Real-Time Object Pose Tracking System With Low Computational Cost for Mobile Devices

Yo-Chung Lau, Kuan-Wei Tseng, Peng-Yuan Kao, I-Ju Hsieh, Hsiao-Ching Tseng, and Yi-Ping Hung

***Abstract*—Real-time object pose estimation and tracking is challenging but essential for some emerging applications, such as augmented reality. In general, state-of-the-art methods address this problem using deep neural networks, which indeed yield satisfactory results. Nevertheless, the high computational cost of these methods makes them unsuitable for mobile devices where real-world applications usually take place. We propose real-time object pose tracking system with low computational cost for mobile devices. It is a monocular inertial-assisted-visual system with a client–server architecture connected by high-speed networking. Inertial measurement unit (IMU) pose propagation is performed on the client side for fast pose tracking, and RGB image-based 3-D object pose estimation is performed on the server side to obtain accurate poses, after which the pose is sent to the client side for refinement, where we propose a bias self-correction mechanism to reduce the drift. We also propose a fast and effective pose inspection algorithm to detect tracking failures and incorrect pose estimation. In this way, the pose updates rapidly even within 5 ms on low-level devices, making it possible to support real-time tracking for applications. In addition, an object pose dataset with RGB images and IMU measurements is delivered for evaluation. Experiments also show that our method performs well with both accuracy and robustness.**

***Index Terms*—Low computational cost, mobile device, object pose estimation, real-time object pose tracking.**

I. Introduction

THE purpose of object pose estimation and tracking is to find the relative six degrees of freedom (6DoF) transformation, including the translation and rotation, between the object and the camera. This important task plays a significant role in real-life applications, such as augmented reality (AR) [1], [2] and robotic manipulation [3], [4].

Object pose tracking, in contrast to object pose estimation, puts emphasis on tracking object pose in consecutive frames [5], [6]. This is challenging since real-time performance is required to ensure coherent and smooth user experience. Despite the seeming prevalence of solutions, whether they are vision-only [6], [7] or visual–inertial [8], [9], [10], such methods are designed to be run on computers or even servers. Hou et al. [11] proposed lightweight neural networks to track objects on mobile devices, but hardware requirements are still critical. Moreover, with the development of head-mounted displays, frame rate demands have increased. Although 60 FPS may be sufficient for smartphone-based applications, more than 90 FPS is expected for AR glasses to prevent the motion sickness, which makes the problem more difficult. In addition, considering tasks running in AR applications, such as rendering and human–computer interaction, also share the same computational resource at the same time, and it is necessary to make the computational cost of the tracking as low as possible.

We then propose an inertial-assisted-visual system for accurate object pose estimation and tracking to support static scene AR applications (e.g., AR museum exhibition [12], [13], [14], [15]) on mobile devices. Unlike traditional visual–inertial methods using pose fusion, we track the object pose mainly with inertial measurements and refine it with visual information coordinately. As shown in Fig. 1, our system uses a client–server architecture that performs fast pose tracking on the client side (mobile device) and accurate pose estimation on the server side. The accumulated error or drift on the client is diminished by data exchanges with the server. Specifically, the client is composed of three modules: a pose propagation module (PPM) to calculate a rough pose estimation via inertial measurement unit (IMU) integration, a pose inspection module (PIM) to detect tracking failures, including lost tracking and large pose errors, and a pose refinement module (PRM) to optimize the pose and update the IMU state vector to correct the drift based on the response from the server, where we run state-of-the-art object pose estimation methods using RGB images. This pipeline not only achieves real-time and accurate tracking on low-end mobile devices, but also saves

Manuscript received 30 September 2023; revised 22 November 2023 and 5 December 2023; accepted 5 December 2023. Date of publication 11 December 2023; date of current version 18 December 2023. This work was supported in part by the National Science and Technology Council, Taiwan, under Grants NSTC 109-2221-E-002 -207-MY3 and NSTC 112-2218-E-119-001-MBK. *(Corresponding author: Yo-Chung Lau.)*

Yo-Chung Lau is with the Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei 10617, Taiwan, and also with the Digital Innovation Laboratory, Chunghwa Telecom Laboratories, Taoyuan 326402, Taiwan (e-mail: lyc0326@cht.com.tw).

Kuan-Wei Tseng is with the Department of Computer Science, Tokyo Institute of Technology, Tokyo 152-8550, Japan (e-mail: kuanwei@g.ntu.edu.tw).

Peng-Yuan Kao and Yi-Ping Hung are with the Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei 10617, Taiwan (e-mail: zbabqr@gmail.com; hung@csie.ntu.edu.tw).

I-Ju Hsieh and Hsiao-Ching Tseng are with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10617, Taiwan (e-mail: r10922094@csie.ntu.edu.tw; r10922022@csie.ntu.edu.tw).

This article has supplementary downloadable material available at https://doi.org/10.1109/JISPIN.2023.3340987, provided by the authors.

Digital Object Identifier 10.1109/JISPIN.2023.3340987

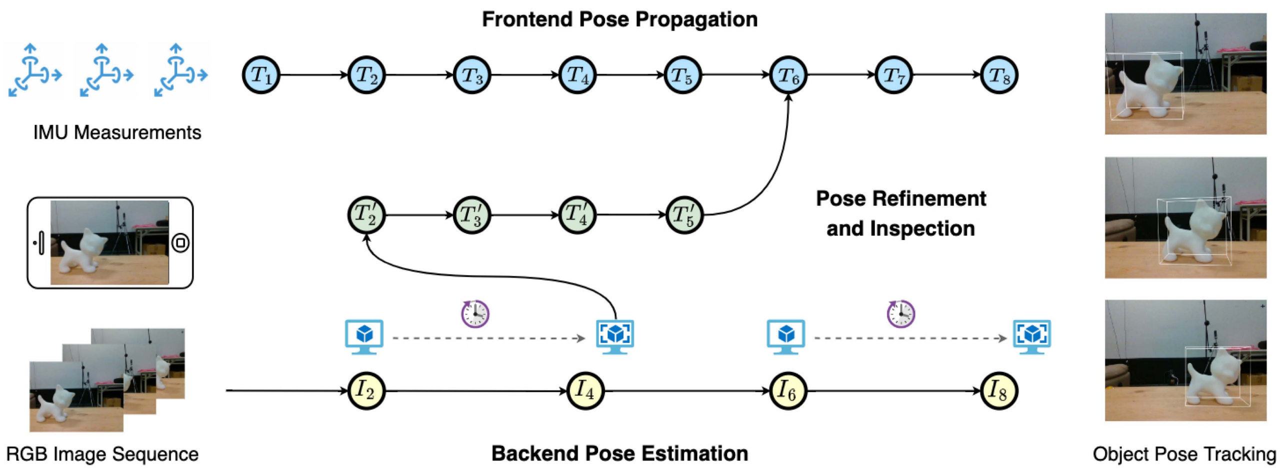

Frontend Pose Propagation
IMU Measurements
T_1 T_2 T_3 T_4 T_5 T_6 T_7 T_8
T'_2 T'_3 T'_4 T'_5
Pose Refinement and Inspection
I_2 I_4 I_6 I_8
RGB Image Sequence
Backend Pose Estimation
Object Pose Tracking

Fig. 1. System overview. We propose an object pose tracking system with a client–server architecture for mobile applications. The input of the system is IMU measurements and RGB image sequences. On the frontend side (mobile device), we perform the fast pose propagation based on IMU measurements. On the backend side (server), we utilize 6DoF object pose estimation models to estimate the object pose based on RGB images. A more accurate object pose from the backend will be sent back to the frontend to refine the pose and calibrate the biases of IMU measurements. Note that T_i and I_i stand for the object pose and image taken at timestamp i, respectively.

the computational cost on the tracking time within 5 ms per frame. The main contributions of our work are summarized as follows.

1) We proposed a monocular inertial-assisted-visual system with a client–server architecture to track the object pose with an extremely low computational cost on low-level mobile devices. To the best of authors' knowledge, our method is the most computationally efficient.
2) We developed a bias self-correction mechanism (BSCM) to improve the accuracy of pose propagation in tracking.
3) We designed a pose inspection algorithm (PIA) to efficiently and quickly exclude probable wrong poses.
4) We collected a real-world object pose dataset with RGB images and IMU measurements to evaluate the tracking.

The rest of this article is organized as follows. In Section II, we review the related work on the object pose estimation and tracking. In Section III, we propose our method and explain the detail specifically. In Section IV, we explore the experiments and discuss the results. Finally, Section V concludes this article.

II. Related Work

A. Object Pose Estimation

Object pose estimation has long been an open issue; of the many studies on this, some [16], [17], [18], [19] use the depth information to address this problem and indeed yield satisfactory results. Unfortunately, RGB-D images are not always supported or practical in most real use cases. As a result, we then focus on methods that do not rely on the depth information.

1) Classical Methods: Conventional methods that estimate the object pose from an RGB image can be classified either as feature-based or template-based. In feature-based methods [20], [21], [22], features in 2-D images are extracted and matched with those on the object 3-D model, and the object pose can be estimated by a perspective-n-point (PnP) solver. This kind of method still performs well in occlusion cases, but fails in textureless objects without distinctive features. Template-based methods [23], [24], [25] can handle both textured and textureless objects. Synthetic images rendered around an object 3-D model from different camera viewpoints are generated as a template database, and the input image is matched against the templates to find the correct object pose. However, these methods are sensitive and not robust when objects are occluded.

2) Deep Learning-Based Methods: Learning-based methods can also be categorized into direct and PnP-based approaches. Direct approaches regress or infer poses with feed-forward neural networks. SSD6D [26] disentangles the 6DoF pose into viewpoint and in-plane rotation, first by estimating the rotation and then by inferring the 3-D translation with a rotation and bounding box. PoseCNN [27] generates semantic labels and localizes the object center with its distance to the camera via a CNN network. It constructs a region of interest (ROI) and regresses the ROI features to estimate 3-D translation and rotation. PnP-based approaches find 2D–3D correspondences by deep learning, and the estimation of object pose is handled by other PnP solvers. BB8 [28] processes the coarse-to-fine segmentation of the object from an image and predicts its pose based on the projection of object bounding box. PVNet [29] selects keypoints using a voting-based algorithm based on the distance from the center to the surface of the 3-D object model, which allows it to effectively tackle occluded objects. Yu et al. [30] proposed differentiable proxy voting loss to reduce the search error of object keypoints. Some studies, such as RePOSE [31] and RNNPose [32], add postrefinement procedures for better pose accuracy. However, these multistage pipelines are too slow for real-time applications.

B. Object Pose Tracking

Unlike the general concept of the object tracking, which means to know just where the target object is in the video or consecutive images, the tracking we discuss in this article is to keep localizing the 6DoF object poses. In addition to a single image, temporal information among frames is also utilized to facilitate estimation. Studies [16], [17], [33], [34] use a stereo/depth camera or Lidar to help tracking, but this is not practical in real use cases in which only a monocular camera is available. In real-world AR applications, instead of using stereo or RGB-D cameras, IMUs are also commonplace solutions. Thus, we briefly introduce vision-based and visual–inertial-based methods.

1) Vision-Based Methods: Classical vision-based methods track features, such as SIFT, SURF, and ORB, to estimate the correct pose by solving a PnP problem. Likewise, these methods may have high accuracy but their high computational overhead and low robustness to the image distortion and self-occlusion are problems [35]. Based on deep learning, Zhong et al. [6] tracked objects in video effectively by segmenting objects from the frame even with heavy occlusion.

2) Visual–Inertial-Based Methods: As the development of simultaneous localization and mapping [36], conventional visual–inertial fusion using extended Kalman filters [10], [37], [38] or nonlinear optimization [9], [39] has been deployed for AR and robotic applications. However, these suffer from problems of low frame rates and the long delay due to their high computational costs. Recently, learning-based methods [8], [40], [41] have been proposed, which regress the fused visual and inertial features for the pose estimation.

III. Proposed Method

Compared with studies on implementations for PCs or servers mentioned above, there is a lack of studies for mobile devices. MobilePose [11] uses two lightweight neural network models to track objects and shows strong results on smartphones. It achieves 36 FPS on a Galaxy S20. However, in order to support the fluent user interaction, applications may need higher requirements of frame rate (e.g., more than 90 FPS for mixed reality on smart glasses) and lower processing time in tracking to save computational resources to support other tasks, such as rendering and human–computer interaction control. These critical requirements and restrictions also further complicate the problem.

A. System Architecture

For real-time object pose tracking, the main factors lie on how to properly use the contextual information between consecutive time points and quickly update the pose based on it. To minimize the computational cost and support fluent object pose tracking on general mid-level or even low-level mobile devices, tasks running on mobile ends should not be overly complicated. Specifically, for static scene AR applications, the change of motion between two continuous frames is usually small; that means, if the object pose in the current frame is known, the pose in the next frame can be updated just based on the motion change between frames from the device.

Inspired by studies of outdoor localization [42], [43], [44], we then use a client–server design to separate out the frontend and backend tasks. Complicated and time-consuming tasks, such as the object recognition and precise pose estimation, are processed on the backend where powerful computers or servers with the high-level hardware are executed. Simple tasks, such as the pose propagation and checking, are handled by the mobile device itself on the frontend. The two ends are connected to each other through a high-speed network, such as 5G or WiFi. In this way, poses update rapidly, making it possible to track the object instantly with a very low computational cost.

The system workflow is shown in Fig. 2. There are three modules running on the mobile device frontend: the PPM, the PIM, and the PRM. When the system starts, the PRM sends an RGB frame to the backend to obtain the object pose for the frontend initialization. Here, the solution implemented on the backend is not restricted but instead kept open, so that hosts can choose the method that best fits their needs. For example, PVNet [29] may be a choice for general use. Once the frontend is initialized, the PPM then regularly updates the pose according to IMU measurements. Meanwhile, the PIM checks the correctness of pose by the proposed PIA. The PRM repeatedly optimizes the pose computed by the PPM using the response from the backend to maintain the accuracy.

B. System Modules

1) Pose Propagation Module: As the PPM periodically updates the object pose according to the IMU data by pose propagation, its processing frequency is equal to the IMU sample rate, which is the maximum supportable tracking rate. The IMU data are also saved in the system for later pose updates in the PRM.

2) Pose Inspection Module: The PIM checks the correctness of the pose by PIA when a frame arrives. It reports back with *fine pose*, *wrong pose*, or *tracking lost*. The pose is accepted in case of fine pose; other statuses are classified as failures. The PIM reinitializes the PRM status when a failure occurs, and the current frame is processed immediately in the PRM in case of wrong pose.

3) Pose Refinement Module: The PRM attempts to retrieve the frame's correct object pose through the cloud. A pose estimation request with the frame is sent to the backend, after which the module waits for a response. If we assume the request sent and response received occurred at time t_0 and t_1 respectively, the correct object pose at time t_1 can be calculated based on the backend result and previously held IMU data from time t_0 to t_1. Meanwhile, we also perform BSCM, which leverages the correct object pose from the backend to compensate for the drift. In general, the PRM is triggered for every frame if it is idle, but it can also be triggered by the PIM when a wrong pose is found. It is noteworthy that the whole processing time should be concerned not to be larger than IMU sample time to ensure the smooth pose updating in the PPM.

C. Pose Propagation by IMU Measurement

For a static scene, given a known camera pose M_t at time t, we have $M_{t+\delta t}$ at time $t + \delta t$ with

$$M_{t+\delta t} = M_t^{t+\delta t} M_t \tag{1}$$

where $M_t^{t+\delta t} = [R_{t+\delta t} \mid T_{t+\delta t}]$ is the camera pose transformation matrix. It represents the transformation of the rotation, $R_{t+\delta t}$, and translation, $T_{t+\delta t}$, from time t to $t + \delta t$. When δt is

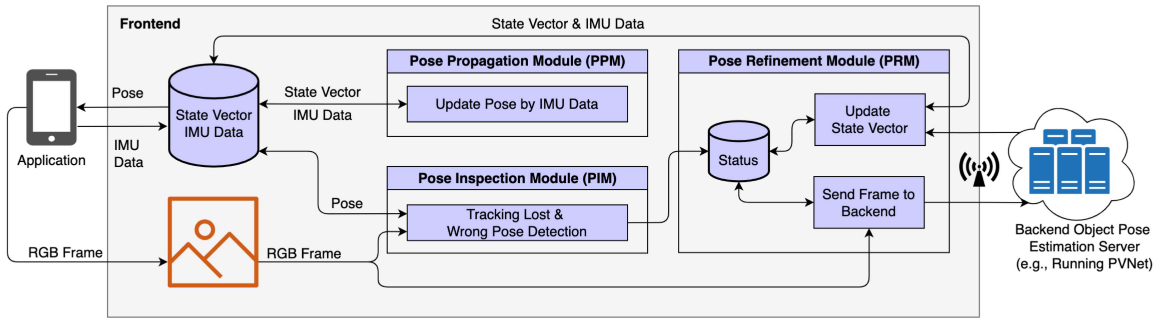


Fig. 2. System workflow. The system is composed of a frontend client and a backend server. The frontend performs the fast pose propagation with IMU measurements and fuses the result of visual pose estimation by the backend server. The client is composed of three modules: a PPM to update the pose via IMU integration, a PIM to detect tracking failures, including lost tracking and wrong pose, and a PRM to optimize the pose and update the IMU state vector to correct the drift based on the pose received from the server, where we run state-of-the-art object pose estimation methods using RGB images. The state vector contains the device pose and motion information, such as the velocity and biases of IMU measurements.

very small, it is safe to calculate $M_t^{t+\delta t}$ by the IMU measurement on the device [45].

Theoretically, $R_{t+\delta t}$ can be calculated by the rotational change of angular velocity from the gyroscope, and $T_{t+\delta t}$ can be determined by the moving offset based on the acceleration from the accelerometer as follows [46]:

$$R_{t+\delta t} = R_t \left(I + \frac{\sin\sigma}{\sigma} B + \frac{1-\cos\sigma}{\sigma^2} B^2 \right) \tag{2}$$

$$B = \begin{bmatrix} 0 & -\omega^z_{t+\delta t}\delta t & \omega^y_{t+\delta t}\delta t \\ \omega^z_{t+\delta t}\delta t & 0 & -\omega^x_{t+\delta t}\delta t \\ -\omega^y_{t+\delta t}\delta t & \omega^x_{t+\delta t}\delta t & 0 \end{bmatrix} \tag{3}$$

$$\sigma = |\omega_{t+\delta t}\delta t| \tag{4}$$

$$\omega_{t+\delta t} = \left[\omega^x_{t+\delta t}, \omega^y_{t+\delta t}, \omega^z_{t+\delta t}\right]^T \tag{5}$$

where I is the identity matrix and $\omega_{t+\delta t}$ is the device local angular velocity sampled at time $t+\delta t$.

$$T_{t+\delta t} = T_t + \delta t V_{t+\delta t} \tag{6}$$

$$V_{t+\delta t} = V_t + \delta t \left(R_{t+\delta t}\, a_{t+\delta t} - g\right) \tag{7}$$

where $V_{t+\delta t}$ is the device velocity at time $t+\delta t$, $a_{t+\delta t}$ is the device local acceleration sampled at time $t+\delta t$, and g is the acceleration of gravity.

Although it would seem that $M_{t+\delta t}$ can be propagated directly from M_t by (1)–(7), there may be a problem about correctness. First, IMU data are usually polluted by noise and bias, which makes the pose calculated unreliable. Second, we lack a good initial estimation of system velocity, which results in the translational error. This is nontrivial because there is no reference of the velocity from the backend. The correctness of IMU data and system velocity is so critical to pose propagation that we propose BSCM to compensate for the error.

D. Bias Self-Correction Mechanism

BSCM improves the pose accuracy during pose propagation by removing the bias of IMU data and system velocity. Generally, bias-dependent pose error accumulates and amplifies as the pose is propagated continuously. The estimated rotation $\widehat{R}_{\text{imu}}$ obtained from IMU data is defined as

$$\widehat{R}_{\text{imu}} = R_{\text{noise}} R_{\text{bias}} R_{\text{real}} \tag{8}$$

where R_{real} is the true rotation, and R_{bias} and R_{noise} are erroneous rotation resulted from the bias and noise. By omitting the noise term, which is essentially random, we can approximate the bias $\widehat{R}_{\text{bias}}$ using the estimated true rotation $\widehat{R}_{\text{real}}$ from the backend as

$$\widehat{R}_{\text{bias}} \approx \widehat{R}_{\text{imu}} \widehat{R}^{-1}_{\text{real}}. \tag{9}$$

The corresponding XYZ Euler angles ($\widehat{E}_{\text{bias}}$) can be decomposed from $\widehat{R}_{\text{bias}}$, and the time difference (Δt) from the time of last triggered backend pose estimation to now is also known. Hence, the bias of angular velocity can be written as

$$\widehat{\omega}_{\text{bias}} = \frac{\widehat{E}_{\text{bias}}}{\Delta t}. \tag{10}$$

As for the velocity and acceleration biases, we use the average velocity calculated by consecutive poses as the reference. Assume we have two consecutive frame poses at time t_0 and t_1 from the backend, the system average velocity at time $\frac{t_0+t_1}{2}$ (denoted as t_0^1) can be represented as

$$\widehat{V}_{\text{avg}} = \frac{\widehat{T}_{t_1} - \widehat{T}_{t_0}}{t_1 - t_0} \tag{11}$$

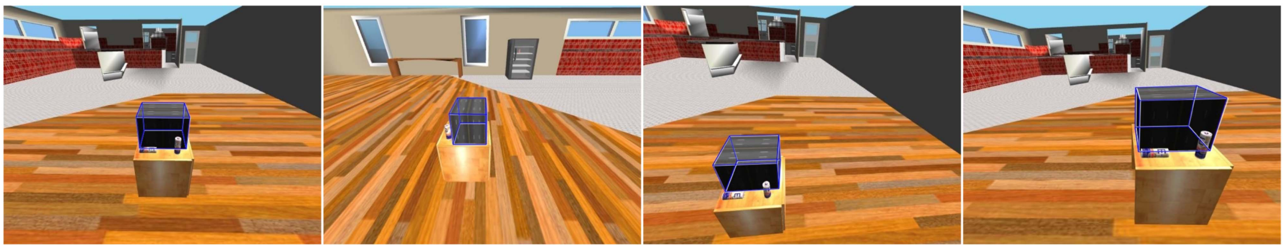
where $\widehat{T}_{t_0}$ and $\widehat{T}_{t_1}$ are the translation at time t_0 and t_1 estimated by the backend. Thus, we have the system velocity bias at time t_0^1 as

$$\widehat{V}_{\text{bias}} = \widehat{V}_{\text{imu}} - \widehat{V}_{\text{avg}} \tag{12}$$

where $\widehat{V}_{\text{imu}}$ is the system velocity closest to time t_0^1 calculated based on IMU data. Therefore, the acceleration bias can be derived as

$$\widehat{a}_{\text{bias}} = \frac{\widehat{V}_{\text{bias}}}{t_1 - t_0}. \tag{13}$$

Through BSCM, we not only estimate and remove the IMU biases but also regularly compensate for the velocity error.


(a)

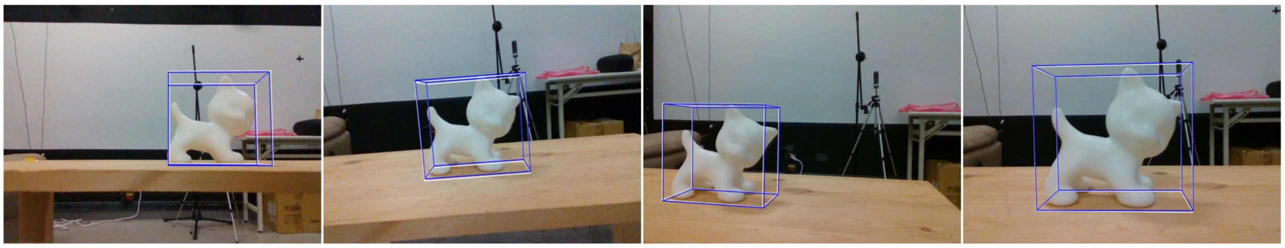

(b)

Fig. 3. Demonstration of experiments. We tracked the gray box and white cat in simulation and real-world experiment, respectively. The figure shows the tracking results in different camera viewpoints, in which we drew the bounding boxes of the object in white with the estimated pose and in blue with the GT, respectively. Note that two bounding boxes in each picture are almost overlapped accurately. (a) Simulation (Noisy backend) (b) Real-world experiment.

E. Pose Inspection Algorithm

A backend pose is reliable and image-independent, but that from the frontend is not always guaranteed. PIA is proposed to quickly check whether the frontend pose is acceptable by excluding probable cases of lost tracking and wrong poses. To satisfy the goal of low computational cost, the first concern of PIA is still its execution speed. Therefore, PIA is not designed for absolutely accurate pose inspection, instead, its effectiveness is reflected in finding out most suspicious poses as fast as possible.

There are two cases in which the propagated pose is unacceptable, including the lost tracking and wrong pose. The tracking can be considered lost when the object is out of the camera view or it is too small in the frame. Once the tracking is lost, the pose calculated becomes meaningless and should not be used, even if it is correct. On the other hand, the pose propagation is sensitive to the motion change captured by IMU and may not be reliable when the device moves too rapidly. As a result, for safe use, we should also regard the pose as a wrong one if a large motion is detected.

According to the above concept, the core PIA algorithm is to first find the bounding box of the object in the image based on the pose. We check the projection area and determine for the tracked object whether the area is not less than a threshold, $\mathrm{THR}_{\mathrm{area}}$, or we consider tracking lost. Afterward, we calculate the mean offset of the vertices from those on last frame. If the offset is less than a threshold, $\mathrm{THR}_{\mathrm{2d}}$, we take it as a fine pose; otherwise, we assume that the motion is too drastic that the pose propagation might be poor, and a wrong pose is returned. Obviously, if the threshold is set too small, the pose inspection will become too sensitive to the motion change, while there may be an opposite result in cases of larger settings. The setting of $(\mathrm{THR}_{\mathrm{area}}, \mathrm{THR}_{\mathrm{2d}})$ is actually a tradeoff problem and empirically defined as $(\frac{\mathrm{framearea}}{100}, 17)$ here. We will have a more detailed discussion about it in Section IV-C2.

IV. Experiments

Since there is no publicly available dataset for both visual and inertial object pose tracking, we first evaluated the proposed system with a simulated dataset using Gazebo (v11.0.0) [47]. Subsequently, we collected a lightweight real-world dataset ourselves and used it to verify our method.

A. Experiments on Simulated Data

As Fig. 3(a) shows, we created an indoor scene in which the target object was placed on a table and its initial distance to the device with an on-board monocular camera and IMU was about 1.2 m. The camera captured images up to 120 FPS, and the IMU sample rate was about 200 Hz throughout the experiment. The IMU noise parameters were 6.63×10^{-5} rps/$\sqrt{\mathrm{Hz}}$ for the gyroscope and 7.35×10^{-4} m/s$^2\sqrt{\mathrm{Hz}}$ for the accelerometer, which resembled the LSM6DSM, a consumer-level IMU chipset in the Google Pixel 2. In our simulations, the camera followed the target object, and its movement followed predefined motion programs. After that, we compared the tracking results with the true poses.

1) Motion Programs: In order to simulate the diverse interaction behaviors of real interactive applications, such as AR, we used translational and circular motion scripts, each with three levels of difficulty. The detail of scripts is given in Table I. Each script ran for 30 s. For the translational motion, we primarily moved the camera using a combination of dolly, pedestal, and trucking; for the circular motion, we moved and rotated the camera around the object. We also applied a random additional

TABLE I
SIMULATION SPECIFICATIONS

Quantity		Translational		Angular	
Script		Average speed (m/s)	Acceleration range (m/s^2)	Average rate (rad/s)	Acceleration range (rad/s^2)
Translational	Easy	0.06	[0, 0.19)	0	0
	Medium	0.12	[0.17, 0.36)	0.01	[0, 0.15)
	Hard	0.18	[0.35, 0.54)	0.04	[0.15, 0.30)
Circular	Easy	0.07	[0.01, 0.02)	0.06	[0, 0.04)
	Medium	0.15	[0.04, 0.06)	0.33	[0, 0.15)
	Hard	0.23	[0.09, 0.14)	0.40	[0, 0.34)

force and torque to the camera, roughly resembling real-world vibration or shaking. Thus, the higher the difficulty of the script, the faster the movement and the larger the force and torque.

2) Networking and Transmission: Considering the updating frequency of backend pose, which is proportional to the network speed may influence the tracking accuracy proportionally, we set the network transmission speed and data propagation delay to 50 Mbps and 10 ms according to the minimum standard of a 5G network [48], which could be considered to be the worst case in verification. Also, as the real-world transmission is imperfect, we added a uniformly distributed (U) extra delay of 0–30 ms. The size of the transmitted data was 100 kB, including a 90-kB 640 × 480 JPEG image with a compression ratio of 10, and a 10 kB of metadata containing the state vector and other information. Once the server received the data, the backend task started instantly without delay. The backend response time was thereby

$$\begin{aligned}\text{Time}_{\text{trans}} &= \frac{100\ (\text{size}) \times 8\ (\text{kB to kb}) \times 10^3\ (\text{s to ms})}{1024\ (\text{kb to Mb}) \times 50\ (\text{network speed})}\\ &\quad + 10\ (\text{propagation delay}) \times 2\ (\text{back/forth})\\ &\quad + U\ (0\ \text{ms},\ 30\ \text{ms})\ (\text{extra delay})\\ &\approx (35\ \text{ms},\ 66\ \text{ms}). \qquad (14)\end{aligned}$$

3) Computational Power Conversion: In order to specifically understand the performance on the computational cost of our method, low-level phones with limited computational power released several years ago, such as the Google Pixel 2 released in 2017, are taken as simulated devices. Thus, we restricted the CPU performance of our testing device so that its computational power was commensurate with the Google Pixel 2. We referred to the CPU multicore scores from Geekbench [49]. The performance ratio of two CPUs can be directly obtained from the ratio of multicore scores. Based on this assumption, we delayed the processing time of each operation on the frontend by adding extra sleep. Specifically, the multicore scores of our device and Google Pixel 2 were 3185 and 1294, respectively. The performance ratio was thereby 3185/1294 ≈ 2.46. Hence, we extended the processing time of each frontend operation by 246%.

4) Simulation Results and Discussion: Fig. 3(a) shows a glimpse of the tracking results in different camera viewpoints. For the more qualitative visualization of our tracking system, refer to the Supplementary Material. Following the convention,

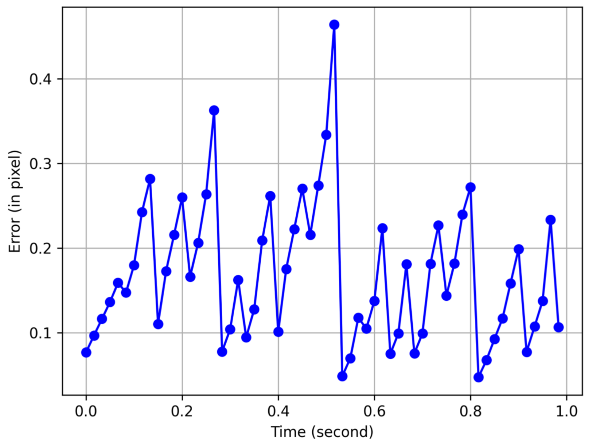


Fig. 4. 2-D projection errors among the pose refinement processes. The tracking error in our method will be fixed periodically by BSCM with the responses from the backend. Each local fluctuation here (between the local maximum and minimum) can be considered as a cycle of the pose propagation and refinement.

we computed the 2-D projection error by measuring the distance projected on the image plane using the estimated pose and the true pose. Besides, considering the perfect pose estimation solution with no error is nonexistent, we noised the backend ground-truth (GT) poses as noisy ones with Gaussian noise, and then compared the tracking performance in these two conditions. The mean 6DoF pose (translation and rotation) and 2-D projection errors were revealed in Table II.

In general, the error increased in harder sequences with higher motion complexity. If GT backend was utilized, the tracking was very smooth and accurate in all experiments. Its 2-D projection error was less than 1 pixel among all sequences regardless of motion complexity. The result has validated our core element that once the true pose is received from the backend, the pose is refined and the error should decrease. More clearly, Fig. 4 shows the object 2-D projection error of frames (points) within a short period of time in tracking. Each fluctuation can be taken as one process of the pose refinement. The pose error first kept increasing to the peak, after which the refinement occurred, leading to fixing the pose error of the next frame. Specifically, for each fluctuation, the pose refinement would happen in the interval between the peak and the foot.

On the other hand, if the backend returned a noisy pose, which reflected the practical situation, then there was an admitted drop in the performance. The average 2-D projection error became about 2 pixels, but was still below the critical threshold of 5 pixels [29], meaning the pose was correct and acceptable. This demonstrated that even with an imperfect backend solution, our system was still capable of maintaining acceptable poses and reducing the impact of inaccurate backend responses in tracking.

Moreover, we analyzed the influence of frame rate on tracking accuracy. If GT backend was used, the error decreased at a higher frame rate due to a shorter waiting time to send a pose estimation request. Consequently, it benefited the pose refinement and pose propagation afterward. Nevertheless, a counterproductive result would happen in noisy backend case. In other words, if the backend pose was noisy, then higher frame rate would lead to

TABLE II
COMPARISON OF MEAN POSE ERROR OF TRACKING IN SIMULATION

Motion Type		Translational			Circular		
Backend	Frame rate	Easy	Medium	Hard	Easy	Medium	Hard
GT	30 *FPS*	1.17/0.00/0.20	2.29/0.00/0.45	3.39/0.01/0.69	0.21/0.02/0.10	0.50/0.03/0.17	1.02/0.06/0.32
	60 *FPS*	1.05/0.00/0.18	2.07/0.00/0.41	3.13/0.01/0.64	0.19/0.02/0.08	0.49/0.03/0.15	0.90/0.05/0.25
	90 *FPS*	1.04/0.00/0.18	2.02/0.00/0.40	2.99/0.01/0.60	0.21/0.02/0.09	0.51/0.03/0.18	0.94/0.05/0.29
	120 *FPS*	1.02/0.00/0.18	1.97/0.00/0.39	2.93/0.01/0.59	0.22/0.02/0.10	0.53/0.04/0.19	0.98/0.06/0.32
Noisy	30 *FPS*	4.89/0.47/1.96	5.16/0.48/2.10	5.78/0.47/2.15	4.68/0.48/2.00	4.73/0.47/2.00	4.80/0.48/2.01
	60 *FPS*	5.16/0.50/2.09	5.39/0.50/2.15	5.86/0.50/2.28	5.03/0.50/2.09	5.04/0.51/2.14	5.12/0.51/2.10
	90 *FPS*	5.24/0.52/2.15	5.53/0.51/2.22	5.92/0.52/2.31	5.13/0.51/2.17	5.12/0.51/2.14	5.21/0.52/2.16
	120 *FPS*	5.26/0.51/2.14	5.52/0.52/2.25	5.98/0.52/2.33	5.16/0.52/2.19	5.27/0.52/2.16	5.26/0.53/2.21

Pose error: position (*mm*) / orientation (°) / 2-D projection (*pixel*)

higher error. We believe the reason behind this contradictory phenomenon was that the error accumulated during the latency was less than the noisy backend poses. In this case, we actually added additional error to the system, while there was no serious drift on the frontend. However, this problem was also mitigated by our BSCM, and the tracking accuracy was maintained.

B. Experiments on Real-World Data

1) Real-World Data Collection: Apart from simulation, we collected our own dataset to validate our system under the real-world scenario. We utilized Intel RealSense D435i camera to capture RGB images, depth images, and IMU measurements in a room with Vicon motion capture system. The average frame rate of RGB images was 52 FPS, while the IMU sample rate was 200 Hz. The target object was a 3-D printed cat model from LineMOD [24]. As shown in Fig. 3(b), we put the cat model on the table and moved the camera in a common movement pattern of general interactive AR applications. We totally recorded five 30 s videos for experiments. A conceptual sketch of the setup could be found in Fig. 5.

To obtain the object pose with respect to the camera is a nontrivial work. Inspired by the work in [50], we first calibrated the extrinsic parameters by using a checkerboard with Vicon markers attached at each corner. Combined with intrinsic parameters, we then had the transformation between image plane and the global coordinate system defined by Vicon. Next, we backprojected depth maps with corresponding camera poses to reconstruct a 3-D point cloud model of the cat under the global coordinate system. We then registered the reconstructed 3-D point cloud model with the original computer aided design (CAD) model of the cat using iterative closest point (ICP) to obtain the transformation between the predefined object-centric coordinate system and the global coordinate system. Eventually, we could compute the 6DoF object pose, which was the transformation between object-centric coordinate system and camera coordinate system. In practice, since the depth maps may fail to precisely segment the silhouette of the object, we performed some postprocessing with ICP by each frame. Although the object poses are still noisy, it serves a great opportunity to examine whether our system would work in the real world where the backend would actually return poses with noise.

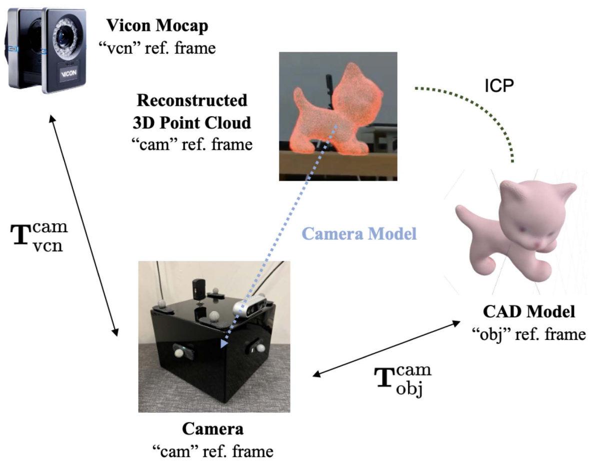


Fig. 5. Real-world dataset construction. The data were collected in a room with the Vicon motion capture system. We obtained the transformation between camera and Vicon $\mathbf{T}_{\text{vcn}}^{\text{cam}}$ by camera calibration and computed the transformation between camera and object $\mathbf{T}_{\text{cam}}^{\text{obj}}$ via the ICP algorithm.

TABLE III
MEAN POSE ERROR OF THE REAL-WORLD EXPERIMENTS

	#1	#2	#3	#4	#5
Position Error (*mm*)	1.65	2.06	2.01	1.25	1.30
Orientation Error (°)	1.54	6.96	8.17	8.23	3.91
2-D Projection Error (*pixel*)	2.56	2.93	3.37	2.21	2.30

2) Real-World Experiment Results and Discussion: We then verified our method with the collected real-word data based on the network setting and computational power conversion mentioned in Sections IV-A2 and IV-A3. Fig. 3(b) demonstrates the tracking results from several camera viewpoints. We also summarized the mean pose and 2-D projection errors of each sequence in Table III. We found that compared with the simulated data, the rotational error was larger. This could be attributed to the shaking of the capturing device during data collection. Videos shot by handheld devices without stabilizer would inevitably contain such movements. As a consequence,

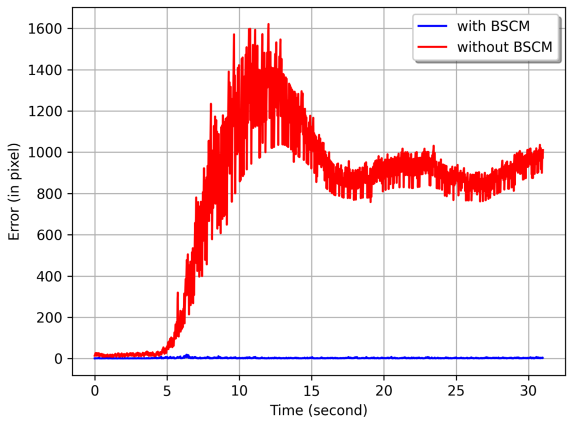


Fig. 6. Ablation study on BSCM by comparing the 2-D projection error in experiments w/wo BSCM.

the average angular velocity (and acceleration) was larger than the simulated data. Besides, the magnitude of translational error was about that of simulated ones with GT backend although it utilized noisy backend poses. This resulted from the slower moving speed in real-world applications. We moved the camera faster in the simulated data to examine the robustness of model. However, for real-world AR applications we focus on what mentioned in Section I, it is less possible that users would move so fast. Thus, the smaller error was due to the slower movement.

C. Ablation Studies

In this section, we discuss the influence about BSCM and provide a detailed analysis of thresholds of PIA in our method.

1) Influence of BSCM: To verify the effectiveness of the proposed pose refinement mechanism, we conducted an ablation study on the PRM by disabling the BSCM and PIA. Note that once the BSCM and PIA are disabled, the IMU biases would never be removed during pose propagation. The result is shown in Fig. 6. The tracking with BSCM performed well with a good pose accuracy, while huge 2-D projection errors of poses were observed in case of no BSCM. We also found that the error could still decrease periodically in tracking without BSCM. This was because the pose would still be repropagated on the frontend if a backend pose was received. However, even with the correct pose from the backend, the repropagated pose was still wrong with a huge error since the incorrect system velocity and IMU measurements with biases were used. BSCM was, thereby, proven important to keep the tracking accuracy.

2) Thresholds in PIA: There are two thresholds in the PIA, including $\mathrm{THR}_{\mathrm{area}}$ and $\mathrm{THR}_{\mathrm{2d}}$. $\mathrm{THR}_{\mathrm{area}}$ is used to roughly screen out reasonable cases of lost tracking, which is actually related to what kind of the object pose estimation algorithm is used on the backend. For example, if the backend solution can still estimate the object pose well even if the object is far away from the device (camera), $\mathrm{THR}_{\mathrm{area}}$ could be defined smaller. As a result, compared with $\mathrm{THR}_{\mathrm{area}}$, the discussion on $\mathrm{THR}_{\mathrm{2d}}$ would be more practical and significant in the PIA.

We investigated the performance of PIA by conducting an ablation study on various configurations of $\mathrm{THR}_{\mathrm{2d}}$ shown in

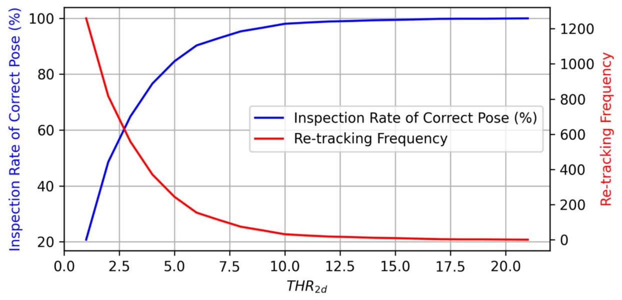


Fig. 7. Ablation study on the performance of PIA with various configurations of thresholds.

Fig. 7. Although it seemed we already had a pretty good result of the correct pose inspection rate when $\mathrm{THR}_{\mathrm{2d}}$ was around 12.5, there was another problem about the retracking. As described in Section III-B2, if the pose does not pass the inspection, the original tracking is interrupted and the system will be reinitialized. Accordingly, the retracking occurs. It is also intolerable that the tracking is interrupted too many times but the tracking pose estimated is acceptable actually. Thus, we must broaden the threshold. Nevertheless, as we mentioned in Section III-E, the inspection would also be meaningless if the threshold is too large. In fact, it is essentially a tradeoff problem and may have different settings depending on different cases. For these reasons, we thereby set $\mathrm{THR}_{\mathrm{2d}}$ as 17 for balance in our experiments eventually.

D. Computational Cost

Compared with the tracking accuracy, for the support on the low-end devices, the frontend workload is always the most important issue we care about actually. We also measured the mean processing time of the pose propagation (in the PPM), inspection (in the PIM), and refinement (in the PRM). The mean pose propagation and inspection time were around 0.22 and 0.88 ms, which were both much smaller than the IMU updating time of 5 ms. The pose refinement, as we described in Section III-B3, should be the most time-consuming in tracking. This is because, after a backend pose is received, lots of pose propagation will be redone again based on the new updated system state vector. However, even in demanding condition of limited computational power, the pose refinement still just took around 2.01 ms in average. Through experiments, our method was proven not only real time but also highly cost effective in computation.

E. Comparisons With state of the art (SOTA)

As given in Tables II and III, the tracking accuracy of our method is around 3 pixels of 2-D projection error on average. Rather than comparing the tracking error of 1 or 2 pixels that does not affect the actual use, we put emphasis on the performance of computational cost in the form of the pose updating time, which is the key we always focus on. We compared our method with other state-of-the-art tracking solutions of VIPose [8] and MobilePose [11] in Table IV. In fact, the comparison is nontrivial since the other two methods both need GPU support, while it is

TABLE IV
COMPARISON OF COMPUTATIONAL COST OF TRACKING

	Pose Update Interval (*ms*)	Computing Device (Type)
MobilePose [11]	27.78	Galaxy S20 (GPU)
VIPose [8]	19.92	RTX 2080 Ti (GPU)
Ours (w/o refinement)	1.10	Google Pixel 2 (CPU)
Ours (full)	**3.11**	Google Pixel 2 (CPU)

We reported the mean pose updating time among different methods. For our method, we also presented the result without the pose refinement.

not to ours. That means, for some current mid-level or even low-level mobile devices without GPU support, these two methods are not usable. The requirement of hardware for our method is the lowest. However, even in such poor hardware condition, the pose updating time of our method (*3.11* ms) was still far less than the others', which demonstrated the extremely good performance in computational cost.

V. CONCLUSION

We review the importance and difficulty of low computational cost real-time object pose tracking using only RGB images taken by mobile devices. To this end, we present a real-time inertial-assisted-visual object pose tracking solution with a very low computational overhead. To ensure the robustness, we propose a BSCM to alleviate the error that accumulates over time. We also devise a PIA to quickly examine the reliability of the object pose. Moreover, an object pose dataset with RGB images and IMU measurements is delivered to evaluate the tracking performance. A pretty low processing time of 3.11 ms in updating pose is measured even in demanding condition of limited computational power, which demonstrates the feasibility of our method. We believe the proposed method and dataset will facilitate the future research.

Future work includes extending this method in pose inspection and refinement. For pose inspection, there may exist other parameters (e.g., the frame rate or the distance between object and camera) to further discuss. For pose refinement, modeling the noise may helpful to eliminate the error in tracking. It could also be an alternative to analyze the system biases on the backend to save more computational power of the frontend.

REFERENCES

[1] J. P. S. do Monte Lima, F. P. M. Simoes, L. S. Figueiredo, and J. Kelner, "Model based markerless 3D tracking applied to augmented reality," *J. Interactive Syst.*, vol. 1, no. 1, 2010.

[2] Y. Su, J. Rambach, N. Minaskan, P. Lesur, A. Pagani, and D. Stricker, "Deep multi-state object pose estimation for augmented reality assembly," in *Proc. IEEE Int. Symp. Mixed Augmented Reality Adjunct*, 2019, pp. 222–227.

[3] C. Choi and H. I. Christensen, "Real-time 3D model-based tracking using edge and keypoint features for robotic manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 4048–4055.

[4] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," in *Proc. 2nd Conf. Robot Learn. Mach. Learn. Research*, vol. 87, Oct. 2018, pp. 306–316.

[5] H.-N. Hu et al., "Joint monocular 3D vehicle detection and tracking," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 5390–5399.

[6] L. Zhong et al., "Seeing through the occluders: Robust monocular 6-DOF object pose tracking via model-guided video object segmentation," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5159–5166, Oct. 2020.

[7] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox, "PoseRBPF: A Rao–Blackwellized particle filter for 6-D object pose tracking," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1328–1342, Oct. 2021.

[8] R. Ge and G. Loianno, "VIPose: Real-time visual-inertial 6D object pose tracking," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 4597–4603.

[9] T. Sandy and J. Buchli, "Object-based visual-inertial tracking for additive fabrication," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1370–1377, Jul. 2018.

[10] K. Eckenhoff, P. Geneva, N. Merrill, and G. Huang, "Schmidt-EKF-based visual-inertial moving object tracking," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 651–657.

[11] T. Hou, A. Ahmadyan, L. Zhang, J. Wei, and M. Grundmann, "MobilePose: Real-time pose estimation for unseen objects with weak shape supervision," 2020, *arXiv:2003.03522*.

[12] M. Ryffel et al., "AR museum: A mobile augmented reality application for interactive painting recoloring," in *Proc. Int. Conf. Interfaces Human Comput. Interact.* 2017, pp. 54–60.

[13] H. Choi, "The conjugation method of augmented reality in museum exhibition," *Int. J. Smart Home*, vol. 8, no. 1, pp. 217–228, 2014.

[14] K. Kitamura, "Case study of digital exhibition of Japanese classical writings and drawings based on AR technology," in *Proc. Int. Conf. Culture Comput.*, 2017, pp. 125–126.

[15] Z. Gong, R. Wang, and G. Xia, "Augmented reality (AR) as a tool for engaging museum experience: A case study on Chinese art pieces," *Digital*, vol. 2, no. 1, pp. 33–45, 2022.

[16] K. Pauwels, L. Rubio, J. Diaz, and E. Ros, "Real-time model-based rigid object pose estimation and tracking combining dense and sparse visual cues," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2013, pp. 2347–2354.

[17] S.-C. Huang et al., "Efficient recognition and 6D pose tracking of markerless objects with RGB-D and motion sensors on mobile devices," in *Proc. VISIGRAPP*, 2019, pp. 375–382.

[18] Y. Liu, J. Zhou, Y. Zhang, C. Ding, and J. Wang, "3DPVNet: Patch-level 3D Hough voting network for 6D pose estimation," 2020, *arXiv:2009.06887*.

[19] Y. He, H. Huang, H. Fan, Q. Chen, and J. Sun, "FFB6D: A full flow bidirectional fusion network for 6D pose estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 3003–3013.

[20] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. IEEE 7th Int. Conf. Comput. Vis.*, 1999, pp. 1150–1157.

[21] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce, "3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints," *Int. J. Comput. Vis.*, vol. 66, no. 3, pp. 231–259, 2006.

[22] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose tracking from natural features on mobile phones," in *Proc. IEEE 7th ACM Int. Symp. Mixed Augmented Reality*, 2008, pp. 125–134.

[23] S. Hinterstoisser et al., "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *Proc. Int. Conf. Comput. Vis.*, 2011, pp. 858–865.

[24] S. Hinterstoisser et al., "Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes," in *Proc. Asian Conf. Comput. Vis.*, 2012, pp. 548–562.

[25] K. Ramnath, S. N. Sinha, R. Szeliski, and E. Hsiao, "Car make and model recognition using 3D curve alignment," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2014, pp. 285–292.

[26] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1521–1529.

[27] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," in *Proc. Robot., Sci. Syst.*, 2018.

[28] M. Rad and V. Lepetit, "BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 3828–3836.

[29] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "PVNet: Pixel-wise voting network for 6DoF pose estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4561–4570.

[30] X. Yu, Z. Zhuang, P. Koniusz, and H. Li, "6DoF object pose estimation via differentiable proxy voting regularizer," in *Proc. 31st Brit. Mach. Vis. Conf.*, 2020.

[31] S. Iwase, X. Liu, R. Khirodkar, R. Yokota, and K. M. Kitani, "RePOSE: Fast 6D object pose refinement via deep texture rendering," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 3303–3312.

[32] Y. Xu, K.-Y. Lin, G. Zhang, X. Wang, and H. Li, "RNNPose: Recurrent 6-DoF object pose refinement with robust correspondence field estimation and pose optimization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 14880–14890.

[33] X. Weng, J. Wang, D. Held, and K. Kitani, "3D multi-object tracking: A baseline and new evaluation metrics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 10359–10366.

[34] B. Wen et al., "BundleSDF: Neural 6-DoF tracking and 3D reconstruction of unknown objects," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 606–617.

[35] U. Neumann and S. You, "Natural feature tracking for augmented reality," *IEEE Trans. Multimedia*, vol. 1, no. 1, pp. 53–64, Mar. 1999.

[36] M. Servières, V. Renaudin, A. Dupuis, and N. Antigny, "Visual and visual-inertial SLAM: State of the art, classification, and experimental benchmarking," *J. Sensors*, vol. 2021, pp. 1–26, 2021.

[37] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2007, pp. 3565–3572.

[38] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct EKF-based approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 298–304.

[39] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018.

[40] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "VINet: Visual-inertial odometry as a sequence-to-sequence learning problem," in *Proc. AAAI Conf. Artif. Intell.*, vol. 31, 2017, pp. 3995–4001.

[41] C. Chen et al., "Selective sensor fusion for neural visual-inertial odometry," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10542–10551.

[42] C. Arth, A. Mulloni, and D. Schmalstieg, "Exploiting sensors on mobile phones to improve wide-area localization," in *Proc. 21st Int. Conf. Pattern Recognit.*, 2012, pp. 2152–2156.

[43] S. Middelberg, T. Sattler, O. Untzelmann, and L. Kobbelt, "Scalable 6-DoF localization on mobile devices," in *Proc. Comput. Vis., 13th Eur. Conf.*, 2014, pp. 268–283.

[44] C. Arth, G. Reitmayr, and D. Schmalstieg, "Full 6DoF pose estimation from geo-located images," in *Proc. Comput. Vis., 11th Asian Conf. Comput. Vis.*, 2012, pp. 705–717.

[45] P. Lang, A. Kusej, A. Pinz, and G. Brasseur, "Inertial tracking for mobile augmented reality," in *Proc. the 19th IEEE Instrum. Meas. Technol. Conf.*, 2002, pp. 1583–1587.

[46] O. J. Woodman, "An introduction to inertial navigation," Comput. Lab., Univ. Cambridge, Tech. Rep. UCAM-CL-TR-696, 2007.

[47] C. Aguero et al., "Inside the virtual robotics challenge: Simulating real-time robotic disaster response," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 494–506, Apr. 2015.

[48] N. Alliance, "5G White paper," *Next Gener. Mobile Netw., White Paper*, vol. 1, 2015.

[49] "Geekbench browser," Sep. 13, 2021. [Online]. Available: https://browser.geekbench.com/

[50] M. Garon, D. Laurendeau, and J.-F. Lalonde, "A framework for evaluating 6-DOF object trackers," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 582–597.

Yo-Chung Lau received the B.S. and M.S. degrees in computer science and information engineering from National Cheng Kung University, Tainan, Taiwan, in 2005 and 2007, respectively. He is currently working toward the Ph.D. degree in computer science with National Taiwan University, Taipei, Taiwan.

From 2008 to 2011, he developed the products of embedded systems in Novatek Microelectronics Corp., Hsinchu, Taiwan. From 2011, he has been a Researcher in Chunghwa Telecom. He is currently with Chunghwa Telecom Laboratories, Taoyuan, Taiwan. His research interest includes the computer vision, pattern recognition, object localization, image processing, augmented/mixed/virtual reality, and metaverse.

Kuan-Wei Tseng received the B.S. degree in engineering from National Taiwan University, Taipei, Taiwan, in 2020. He is currently working toward the M.S. degree in artificial intelligence with the Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan.

Prior to his graduate study, he was a Full-Time Research Associate with the AI Applications and Integration Lab (AI^2Lab) for one year and as a Research Assistant with the Image and Vision Lab (imLab) for 1.5 years. He is currently in imLab, National Taiwan University, as a Graduate Research Assistant. His research interests include visual computing, focusing on 3-D computer vision, deep learning for computer vision, and their applications to augmented and virtual reality.

Peng-Yuan Kao received the B.S. degree in computer science and information engineering from National Taiwan University of Science and Technology, Taipei, Taiwan, in 2014, and the Ph.D. degree in computer science from the Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, in 2023.

His research interests include computer vision, deep learning, and multimedia.

I-Ju Hsieh received the B.S. and M.S. degrees in computer science from the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, in 2021 and 2023, respectively.

In 2021, he was a Product Developer Intern with Synology and a Software Development Engineer Intern with Amazon Ring in 2022. His research interests include object pose estimation, visual localization, and unsupervised domain adaptation.

Hsiao-Ching Tseng received the B.S. degree in computer science from the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, in 2021, and the M.S. degree in computer science from the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, in 2023.

Her research focuses on computer vision, focusing on object tracking and unmanned aerial vehicle localization.

Yi-Ping Hung received the B.Sc. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1982, and the M.Sc. degree in engineering, M.Sc. degree in applied mathematics, and the Ph.D. degree in engineering from Brown University, Providence, RI, USA, in 1987, 1988, and 1990, respectively.

He served as the Director of the Graduate Institute of Networking and Multimedia from 2007 to 2013. He is currently a Professor with the Graduate Institute of Networking and Multimedia and the Department of Computer Science and Information Engineering at the National Taiwan University, as well as the Academic Vice President of the Taipei National University of Arts. His current research interests include VR/AR/XR, metaverse, image processing, multimedia, and human–computer interaction.