thus the mapping relationship from feature point $\mathbf{P}'_{i,t_j}$ to feature point $\mathbf{P}'_{i,t_{j+1}}$ can be represented as:

$$\mathbf{P}'_{i,t_{j+1}} = \mathbf{K}\mathbf{R}_{t_0,t_{j+1}}\mathbf{R}^{-1}_{t_0,t_j}\mathbf{K}^{-1}\mathbf{P}'_{i,t_j}, \tag{17}$$

where the coordinates of the feature point pair $(\mathbf{P}'_{i,t_j}, \mathbf{P}'_{i,t_{j+1}})$ can be obtained by feature tracking, the rotation $\mathbf{R}_{t_0,t_j}$ and $\mathbf{R}_{t_0,t_{j+1}}$ can be obtained by rotation estimation aforementioned in Section 5.3.2. Then, the only unknown factor in Eq.(17) is $\mathbf{K}$, which has five unknown parameters, i.e., $[c_x, c_y]^T$, $\alpha$, $\beta$ and $f$. To solve the above five parameters, we formulate camera calibration as an optimization problem, where we want to minimize the reprojection error of all feature point pairs:

$$\mathbf{K}^* = \arg\min_{\mathbf{K}} \sum_{j=1}^{N-1}\sum_{i=1}^{N_j} \|\mathbf{P}'_{i,t_{j+1}} - \mathbf{K}\mathbf{R}_{t_0,t_{j+1}}\mathbf{R}^{-1}_{t_0,t_j}\mathbf{K}^{-1}\mathbf{P}'_{i,t_j}\|^2, \tag{18}$$

where $N$ is the number of frames, $N_j$ is the number of feature point pairs of $j-$th consecutive frame. By solving this optimization problem based on the *Least Square Error*(LSE) method, we can calculate the camera intrinsic parameters. Note that the camera calibration only needs to be done once for each camera.

### 6.3 Video Stabilization

#### 6.3.1 Camera Translation Estimation

According to the method aforementioned in Section 5.3.3, we can estimate the camera's 3D translation $\{\mathbf{T}_{t_0,t}\}$, from the time $t_0$ to the time $t$. In particular, during the procedure of the translation estimation, we use the pairs of feature points as reference points. We expect the 3D target point of feature points to be stationary in regard to the earth coordinate system, such that the camera motion and frame warping can be accurately performed based on the coordinate variation of feature points. However, in real applications, these 3D target points can be dynamically moving instead of keeping stationary. For example, the feature points can be extracted from the moving human subjects in the scene. These feature points should be regarded as outliers. Therefore, we use the Random Sample Consensus (RANSAC) algorithm [28] to detect the outliers. Specifically, at any time $t$, we calculate the translation $\mathbf{T}_{t_0,t}$ through multiple iterations. In each iteration, e.g., the kth iteration, we first select two pairs of matching points randomly to calculate the translation $\mathbf{T}^k_{t_0,t}$. Then, we use the translation $\mathbf{T}^k_{t_0,t}$ to calculate the reprojection error $E_{i,k}$ for each matching point pair $(\mathbf{P}_{i,t_0}, \mathbf{P}'_{i,t})$, as shown in Eq. (19).

$$E_{i,k} = \|\mathbf{P}'_{i,t} - \mathbf{K}(\mathbf{R}_{t_0,t}\mathbf{P}_{i,t_0} + \mathbf{T}^k_{t_0,t})\|^2. \tag{19}$$

If the average reprojection error of all matching pairs at the kth iteration is below a certain threshold, we add the calculated $\mathbf{T}^k_{t_0,t}$ to the candidate translations. In addition, the matching point pairs whose reprojection errors are below the certain threshold are classified as inliers. While the matching point pairs whose reprojection errors are above the certain threshold are classified as outliers. If we have enough inliers or the iteration is repeated a fixed number of times, the iteration stops. Finally, we choose the candidate translation with minimal rejection error as the translation at the time $t$.

In addition, due to the motion of camera, some 3D points will move out of view and cannot be used to estimate the following translation. Therefore, when the number of available 3D points is less than a threshold, we will detect new 3D points based on the feature point pairs of frames at time $t-1$ and time $t$, as mentioned before.

#### 6.3.2 Camera Motion Smoothing

Due to the sudden movement of the mobile devices during the camera shooting, there might exist a number of jitters in regard to the measurements related to the rotation and translation. Specifically, suppose that we set the time $t_0$ as the initial time, according to the method aforementioned in Section 5.3, we can calculate the rotation matrix $\{\mathbf{R}_{t_0,t}\}$ and the translation vector $\{\mathbf{T}_{t_0,t}\}$, respectively, from the time $t_0$ to the time $t$. We set the sequence of $\{\mathbf{R}_{t_0,t}\}$ and $\{\mathbf{T}_{t_0,t}\}$ as the *original camera motion*, as it represents the

original moving path of the camera which involves the unexpected jitters. Then, to smooth the unexpected jitters from the *original camera motion*, which is usually existing in high frequency band, we apply the low-pass filter on the sequence of $\{\mathbf{R}_{t_0,t}\}$ and $\{\mathbf{T}_{t_0,t}\}$, respectively, to obtain the *smoothed camera motion*.

To smooth the unexpected jitters in the translation, it is known that the translation vector $\{\mathbf{T}_{t_0,t}\}$ has three degrees of freedom, we can apply the low-pass filter on each of the three dimensions directly. Without loss of generality, we describe our solution in regard to the $x$-dimension of the translation vector $\{\mathbf{T}_{t_0,t}\}$, i.e., $\{\mathbf{T}^x_{t_0,t}\}$. Specifically, we apply a *weighted moving average* filter on the sequence of $\{\mathbf{T}^x_{t_0,t}\}$ using a sliding window. To calculate the smoothed translation, we provide different weights at different positions of the sliding window. Here, we use a Gaussian function to calculate the weights, so that for the time $t$, the weight given to the close positions is higher than those distant positions. Assume the length of the sliding window is $n$, then at the time $t_i$, the smoothed translation $\hat{\mathbf{T}}^x_{t_0,t_i}$ can be

$$\hat{\mathbf{T}}^x_{t_0,t_i} = \sum_{j=1}^{n} \frac{w_j}{\sum_{j=1}^{n} w_j} \mathbf{T}^x_{t_0,t_{i-j+1}}, \tag{20}$$

where $w_j$ is the weight at the position $j$. In this way, we are able to smooth the translation-based jitters of the camera.

To smooth the unexpected jitters in the rotation, it is known that the $3 \times 3$ rotation matrix $\mathbf{R}_{t_0,t}$ involves 9 parameters. As the camera coordinate system is rotating, we obtain 9 streams of these parameters over time. However, these parameters are not mutually independent to each other, since the rotation usually has three degrees of freedom in the 3D space. Therefore, in order to smooth the jitters in the rotation measurement, we first transform the $3 \times 3$ rotation matrix $\mathbf{R}_{t_0,t}$ into the corresponding Euler angles $[\phi_{t_0,t}, \theta_{t_0,t}, \psi_{t_0,t}]$. Specifically, suppose $\mathbf{R}_{t_0,t} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$, then, the Euler angles $(\phi_{t_0,t}, \theta_{t_0,t}, \psi_{t_0,t})$ can be calculated by

$$\begin{cases} \phi_{t_0,t} = \arctan(\frac{R_{32}}{R_{33}}), \\ \theta_{t_0,t} = \arctan(\frac{-R_{31}}{\sqrt{R_{32}^2 + R_{33}^2}}), \\ \psi_{t_0,t} = \arctan(\frac{R_{21}}{R_{11}}). \end{cases} \tag{21}$$

Then we use the low pass filter such as the same weighted moving average filter to smooth the jitters in the Euler angles. After that, we further transform the smoothed Euler angles to the rotation matrix $\hat{\mathbf{R}}_{t_0,t}$ in a similar manner as follows:

$$\hat{\mathbf{R}}_{t_0,t} = \mathbf{R}^z_{t_0,t} \mathbf{R}^y_{t_0,t} \mathbf{R}^x_{t_0,t}. \tag{22}$$

Here, $\mathbf{R}^x_{t_0,t}$, $\mathbf{R}^y_{t_0,t}$, $\mathbf{R}^z_{t_0,t}$ mean the rotation matrix transformed from the Euler angles $\phi_{t_0,t}$, $\theta_{t_0,t}$, $\psi_{t_0,t}$, respectively. They also correspond to rotation around $x$-axis, $y$-axis, $z$-axis, respectively. In this way, we are able to smooth the rotation-based jitters of the camera.

Fig. 8 shows an example of the original moving path and the smoothed moving path in a global coordinate system, the moving path involves the rotation and translation of the camera in the 3D space. It is found that the original moving path is full of jitters, while the smoothed moving path is fairly flat and smooth in the contour. The red arrows show the mapping relationship between the positions in the original moving path and the positions in the smoothed moving path.

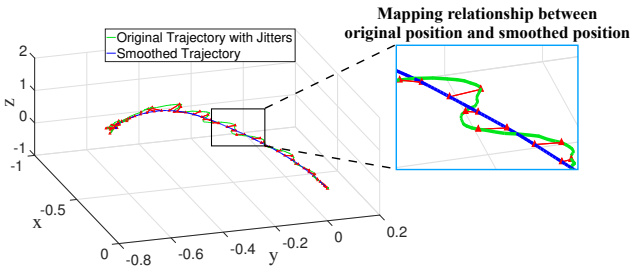



Fig. 8. The original moving path vs smoothed moving path

#### 6.3.3 Frame Warping for Video Stabilization

During a video shoot, if we do not introduce any stabilization operation, each image frame generated from the camera view is called *original frame*. However, if we introduce the estimated camera motion to calibrate the original image frame, we can replace the original frame with a new image frame, which is called *stabilized frame*. At first, we transform the pixels $\mathbf{P}'_{i,t}$ corresponding to the feature points in the original frame to the pixels $\hat{\mathbf{P}}'_{i,t}$ in the stabilized frame. For the feature point $\mathbf{P}'_{i,t}$ in the original frame, the coordinate of its corresponding 3D point at time $t_0$, i.e., $\mathbf{P}_{i,t_0}$, can be calculated based on the original camera motion. With the known coordinate $\mathbf{P}_{i,t_0}$ at time $t_0$, and the known *smoothed camera motion* $(\hat{\mathbf{R}}_{t_0,t}, \hat{\mathbf{T}}_{t_0,t})$ at the time $t$ by referring to the status at the time $t_0$, we can transform $\mathbf{P}_{i,t_0}$ to $\hat{\mathbf{P}}_{i,t}$, then to $\hat{\mathbf{P}}'_{i,t}$, based on the camera projection model shown in Eq. (23).

$$\hat{Z}_{i,t} \cdot \hat{\mathbf{P}}'_{i,t} = \mathbf{K}\hat{\mathbf{P}}_{i,t} = \mathbf{K}(\hat{\mathbf{R}}_{t_0,t}\mathbf{P}_{i,t_0} + \hat{\mathbf{T}}_{t_0,t}). \tag{23}$$

Here, $\hat{\mathbf{P}}_{i,t}$ is the coordinate of $\mathbf{P}_i$ in the smoothed camera coordinate system at the time $t$, $\hat{Z}_{i,t}$ is the coordinate of $\hat{\mathbf{P}}_{i,t}$ in the $z$-axis, i.e., the depth value of the pixel $\hat{\mathbf{P}}'_{i,t}$. Among the parameters, $\mathbf{P}_{i,t_0}$ is obtained through initialization, $\hat{\mathbf{R}}_{t,t_0}$ and $\hat{\mathbf{T}}_{t,t_0}$ are obtained through *smoothed camera motion*, while $\mathbf{K}$ is a known parameter, thus we can calculate $\hat{\mathbf{P}}_{i,t}$. After that, we can obtain $\hat{Z}_{i,t}$, which is the coordinate of $\hat{\mathbf{P}}_{i,t}$ in $z$-axis. That is to say, there is only one unknown parameter $\hat{\mathbf{P}}'_{i,t}$, which can be solved based on Eq. (23). In this way, we can transform each feature point $\mathbf{P}'_{i,t}$ in the original frame to the 3D point $\mathbf{P}_{i,t_0}$, and then transform $\mathbf{P}_{i,t_0}$ to the corresponding pixel $\hat{\mathbf{P}}'_{i,t}$ in the stabilized frame.

For the other pixels not belonging to feature points, it is difficult to transform them into the corresponding pixels in the stabilized frame, because of the unknown 3D points corresponding to the pixels. At this time, we combine the original frame and the projected feature points in the stabilized frame to stabilize the video. That is to say, we introduce a standard texture mapping algorithm according to the warped mesh [15], which proposed content-preserving warps to transform each pixel in the original frame to the stabilized frame.

### 6.4 Multi-Thread Optimization for Video Stabilization

In order to implement real-time video stabilization, our system needs to output the stabilized frame without noticeable

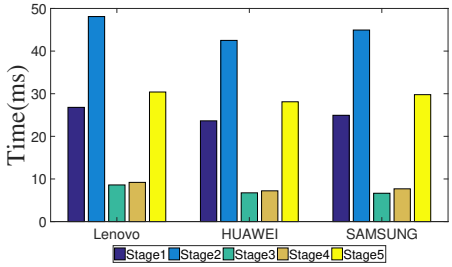



Fig. 9. Time cost in different stages

latency. If the frame rate is 30fps, we want the time latency to be lower than the waiting time between two frames, i.e., 33ms. According to Section 6.1, in order to output a stabilized video, we need to capture original frame, extract and track features, estimate and smooth camera motion, perform frame warping to obtain the stabilized frame, and finally write the stabilized frame to the stabilized video. The large time cost in image processing leads to large time latency for video stabilization. To solve this problem, we first profile the time cost of each stage in video stabilization, and then introduce multi-thread optimization to reduce the time cost.

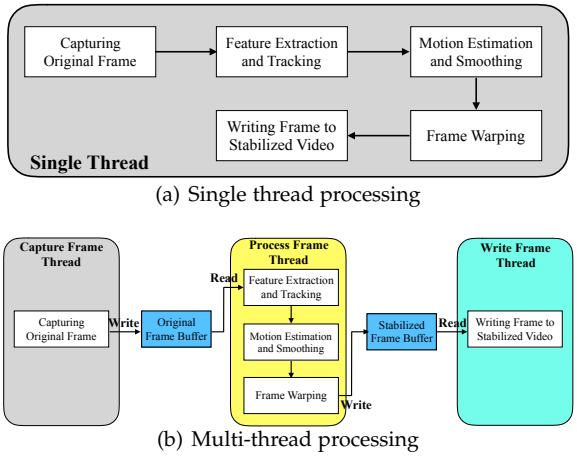
### 6.4.1 Time Cost in Different Stages

There are five main stages in our video stabilization system, i.e., capturing original frame, feature extraction and tracking, camera motion estimation and smoothing, frame warping, and writing the stabilized frame to the stabilized video. The stages are respectively called 'Stage1', 'Stage2', 'Stage3', 'Stage4' and 'Stage5' for short. We first set the image size to $1920 * 1080$ pixels, and then measure the time cost of processing one original frame in each stage. Without loss of generality, we use the smartphones Lenovo PHAB2 Pro, HUAWEI MATE20 Pro and SAMSUNG Galaxy Note8 as the testing platforms. We repeat the measurement for 500 times to get the average time cost. According to the measurement shown in Fig. 9, without loss of generality, we use the time cost with the Lenovo PHAB2 Pro by default, and the time cost in five stages with the Lenovo PHAB2 Pro smartphone is 26.8, 48.1, 8.6, 9.2, 30.4 ms, respectively. Thus the time cost of processing one original frame can be calculated by Eq.(24). Obviously, 123.1 ms is very large latency, thus more optimizations are expected for our video stabilization system to realize real time processing.

$$T = (26.8 + 48.1 + 8.6 + 9.2 + 30.4)ms = 123.1ms. \quad (24)$$

### 6.4.2 Multi-Thread Processing

As shown in Fig. 10(a), according to Eq.(24), if our system works with a single thread, it takes 123.1 ms to process one frame. Thus, we introduce multi-thread optimization to reduce the time cost. We use three threads to capture frame, process frame and write frame in parallel. As shown in Fig. 10(b), the *Capture Frame* thread captures frames from the original video and writes the original frames to the buffer. The *Process Frame* thread first reads the original frame from the buffer, then performs feature extraction and tracking, motion estimation and smoothing, frame warping to obtain the stabilized frame, and finally writes the stabilized frame




(a) Single thread processing

(b) Multi-thread processing

Fig. 10. Multi-thread processing

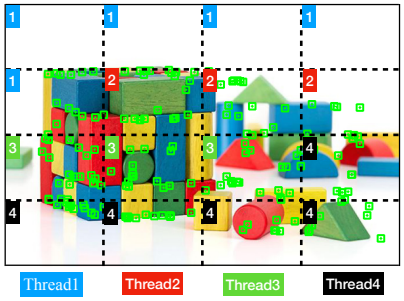



Fig. 11. Multi-thread feature extraction

to the buffer. The *Write Frmae* thread reads the stabilized frames from the buffer and writes them to video. By adopting multiple threads, the time cost of processing one frame can be calculated by

$$T = \max(26.8, 48.1 + 8.6 + 9.2, 30.4)ms = 65.9ms. \quad (25)$$

Compared to single-threaded processing, the time cost is reduced from 123.1 ms to 65.9 ms. However, the time latency is still higher than the waiting time between two frames, i.e., 33ms. According to Eq.(25), we can find that the time latency is determined by the time cost of *Process Frame* thread, where the processing time of feature extraction and tracking takes the largest proportion, i.e., $73\%$. Therefore, it is better to optimize the operators of feature extraction and tracking.

As aforementioned in Section 5.3.3, feature extraction only requires local information, i.e., for each pixel, feature extractor only uses the pixels around it to classify whether it is a feature point. Therefore, we can split a frame into blocks without affecting the locality of feature extraction, and then use multiple threads to extract features for each block separately. Specifically, as shown in Fig. 11, without loss of generality, we first divide the frame into sixteen blocks, and for each block, we calculate the color variance inside the block to measure its salience. A larger color variance implies higher salience. Hence, according to the value of color variance, we then divide the blocks into three categories, i.e, low salience block, medium salience block and high salience block. For low, medium and high salience block, we extract 10, 20 and 40 features respectively. Finally, these sixteen blocks are distributed to four threads sequentially from left to right and then from top to bottom. In order to achieve load balancing, we let each thread

process different number of blocks, and make the number of features that each thread needs to extract are roughly the same. By adopting multiple threads, the time cost of feature extraction and tracking is reduced from 48.1 ms to 14.8 ms. And the time cost of processing one original frame is reduced from 65.9 ms to 32.6 ms, which is lower than the waiting time between two frames, i.e., 33 ms. Unit now, the time latency meets the real-time requirement.

## 7 Discussion

**Translation Estimation:** As described in Section 5.3.3, we are expected to select the feature point pairs generated from fixed 3D points to calculate the translation. If the camera takes fast moving objects, our method may fail to estimate the camera translation, due to the failure of extracting and tracking enough feature points. This is also a common challenge faced by many CV-based methods [1], [15], [29], [30]. To tackle this issue, we can introduce the Extended Kalman Filter (EKF) to further fuse the IMU-based method and CV-based method, to mitigate this problem. As we know, the IMU-based method has the problem of measurement noise and error accumulation. However, by fusing the CV-based method and introducing the EKF, we may quantify and calibrate the measurement noise in acceleration, and then estimate the translation with higher accuracy.

**Frame Warping:** As described in Section 6.3.3, when we perform frame warping to obtain stabilized frames, because the position of each frame is changed from the original moving path to the smoothed moving path, there are some missing areas within stabilized frames. In order to hide these missing areas, we crop stabilized frames. This operation causes the loss of information at video boundaries. To tackle this issue, we can use inpainting algorithms [14], [31], [32] to perform full-frame video stabilization. The idea of image inpainting is using the pixel information from the surrounding areas and nearby frames to complete the missing pixels. For example, according to the method described in [31], we can search the most similar patch in color space among the nearby frames, and use it to complete the missing areas.

## 8 Performance Evaluation

### 8.1 Experimental Setup

We have implemented a prototype system for video stabilization using an Android phone (Lenovo PHAB2 Pro), which is embedded with the inertial sensors including an accelerometer and a gyroscope. In the experiment, we used the Android phone to capture the 1080p videos, where the sampling rate of the camera is 30 frames per second and the sampling rate of the inertial sensors is 200Hz.

### 8.2 Evaluate the Performance of Motion Estimation

#### 8.2.1 Accuracy

We first evaluated the accuracy of camera motion estimation. We compared our solution with three baseline solutions, i.e., the Gyro-based solution which estimates the rotation via the gyroscope, the Acc-based solution which estimates the translation via the accelerometer, and the CV-based solution called eight-point-algorithm [11]. We use the OptiTrack system [22] to capture the ground-truth of the camera motion. For the rotation estimation, as shown in Fig. 12(a), the Gyro-based solution and our solution outperform the CV-based solution. Specifically, as the jitter's range is increasing, the estimation error of the Gyro-based solution and our solution is increasing from 0.7° to 1.7°, whereas the rotation error of CV-based solution is increasing from 2.7° to 4.6°. For the translation estimation, as shown in Fig. 12(b), the estimation error of Acc-based solution is rather large, whereas the estimation error of the CV-based solution and our solution is rather small. Moreover, by fusing the IMU-based method and the CV-based method, our solution further outperforms the CV-based solution in the translation estimation. Specifically, as the jitter's range is increasing, the estimation error of CV-based solution is increasing from 2.5cm to 4.0cm, whereas the estimation error of our solution is always less than 2.3cm.

#### 8.2.2 Time Efficiency

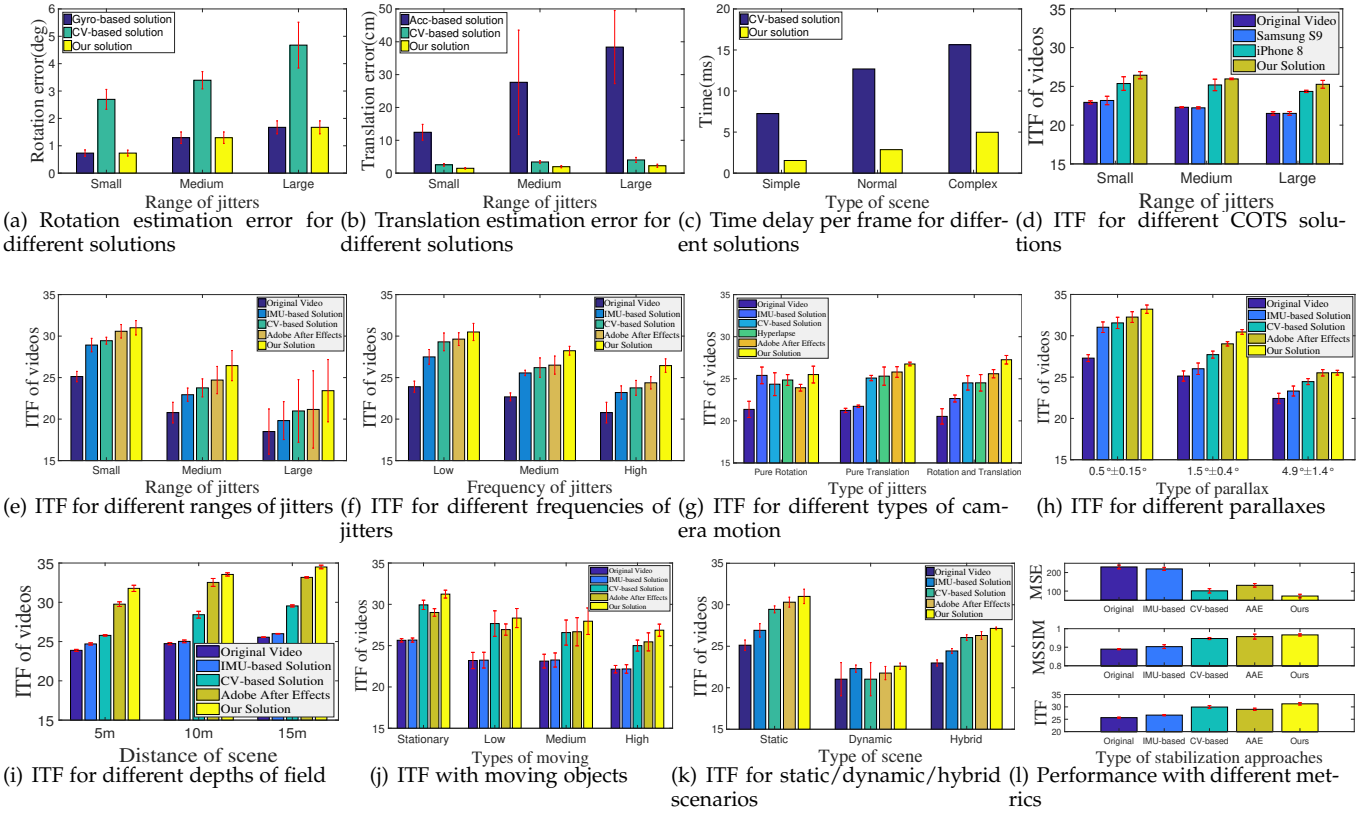
We then evaluated the time delay of camera motion estimation per frame (one frame lasts 33ms in our setting). We compared our solution with a traditional CV-based solutions called eight-point-algorithm [11], which estimates the rotation and translation simultaneously, by using eight pairs of feature points. We applied our solution and the CV-based solution in 30 videos, which are classified into three categories based on scene type, i.e., simple, normal, and complex. For simple, normal and complex scene, we extracted 50∼100, 200∼300 and 500∼600 pairs of features points, respectively, for camera motion estimation. As shown in Fig. 12(c), our solution using fewer feature point pairs clearly outperforms the CV-based solution. Specifically, as the scene varies from simple to complex, the time delay for our solution increases from 1.5ms to 5ms per frame, whereas the time delay for CV-based solution increases from 7ms to 16ms per frame.

### 8.3 Evaluate the Performance of Video Stabilization

To evaluate the performance of video stabilization, we used the metric of Inter-frame Transformation Fidelity (ITF) [29], i.e., $ITF = \frac{1}{N_F-1}\sum_{k=1}^{N_F-1} PSNR(k)$. Here, $N_F$ is the number of video frames, while PSNR(k) is the Peak Signal-to-Noise Ratio between two consecutive frames $F_k$ and $F_{k+1}$. PSNR measures how an image is similar to another one, as the consecutive frames in the stabilized video should be more continuous than the original video, thus ITF can be used to evaluate the stabilization degree of a video. Hence, larger ITF implies that better performance is achieved for video stabilization. We compared our solution with three baseline solutions, i.e., 1) IMU-based solution: it estimates the camera rotation via the gyroscope and estimates the camera translation via the accelerometer [12]; 2) CV-based solution: it estimates the camera rotation and translation via CV-based method [13], [15]; 3) Warp Stabilizer: it is built upon CV-based method [30] and adopted in the state-of-art commercial offline system Named Adobe After Effects CC 2018.

#### 8.3.1 Performance comparison with COTS solutions

We compared our solution with the optical image stabilization method, which is widely adopted in commercial mobile devices, e.g., iPhone 8 and Samsung S9. Specifically, by




(a) Rotation estimation error for different solutions (b) Translation estimation error for different solutions (c) Time delay per frame for different solutions (d) ITF for different COTS solutions

(e) ITF for different ranges of jitters (f) ITF for different frequencies of jitters (g) ITF for different types of camera motion (h) ITF for different parallaxes

(i) ITF for different depths of field (j) ITF with moving objects (k) ITF for static/dynamic/hybrid scenarios (l) Performance with different metrics

Fig. 12. The experiment results

changing the ranges of jitters, we evaluated the performance of each approach while shooting the same scene. As shown in Fig.12(d), the video is stabilized by each approach. Moreover, the ITF index of our solution, i.e., 26.4, 26.0 and 25.3 for small, medium and large jitters, respectively, is higher than that of the other two approaches. It indicates that our solution outperforms the existing approaches adopted in commercial devices.

### 8.3.2 Performance comparison with different ranges and frequencies of jitters

We evaluated the *ITF* by varying the range of movement jitters from small (5°, 1cm), medium (10°, 3cm) and large (15°, 5cm) ranges. As shown in Fig. 12(e), it is found that, as the range of movement jitters increases, the performance of all solution all slightly decreases. We further evaluated the *ITF* by varying the frequency of movement jitters from low (1HZ), medium (3HZ) and high (4∼5HZ) frequencies. As shown in Fig. 12(f), it is found that, as the frequency of movement jitters increases, the performance of all solution all slightly decreases. Nevertheless, in all situations, our solution achieves the best performance, whereas the IMU-based solution achieves the worst performance.

### 8.3.3 Performance with different types of camera motion

We evaluated the *ITF* by varying the types of camera motion in three modes, i.e., pure rotation, pure translation, and hybrid motion (rotation and translation). As shown in Fig. 12(g), when the camera motion is pure rotation, the IMU-based solution and our solution outperforms the CV-based solution, as the gyroscope can estimate the rotation more accurately than CV-based method. When the camera motion is pure translation or hybrid motion, the CV-based solution achieves better performance than the IMU-based solution, as the accelerometer cannot accurately estimate the translation. Moreover, our solution achieves the best performance. Specifically, for pure translation and hybrid motion, the IMU-based solution increases ITF by 2% and 10%, the CV-based solution increases ITF by 18% and 19%, the Warp Stabilizer at Adobe After Effects increases ITF by 21% and 25%, whereas our solution increases ITF by 26% and 32%, respectively.

We also compared our solution with the Hyperlapse [33], which is designed to create a stabilized time lapse for a long video. As a CV-based solution, the Hyperlapse shows similar performance to CV-based solution [15] under different camera motions, and it is better than the CV-based solution because of its additional rolling-shutter correction. Overall, our solution achieves better performance than Hyperlapse. Specifically, our solution achieves 3%, 6%, and 13% better performance in ITF than Hyperlapse, respectively.

### 8.3.4 Performance with different parallaxes

We evaluated our solution in the scenarios with different parallaxes. Parallax means the difference in the apparent position of an object viewed along two different lines of sight, and is measured by the angle of inclination between those two lines. In the experiments, we evaluated the performance of video stabilization by varying the average distance between the shooting target and the camera from 15cm to 150cm. We then evaluate the average parallax for each frame (33ms per frame) according to the average moving speed of the camera. During the process of video shooting, since nearby objects have a larger parallax than more distant objects when observed from different positions, hence, the parallax is varied from $0.5°$ to $4.9°$ per frame on average.

Note that, when the parallax is small, e.g., $0.5°$ per frame, all solutions including the IMU-based solution, CV-based solution, Warp Stabilizer and our solution achieve good performance in regard to ITF. When the parallax is large, e.g., $4.9°$ per frame, the ITF is decreased to a certain extent for all solutions. This implies the video stabilization performance is sensitive to the parallax. Nevertheless, our solution achieves the best performance among all solutions, even if when the parallax is $4.9°$ per frame, the ITF for our solution is greater than 25.

#### 8.3.5 Performance with different depths of field

We evaluated our solution in the scenarios with different depths of field. Here, the depth means the distance between the object and the camera. Different depths indicate that the feature points will be extracted from different distances. In the experiment, we vary the depth between the shooting target and the camera from 5m to 15m, as shown in Fig. 12(i). The experimental results show that, for all three cases, the videos are effectively stabilized by our solution with different depths. Moreover, our solution outperforms the other video-stabilization approaches.

#### 8.3.6 Performance comparison with moving objects

In order to evaluate how the moving objects in the video frames affect the system performance, we evaluate the ITF of video shoot by changing the states of moving objects, i.e., keeping stationary and moving in low, medium or high speed. Here, "stationary" means that no moving objects appear in video frames. In regard to the other three scenarios, three volunteers are asked to walk at 0.75m/s, 1.57m/s and run at 2.85m/s, respectively. As shown in Fig. 12(j), our solution outperforms the other solutions in regard to the ITF metric. Besides, as the moving speed increases, the ITF of our solution gradually decreases from 32 to 27. Nevertheless, the ITFs are all greater than 25 in four scenarios, which are not greatly affected by the moving speed. This is mainly because the moving objects usually appear in the foreground in daily video shootings, while the background keeps stationary. Even if some objects move in high speed, the number of feature points extracted from stationary background is enough to stabilize the video with fairly good performance.

#### 8.3.7 Performance with static/dynamic/hybrid scenarios

We evaluated the *ITF* in three different scenarios of video shoot, i.e., 1) static scenario: most of the target points are stationary, 2) dynamic scenario: most of the target points are dynamically moving, 3) hybrid scenario: the target points can be stationary or dynamically moving. As shown in Fig. 12(k), for the static or hybrid scenario, the CV-based solution and our solution achieve better performance than the IMU-based solution, since they can use enough feature points to accurately estimate the camera motion. However, for the dynamic scenario, there are not enough feature points and long feature point tracks, the CV-based solution can not accurately estimate the camera motion, thus it achieves the worst performance. Nevertheless, our solution achieves the best performance among four solutions. Specifically, the IMU-based solution increases ITF by 6%, the CV-based solution increases ITF by 1%, the Warp Stabilizer at Adobe After Effects increases ITF by 3%, whereas our solution increases ITF by 7%.

#### 8.3.8 Performance with different metrics

We evaluated the quality of video stabilization with different metrics. Specifically, we choose the MSE [34], MSSIM [35] and ITF to evaluate four stabilization approaches. MSE [34] denotes the mean square error of a video, i.e. MSE $= \frac{1}{N_F-1}\sum_{i=1}^{N_F-1}\text{absdiff}(F_{i+1},F_i)^2$. Here, absdiff means the absolute difference between two consecutive frames. Clearly, smaller MSE implies that better performance is achieved for video stabilization. MSSIM [35] is mean structural similarity, which separates the task of similar measurement into three comparisons: luminance, contrast and structure. Larger MSSIM means better video quality. As show in Fig. 12(l), the MSE of original videos is largely reduced by our solution (from 230 to 73), the MSSIM is increased from 0.89 to 0.97, and our solution increases the ITF by 22%. Overall, according to three different evaluation metrics, our solution achieves the best performance in all cases.

## 9 CASE STUDY

### 9.1 Video Stabilization for Different Human Movements

To evaluate the video stabilization performance under real scenarios, in the following experiments, the human subject used handheld devices to shoot videos while he/she was walking, climbing or riding. Then we compared our solution with the optical image stabilization (OIS) technology adopted in the commercial smartphone, e.g., SAMSUNG Galaxy Note8 (A video demo can be found in the supplemental video).


Fig. 13. Human movements

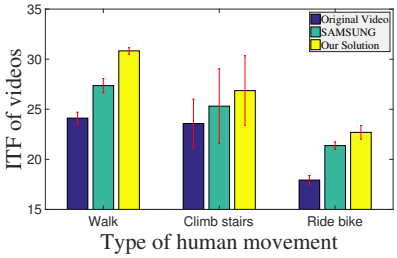



Fig. 14. The experiment results

As shown in Fig. 13, the volunteers were told to use two mobile phones, i.e., Lenovo PHAB2 Pro and SAMSUNG Galaxy Note8, to capture videos respectively, when they were climbing stairs in indoor environment, or walking, riding a bike in outdoor environment. With the Lenovo

mobile phone, we obtained the videos (Original video for short) not been processed by any video stabilization solutions. With the SAMSUNG mobile phone, we obtained the videos (SAMSUNG video for short) been processed by SAMSUNG's optical image stabilization techniques. For each type of human movement, we captured 20 videos separately. After that, we performed our video stabilization solution for videos captured by Lenovo phone and got stabilized videos (Our video for short). We then compared the three types of videos, i.e., Original video, SAMSUNG video and our video, by using the same metric *ITF* mentioned in Section 8.3. As shown in Fig. 14, whatever the human movement was, our solution achieved the best performance. On average, compared to original video, SAMSUNG video increased ITF by 13%, whereas our video increased ITF by 22%.

### 9.2 Video Stabilization for Google Glass

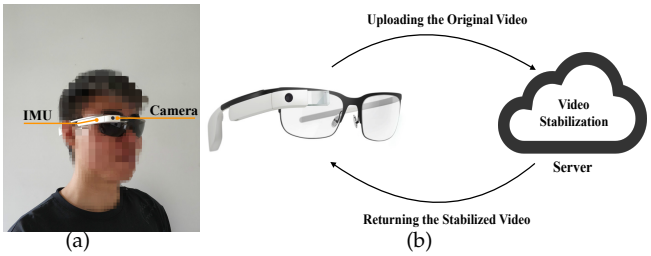
We had also implemented a video stabilization system on Google Glass. Due to the limitations of Google Glass in hardware performance, different from smartphones, we implemented the video stabilization system for Google Glass based on the C/S architecture. Specifically, in the experiment, we first used the Google Glass to capture the 720p videos, where the sampling rate of the camera was 30 frames per second. Moreover, during the process of video shoot, we used the built-in accelerometer and gyroscope to capture the linear acceleration and the angular rate at a frequency of 100Hz, respectively. Then as shown in Fig.15, we uploaded the video captured by Google Glass to the server. The server performed our video stabilization system to obtain the stabilized video, and returned the stabilized video to Google Glass. Finally, we evaluated the time efficiency and performance of video stabilization for Google Glass.

#### 9.2.1 Time Efficiency

There are three main stages in the video stabilization system of Google Glass, i.e., uploading the original video to the server, waiting for the server to process the original video, and downloading the stabilized video from the server. The stages are respectively called 'Uploading', 'Waiting' and 'Downloading' for short. We first set the duration of the original videos to 30 seconds, and without loss of generality, we used the Lenovo PHAB2 Pro smartphone as the server. Then we measured the time cost of processing one video in each stage. We repeated the measurement for 50 times to get the average time cost. According to the measurement shown in Fig. 16(a), the time cost in three stages was 13.31, 21.51 and 16.21 s, respectively. Thus the average time cost of processing a 30 seconds video was 51.03 seconds. This case study shows that, our video stabilization method can be effectively executed on the resource-limited wearable devices, such as Google Glass.

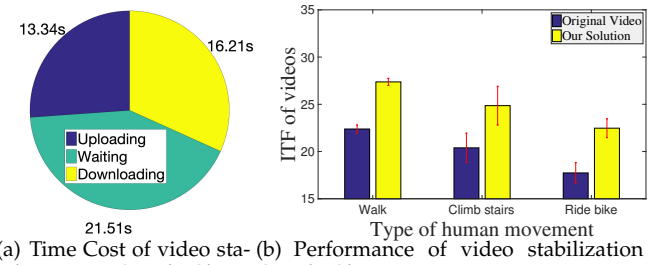
#### 9.2.2 Performance of Video Stabilization

To evaluate the performance of video stabilization in Google Glass, the volunteers were told to wear Google Glass to shoot videos while they were walking, climbing or riding. Then we compared the original videos and the stabilized videos by using the same metric *ITF*. As shown in Fig.16(b), on average, compared to original video, stabilized video increased ITF by 24%. The experiment results show that, our video stabilization method can effectively tackle the sudden movement from the head.




(a) (b)

Fig. 15. The video stabilization system for Google Glass




(a) Time Cost of video stabilization in Google Glass (b) Performance of video stabilization in Google Glass

Fig. 16. The experiment results

## 10 Conclusion

In this paper, we present a video stabilization solution in mobile devices via inertial-visual state tracking. By fusing the IMU-based method and the CV-based method, our solution is robust to the fast movement and violent jitters, moreover, it greatly reduces the computation overhead in video stabilization. In the context of recent visual-inertial based video stabilization methods [12], [13], our solution is able to estimate the translation and rotation in a more accurate manner, and meets the real time requirement for online processing, by directly reducing the number of undetermined degrees of freedom from 6 to 3 for CV-based processing. We implemented a prototype system on smart glasses and smart phones, and evaluated the performance under various real scenarios. The experiment results show that our solution achieves 32% better performance than the state-of-art solutions in regard to video stabilization. Moreover, the average processing time latency is 32.6ms, which is lower than the conventional inter-frame time interval, i.e., 33ms, and thus meets the real-time requirement for online processing.

## Acknowledgments

This work is supported in part by National Natural Science Foundation of China under Grant Nos. 61872174, 61832008, 61802169, 61902175; The Key R&D Program of Jiangsu Province under Grant No. BE2018116; JiangSu Natural Science Foundation under Grant No. BK20180325, BK20190293. This work is partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization. Lei Xie is the corresponding author.

## REFERENCES


[1] S. Liu, L. Yuan, P. Tan, and J. Sun, "Steadyflow: Spatially smooth optical flow for video stabilization," in *Proc. of IEEE CVPR*, 2014.

[2] S. Ji, M. Zhu, Y. Lei *et al.*, "Video stabilization with improved motion vector estimation," *Opt Precision Eng*, vol. 23, no. 5, pp. 1458–1465, 2015.

[3] J. Kopf, M. Cohen, and R. Szeliski, "First-person hyper-lapse videos," *ACM Transactions on Graphics*, vol. 33, no. 4, p. 78, 2014.

[4] T. Lee, Y. Lee, and B. Song, "Fast 3d video stabilization using roi-based warping," *Journal of Visual Communication and Image Representation*, vol. 25, no. 5, pp. 943–950, 2014.

[5] Z. Jiang, J. Han, C. Qian, W. Xi, K. Zhao, H. Ding, S. Tang, J. Zhao, and P. Yang, "Vads: Visual attention detection with a smartphone," in *Proc. of IEEE INFOCOM*, 2016.

[6] Z. Li, M. Li, P. Mohapatra, J. Han, and S. Chen, "itype: Using eye gaze to enhance typing privacy," in *Proc. of IEEE INFOCOM*, 2017.

[7] W. Aguilar and C. Angulo, "Real-time model-based video stabilization for microaerial vehicles," *Springer Neural Processing Letters*, vol. 43, no. 2, pp. 459–477, 2016.

[8] A. Karpenko, D. Jacobs, J. Baek, and M. Levoy, "Digital video stabilization and rolling shutter correction using gyroscopes," *CSTR*, vol. 1, p. 2, 2011.

[9] G. Hanning, N. Forslöw, P. Forssén *et al.*, "Stabilizing cell phone video using inertial measurement sensors," in *Proc. of IEEE ICCV Workshops*, 2011.

[10] S. Bell, A. Troccoli, and K. Pulli, "A non-linear filter for gyroscope-based video stabilization," in *Proc. of ECCV*, 2014.

[11] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[12] A. Karpenko, "Systems and methods for digital video stabilization via constraint-based rotation smoothing," Patent 9 071 756, June, 2015. [Online]. Available: http://www.freepatentsonline.com/9071756.html

[13] K. V. E. I. Grundmann, Matthias, "Cascaded camera motion estimation, rolling shutter detection, and camera shake detection for video stabilization," Patent 9 374 532, June, 2016.

[14] Y. Matsushita, E. Ofek, W. Ge, X. Tang, and H. Shum, "Full-frame video stabilization with motion inpainting," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 28, no. 7, pp. 1150–1163, 2006.

[15] F. Liu, M. Gleicher, H. Jin, and A. Agarwala, "Content-preserving warps for 3d video stabilization," in *Proc. of ACM TOG*, 2009.

[16] H. Ovrén and P. Forssén, "Gyroscope-based video stabilisation with auto-calibration," in *Proc. of IEEE ICRA*, 2015.

[17] C. Jia and B. Evans, "3d rotational video stabilization using manifold optimization," in *Proc. of IEEE ICASSP*, 2013.

[18] S. Zhou, F. Fei, G. Zhang, Y. Liu, and W. J. Li, "Hand-writing motion tracking with vision-inertial sensor fusion: calibration and error correction." *Sensors*, vol. 14, no. 9, pp. 15 641–15 657, 2014.

[19] X. Yang, X. Si, T. Xue, L. Zhang, and K. Cheng, "Vision-inertial hybrid tracking for robust and efficient augmented reality on smartphones," in *Proc. of ACM ICME*, 2015.

[20] C. Jia and L. Brian, "Probabilistic 3-d motion estimation for rolling shutter video rectification from visual and inertial measurements," in *Proc. of IEEE MMSP*, 2012.

[21] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[22] OptiTrack, http://www.optitrack.com/.

[23] O. Woodman, "An introduction to inertial navigation," University of Cambridge, Computer Laboratory, Tech. Rep., 2007.

[24] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Proc. of Springer ECCV*, 2010.

[25] K. K. E. Rublee, V. Rabaud and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Proc. of IEEE ICCV*, 2012.

[26] M. Norouzi, D. Fleet, and R. Salakhutdinov, "Hamming distance metric learning," in *Proc. of NIPS*, 2012.

[27] S. C. Reina, A. Solin, and J. Kannala, "Robust gyroscope-aided camera self-calibration," in *Proc. of IEEE FUSION*, 2018.

[28] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[29] J. Xu, H. Chang, S. Yang, and M. Wang, "Fast feature-based video stabilization without accumulative global motion estimation," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 3, pp. 993–999, 2012.

[30] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala, "Subspace video stabilization," *Acm Transactions on Graphics*, vol. 30, no. 1, pp. 1–10, 2011.

[31] Y. Wexler, E. Shechtman, and M. Irani, "Space-time video completion," in *Proc. of IEEE CVPR*, 2004.

[32] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, "High-resolution image inpainting using multi-scale neural patch synthesis," in *Proc. of IEEE CVPR*, 2017.

[33] N. Joshi, W. Kienzle, M. Toelle, M. Uyttendaele, and M. F. Cohen, "Real-time hyperlapse creation via optimal frame selection," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 63:1–63:9, Jul. 2015.

[34] A. M. Eskicioglu and P. S. Fisher, "Image quality measures and their performance," *IEEE Transactions on Communications*, vol. 43, no. 12, pp. 2959–2965, Dec 1995.

[35] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

**Fei Han** received his B.S. degree in information security from Nanjing University of Science and Technology, China in 2016 and received his M.S. degree in the Department of Computer Science and Technology at Nanjing University in 2019. His research interests include mobile sensing, and wearable computing.

**Lei Xie** received his B.S. and Ph.D. degrees from Nanjing University, China in 2004 and 2010, respectively, all in computer science. He is currently an associate professor in the Department of Computer Science and Technology at Nanjing University. He has published over 80 papers in ACM/IEEE Transactions on Networking, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, ACM Transactions on Sensor Networks, ACM MobiCom, ACM UbiComp, ACM MobiHoc, IEEE INFOCOM, IEEE ICNP, IEEE ICDCS, etc.

**Yafeng Yin** received her B.E. degree in network engineering from Nanjing University of Science and Technology, and received her Ph.D. degree in computer science from Nanjing University, China in 2011 and 2017, respectively. She is currently an assistant professor in the Department of Computer Science and Technology at Nanjing University. Her research interests include human activity recognition, mobile sensing, wearable computing, etc.

**Hao Zhang** received his B.S. degree in Computer Science and Technology from Nanjing University, China in 2018 and currently is a first year graduate student of Department of Computer Science and Technology in Nanjing University. His research interests include mobile sensing, and wearable computing.

**Guihai Chen** received the BS degree in computer software from Nanjing University, China, in 1984, the ME degree in computer applications from Southeast University in 1987, and the PhD degree in computer science from the University of Hong Kong in 1997. He is currently a professor of the Department of Computer Science, Nanjing University. He had been invited as a visiting professor by many foreign universities including the Kyushu Institute of Technology, Japan, in 1998, University of Queensland, Australia, in 2000, and Wayne State University, USA, from September 2001 to August 2003. He has a wide range of research interests with focus on sensor networks, peer-to-peer computing, high-performance computer architecture, and combinatorics. He is a member of the IEEE.

**Sanglu Lu** received her B.S., M.S. and Ph.D. degrees from Nanjing University, China in 1992, 1995 and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology at Nanjing University. Her research interests include distributed computing and pervasive computing.