·Review·

Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality

Jinyu LI[1], Bangbang YANG[1], Danpeng CHEN[2], Nan WANG[2], Guofeng ZHANG[1*], Hujun BAO[1*]

1. *State Key Lab of CAD&CG, Zhejiang University, Hangzhou* 310058*, China*

2. *SenseTime Research, Hangzhou* 311215*, China*

* **Corresponding author,** bao@cad.zju.edu.cn; zhangguofeng@cad.zju.edu.cn
Received: 15 December 2018 **Accepted:** 1 February 2019

Supported by the National Key Research and Development Program of China (2016YFB1001501); NSF of China (61672457); the Fundamental Research Funds for the Central Universities (2018FZA5011); Zhejiang University-SenseTime Joint Lab of 3D Vision.

Citation: Jinyu LI, Bangbang YANG, Danpeng CHEN, Nan WANG, Guofeng ZHANG, Hujun BAO. Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality. Virtual Reality & Intelligent Hardware
DOI: 10.3724/SP.J.2096-5796.2018.0011

Abstract Although VSLAM/VISLAM has achieved great success, it is still difficult to quantitatively evaluate the localization results of different kinds of SLAM systems from the aspect of augmented reality due to the lack of an appropriate benchmark. For AR applications in practice, a variety of challenging situations (e.g., fast motion, strong rotation, serious motion blur, dynamic interference) may be easily encountered since a home user may not carefully move the AR device, and the real environment may be quite complex. In addition, the frequency of camera lost should be minimized and the recovery from the failure status should be fast and accurate for good AR experience. Existing SLAM datasets/benchmarks generally only provide the evaluation of pose accuracy and their camera motions are somehow simple and do not fit well the common cases in the mobile AR applications. With the above motivation, we build a new visual-inertial dataset as well as a series of evaluation criteria for AR. We also review the existing monocular VSLAM/VISLAM approaches with detailed analyses and comparisons. Especially, we select 8 representative monocular VSLAM/VISLAM approaches/systems and quantitatively evaluate them on our benchmark. Our dataset, sample code and corresponding evaluation tools are available at the benchmark website http://www.zjucvg.net/eval-vislam/.

Keywords Visual-inertial SLAM; Odometry; Tracking; Localization; Mapping; Augmented reality

1 Introduction

In recent years, AR (Augmented Reality) technology has developed rapidly and become more and more mature. International IT giants Apple, Google and Microsoft launched the mobile AR software development platforms (i.e. ARKit and ARCore), as well as AR helmet display HoloLens, respectively. In particular, with the popularization of mobile communications and intelligent terminals, AR technology has gradually expanded from high-end applications such as industrial production, medical rehabilitation, and urban management to electronic commerce, cultural education, digital entertainment and other popular

applications, and has become a basic tool for people to recognize and transform the world.

AR is a kind of technique which can seamlessly fuse virtual objects or information with real physical environment together and present the compositing effect to the user. 3D registration (i.e., accurate pose registration/localization) is the key fundamental technique for achieving immersive AR effects. Early AR solutions like ARToolkit[1] use fiducial markers for pose registration, which limits AR objects to specific places. Later on, some camera tracking methods based on natural features are developed. The most important 3D registration technique is SLAM (Simultaneous Localization and Mapping), which can real-time recover the device pose in an unknown environment. According to the use of different sensors, SLAM techniques can be divided into VSLAM (visual SLAM), VISLAM (visual-inertial SLAM), RGB-D SLAM and so on. There are already some general reviews about them[1−5]. In this paper, we mainly review the publicly available VSLAM and VISLAM approaches with quantitative evaluation.

SLAM technology originates from the field of robotics. Over the past few decades, many researchers have studied its modeling, optimization, and engineering but with simplified assumptions. In AR, new challenges arise: applications require rapid initialization with an accurate scale. Accidental jittering and pure rotational motion often occur. The measurements of consumer-level sensors are easily polluted by noise and drift. Hardware synchronization is not easy to achieve. Unfortunately, there is no prior evaluation sufficiently addressing these problems, making it hard to quantitatively compare the performance of different VISLAM systems for AR applications.

Mobile devices (i.e., smartphones), generally have cameras and IMU (inertial measurement unit) sensors, which are ideal for localization with VISLAM technology. For example, Apple's ARKit and Google's ARCore both use VISLAM for 3D pose registration. Although there are already some datasets like EuRoC[6] and KITTI[7], they do not aim at evaluating AR effects. Different from other applications, good AR experience requires that the SLAM system can handle kinds of complex camera motion, allowing easy use for a novice home user. For example, many AR applications need rapid initialization with an accurate scale. The user may freely move the AR device and encounter a variety of unexpected situations, such as occasional camera jittering, camera lost, rapid camera motion with severe motion blur, and dynamic interference. Unfortunately, none of the existing benchmarks specifically address these issues and establish corresponding evaluation criteria.

In this paper, we publish a new monocular VSLAM/VISLAM benchmark for evaluating SLAM performance in AR applications. Specifically, we implemented a full pipeline for visual-inertial data acquisition on mobile phones. We define a series of evaluation criteria for SLAM in AR. In addition, we review the existing mainstream monocular VSLAM/VISLAM approaches, with detailed analysis and comparisons. We perform a quantitative evaluation of public monocular VSLAM/VISLAM systems on our benchmark[2].

2 Basic theory of VSLAM and VISLAM

VISLAM is a technology which uses visual and inertial sensors to infer the device's pose and scene map in an unknown environment. In contrast, VSLAM uses only visual sensor (i.e., monocular or multiple cameras) to estimate the camera pose and scene structure according to multi-view geometry theory[8,9]. Inertial information (i.e., linear acceleration and rotational velocity measured by the IMU sensor) is modeled by inertial navigation and can make up for the defects of visual information. So by fusing visual

[1] https://github.com/artoolkit

[2] The benchmark website is at http://www.zjucvg.net/eval-vislam/

and inertial information, a VISLAM system generally can be more robust than a VSLAM system in the same situations.

VSLAM can be regarded as the online version of structure-from-motion (SfM), which is also a key problem in computer vision. Given the input multiple images or video sequences, SfM can automatically recover the camera poses and the 3D points of matched features. The camera motion state of image i can be denoted as $\mathbf{C}_i = (\mathbf{R}_i, \mathbf{p}_i)$, where $\mathbf{R}_i$ and $\mathbf{p}_i$ are the rotation matrix and camera position of image i respectively. As illustrated in , a 3D point $\mathbf{X}_j$ can be projected to image i as:

$$\mathbf{x}_{ij} = h(\mathbf{C}_i, \mathbf{X}_j) = \pi(\mathbf{K}\mathbf{R}_i^{\top}(\mathbf{X}_j - \mathbf{p}_i)), \tag{1}$$

where $\mathbf{K}$ is the camera intrinsic matrix, and $\pi(x, y, z) = (x/z, y/z)^{\top}$ is the projection function. Eq. (1) relates the 3D point $\mathbf{X}_j$ in the world coordinate to a 2D point $\mathbf{x}_{ij}$ on the image I_i. In reality, the matching is not perfect. Let $\tilde{\mathbf{x}}_{ij}$ be the actual keypoint on the image, the reprojection error of $\mathbf{X}_j$ on image i can be computed as $\epsilon_{ij} \equiv \mathbf{x}_{ij} - \tilde{\mathbf{x}}_{ij}$. For m images and n 3D points, we can simultaneously solve the camera poses and 3D points by minimizing the following energy function:

$$\underset{\mathbf{C}_1 \cdots \mathbf{C}_m, \mathbf{X}_1 \cdots \mathbf{X}_n}{\arg\min} \sum_{i=1}^{m} \sum_{j=1}^{n} \| h(\mathbf{C}_i, \mathbf{X}_j) - \mathbf{x}_{ij} \|^2 \tag{2}$$

This optimization is called bundle adjustment (BA)[10], which is the core component of SfM and VSLAM.

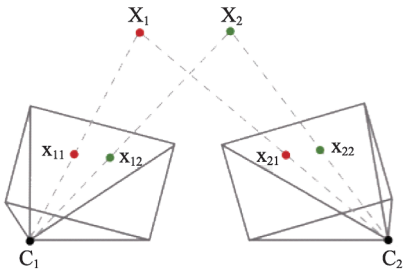

Figure 1 Multiple view geometry.

For monocular VSLAM, the absolute scale cannot be solved by minimizing reprojection error. Fortunately, IMU sensor can give metric measurements, so we can recover the absolute scale by integrating and optimizing IMU data. Typically, the IMU sensor measures the rotational velocity $\boldsymbol{\omega}(t)$ and the linear acceleration $\mathbf{a}(t)$ with respect to its local frame. Its common model[11] is as follows:

$$\begin{cases} \boldsymbol{\omega}(t) = \tilde{\boldsymbol{\omega}}(t) + \mathbf{b}_{\omega} + \mathbf{n}_{\omega} \\ \mathbf{a}(t) = \mathbf{R}_t^{\top}(\tilde{\mathbf{a}}_W(t) - \mathbf{g}) + \mathbf{b}_{\mathbf{a}} + \mathbf{n}_{\mathbf{a}} \\ \dot{\mathbf{b}}_{\omega} = \boldsymbol{\eta}_{\omega} \\ \dot{\mathbf{b}}_{\mathbf{a}} = \boldsymbol{\eta}_{\mathbf{a}} \end{cases} \tag{3}$$

where $\tilde{\boldsymbol{\omega}}(t)$ denotes the true rotational velocity in the IMU's frame, $\tilde{\mathbf{a}}_W(t)$ denotes the true acceleration in the world frame, and $\mathbf{R}_t$ is the rotation matrix of IMU at time t. $\mathbf{n}_{\omega} \sim N(0, \Sigma_{\omega})$ and $\mathbf{n}_{\mathbf{a}} \sim N(0, \Sigma_{\mathbf{a}})$ are the measurement noises of gyroscope and accelerometer, respectively. $\mathbf{b}_{\omega}$ and $\mathbf{b}_{\mathbf{a}}$ are random walk contaminations in the measurements, called drift error. Their random walk noises are $\boldsymbol{\eta}_{\omega} \sim N(0, \Sigma_{\mathbf{b}_{\omega}})$ and $\boldsymbol{\eta}_{\mathbf{a}} \sim N(0, \Sigma_{\mathbf{b}_{\mathbf{a}}})$. Obviously, direct integration of $\boldsymbol{\omega}(t)$ and $\mathbf{a}(t)$ will lead to significant accumulation error. In VSLAM, the accumulation error can be eliminated by loop closure detection and global optimization like bundle adjustment. VISLAM combines visual and inertial measurements, and can be regarded as the direct extension of VSLAM. So the BA function in VISLAM can be defined as follows:

$$\underset{\mathbf{C}_1, \cdots, \mathbf{C}_m, \mathbf{X}_1, \cdots, \mathbf{X}_n}{\arg\min} \left\{ \sum_{i=1}^{m} \sum_{j=1}^{n} \left\| h\left(\mathbf{C}_i, \mathbf{X}_j\right) - \mathbf{x}_{ij} \right\|_{\Sigma_h^{-1}}^2 + \sum_{i=1}^{m-1} \left\| s\left(\mathbf{C}_i | \boldsymbol{\omega}, \mathbf{a}\right) \ominus \mathbf{C}_{i+1} \right\|_{\Sigma_m^{-1}}^2 \right\} \tag{4}$$

The new term $s(\mathbf{C}_i | \boldsymbol{\omega}, \mathbf{a})$ in Eq. (4) represents the pose prediction of $\mathbf{C}_{i+1}$ based on $\mathbf{C}_i$ and the measurements of $\boldsymbol{\omega}(t)$ and $\mathbf{a}(t)$. This is usually achieved by iteratively integrating the IMU measurements into the current pose prediction. Another method for fusing the inertial measurements into VSLAM is to summarize a group of sequential IMU readings into one single pre-integrated IMU measurements, making it convenient to incorporate the bias updates during optimization, such as [12, 13]. The binary operator

gives the error between the prediction and the real value, typically via lie-algebra. Σ_h, Σ_m are covariance matrices modeling the uncertainty of each error. In this way, it can give the best estimation for camera poses which fuses camera and IMU. Because velocity and biases are also involved in the prediction s, we need to solve camera poses, velocities, and IMU biases together in Eq. (4).

3 Representative monocular VSLAM/VISLAM approaches

As we know, SLAM systems can solve the states by filtering or optimization. Based on this, the SLAM methods can be divided into filtering-based methods and optimization-based methods. The visual information used for tracking may also be quite different. Some methods use keypoint matching and optimize the reprojection error. Some other methods use image pixels directly and minimize the photometric error. In this section, we introduce some of representative monocular VSLAM/VISLAM approaches.

3.1 Filtering-based SLAM

MonoSLAM[14] is one of the earliest monocular VSLAM systems. Since it solves the camera pose using the extended Kalman filter, it is a filtering-based SLAM system. For the Kalman update step, the observation used is the reprojection from the standard pinhole model.

The EKF in MonoSLAM gives the maximum-a-posteriori estimation to all 3D points and the latest camera pose. From a modern perspective, it repeatedly marginalizes out old camera states. In this way, the number of states can be limited to $O(N)$ size, where N is the number of landmarks. Therefore, the total computation cost is bounded. It also reveals one of the main drawbacks of filtering-based methods: EKF usually cannot give the global optimum estimation to the camera state. Premature marginalization of this sub-optimal state will introduce permanent error to the system, resulting in large drift. In MonoSLAM, its marginalization scheme also builds a dense covariance matrix. Each EKF iteration needs to take $O(N^3)$ time, making it intractable for processing lots of map points.

As another early Kalman filter based SLAM method, MSCKF[15] used a different way to estimate camera states. They kept a sliding window of M frames. The state vector of MSCKF contains the poses of M frames and the latest IMU states. To avoid including 3D points in state vector, MSCKF triangulates points from current camera states for updating the filter estimation, and then marginalize them out immediately. Different from MonoSLAM, MSCKF uses IMU to estimate the pose of the new frame. However, since there are no relocalization or loop-closure modules, it is actually visual-inertial odometry (VIO). By using the sliding window, the camera states in MSCKF will be refined many times before they are marginalized out. Also, the size of the state only depends on the size of the sliding window, so each update in MSCKF takes only $O(M^3)$ time. Hence, MSCKF can track in a wide area in real-time, while having relatively small drift. In its later extension, MSCKF 2.0[16] investigated the observability of the system. They found 4 dimensions in camera states that are unobservable. Noise in these dimensions can introduce additional error. So they used first-estimate-Jacobians[17] to avoid leaking errors into these extra dimensions. There are other works addressing the problem of observability and consistency[18-21]. For example in [21], the linearization points used to calculating Jacobians are selected under observability constraints, then a variant of the EKF is used to remedy the consistency. MSCKF 2.0 has been used in mobile AR products, due to its limited computation demand.

3.2 Optimization-based SLAM

Filtering-based SLAM systems are inevitably suffering from accumulation error. As investigated,

optimization-based methods can have superior accuracy over filtering-based ones[22]. When there are visual-loops, additional constraints can be made in the optimization to connect the non-consecutive overlapping frames, thus eliminating the accumulation error. However, the computational cost of the global optimization will grow rapidly along with the increasing frames. Existing literature focus on improving the efficiency of the optimization, most of which aims at utilizing the sparsity of the relationship between variables and the locality of the SLAM problem. Early works[23], already proposed to interpret the factorization of information matrix or measurement Jacobian as the elimination progress of the factor graph[24]. And the use of variable reordering heuristics like CHOLMOD[25] and COLAMD[26] dramatically reduces the fill-in during the elimination, thus maintaining the sparsity. Based on these theories, iSAM[27] was proposed which further took advantages of the locality and updated the factorization of the measurement Jacobian incrementally. In order to better combine the variable reordering progress and the incremental factorization, iSAM2[28], furthermore, presented Bayes tree structure to help to analyze the causality. Other methods, like SLAM ++[29,30] and ICE-BA[31], for example, employ incremental Schur complement algorithm, which always eliminates the landmark variables before the camera/IMU variables to minimize fill-in.

PTAM[32] is a ground-breaking VSLAM system which uses keyframe-based optimization. It puts local tracking and global mapping in two parallel threads. In the camera tracking thread, they used a decaying velocity model to predict the camera pose. The pose prediction also helps to project the 3D map points onto the new image. So new keypoints are searched in the neighborhood region of the projections. Given the matching result, they minimize the reprojection error to update the camera pose. Since only the pose is solved, this can be done in real-time. In the other thread, the global mapping is done through bundle adjustment. When camera tracking nominates a good keyframe, it will be added for bundle adjustment. If sparsity is not considered, the computational complexity of bundle-adjustment with M keyframes is $O(M^3)$, which grows over time. Its processing will become very expensive as the map expands. Running as a separate thread can prevent mapping from blocking camera tracking, hence achieving real-time performance. Despite that, the complexity of this global bundle adjustment still imposes limitations on PTAM. In their original paper, the map can contain up to only hundreds of keyframes.

There is another caveat in the original PTAM system: its initialization requires user interaction. During the start-up, the user must select two initial keyframes. Nevertheless, PTAM's parallel tracking and mapping framework has inspired a lot of SLAM systems. Nowadays, almost all keyframe-optimization based SLAM systems use a similar framework.

ORB-SLAM[33,34] is a state-of-the-art SLAM system, which used ORB features throughout its whole system to improve the system robustness. Following PTAM, it puts camera tracking, local mapping and loop closing in three threads.

In PTAM, there is no explicit handling of loop closure or relocalization, and the global map is a soup of keyframes connected by keypoint matches. ORB-SLAM takes steps further. They separated the optimization process into a local-window bundle adjustment and a loop-closing optimization. The local-window bundle adjustment optimizes the latest keyframe and all keyframes that share observations with the latest one. Since it only involves limited frames, the computational cost is bounded. The loop-closing optimization is based on an Essential Graph as:

$$\arg\min_{\{\mathbf{S}_i\}} \sum_{i,j} \left\| \log_{Sim(3)} \left(\mathbf{S}_i \cdot \Delta\mathbf{S}_{ij} \cdot \mathbf{S}_j^{-1} \right) \right\|^2 \tag{5}$$

where $\mathbf{S}_i$, $\mathbf{S}_j$ are the nodes, representing keyframe poses. $\Delta\mathbf{S}_{ij}$ is the edge between node i and j, representing similarity transformation between the corresponding keyframes. Using similarity transforms can help

reduce scale drift, which is a common problem in visual only SLAM. Also, since 3D points are not involved in Eq. (5), the number of variables is reduced, so the overall performance is improved.

To avoid user interaction, ORB-SLAM simultaneously estimates a homography model and an epipolar model and chooses the best one for initializing the first two keyframes. So the system automatically initializes when there is sufficient movement. ORB-SLAM open-sourced its implementation, and has inspired many new works, including a visual-inertial version of ORB-SLAM[35].

OKVIS[36] is another VISLAM system designed to fuse inertial measurements. The core optimization of OKVIS is a sliding-window optimization with both reprojection errors and IMU motion errors. And it uses marginalization to preserve information that goes out of the window. This sliding-window plus marginalization strategy gives good accuracy with bounded computational cost.

VINS-Mono[37] is a robust visual-inertial SLAM system. It is also open-sourced. It has many new highlights comparing to ORB-SLAM and uses marginalization techniques like OKVIS to improve accuracy. It has a robust initialization with scale estimation. The odometry tracking used a two-way marginalization for the local sliding window. And the global pose-graph[38] has only 4DoF for each frame. The resulting system gives a very good estimation to camera and IMU states, as well as the physical scale. A mobile version[39] is also publicly available, which can run smoothly on iPhone 6s.

VINS-Mono initializes in a loosely-coupled way. First, a SfM-based reconstruction is built on several keyframes. Then the poses recovered by SfM are aligned with IMU measurements. This visual-inertial alignment estimates the gyroscope bias, the gravity direction, a rough scale, and all the velocities of the keyframes. These estimations are then used to initialize the system.

The tracking of VINS-Mono consists of a local visual-inertial odometry thread, a relocalization thread, and a global pose-graph optimization thread. For initialization and local odometry tracking, VINS-Mono tracks KLT features with optical flow.

The local visual-inertial odometry thread manages a sliding window of M recent keyframes. Upon the arrival of a new frame, it predicts the frame pose by fusing the keypoint matching and IMU measurements. Then the frame is added into the sliding window for a tightly-coupled bundle-adjustment where the errors being minimized are reprojection error and motion preintegration error. To bound the computational cost, VINS-Mono uses a two-way marginalization strategy: if the second-latest frame inside the window is not a keyframe, its states are marginalized out, and the frame is removed from the window. Otherwise, the oldest frame inside the window is marginalized and removed. By doing this, the information of the removed frame turns into a prior term, and the total number of frames in the window is fixed. Removed keyframes are added to the global pose-graph optimization as a node, and will also be used for relocalization.

The relocalization thread first detects loop closure with DBoW2[40]. When a loop-closing frame is detected, BRIEF feature matching is computed. The feature-matching usually contains outliers, which are filtered based on geometric criteria. Once the feature matching is reliable, the loop-closing frame is added to the local visual-inertial odometry as a constraint. When there are more loop-closing frames, all of them are used as constraints to improve relocalization accuracy.

The global pose-graph optimization thread keeps the historical keyframes. They add sequential edges and loop closure edges between frames. The edge they used captures 3D relative position and 1D relative rotation in the yaw direction, so the optimization is 4DoF for each keyframe. This is reasonable because the other two rotational directions are observable from IMU measurements, and can be estimated during the local-visual-inertial odometry.

3.3 SLAM with direct tracking

The systems introduced before use feature points to provide visual measurements. More specifically, in

their optimization, visual factors are reprojection errors. And they are usually called "feature-based" or "indirect" method. Some other SLAM systems try to minimize measurements based on image appearance like the photometric error. These systems are known as direct methods. Unlike the indirect ones, these systems skip the pre-computation step (e.g., forming visual measurements from feature matching), and directly use the light intensity from the camera as measurements.

Both direct and indirect methods have their advantages and disadvantages. In most cases, indirect methods are more robust to geometric noise, like lens distortions or rolling shutter effects, while direct methods can be sensitive to them. On the other hand, direct methods are more robust to photometric noise because all image regions having intensity gradient are utilized (edges, featureless walls).

Direct Sparse Odometry (DSO) [41] is a state-of-the-art visual odometry algorithm based on direct tracking. DSO uses a sparse and direct formulation proposed by [42], whereas previous works are mostly dense[43,44]. Moreover, DSO uses a fully direct probabilistic model to jointly optimize all model parameters, including geometric structure and camera motion, making it convenient to incorporate other kinds of sensors. Another difference between DSO and other direct method systems is the visual measurement model. DSO proposed a novel visual measurement model that integrates the standard light intensity with exposure time, lens vignetting, and a non-linear response function in order to improve accuracy and robustness.

Like OKVIS, the optimization in DSO is performed in a sliding window of up to N keyframes. When the active set of frames exceeds N, the old camera poses as well as points becoming invisible are removed by marginalization. DSO uses a heuristically designed scoring function to determine which keyframes to remove, in order to keep active keyframes well-distributed in 3D space. Once a keyframe is chosen, all points represented in it are marginalized first, then the frame itself. To keep the sparsity of the problem, DSO employs a suboptimal marginalization strategy where only a part of the residual terms is marginalized. All observations that will affect the sparsity pattern are dropped directly. This is also inspired by OKVIS.

Besides the systems introduced above, there are also other types of SLAM methods like RGB-D based[45,46] or event-based methods[47]. Also, some systems use lines and planes for better regularization[48,49]. The recent development of deep learning also gives birth to some learning-based systems[50−52]. However, only cameras and IMUs are commonly available on mobile phones, while the detection and tracking of lines and planes are relatively expensive. Learning based methods are still not quite ready for being applied in mobile AR applications. So we will only focus on evaluating the feature-based and direct-based monocular VSLAM/VISLAM systems.

4 Visual-inertial dataset

There are already a few datasets and benchmarks[6−7,53−55]. For example, the EuRoC MAV dataset[6] is a hexacopter-based dataset which has 11 sequences captured from 3 scenarios: two rooms and a machine hall. There are stereo images captured at 752×480×20Hz and IMU sensor data captured at 200Hz. Ground-truth poses are obtained from VICON and Leica MS50, with accuracy around 1mm. All the data are hardware synchronized to a common clock. The dataset uses global shutter cameras. It also has good synchronization and high-accuracy ground truth. These characteristics make it very popular among recent VISLAM research. More datasets for VISLAM include the TUM VI benchmark dataset[53], the KITTI Vision benchmark suite[7] and the PennCOSYVIO dataset[54]. However, these datasets are still too ideal for evaluating VISLAM in real applications, especially for mobile AR applications. ADVIO dataset[55] is

perhaps the only dataset so far, which captures data from real mobile phones. Although the dataset comes with ground truth, its accuracy is around a few cm/m according to their paper.

Despite the abundance of datasets and benchmarks, none have been dedicated to test the performance of VISLAM systems for AR applications. In order to fill this gap, we propose to build a new visual-inertial dataset for evaluating SLAM in AR applications. Figure 2 illustrates the scheme of dataset processing. Our visual-inertial dataset is collected with two mobile phones and a VICON motion capture system[3]. Firstly, we gather raw data from devices, as shown in blue blocks. The raw data will be fed into the synchronization/calibration process for spatial/temporal alignment. The ground truth data will be produced utilizing the synchronization and calibration. The calibration data and ground truth data, as shown in red and green blocks, together with raw data, will be presented in the final dataset. In the following subsections, we will introduce the details of our hardware setup, conventions, calibration process, and the dataset organization. Table 1 lists the characteristics of our dataset and the commonly used datasets for comparison.

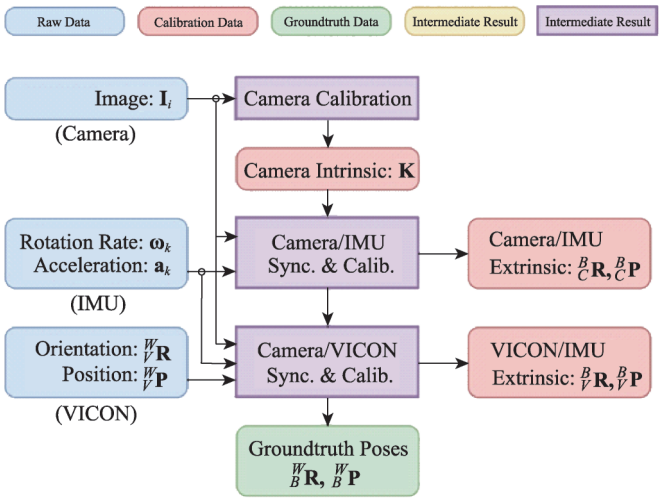

Figure 2 The scheme of dataset processing.

Table 1 Comparison of commonly used VISLAM datasets

Dataset	KITTI[7]	EuRoC[6]	TUM VI[53]	ADVIO[55]	Ours
Hardware	Car	MAV	Custom Handheld	iPhone 6s	iPhone X / Xiaomi Mi 8
Camera	2×1392×512 10FPS Global Shutter	2×768×480 20FPS Global Shutter	2×1024×1024 20FPS Global Shutter	1×1280×720 60FPS RollingShutter	1×640×480 30FPS Rolling Shutter
IMU	OXTS RT 3003 10Hz	ADIS 16488 200Hz	BMI160 200Hz	The IMU of iPhone 6s 100Hz	The IMU of iPhone X / The IMU of Xiaomi Mi8 100Hz/400Hz
Ground- truth	OXTS RT 3003 10Hz	VICON/Leica 200Hz	OptiTrack 120Hz (Partially)	Sensor Fusion 100Hz	VICON 400Hz
Environment	Outdoors	Indoors	In-/outdoors	In-/outdoors	Indoors
Total Distance	39.2 km	0.9 km	20 km	4.5 km	377 m
Accuracy	~10 cm	~1 mm	~1 mm	~few dm	~1 mm
Sync	Software	Hardware	Hardware	Software	Software

4.1 Hardware setup

We used two different mobile phones (i.e., an iPhone X and a Xiaomi Mi 8) to collect visual-inertial data.

[3] https://www.vicon.com/

Specifically, we capture 640×480 monochrome images at 30fps with their rear camera. IMU data are recorded at different frequencies. For Xiaomi Mi 8, IMU data are coming at 400Hz. For iPhone X, its IMU data frequency is capped at 100Hz due to the limitation of CoreMotion API.

Ground truth data is obtained from a VICON motion capture system. It provides 6D pose measurements of the phone at 400Hz. The body frame of the phone is determined from a set of special markers. Figure 3 shows one of our colleagues capturing data. We will register the body frame to the camera and IMU's local frame through calibration. Since VICON data are recorded by a PC, there is a second synchronization problem. Since the time-offset may be different across sequences, we need to recalibrate the time-offset for each sequence.

4.2 Convention

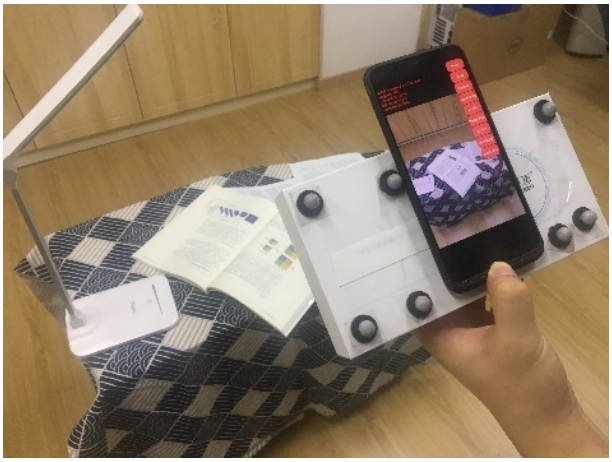

Figure 3 Our data capture equipment. The phone is rigidly attached to a marker object for VICON localization.

Before introducing our dataset and the calibration, we first define the coordinate frame convention used in our dataset. We use ${}^{B}_{A}\mathbf{R}$, ${}^{B}_{A}\mathbf{p}$ to represent the orientation and 3D position of a coordinate frame A with respect to coordinate B, respectively. Let ${}^{A}\mathbf{x}$ and ${}^{B}\mathbf{x}$ be the coordinates of the same point with respect to frame A and B respectively, then we have ${}^{B}\mathbf{x} = {}^{B}_{A}\mathbf{R}\,{}^{A}\mathbf{x} + {}^{B}_{A}\mathbf{p}$. In each data sequence, there will be 4 coordinate frames: the phone body frame B, the camera frame C, the VICON object frame V, and the VICON world frame W. B is attached to the IMU, representing the pose of the IMU as well as the phone itself. C represents the camera pose. V is defined by the reflective marker of VICON, and W can be chosen arbitrarily during the initialization of VICON. We glued the reflective markers on a rigid box, and then fix the phone on the box, so V is fixed on the phone too. During data recording, VICON gives the pose of V in W, i.e., $({}^{W}_{V}\mathbf{R}, {}^{W}_{V}\mathbf{p})$. The ground truth pose is represented as $({}^{W}_{B}\mathbf{R}, {}^{W}_{B}\mathbf{p})$.

Ultimately, we have to make spatial-temporal registration among the VICON object frame, camera frame, and the body (IMU) frame. The VICON marker is rigidly attached to the phone. Once the clocks are synchronized, both $({}^{W}_{V}\mathbf{R}, {}^{W}_{V}\mathbf{p})$ and $({}^{W}_{B}\mathbf{R}, {}^{W}_{B}\mathbf{p})$ should be constant. Figure 4 illustrates the spatial relationship between all these coordinate frames.

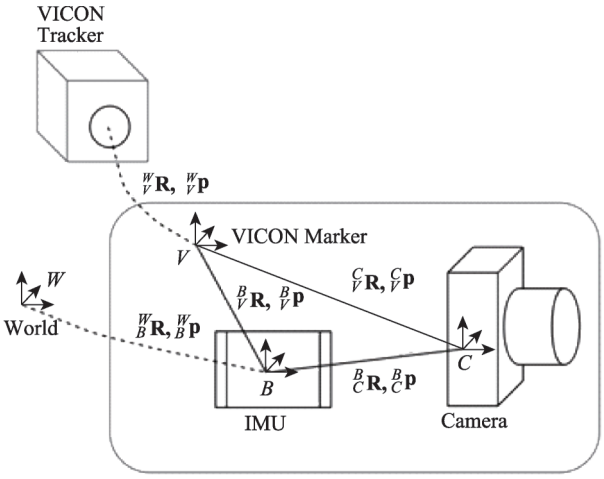

Figure 4 The relationship among VICON tracker, IMU and camera.

4.3 Calibration

4.3.1 Camera-IMU synchronization and calibration

We calibrate the camera intrinsic with the MATLAB Calibration Toolbox[4]. The relative rotation ${}^{B}_{C}\mathbf{R}$ and translation ${}^{B}_{C}\mathbf{p}$ between IMU and camera, as well as the time offset ${}^{B}_{C}t$, can be

[4] http://www.vision.caltech.edu/bouguetj/calib_doc/

obtained using Kalibr[56]. The camera timestamp is then corrected to the IMU clock.

4.3.2 VICON-IMU synchronization

Before calibrating the extrinsics between VICON and camera, synchronization must be done. Since we already synchronized the Camera and the IMU, we only need to synchronize IMU with VICON now. Let T be a specific time window. For any given VICON time ${}_V t$, the relative rotation between VICON pose at ${}_V t$ and VICON pose at ${}_V t + T$ is defined by:

$$ {}^{V({}_V t + T)}_{V({}_V t)}\mathbf{R} = {}^{V({}_V t + T)}_{W}\mathbf{R} \cdot {}^{V({}_V t)}_{W}\mathbf{R}^{\top} \tag{6} $$

Meanwhile, the same relative rotation can be found by integrating IMU measurements between the two time-points. Let ${}_B t = {}_V t + {}^{B}_{V}t$ be the IMU time corresponding to ${}_V t$, where ${}^{B}_{V}t$ be the time-offset between two sensors. The relative rotation as measured by IMU can be found by:

$$ {}^{B({}_B t + T)}_{B({}_B t)}\mathbf{R} = {}^{B({}_V t + {}^{B}_{V}t + T)}_{B({}_V t + {}^{B}_{V}t)}\mathbf{R} = \prod_{t = {}_B t}^{{}_B t + T} \exp\left(\omega(t)\,\Delta t\right) \tag{7} $$

Here, $\omega(t)$ is the IMU measurement at time t, and Δt is the sample interval of IMU. Now, since there is a rigid relative rotation ${}^{B}_{V}\mathbf{R}$ between VICON and IMU, these two relative rotations are related as:

$$ {}^{V({}_V t + T)}_{V({}_V t)}\mathbf{R} = {}^{B}_{V}\mathbf{R}^{\top} \cdot {}^{B({}_B t + T)}_{B({}_B t)}\mathbf{R} \cdot {}^{B}_{V}\mathbf{R} \tag{8} $$

However, the relative rotation ${}^{B}_{V}\mathbf{R}$ is unknown before we calibrated the extrinsics between the VICON and the IMU. To get rid of it, we compute the angle of the two relative rotations. Let $\theta_V(t) = \log\left({}^{V(t+T)}_{V(t)}\mathbf{R}\right)$, $\theta_B(t) = \log\left({}^{B(t+T)}_{B(t)}\mathbf{R}\right)$ be their angle-axis rotations. We have:

$$ \begin{aligned} {}^{V({}_V t + T)}_{V({}_V t)}\mathbf{R} &= {}^{B}_{V}\mathbf{R}^{\top} \cdot {}^{B({}_B t + T)}_{B({}_B t)}\mathbf{R} \cdot {}^{B}_{V}\mathbf{R} \\ \exp\left(\theta_V({}_V t)\right) &= {}^{B}_{V}\mathbf{R}^{\top} \cdot \exp\left(\theta_B({}_B t)\right) \cdot {}^{B}_{V}\mathbf{R} = \exp\left({}^{B}_{V}\mathbf{R}^{\top}\theta_B({}_B t)\right) \\ \theta_V({}_V t) &= {}^{B}_{V}\mathbf{R}^{\top}\theta_B({}_B t) \\ \left\|\theta_V({}_V t)\right\| &= \left\|{}^{B}_{V}\mathbf{R}^{\top}\theta_B({}_B t)\right\| = \left\|\theta_B({}_B t)\right\| \end{aligned} \tag{9} $$

So the angle of rotation can be extracted regardless of extrinsics. In reality, ${}_V t$ and ${}_B t$ need to be synchronized. So, we find the time-offset ${}^{B}_{V}t$ which maximizes the cross-correlation between their angle of rotations:

$$ \arg\max_{{}^{B}_{V}t} \frac{\sum \left\|\theta_V({}_V t)\right\| \left\|\theta_B({}_V t + {}^{B}_{V}t)\right\|}{\sqrt{\sum \left\|\theta_V({}_V t)\right\|^2}\sqrt{\sum \left\|\theta_B({}_V t + {}^{B}_{V}t)\right\|^2}} \tag{10} $$

In practice, we set T = 1s, and solves Eq. (9) by a linear sweeping over all the time-range of the dataset. In the following discussions, we will drop the timestamp notations by assuming a good synchronization.

4.3.3 VICON-Camera calibration

Now, with good synchronization, we can solve the relative transform $({}^{B}_{V}\mathbf{R}, {}^{B}_{V}\mathbf{p})$ by aligning the VICON measurements with the camera measurements. We put a planar board with multiple AprilTags[57] printed on it. Then we take several images of the AprilTag board from different viewpoints. At the same time, we record the VICON pose. With the help of AprilTags, we can obtain reliable correspondence between images. So we solve the following bundle-adjustment problem:

$$ \arg\min_{\{{}^{C}_{V}\mathbf{R}, {}^{C}_{V}\mathbf{p}, \{\mathbf{X}_j\}\}} \sum_i \sum_j \left\| \pi\left({}^{C}_{V}\mathbf{R}\,{}^{W}_{V}\mathbf{R}_i^{\top}\left(\mathbf{X}_j - {}^{W}_{V}\mathbf{p}_i\right) + {}^{C}_{V}\mathbf{p}\right) - \mathbf{x}_{ij} \right\|^2 . \tag{11} $$

Here, the device's poses are known from the VICON measurement. We only need to solve the VICON-Camera extrinsics $({}^{C}_{V}\mathbf{R}, {}^{C}_{V}\mathbf{p})$ and the spatial location of AprilTags $\mathbf{X}_j$. $\mathbf{x}_{ij}$ denotes the feature point in frame i which corresponds to $\mathbf{X}_j$. The VICON-IMU extrinsics $({}^{B}_{V}\mathbf{R}, {}^{B}_{V}\mathbf{p})$, can be computed using the VICON-Camera and Camera-IMU extrinsics. So, we will be able to register all three types of sensor data together, as shown

in Figure 4.

4.4 Motion type

For each phone, we captured several sequences. We focus on two aspects, i.e., motion type and scene complexity. The sequences are different combinations of these two types.

We define 5 motion types, i.e., hold, wave, aiming, inspect and patrol. "Hold" means the user keeps the phone almost stationary by hand. "Wave" means the user turns the phone around while keeping the wrist at an almost fixed position, so there will be strong rotation but small translation. "Aiming" will mostly appear in First-Person Shooting-like AR gaming, where the user is holding the phone away and pointing around. In this case, the rotation is moderate but the translational movement may be relatively fast. "Inspect" means the phone is moving around while looking at a particular place. This is a usual movement pattern when the user is inspecting some virtual objects. At last, for "patrol" motion type, the phone is firstly looking forward and then circling around. There may be fast rotation in this movement pattern, which is challenging for SLAM systems to work robustly.

We set up the VICON capture room with random messing-around objects, featureless or repetitive walls, PC desktop and glossing wooden floor. They are named "mess", "clean", "desktop" and "floor" correspondingly.

All sequences are composed of 3 segments: the static segment, initialization segment, and main segment. The static segment has at least 5s for some SLAM algorithms like MSCKF being able to accurately initialize the IMU biases. For the initialization segment, the mobile phone moves slowly and the captured content has sufficient texture so that most VSLAM/VISLAM approaches are able to initialize accurately. The main segment is designed for testing the tracking ability under kinds of camera motion that easily appear in mobile AR applications.

We recorded 16 sequences in total. Table 2 lists all the sequences and describes their properties, and Figure 5 presents some of the ground truth trajectories. In A0 and A1, the mobile phone is walking around the room and having strong rotational movements. A0 mainly contains glossy wooden floor, while A1 contains a white board and repetitive textured carpets. These are the two most difficult datasets. A2−A4 simulate different motions from daily use cases, and having more objects in the room. A5 also has many objects in the room, but we slowly wave the hand while holding the phone. A6−A7 are the easiest sequences where the phone fixates at a small desk with rich objects. B0−B7 are captured for evaluating dedicated criteria, they all circulate around a small desktop, plus some additional movements like fast rotations. These sequences will be introduced in section 5.

5 Evaluation criteria

To evaluate the SLAM systems for mobile AR, we define three criteria: the tracking accuracy, the initialization quality, and the tracking robustness. We will first introduce these criteria in detail, and then present our quantitative evaluation of 8 representative monocular VSLAM/VISLAM approaches/systems.

5.1 Tracking accuracy

Tracking accuracy, including absolute error, relative error, and completeness, is crucial to AR application. The absolute error measures how good a SLAM system localizes itself in the world. The absolute positional error, APE for brevity, is defined as:

Table 2 The motion and scene types of the captured sequences

Sequence		Motion	Scene	Description
Xiaomi	A0	inspect+patrol	floor	Walking and looking around the glossy floor.
	A1	inspect+patrol	clean	Walking around some texture-less areas.
	A2	inspect+patrol	mess	Walking around some random objects.
	A3	aiming+inspect	mess+floor	Random objects first, and then glossy floor.
	A4	aiming+inspect	desktop+clean	From a small scene to a texture-less area.
	A5	wave+inspect	desktop+mess	From a small scene to a texture-rich area.
	A6	hold+inspect	desktop	Looking at a small desktop scene.
	A7	inspect+aiming	desktop	Looking at a small desktop scene.
iPhone	B0	rapid-rotation	desktop	Rotating the phone rapidly at some time.
	B1	rapid-translation	desktop	Moving the phone rapidly at some time.
	B2	rapid-shaking	desktop	Shaking the phone violently at some time.
	B3	inspect	moving people	A person walks in and out.
	B4	inspect	covering camera	An object occasionally occluding the camera.
	B5	inspect	desktop	Similar to A6 but with black frames.
	B6	inspect	desktop	Similar to A6 but with black frames.
	B7	inspect	desktop	Similar to A6 but with black frames.

$$\epsilon_{\mathrm{APE}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \| \mathbf{p}_{\mathrm{SLAM}}[i] - \mathbf{p}_{\mathrm{GT}}[i] \|^2} \tag{12}$$

where $\mathbf{p}_{\mathrm{SLAM}}[i]$ and $\mathbf{p}_{\mathrm{GT}}[i]$ are the estimated camera position and ground truth position for frame i respectively.

The absolute rotational error or ARE, is defined as:

$$\epsilon_{\mathrm{ARE}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \| \log(\mathbf{R}_{\mathrm{SLAM}}^{-1}[i] \cdot \mathbf{R}_{\mathrm{GT}}[i]) \|^2} \tag{13}$$

Here, $\mathbf{R}_{\mathrm{SLAM}}[i]$ and $\mathbf{R}_{\mathrm{GT}}[i]$ are the estimated camera orientation and ground truth orientation for frame i, respectively. $\log(\mathbf{R})$ maps a rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$ to its Lie algebra.

Since there is gauge ambiguity in the world coordinate, we must align the result with the ground truth first. The optimal alignment transformation can be found using Umeyama's method[58]. For the poses from t_1 to t_2, we use $\mathrm{U}(t_1,t_2)$ to denote the best similarity transform by minimizing the following energy function:

$$\mathrm{U}(t_1,t_2) = \underset{s,\mathbf{R},\mathbf{T}}{\arg\min} \sum_{t=t_1}^{t_2} \| (\mathbf{R}\mathbf{p}_{\mathrm{SLAM}}[t] + \mathbf{T}) - s\mathbf{p}_{\mathrm{GT}}[t] \|^2 \tag{14}$$

The estimated camera trajectory from VSLAM will be transformed with the result of Eq. (14) before evaluating ϵ_{APE} and ϵ_{RPE}. For VISLAM result, the scale should be estimated by the algorithm. So we force $s = 1$ while aligning the camera trajectory with the ground truth. We will also use $\mathrm{U}_S(t_1,t_2)$ to denote only the scale component in$\mathrm{U}(t_1,t_2)$.

The relative positional error (RPE) and relative rotational error (RRE) are defined as follows:

$$\epsilon_{\mathrm{RPE}} = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m-1} \left(\| \mathbf{p}_{\mathrm{SLAM}}[i+1] - \mathbf{p}_{\mathrm{SLAM}}[i] \| - \| \mathbf{p}_{\mathrm{GT}}[i+1] - \mathbf{p}_{\mathrm{GT}}[i] \| \right)^2} \tag{15}$$

$$\epsilon_{\mathrm{RRE}} = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m-1} \left(\| \log(\mathbf{R}_{\mathrm{SLAM}}^{-1}[i+1] \cdot \mathbf{R}_{\mathrm{SLAM}}[i]) \| - \| \log(\mathbf{R}_{\mathrm{GT}}^{-1}[i+1] \cdot \mathbf{R}_{\mathrm{GT}}[i]) \| \right)^2} \tag{16}$$

Since the SLAM systems may get lost in some time, the evaluation of APE/RPE/ARE/RRE are conducted on all available poses, excluding the poses that are not initialized or in the lost status.

Besides position and rotation error, the completeness of the recovered camera trajectory is also important. The completeness is defined by the ratio between the number of valid poses and the total

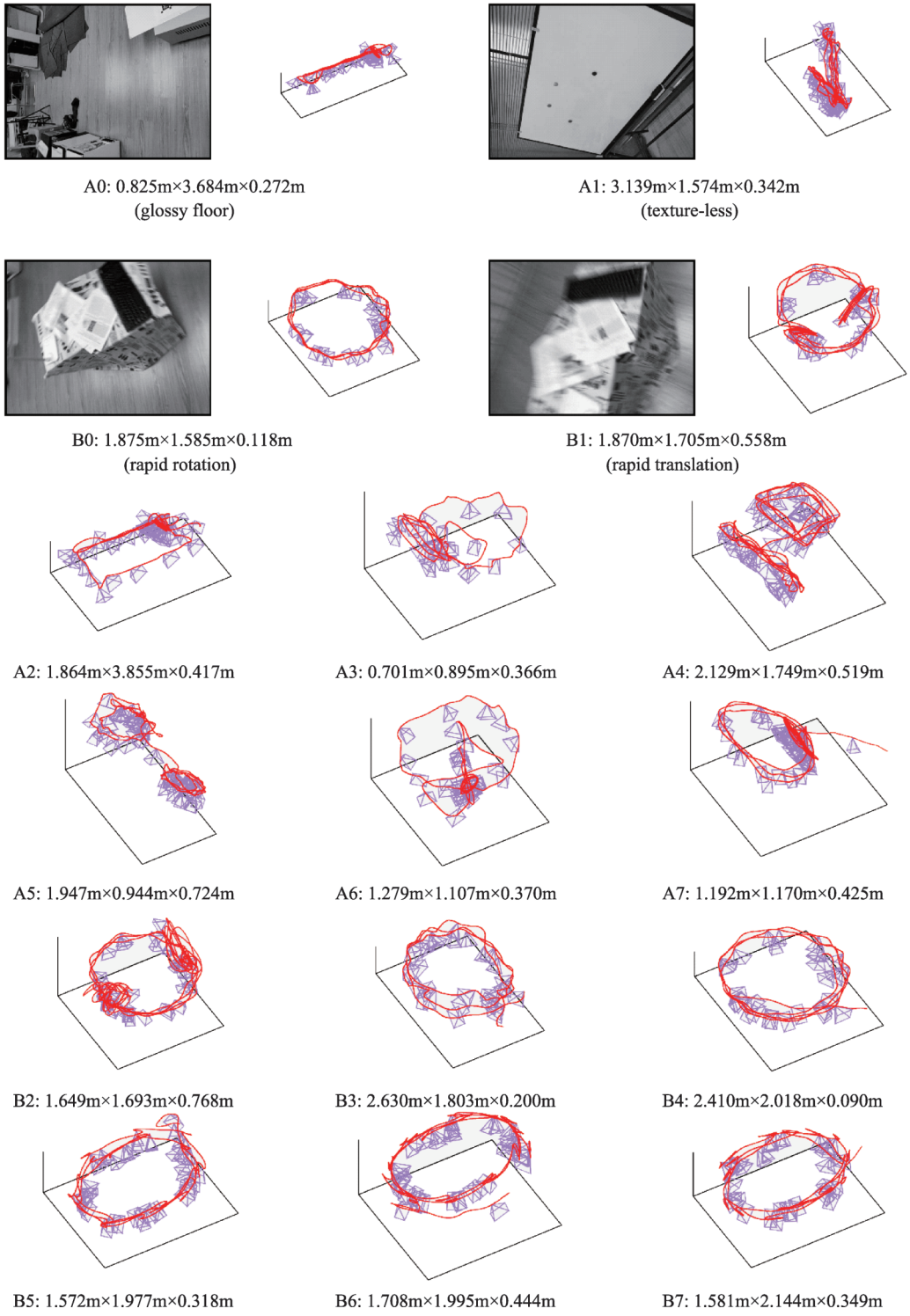

Figure 5 The ground truth trajectories of all 16 sequences in our dataset. We also show the representative images from A0, A1, B0, and B1.

number of all poses (the poses before the first initialization are not included). In our setting, we think the recovered pose is valid only when its absolute position error is not larger than 0.1m. In addition, we exclude the frames before the first initialization. But since the tracking may be completely lost and need re-initialize, the frames between lost and re-initialization are considered invalid and are used for completeness computation.

5.2 Initialization quality

For mobile AR, a fast and accurate initialization is important for good user experience. Yet, different SLAM systems may use quite different initialization strategies. The original PTAM algorithm requires user-interaction, and filtering-based MSCKF requires the device to keep static for a while. ORB-SLAM can automatically select two frames for initialization. If we treat the first pose returned by SLAM system as initialization, the underlying motion state may not fully converge to a good result. Hence, we take a different approach to evaluate the initialization.

We define the scale of the cumulative moving window $s_{\mathrm{cmw}}(t)$ as the scale estimated from frames until time t:

$$s_{\mathrm{cmw}}(t) = \mathrm{U}_S(0, t) \tag{17}$$

where $\mathrm{U}_S(0,t)$ denotes the scale component of $\mathrm{U}(0,t)$. The full initialization of a SLAM system needs to reach a stable scale, i.e., $s_{\mathrm{cmw}}(t)$ is becoming stable.

When evaluating a tracking result, we can find its starting and ending time. Let t_{end} be the ending time of the trajectory. The finally estimated scale at t_{end} will be $s_{\mathrm{cmw}}(t_{\mathrm{end}})$. If there are tracking lost events in the middle, we use the trajectory of the first continuous segment for determining t_{end} and $s_{\mathrm{cmw}}(t_{\mathrm{end}})$. In the tracking, the scale $s_{\mathrm{cmw}}(t)$ will gradually approach $s_{\mathrm{cmw}}(t_{\mathrm{end}})$, as illustrated in Figure 6. We can compute the relative deviation of s_{cmw} from the converged value as:

$$r_{\mathrm{cmw}}(t) = \frac{s_{\mathrm{cmw}}(t) - s_{\mathrm{cmw}}(t_{\mathrm{end}})}{s_{\mathrm{cmw}}(t_{\mathrm{end}})} \tag{18}$$

When $|r_{\mathrm{cmw}}|$ is bounded in range r_{init}, we believe s_{cmw} converged. Therefore, we define the time of initialization as $t_{\mathrm{init}} = \min\left\{t:\forall\tau \geqslant t, |r_{\mathrm{cmw}}(\tau)| \leqslant r_{\mathrm{init}}\right\}$. In our benchmark, we set $r_{\mathrm{init}} = 20\%$. In order to make some algorithms like MSCKF to accurately initialize the IMU biase, in our captured sequences, the camera always keeps stationary for 5s in the beginning. So we subtract t_{init} by 5s as the final initialization time.

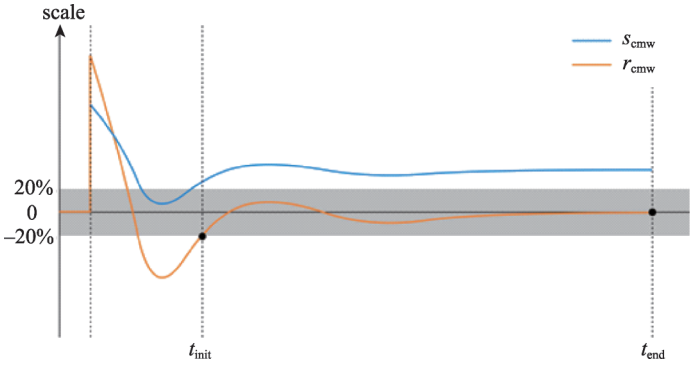

Figure 6 The convergence of scale.

Besides the initialization time, the accuracy of the estimated scale during initialization is also very important. So we compute the following symmetric relative scale error at t_{init}:

$$\epsilon_{\mathrm{scale}} = \frac{1}{2}\left(\left|\frac{s_{\mathrm{cmw}}(t_{\mathrm{init}})}{s_{\mathrm{g}}} - 1\right| + \left|\frac{s_{\mathrm{g}}}{s_{\mathrm{cmw}}(t_{\mathrm{init}})} - 1\right|\right) \times 100\% \tag{19}$$

where s_{g} represents the global scale of each trajectory. For VSLAM systems, we solve s_{g} by aligning all the results with the ground-truth. For VISLAM systems, we set $s_{\mathrm{g}} = 1$, because it is important for VISLAM systems to initialize with accurate scale.

By combining the initialization time and the scale error after initialization, the initialization quality is finally defined as:

$$\epsilon_{\text{init}} = t_{\text{init}} (\epsilon_{\text{scale}} + \beta)^{\alpha} \tag{20}$$

where $\alpha = 0.5$ and $\beta = 1\%$ in our experiments.

5.3 Tracking robustness

In extreme cases, visual tracking may be lost. A robust system should recover from the lost status as soon as possible, and accurately register the new pose with the past trajectory. For the next discussion, we assume there is one lost event during tracking. Let $[t_1,t_2]$ and $[t_3,t_4]$ be the trajectory before and after this lost event, we can register the two sub-trajectories as:

$$\begin{aligned} \{s_{12}, \mathbf{R}_{12}, \mathbf{T}_{12}\} &= \mathrm{U}(t_1, t_2) \\ \{s_{34}, \mathbf{R}_{34}, \mathbf{T}_{34}\} &= \mathrm{U}(t_3, t_4) \end{aligned} \tag{21}$$

For a good recovery, the sub-trajectory $[t_3,t_4]$ must be aligned well with $[t_1, t_2]$, hence $\{s_{34}, \mathbf{R}_{34}, \mathbf{T}_{34}\}$ should be close to $\{s_{12}, \mathbf{R}_{12}, \mathbf{T}_{12}\}$. Otherwise, the new sub-trajectory can have a much different transformation, and the global consistency may be broken, as illustrated in Figure 7.

Given the whole tracking result, there can have multiple lost events. We first segment the trajectory according to these lost events. Then we register each trajectory with the ground truth. Let $\{\xi_i | \xi_i = (s_i, \mathbf{R}_i, \mathbf{T}_i)\}$ be the similarity transformation of each sub-trajectory, we compute the relocalization error by comparing two neighboring transformations and add them together:

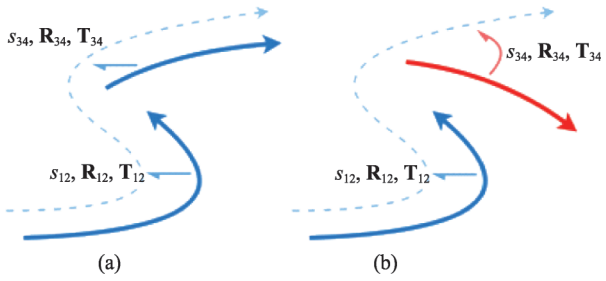

Figure 7 Different relocalization quality.

$$\epsilon_{\text{RL}} = \sum_{i=1}^{n-1} \| \log_{\text{Sim}(3)} (\xi_i^{-1} \xi_{i+1}) \| \tag{22}$$

Here, summation is used instead of averaging. The more an algorithm is getting lost, the more relocalization error it is accumulating.

Beside the relocalization error, the lost time is also crucial to user experience since AR would be impossible when lost. Hence, we also count the ratio of lost time to the total tracking time α_{lost}. The lost status is defined by each system itself. In order to get clear indication of tracking lost, we force all systems to output poses for all frames, and if the tracking is lost, the systems output a pose with invalid rotation. Thus, we can compute α_{lost} with the output poses. Also, the positional error ϵ_{APE} is directly related to the quality of augmentation effect. As a final score, the lost time ratio α_{lost}, the relocalization error ϵ_{RL}, and the positional errorϵ_{APE} are summarized into one robustness error as:

$$\epsilon_{\text{R}} = (\alpha_{\text{lost}} + \eta_{\text{lost}})(\epsilon_{\text{RL}} + \eta_{\text{APE}} \epsilon_{\text{APE}}) \tag{23}$$

Here, η_{lost} is used as a damping factor to prevent $\alpha_{\text{lost}} = 0$ from canceling out other two errors. η_{APE} is another weighting factor to balance between ϵ_{RL} and ϵ_{APE}. In our experiments, we set $\eta_{\text{lost}} = 5\%$ and $\eta_{\text{APE}} = 0.1$.

In our experiments, we consider three common ill-scenarios when using AR: rapid motion, moving people and camera occlusion. The algorithm should be resilient to rapid motion since such motion can be common on mobile phones. Also, it should allow moving people or moving objects to appear in the image. In either case, SLAM tracking may become unreliable and easily lost. In this situation, the system should be able to recover from the lost status and get back to the correct location. The total time of lost should be as short as possible, the relocalization error should be as small as possible, and of course, the result positions should be as accurate as possible.

To simulate these scenarios, 5 dedicated sequences were captured. They are sequences B0-B4 listed in

Table 2. We introduce rapid rotation/translation and random shake in sequence B0−B2. These three sequences are used for quantitative evaluation of the robustness of rapid motion. In sequence B3, we had a person walking in and out of the room to create a dynamic scene. The tracking may get distracted from the moving people. In sequence B4, we deliberately covered the camera for some time, make it even more challenging for tracking. Upon test, the optimization of the algorithms may diverge. In order to recognize this case, when the estimated pose jumps for more than 5cm, we consider the result as lost.

5.4 Relocalization time

The tracking robustness is designed to reflect AR quality. From a technical perspective, we would like to know how much time the algorithms spent to accomplish relocalization. To get relocalization time, three additional sequences were designed. The sequences are B5−B7 in Table 2. In these sequences, we looped around a textured desk for several rounds. And the movement taken was as steady as possible. After the first round of loop, we black-out some frames. In sequence B5, each black-out takes 1s. Similarly, 2s and 3s black-outs are used for sequence B6 and B7, respectively. A SLAM system will have the chance to create their visual database in the first 30s, then forced to enter relocalization state when black frames come. After black frames are passed, there are 10s of original frames. The SLAM systems should re-localize themselves in this period. By manually adding black-frames, we can precisely measure the time used for relocalization as the time-difference between the end of black-frames and the begin of the re-localized results. Some VISLAM systems like VINS-Mono do not have a clear "lost" indication. Upon visual information is lost, they just keep tracking with IMU integration. We detect the "jump" in the estimated poses, as these jumps correspond to the underlying relocalization event.

Formally, let $t_{\mathrm{K}[i]}$ be the end time of i-th black-out, $t_{\mathrm{SLAM}[i]}$ be the time of the first valid pose immediately after the black-out. Assuming there are N black-outs, we can compute the average relocalization time as $t_{\mathrm{RL}} = \frac{1}{N}\sum_{i=1}^{N}\left(t_{\mathrm{SLAM}[i]} - t_{\mathrm{K}[i]}\right)$. For VISLAM systems, we define $t_{\mathrm{SLAM}[i]}$ by detecting the pose jump:

$$t_{\mathrm{SLAM}[i]} \equiv \min\left\{t_k > t_{\mathrm{K}[i]} \,|\, \left\|\mathbf{p}_{\mathrm{SLAM}}[k+1] - \mathbf{p}_{\mathrm{SLAM}}[k]\right\| > \delta\right\} \tag{24}$$

i.e., the first pose after $t_{\mathrm{K}[i]}$ which jumped for more than δ. In our experiments, we set $\delta = 5$ cm.

6 Experimental results

We selected 8 monocular VSLAM/VISLAM systems to perform the quantitative evaluation with our benchmark. PTAM and ORB-SLAM2[34] are the most famous VSLAM systems. LSD-SLAM and DSO are representative direct methods. MSCKF is a representative filtering-based VISLAM method. OKVIS and VINS-Mono are two representative optimization-based VISLAM methods which both use sliding window optimization and marginalization technique. Specifically, VINS-Mono has global optimization and relocalization modules. We also selected a commercial VISLAM system SenseSLAM[5] which is developed by us in cooperation with SenseTime Group Limited. For other commercial AR systems like ARCore and ARKit, since they cannot run on a PC in offline-mode, we cannot quantitatively evaluate them on our benchmark.

We used a modified PTAM which is provided as a ROS package[6]. This version improved the overall robustness. It also has an automatic initialization process[59]. Since the original implementation of MSCKF

[5] The binary executable of SenseSLAM v1.0 we tested in this paper can be downloaded from the website at http://www.zjucvg.net/senseslam/

[6] http://wiki.ros.org/ethzasl_ptam

is not publicly available, we also used a third-party implementation[7] instead, whose validity is demonstrated[60]. For other SLAM systems, we used the public implementations provided by the authors. All the experiments are conducted on a desktop PC with an Intel i7-7700 3.6GHz CPU and 8GB of memory. The operating system is Ubuntu 16.04 with ROS Kinetic.

We run all 8 SLAM systems on all sequences. All these systems can perform in real-time. It is worth mentioning that although all the systems run on PC at over 30fps, some of them may not be capable of running on mobile devices at real-time. These systems run in different manners. Some systems like ORB-SLAM2 will sleep the system to keep tracking pace at 30fps, while VINS-Mono is synchronized to ROS messages. Due to different internal implementation, there lacks a common definition for processing time per frame, so we will not report the detailed timings for these systems. Due to the randomness in these SLAM systems, they may not always produce the same results. So, we run each benchmark 10 times to take the average of the results. Some algorithms, when run against some sequences, may produce inconsistent results. For example, the DSO algorithm failed to converge on sequence A1 for 6 out of 10 times, and OKVIS also failed several times on some sequences (e.g., 6 times on sequence A5). If we directly compute the average, the computed tracking accuracy may be unusually large. Therefore, we remove these failure cases (i.e., APEs were unusually large) by inspection, and then compute the average of the remaining ones. The number of failures for each sequence is reported in our benchmark website. It is worth mentioning that sequence A1 appeared to be the most difficult one during our benchmark. All VSLAM systems failed several times on sequence A1. We ran MSCKF 10 times on sequence A1, and found that the recovered trajectories were abnormal (the corresponding APEs were larger than 0.5m) for 3 times. In contrast, VINS-Mono and SenseSLAM are rather robust. They can successfully run all sequences and produce consistent results.

Tables 3 and 4 summarize the tracking accuracy of different SLAM systems on sequences A0−A7. For VSLAM systems, ORB-SLAM2 clearly has the best accuracy in most sequences. Comparing with the patch-based feature tracking used by PTAM, ORB feature-based tracking method is more robust especially in the cases of fast camera motion. In addition, ORB-SLAM2 has more advanced global mapping and loop closure modules. In combination with ORB features, it can reliably reconstruct a globally consistent map, hence having better accuracy. Although PTAM is also optimizing a global map, its localization strategy is relatively simple, so the tracking gets lost more easily. This is directly reflected in the completeness results, as reported in Table 3.

The errors for LSD-SLAM and DSO are also quite large compared to PTAM and ORB-SLAM2. Typically, direct methods are easily influenced by the factors of calibration, synchronization, varying illumination, rolling-shutter, ill movements, etc. These problems might limit the direct methods from being directly used in mobile AR since it is challenging to calibrate all these factors for mobile phones and the computation cost is generally large. Since DSO does not have relocalization module, the camera pose could not be re-localized once the tracking is lost. This leads to the result that the tracking completeness of DSO is quite small on challenging sequences.

In the results of VISLAM systems, SenseSLAM is generally better than the other three. The possible reason is that SenseSLAM optimizes the initialization module for AR in particular and uses a more complex framework with the modules of local/global optimization and relocalization. We noticed that, for APE errors, SenseSLAM is significantly better than VINS-Mono on challenging sequences like A0−A2. In terms of APE, OKVIS is also very competing. It outperforms VINS-Mono on many sequences and even beats

[7] https://github.com/daniilidis-group/msckf_mono

Table 3 The tracking accuracy of VSLAM systems

Sequence		PTAM		ORB-SLAM2		LSD-SLAM		DSO	
APE/RPE (mm)	A0	**75.442**	6.696	96.777	**5.965**	105.963	11.761	231.860	10.456
	A1	113.406	16.344	**95.379**	**10.285**	221.643	23.833	431.929	12.555
	A2	**67.099**	6.833	69.486	5.706	310.963	8.156	216.893	**5.337**
	A3	**10.913**	4.627	15.310	7.386	199.445	10.872	188.989	**4.294**
	A4	21.007	4.773	**10.061**	**2.995**	155.692	10.756	115.477	4.595
	A5	40.403	8.926	**29.653**	11.717	249.644	12.302	323.482	**7.978**
	A6	19.483	3.051	**12.145**	6.741	49.805	3.018	14.864	**2.561**
	A7	13.503	2.462	**5.832**	**1.557**	38.673	2.662	27.142	2.213
ARE/RRE (deg)	A0	12.051	**0.257**	**5.119**	0.342	20.589	0.371	9.983	0.401
	A1	53.954	0.291	**8.534**	**0.242**	51.122	0.288	39.007	0.524
	A2	8.789	0.301	**5.550**	0.255	30.282	0.296	10.584	**0.253**
	A3	6.225	0.293	**1.431**	0.264	31.370	0.475	20.580	**0.241**
	A4	6.295	0.255	**1.015**	**0.157**	9.592	0.498	5.217	0.180
	A5	14.030	0.452	**1.963**	0.546	36.789	0.810	40.939	**0.324**
	A6	2.348	0.217	**0.892**	**0.169**	5.012	0.207	1.435	0.189
	A7	1.218	0.153	**0.569**	**0.115**	3.052	0.147	2.239	0.135
Completeness (%)	A0	**79.386**		65.175		49.513		14.476	
	A1	60.893		**68.303**		11.511		0.869	
	A2	**85.348**		79.263		21.804		22.878	
	A3	71.635		**98.497**		27.112		43.493	
	A4	95.418		**100.000**		64.283		80.371	
	A5	87.399		**97.785**		25.033		2.059	
	A6	97.399		99.786		94.883		**100.000**	
	A7	**100.000**		**100.000**		98.663		**100.000**	

SenseSLAM on sequence A0. However, its trajectory is somehow jaggy, which is reflected in its high RPE results. Overall, SenseSLAM achieves the best accuracy among the four VISLAM systems on our benchmark.

The completeness results reveal more about the tracking quality. Due to their high positional error, many poses from LSD-SLAM and DSO are considered invalid on sequences A0−A3. Especially, the harsh texture and motion condition of sequence A1 almost disabled DSO. Similar situation happened on the sequence A5. PTAM and ORB-SLAM2 also suffered from difficulties on these challenging sequences, but performed much better than direct methods. On VISLAM systems, sequence A1 is also quite challenging. VINS-Mono and MSCKF cannot track well on sequence A1, hence having low completeness. Except occasional failures, OKVIS achieved quite good completeness results on all sequences A0−A7. On sequence A1, it has the best completeness. For SenseSLAM, its overall tracking quality is good, and can produce valid poses for most of the time. Overall, ORB-SLAM has the best completeness among these VSLAM systems, while SenseSLAM is the best among four VISLAM systems.

Figure 8 lists the initialization time and scale error after initialization of different SLAM systems. Table 5 lists the finally computed initialization quality for kinds of SLAM systems according to Eq. (20). In VSLAM systems, PTAM has the fastest initialization time, but the scale error is so poor that saturates in the figure. As a result, it has the worst initialization quality. The other three VSLAM systems have similar initialization times. However, ORB-SLAM2 has the smallest error, so its overall initialization quality ranked the first among the four VSLAM systems.

Table 4 The tracking accuracy of VISLAM systems

	Sequence	MSCKF		OKVIS		VINS-Mono		SenseSLAM	
ARE/RRE(mm)	A0	156.018	7.436	71.677	7.064	160.334	2.798	**58.995**	**2.525**
	A1	294.091	14.580	87.730	4.283	253.554	2.723	**55.097**	2.876
	A2	102.657	10.151	68.381	5.412	102.263	1.976	**36.370**	**1.560**
	A3	44.493	3.780	22.949	8.739	29.587	1.278	**17.792**	**0.779**
	A4	114.845	8.338	146.890	12.460	37.580	1.042	**15.558**	**0.930**
	A5	82.885	8.388	77.924	7.588	40.423	1.660	**34.810**	1.954
	A6	66.001	6.761	63.895	6.860	80.062	1.404	**20.467**	**0.569**
	A7	105.492	4.576	47.465	6.352	25.082	1.138	**10.777**	**0.831**
ARE/RRE(deg)	A0	6.584	0.203	**3.637**	0.741	4.181	0.206	3.660	**0.197**
	A1	8.703	0.135	5.140	1.098	4.171	0.095	**2.676**	**0.092**
	A2	3.324	0.195	2.493	0.869	3.193	0.202	**1.674**	**0.181**
	A3	6.952	0.186	2.459	0.825	2.276	**0.180**	**1.642**	0.182
	A4	4.031	0.104	3.765	0.603	1.439	**0.070**	**1.129**	0.071
	A5	4.928	0.167	8.843	0.360	2.197	**0.059**	**2.041**	0.089
	A6	2.625	0.170	2.275	0.629	1.840	**0.132**	**1.656**	0.134
	A7	6.810	0.120	3.536	0.602	2.091	**0.080**	**0.502**	0.082
Completeness(%)	A0	40.186		94.255		35.256		97.317	
	A1	1.646		**98.235**		17.902		95.072	
	A2	61.423		94.959		63.449		**99.707**	
	A3	97.814		95.972		**100.000**		**100.000**	
	A4	76.629		97.429		**100.000**		**100.000**	
	A5	76.738		98.162		**99.866**		99.143	
	A6	94.128		97.805		81.763		**100.000**	
	A7	68.341		96.690		**100.000**		**100.000**	

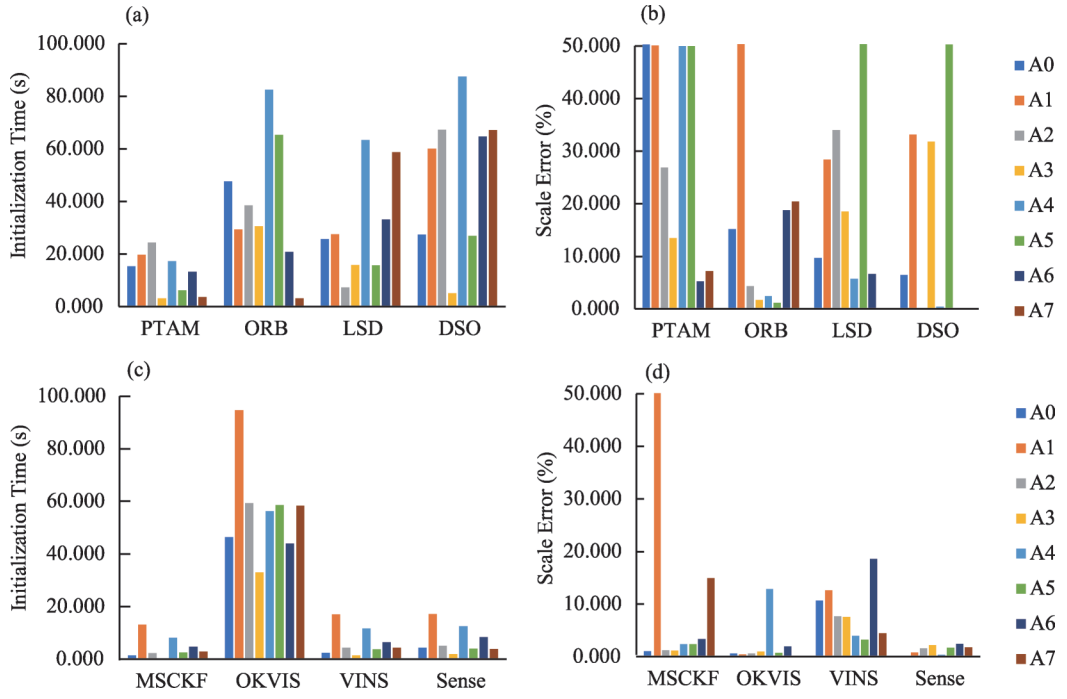

Figure 8 The initialization time and scale error. Scale errors that are too large are trimmed here.

In VISLAM systems, SenseSLAM again shows its good initialization quality. Although the initialization times of MSCKF and VINS-Mono are comparable to that of SenseSLAM, SenseSLAM has more

Table 5 The initialization quality ϵ_{init} for different VSLAM/VISLAM systems

Sequence	PTAM	ORB-SLAM2	LSD-SLAM	DSO	MSCKF	OKVIS	VINS-Mono	SenseSLAM
A0	41.387	19.172	8.423	**7.460**	**0.211**	5.913	0.783	0.449
A1	22.265	28.141	**14.877**	35.062	49.660	11.487	6.265	**2.324**
A2	12.828	8.913	**4.311**	6.837	**0.331**	7.506	1.300	0.804
A3	**1.193**	5.009	6.960	2.920	**0.035**	4.607	0.441	0.340
A4	16.725	15.324	16.478	**10.450**	1.497	20.964	2.584	**1.456**
A5	14.223	**9.512**	41.941	28.801	**0.463**	7.732	0.763	0.652
A6	**3.322**	9.275	9.158	6.550	**0.991**	7.613	2.857	1.553
A7	**1.027**	1.458	6.176	6.766	1.159	6.265	1.026	**0.650**
Average	14.121	**12.101**	13.541	13.106	6.793	9.011	2.002	**1.029**
Max	41.387	**28.141**	41.941	35.062	49.660	20.964	6.265	**2.324**

consistent scale errors across all sequences. Interestingly, on many sequences, MSCKF has good initialization quality, but it also may suffer from catastrophic initialization, like in sequence A1. We also found that OKVIS usually took a long time to converge the scale. The major reason should be that OKVIS simply initializes IMU biases with prior values, which might be inaccurate and need more time to be optimized. It is not fair to make a direct comparison between VSLAM and VISLAM since their scale errors are evaluated in different settings. But we can see VISLAM systems generally finish initialization much faster than VSLAM systems and have much more consistent scale error, thanks to the additional scale information from the IMU.

Table 6 lists ϵ_R for sequences B0−B4. Both MSCKF and DSO fail in some challenging cases and are completely lost. In contrast, SenseSLAM can handle these ill tracking conditions. Since SenseSLAM uses IMU information, its accelerometer measurements under rapid translation / shaking can be quite noisy, hence having a much larger error. While the gyroscope is generally reliable for short-time orientation estimation. This might explain why SenseSLAM gets relatively poor results in sequences B1 and B2. ORB-SLAM2, on the other hand, does not depend on IMU measurements, so can achieve a good localization quality on sequence B2. However, on sequence B3, we can see the tracking result of ORB-SLAM2 is less robust, because the moving people in the sequence misleads ORB-SLAM2.

Table 6 The tracking robustness of different VSLAM/VISLAM systems

Sequence	PTAM	ORB-SLAM2	LSD-SLAM	DSO	MSCKF	OKVIS	VINS-Mono	SenseSLAM
B0 (Rapid Rotation)	16.088	3.396	2.068	1.848	−	5.328	16.774	**0.511**
B1 (Rapid Translation)	26.887	7.128	12.739	16.127	−	**5.448**	9.024	7.199
B2 (Rapid Shaking)	36.140	**3.875**	12.476	−	−	24.024	18.062	9.743
B3 (Moving People)	12.779	16.670	22.882	41.294	−	1.636	16.741	**1.089**
B4 (Covering Camera)	20.062	8.265	17.368	−	3.119	13.051	18.619	**1.192**

With sequences B5−B7, we evaluate the relocalization time. Table 7 lists the average relocalization time of PTAM, ORB-SLAM2, LSD-SLAM, VINS-Mono and SenseSLAM. MSCKF, OKVIS and DSO simply cannot re-localize after blacking-out the images, so we do not evaluate them in this criterion.

According to the experimental results, we found that ORB-SLAM2 always re-localized rapidly. PTAM also re-localized fast. VISLAM systems seem to take more time, while SenseSLAM again performs very well and is faster than VINS-Mono. LSD-SLAM, as a VSLAM system based on direct method, generally takes more time compared to the other four systems.

Table 7 The relocalization time (s) of different VSLAM/VISLAM systems

Sequence	PTAM	ORB SLAM2	LSD-SLAM	VINS-Mono	SenseSLAM
B5 (1s black-out)	1.032	**0.077**	1.082	1.452	0.592
B6 (2s black-out)	**0.366**	0.465	5.413	1.833	1.567
B7 (3s black-out)	0.651	**0.118**	1.834	0.841	0.332
Average	0.683	**0.220**	2.776	1.375	0.830

7 Conclusion and discussion

As the key fundamental technique of AR, VSLAM/VISLAM has achieved significant progress in the past decade. However, due to the lack of benchmarks for AR, it is hard to quantitatively compare the performance of different VSLAM/VISLAM systems for AR applications. In order to address this problem, we publish the first VISLAM benchmark for AR, including visual-inertial dataset as well as evaluation criteria. We review the existing monocular VSLAM and VISLAM approaches, and select 8 representative systems to perform a quantitative evaluation on our benchmark.

Although our visual-inertial data are captured by mobile phones, the evaluation is still conducted on a PC. Since the computation power of a PC is much bigger than a mobile phone, the SLAM results could not faithfully reflect the actual SLAM effect on a mobile phone. Actually, many SLAM systems cannot perform in real-time on a mobile phone. The mobile versions of PTAM and VINS-Mono are already available[61,39]. SenseSLAM is specifically developed for mobile AR and is capable of tracking in real-time on a mobile phone. In the future, we would like to better evaluate the performance of SLAM systems on mobile phones in two ways. On the one hand, we would like to port some algorithms into iOS or Android, so that we can directly run SLAM and perform the quantitative evaluation on mobile phones. On the other hand, we would like to simulate the online stream and strictly control the FPS of the input (some frames will be discarded if the tracking speed could not keep up the input rate), so that the SLAM result on a PC can be closer to that on mobile phones. In addition, the sequences in our current dataset are all captured in a small indoor scene. In the future, we would like to capture more diverse sequences in a larger outdoor environment.

Acknowledgements

We would like to thank Shuai Wang, Kai Huang, Jinle Ke, Pengpeng Jin for their kind help in capturing the visual-inertial data, calibration and other post-processing. Thanks Ru Wang to help to organize the bibliography.

References

1 Cadena C, Carlone L, Carrillo H, Latif Y, Scaramuzza D, Neira J, Reid I, Leonard J J. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics, 2016, 32(6): 1309−1332
DOI:10.1109/tro.2016.2624754

2 Fuentes-Pacheco J, Ruiz-Ascencio J, Rendón-Mancha J M. Visual simultaneous localization and mapping: A survey. Artificial Intelligence Review, 2015, 43(1): 55−81
DOI:10.1007/s10462-012-9365-8

3 Durrant-Whyte H, Bailey T. Simultaneous localization and mapping: Part I. IEEE Robotics & Automation Magazine, 2006, 13(2): 99−110
DOI:10.1109/mra.2006.1638022

4 Bailey T, Durrant-Whyte H. Simultaneous localization and mapping (SLAM): Part II. IEEE Robotics & Automation

Magazine, 2006, 13(3): 108−117
DOI:10.1109/mra.2006.1678144

5 Liu H M, Zhang G F, Bao H J. A survey of monocular simultaneous localization and mapping. Journal of Computer-Aided Design & Computer Graphics, 2016, 28(6): 855−868
DOI:10.3969/j.issn.1003-9775.2016.06.001

6 Burri M, Nikolic J, Gohl P, Schneider T, Rehder J, Omari S, Achtelik M W, Siegwart R. The EuRoC micro aerial vehicle datasets. The International Journal of Robotics Research, 2016, 35(10): 1157−1163
DOI:10.1177/0278364915620033

7 Geiger A, Lenz P, Urtasun R. Are we ready for autonomous driving? The KITTI vision benchmark suite. In: IEEE Conference on Computer Vision and Pattern Recognition. Providence, RI, USA: 2012, 3354−3361
DOI:10.1109/CVPR.2012.6248074

8 Hartley R, Zisserman A. Three-View Geometry. Multiple View Geometry in Computer Vision. Cambridge: Cambridge University Press, 363−364
DOI:10.1017/cbo9780511811685. 020

9 Ma Y, Soatto S, Kosecka J, Sastry S S. An Invitation to 3-D Vision: From Images to Geometric Models. Springer Science & Business Media, 2012, 26

10 Triggs B, McLauchlan P F, Hartley R I, Fitzgibbon A W. Bundle Adjustment—A Modern Synthesis. Vision Algorithms: Theory and Practice. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000: 298−372
DOI:10.1007/3-540-44480-7_21

11 Sola J. Quaternion kinematics for the error-state KF. Tech Rep, 2012

12 Forster C, Carlone L, Dellaert F, Scaramuzza D. On-manifold preintegration for real-time visual: Inertial odometry. IEEE Transactions on Robotics, 2017, 33(1): 1−21
DOI:10.1109/tro.2016.2597321

13 Eckenhoff K, Geneva P, Huang G. Continuous preintegration theory for graph-based visual-inertial navigation. arXiv: 1805. 02774, 2018

14 Davison A J, Reid I D, Molton N D, Stasse O. MonoSLAM: real-time single camera SLAM. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2007, 29(6): 1052−1067
DOI:10.1109/tpami.2007.1049

15 Mourikis A I, Roumeliotis S I. A multi-state constraint kalman filter for vision-aided inertial navigation. In: IEEE International Conference on Robotics and Automation. Roma, Italy, 2007: 3565−3572
DOI:10.1109/ROBOT.2007.364024

16 Li M Y, Mourikis A I. Improving the accuracy of EKF-based visual-inertial odometry. In: IEEE International Conference on Robotics and Automation. Saint Paul, MN, USA: 2012, 828−835
DOI:10.1109/ICRA.2012.6225229

17 Huang G P, Mourikis A I, Roumeliotis S I. Analysis and improvement of the consistency of extended Kalman filter based SLAM. In: IEEE International Conference on Robotics and Automation. Pasadena, CA, USA: 2008, 473−479
DOI:10.1109/ROBOT.2008.4543252

18 Jones E S, Soatto S. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. The International Journal of Robotics Research, 2011, 30(4): 407−430
DOI:10.1177/0278364910388963

19 Huang G P, Mourikis A I, Roumeliotis S I. An observability-constrained sliding window filter for SLAM. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. San Francisco, CA, USA: 2011, 65−72
DOI:10.1109/IROS.2011.6095161

20 Huang G P, Mourikis A I, Roumeliotis S I. A quadratic-complexity observability-constrained unscented kalman filter for SLAM. IEEE Transactions on Robotics, 2013, 29(5): 1226−1243
DOI:10.1109/tro.2013.2267991

21 Barrau A, Bonnabel S. An EKF-SLAM algorithm with consistency properties. arXiv: 1510. 06263, 2015

22 Strasdat H, Montiel J M M, Davison A J. Visual SLAM: why filter? Image and Vision Computing, 2012, 30(2): 65−77
DOI:10.1016/j.imavis.2012.02.009

23 Dellaert F, Kaess M. Square root SAM: Simultaneous localization and mapping via square root information smoothing.

The International Journal of Robotics Research, 2006, 25(12): 1181−1203
DOI:10.1177/0278364906072768

24 Thrun S, Montemerlo M. The graph SLAM algorithm with applications to large-scale mapping of urban structures. The International Journal of Robotics Research, 2006, 25(5/6): 403−429
DOI:10.1177/0278364906065387

25 Chen Y, Davis T A, Hager W W, Rajamanickam S. Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. ACM Transactions on Mathematical Software, 2008, 35(3): 1−14
DOI:10.1145/1391989.1391995

26 Davis T A, Gilbert J R, Larimore S I, Ng E G. A column approximate minimum degree ordering algorithm. ACM Transactions on Mathematical Software, 2004, 30(3): 353−376

27 Kaess M, Ranganathan A, Dellaert F. iSAM: incremental smoothing and mapping. IEEE Transactions on Robotics, 2008, 24(6): 1365−1378
DOI:10.1109/tro.2008.2006706

28 Kaess M, Johannsson H, Roberts R, Ila V, Leonard J J, Dellaert F. iSAM2: Incremental smoothing and mapping using the Bayes tree. The International Journal of Robotics Research, 2012, 31(2): 216−235
DOI:10.1177/0278364911430419

29 Ila V, Polok L, Solony M, Svoboda P. SLAM ++- A highly efficient and temporally scalable incremental SLAM framework. The International Journal of Robotics Research, 2017, 36(2): 210−230
DOI:10.1177/0278364917691110

30 Ila V, Polok L, Solony M, Istenic K. Fast incremental bundle adjustment with covariance recovery. International Conference on 3D Vision (3DV). Qingdao, China: 2017, 175−184
DOI:10.1109/3DV.2017.00029

31 Liu H M, Chen M Y, Zhang G F, Bao H J, Bao Y Z. ICE-BA: incremental, consistent and efficient bundle adjustment for visual-inertial SLAM. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition. Salt Lake City, UT, USA: 2018, 1974−1982
DOI:10.1109/CVPR.2018.00211

32 Klein G, Murray D. Parallel tracking and mapping for small AR workspaces. In: 6th IEEE and ACM International Symposium on Mixed and Augmented Reality. Nara, Japan, 2007: 225−234
DOI:10.1109/ISMAR.2007.4538852

33 Mur-Artal R, Montiel J M M, Tardos J D. ORB-SLAM: A versatile and accurate monocular SLAM system. IEEE Transactions on Robotics, 2015, 31(5): 1147−1163
DOI:10.1109/tro.2015.2463671

34 Mur-Artal R, Tardos J D. ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-D cameras. IEEE Transactions on Robotics, 2017, 33(5): 1255−1262
DOI:10.1109/tro.2017.2705103

35 Mur-Artal R, Tardos J D. Visual-inertial monocular SLAM with map reuse. IEEE Robotics and Automation Letters, 2017, 2(2): 796−803
DOI:10.1109/lra.2017.2653359

36 Leutenegger S, Lynen S, Bosse M, Siegwart R, Furgale P. Keyframe-based visual−inertial odometry using nonlinear optimization. The International Journal of Robotics Research, 2015, 34(3): 314−334
DOI:10.1177/0278364914554813

37 Qin T, Li P L, Shen S J. VINS-mono: A robust and versatile monocular visual-inertial state estimator. IEEE Transactions on Robotics, 2018, 34(4): 1004−1020
DOI:10.1109/tro.2018.2853729

38 Lu F, Milios E. Globally consistent range scan alignment for environment mapping. Autonomous Robots, 1997, 4(4): 333−349
DOI:10.1023/A:1008854305733

39 Li P L, Qin T, Hu B T, Zhu F Y, Shen S J. Monocular visual-inertial state estimation for mobile augmented reality. In: IEEE International Symposium on Mixed and Augmented Reality (ISMAR). Nantes, France, 2017: 11−21
DOI:10.1109/ISMAR.2017.18

40 Galvez-López D, Tardos J D. Bags of binary words for fast place recognition in image sequences. IEEE Transactions on Robotics, 2012, 28(5): 1188–1197
DOI:10.1109/tro.2012.2197158

41 Engel J, Koltun V, Cremers D. Direct sparse odometry. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018, 40(3): 611–625
DOI:10.1109/tpami.2017.2658577

42 Jin H L, Favaro P, Soatto S. A semi-direct approach to structure from motion. The Visual Computer, 2003, 19(6): 377–394
DOI:10.1007/s00371-003-0202-6

43 Newcombe R A, Lovegrove S J, Davison A J. DTAM: Dense tracking and mapping in real-time. In: International Conference on Computer Vision. Barcelona, Spain: 2011, 2320–2327
DOI:10.1109/ICCV.2011.6126513

44 Engel J, Schöps T, Cremers D. LSD-SLAM: Large-Scale Direct Monocular SLAM. Computer Vision–ECCV 2014. Cham: Springer International Publishing, 2014: 834–849
DOI:10.1007/978-3-319-10605-2_54

45 Newcombe R A, Izadi S, Hilliges O, Molyneaux D, Kim D, Davison AJ, Kohi P, Shotton J, Hodges S, Fitzgibbon A. KinectFusion: Real-time dense surface mapping and tracking. In: IEEE International Symposium on Mixed and Augmented Reality (ISMAR). Basel, Switzerland: 2011, 127–136

46 Whelan T, Salas-Moreno R F, Glocker B, Davison A J, Leutenegger S. ElasticFusion: Real-time dense SLAM and light source estimation. The International Journal of Robotics Research, 2016, 35(14): 1697–1716
DOI:10.1177/0278364916669237

47 Zhu A Z, Atanasov N, Daniilidis K. Event-based visual inertial odometry. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI, USA: 2017, 5816–5824
DOI:10.1109/CVPR.2017.616

48 Zhou H Z, Zou D P, Pei L, Ying R D, Liu P L, Yu W X. StructSLAM: visual SLAM with building structure lines. IEEE Transactions on Vehicular Technology, 2015, 64(4): 1364–1375
DOI:10.1109/tvt.2015.2388780

49 Hsiao M, Westman E, Kaess M. Dense planar-inertial SLAM with structural constraints. In: IEEE International Conference on Robotics and Automation (ICRA). Brisbane, QLD, Australia: 2018, 6521–6528
DOI:10.1109/ICRA.2018.8461094

50 Radwan N, Valada A, Burgard W. VLocNet++: deep multitask learning for semantic visual localization and odometry. In: IEEE Robotics and Automation Letters, 2018, 3(4): 4407–4414
DOI:10.1109/lra.2018.2869640

51 Wang S, Clark R, Wen H K, Trigoni N. DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. In: IEEE International Conference on Robotics and Automation (ICRA). Singapore, Singapore: 2017, 2043–2050
DOI:10.1109/ICRA.2017.7989236

52 Lianos K N, Schönberger J L, Pollefeys M, Sattler T. VSO: Visual Semantic Odometry. Computer Vision—ECCV 2018. Cham: Springer International Publishing, 2018: 246–263.
DOI:10.1007/978-3-030-01225-0_15

53 Schubert D, Goll T, Demmel N, Usenko V, Stückler J, Cremers D. The TUM VI benchmark for evaluating visual-inertial odometry. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Madrid, Spain: 2018, 1680–1687
DOI:10.1109/IROS.2018.8593419

54 Pfrommer B, Sanket N, Daniilidis K, Cleveland J. PennCOSYVIO: A challenging visual inertial odometry benchmark. In: IEEE International Conference on Robotics and Automation (ICRA). Singapore, Singapore: 2017, 3847–3854
DOI:10.1109/ICRA.2017.7989443

55 Cortés S, Solin A, Rahtu E, Kannala J. ADVIO: An Authentic Dataset for Visual-Inertial Odometry. Computer Vision—ECCV 2018. Cham: Springer International Publishing, 2018, 425–440
DOI:10.1007/978-3-030-01249-6_26

56 Furgale P, Rehder J, Siegwart R. Unified temporal and spatial calibration for multi-sensor systems. In: IEEE/RSJ

International Conference on Intelligent Robots and Systems. Tokyo, Japan: 2013, 1280−1286
DOI:10.1109/IROS.2013.6696514

57 Olson E. AprilTag: A robust and flexible visual fiducial system. In: IEEE International Conference on Robotics and Automation. Shanghai, China: 2011, 3400−3407
DOI:10.1109/ICRA.2011.5979561

58 Umeyama S. Least-squares estimation of transformation parameters between two point patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1991, 13(4): 376−380
DOI:10.1109/34.88573

59 Weiss S, Achtelik M W, Lynen S, Chli M, Siegwart R. Real-time onboard visual-inertial state estimation and self-calibration of MAVs in unknown environments. In: IEEE International Conference on Robotics and Automation. Saint Paul, MN, USA: 2012, 957−964
DOI:10.1109/ICRA.2012.6225147

60 Delmerico J, Scaramuzza D. A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In: IEEE International Conference on Robotics and Automation (ICRA). Brisbane, QLD, Australia: 2018, 2502−2509
DOI:10.1109/ICRA.2018.8460664

61 Klein G, Murray D. Parallel tracking and mapping on a camera phone. In: 8th IEEE International Symposium on Mixed and Augmented Reality. Orlando, FL, USA: 2009, 83−86
DOI:10.1109/ISMAR.2009.5336495