2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022)
October 23-27, 2022, Kyoto, Japan

Towards Robust Visual-Inertial Odometry with Multiple Non-Overlapping Monocular Cameras

Yao He[1,2], Huai Yu[2,3,*], Wen Yang[2] and Sebastian Scherer[3]

***Abstract*— We present a Visual-Inertial Odometry (VIO) algorithm with multiple non-overlapping monocular cameras aiming at improving the robustness of the VIO algorithm. An initialization scheme and tightly-coupled bundle adjustment for multiple non-overlapping monocular cameras are proposed. With more stable features captured by multiple cameras, VIO can maintain stable state estimation, especially when one of the cameras tracked unstable or limited features. We also address the high CPU usage rate brought by multiple cameras by proposing a GPU-accelerated frontend. Finally, we use our pedestrian carried system to evaluate the robustness of the VIO algorithm in several challenging environments. The results show that the multi-camera setup yields significantly higher estimation robustness than a monocular system while not increasing the CPU usage rate (reducing the CPU resource usage rate and computational latency by 40.4% and 50.6% on each camera). A demo video can be found at https://youtu.be/r7QvPth1m10.**

I. INTRODUCTION

Visual-inertial state estimation serves as the most fundamental role for a wide range of applications, such as robotic navigation, autonomous driving, and augmented reality (AR) [1]. Current monocular and stereo VIO algorithms have achieved high precision and stability for state estimation on most public datasets [1]–[3]. However, improving the robustness of these VIO algorithms remains a significant problem. The limited scenarios and motion patterns in public datasets make such stability unreliable [4]. Current VIO algorithms are prone to failure in specific environments, such as those with homogeneous texture, inconsistent lighting, and aggressive motions.

To account for this problem, numerous works [5]–[9] have been proposed to use multiple cameras to increase the robustness. With more information captured by multiple non-overlapping cameras, stable features can be selected among these cameras for state estimation. If the tracked features from one of the cameras are unstable or limited, the VIO algorithm can still maintain reliable feature tracking with other cameras.

However, introducing additional cameras will increase the CPU computation usage. The VIO frontend is usually used to extract and track landmark features among video frames. Because of the substantial visual data stream, the frontend usually takes extensive CPU resources, yielding the computation latency and incompatibility with other tasks on a computation-limited chip. Jeon et al. [10] tests the CPU usage of different VIO algorithms on various NVIDIA hardware (Jetson TX2, Xavier NX, and AGX Xavier). His work shows that most VIO algorithms have extensive CPU usages around 150%∼250%. This kind of computation usage rate prevents other applications, such as object detection and motion planning, from running on the same computer. Moreover, the frontend computation occupation will increase dramatically by adding more cameras.

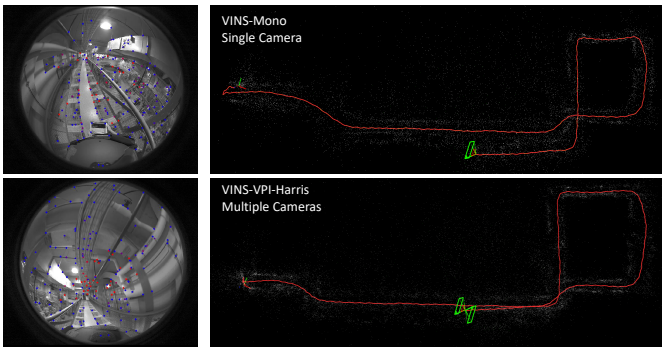

Fig. 1. *Left (top-bottom):* two non-overlapping images of multi-camera system and the tracked features (red). *Right-top:* the trajectory of original VINS-Mono [1] on a single camera. *Right-bottom:* the trajectory of our proposed efficient multi-camera VIO system (loop closure is disabled for both methods, dataset is collected using system with two non-overlapping fisheye cameras.)

In our work, we propose a new VIO algorithm using multiple cameras to improve robustness while reducing the additional CPU burden brought by multiple cameras. Specifically, our algorithm uses multiple non-overlapping monocular cameras for individual feature tracking. Based on VINS-Mono [1], the feature extraction, backend initialization, and state estimation are altered to satisfy a multi-camera hardware configuration. To reduce the CPU burden, we utilize the parallel computing feature of GPU to process multiple video streams. We test our algorithm on both the public and our collected datasets in several challenging situations.

The major contributions are listed as follows:

- We propose a VIO algorithm with multiple non-overlapping monocular cameras. The stable features observed in multiple cameras are fed into the backend for optimization, which provides a higher initialization success rate and more robust state estimation.
- To improve the efficiency of multi-camera VIO, we design and set up the frontend feature extraction and

[1]Yao He is with the School of Science and Engineering, The Chinese University of Hongkong, Shenzhen and Shenzhen Institute of Artificial Intelligence and Robotics for Sociaty, Shenzhen, China. yaohe@link.cuhk.edu.cn
[2]Huai Yu, Wen Yang is with the Electronic Information School, Wuhan University, Wuhan 430072, China. {yuhuai,yangwen}@whu.edu.cn
[3]Huai Yu and Sebastian Scherer is with the AirLab, Carnegie Mellon University, Pittsburgh, PA 15213, USA. basti@andrew.cmu.edu
*Corresponding Author.

tracking on GPU with the NVIDIA Vision Programming Interface (VPI), which allows the VIO to process multi-camera data with a relatively low CPU resource.

The rest of this paper is organized as follows. Section II provides the related literature. Section III describes the VPI accelerated frontend and the multi-camera VIO. Section IV provides the experimental results and discussions. Finally, this paper is concluded in Section V.

II. RELATED WORK

A. Multiple Camera Visual Odometry

Although the SOTA monocular and stereo VIO algorithms have impressive performance, they are still not robust enough for autonomous state estimation on real robots with complex motions and complicated environments over a long period. Multiple cameras can significantly add visual constraints with wider FoV and thus provide redundancy for backend optimization. Taragay *et al.* proposed the loosely-coupled VIO on two pairs of backward and forward stereo cameras with an IMU [5]. Tim *et al.* introduced a real-time 6D stereo visual odometry with two non-overlapping FoV cameras [6]. Liu *et al.* presented a pose tracker and a local mapper supporting an arbitrary number of stereo cameras [11]. Jaekel *et al.* presented a multi-stereo VIO with a fixed-lag smoother, which jointly optimizes poses and landmarks across the stereo pairs. [8]. Zhang *et al.* proposed a feature selection and tracking scheme between overlapping cameras for multi-camera VIO [9]. These methods demonstrate the robustness of multi-camera visual odometry over a long traveling distance and complex environments. However, most of these work focuses on multiple stereo/stereo+monocular cameras with certain FoV overlap. Our goal is to explore the possibility of using multiple non-overlapping monocular cameras. It does not require any specific sensor layout and can be generalized to cameras with overlaps. Another problem is that most of the multi-camera processing is based on the multi-core CPU parallel computing, adding great computation burden to CPU and inevitably bringing computation latency to the state estimation.

B. VIO Algorithms and GPU Acceleration

Numerous VIO algorithms have been proposed in recent years, such as VINS-Mono [1], MSCKF [12], ROVIO [2] and ORB-SLAM3 [13]. Most of them consist of a frontend for feature association and a backend for pose/map optimization. Although these VIO methods have achieved good results for online state estimation, The CPU usages are high because massive visual images need to be processed. According to the experiments conducted on Xavier [10], the CPU usage of VINS-Mono is about 150-170% on multi-core processing, MSCKF is above 170%. ROVIO has the least CPU usages (around 60%), but the backend EKF is less efficient compared to the bundle adjustment method [14]. The computational latency on the backend will be larger in ROVIO. ORB-SLAM has the largest CPU usage ranging about 150-240%. Additionally, the limitation to the CPU usage will increase the computational latency. Therefore, the trade-off between CPU usage and efficiency for VIO algorithms has always been a difficult problem.

With the hardware revolution in the last decade, GPUs are installed in embedded platforms, such as NVIDIA Xiaver, which enables more parallel computing capacity. Recently, several works have utilized GPUs to improve the VIO frontend video processing capability and reduce latency. The commonly used feature detectors, such as Shi-Tomasi [15], Harris [16], and FAST [17] are implemented in CUDA Visual Library (VILIB) [18] for fast feature extraction. VILIB applies efficient low-level, GPU hardware-specific programming for feature detection and feature tracking. Recently, NVIDIA proposed the VPI [19], which enables lots of basic computer vision algorithms (such as Harris detector and LK optical flow) on either CPU, GPU, VPA (Programmable Vision Accelerator), and VIC (Video Image Compositor). It supports an easy switch between different processing chips. These GPU implementations have great potentials to shift the computational-expensive VIO frontend to GPU, reducing the CPU usage and computation latency. With the GPU accelerated frontend, our goal is to set the frontends of multiple cameras conducted on GPU so that the backend optimization can have enough CPU resources to ensure non-latency estimation.

III. METHODOLOGY

A. Overview

The structure of the proposed multi-camera visual-inertial state estimator is shown in Fig. 2. We first extract and track visual features from non-overlapping cameras individually on GPU. IMU measurements between two consecutive frames are pre-integrated as in VINS-Mono [1]. The initialization procedure selects one principal camera to initialize and ends with initialization on all the subordinate cameras. It is regarded as a success if at least one camera is initialized successfully. All the reliable features from multiple cameras will be fed into the visual-inertial bundle adjustment. In terms of re-localization and pose graph, all the initialized frames will be stored in the keyframe database for loop detection.

The notations and frame definitions that we use throughout the paper are defined as follows. $(\cdot)^{c_{it}}$, $(\cdot)^{b}$, and $(\cdot)^{w}$ represent the *i*-th camera frame at time t, body (IMU) frame and world frame, respectively. Rotation matrices $\boldsymbol{R}$ and quaternions $\boldsymbol{q}$ both represent rotation. $\otimes$ represents the multiplication operation between two quaternions. We assume that the IMU and cameras are already calibrated in our algorithm. Thus the extrinsic parameters are treated as known quantities.

B. VPI-Enhanced Frontend for Each Camera

The VIO frontend detects and tracks features, and streams the landmarks to the backend for optimization. VPI provides Harris corner detector to detect features and pyramidal LK optical flow to track features. The overall architecture of the frontend is demonstrated in Fig. 3.

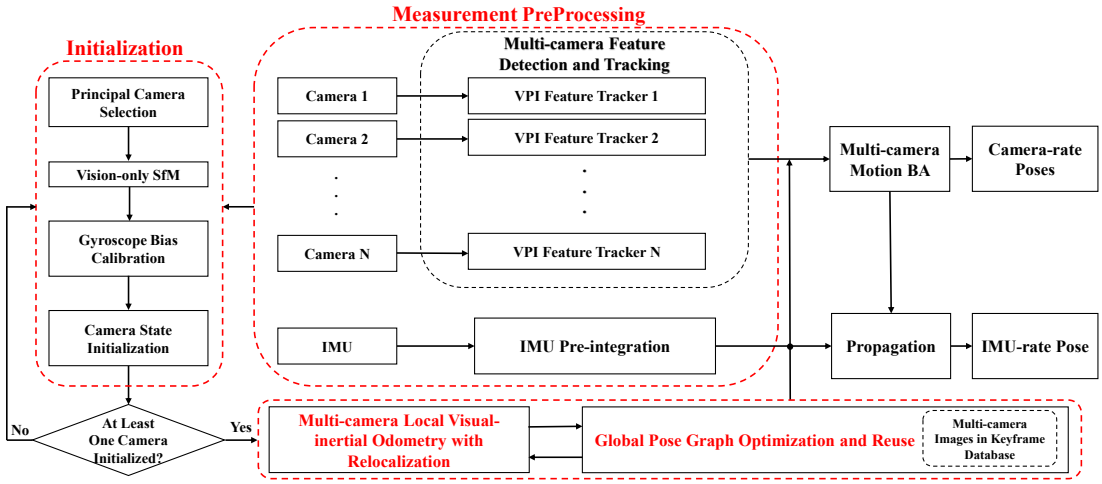

Fig. 2. Block diagram illustrating the pipeline of the proposed multi-camera VIO with VPI enhanced frontend. In the Multi-camera Local Visual-inertial Odometry with Relocalization and Global Pose Graph Optimzation section, the strategies follow VINS-mono except for the multi-camera configuration.

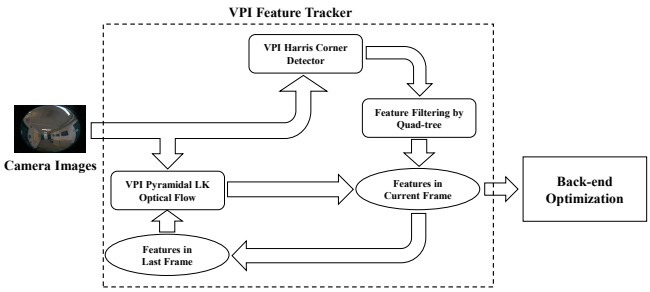

Fig. 3. The overall frontend architecture of VINS-Mono with VPI.

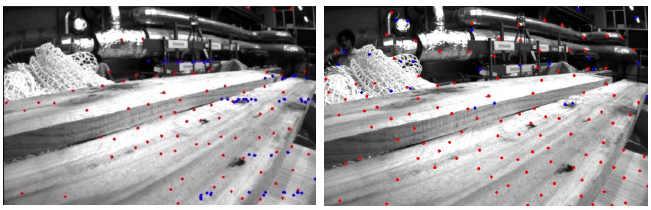

Fig. 4. Distribution of feature points. The scene is from EuRoC dataset [20]. *Picture on the left*: feature points picked by their scores. *Picture on the right*: feature points after the quad-tree filter. *Red points*: tracked features. *Blue points*: newly detected features.

To use VPI Harris corner and LK flow, we first initialize a VPIstream, Harris corner detector VPIpayload and Optical LK flow VPIpayload. Images are converted to VPIimages and VPI image pyramid is constructed for further processing on GPU. Then we submit the VPIimage to the VPI Harris detector to detect features. The feature detection algorithm provided in VPI associated each feature point with a score. Intuitively, the best approach to select feature points is picking points with higher scores. However, the Harris feature distribution will be concentrated in richly textured areas if they are picked directly according to their scores. This reduces the parallax, thus reducing the robustness in initialization and state estimation. Therefore we implement the quad-tree algorithm to filter the feature points, as shown in Algorithm 1 and Fig. 4. The point with the highest response score will be selected regardless of how many points are in a local patch. It ensures a more uniform distribution of feature points and thus improves the robustness. After the feature selection, we submit the detected feature points VPI image pyramid to the VPI Optical LK flow module to track the feature points. The newly detected feature points and the tracked points constitute the feature points in the current image. They will be streamed to the backend for further processing.

Algorithm 1 Selecting Feature points by Quad-tree.

Input: P_{in}: the set of detected feature points; S: the set of scores corresponding to detected feature points; k: the number of needed feature points; M: mask of fisheye.
Output: P_{out}: the set of selected feature points;

Select the feature points inside M into set P'_{in};
if $|P'_{in}| < n$ **then**
  return P'_{in};
end if
Initialize image node set N, first image node n_1;
Store all feature points in P'_{in} into n_1;
Add n_1 into N;
while $|N| < k$ **do**
  for $n_i \in N$ **do**
    if n_i has no feature point **then**
      Remove n_i from N
    end if
    if n_i has at least 2 feature point **then**
      Split n_i to four sub-nodes with the same image size;
    end if
  end for
end while
for n_i in N **do**
  Select the feature point p_i having the highest score;
  Add p_i to P_{out};
end for
return P_{out};

C. Estimator Initialization

Since the cameras have non-overlapping FoV, the algorithm needs an accurate scale initialization at the beginning, similar to the monocular tightly coupled VIO [1]. We get the necessary initial values by loosely aligning IMU pre-integration with the vision-only structure. The estimator initialization for the multi-camera system consists of principal camera selection, vision-only SfM on principal camera, gyroscope calibration, and camera state initialization.

1) Principal Camera Selection: The initialization starts with a principal camera selection. In the sliding window, we check the feature correspondences in each camera between the latest frame and all previous frames. After we find stable feature tracking (more than 30 tracked features) between the latest frame and one typical frame in the sliding window, the camera with the most visual features is selected as the candidate of the principal camera. If the parallax is sufficient (more than 20 pixels), it is regarded as the principal camera. Otherwise, we continue the selection within other cameras. Once the principal camera is identified, the other cameras are characterized as subordinate cameras, and their initialization depends on the initialization of the principal camera.

2) Vision-only SfM on Principal Camera: Once we identify a principal camera and the corresponding two frames, we perform a vision-only SfM [1] on the principal camera. We set the first principal camera frame $(\cdot)^{c_p 0}$ as the reference world frame for SfM. Here we denote c_p as the principal camera and c_i as subordinate cameras. Given the extrinsic parameters $(\boldsymbol{p}_{c_i}^b, \boldsymbol{q}_{c_i}^b)$ between the camera and the IMU, we can recover the frame rotations of subordinate cameras with respect to $(\cdot)^{c_p 0}$ as

$$\boldsymbol{q}_{c_i k}^{c_p 0} = \boldsymbol{q}_{c_p k}^{c_p 0} \otimes (\boldsymbol{q}_{c_p}^b)^{-1} \otimes \boldsymbol{q}_{c_i}^b \tag{1}$$

Note that the scale of principal camera s_p is unknown at this stage, so we cannot directly obtain the translation of subordinate cameras using extrinsic parameters. s_p will be solved in the next.

3) Gyroscope Bias Calibration: While we perform a similar minimization as in VINS-Mono to calibrate the gyroscope bias, rotations from all the cameras rather than one camera are added into the minimization formulation to constrain the bias. Then we follow the same way as in [1] to repropagate all IMU preintegration terms using the new gyroscope bias.

4) Camera States Initialization: After the gyroscope bias is initialized, we move on to initialize velocity $\boldsymbol{v}_{b_k}^{b_k}$ in the body frame, gravity vector $\boldsymbol{g}^{c_p}$ and metric scale s_p by aligning the up-to-scale visual structure with IMU measurements and refining the gravity [1].

After the visual alignment on the principal camera, we move on to initialize the subordinate cameras. We first proceed to recover the translation of subordinate cameras using the extrinsic between cameras and IMU

$$\boldsymbol{p}_{c_i k}^{c_p 0} = \boldsymbol{R}_{c_p k}^{c_p 0} \boldsymbol{R}_{c_p}^b (\boldsymbol{p}_{c_k}^b - \boldsymbol{p}_{c_p}^b) + s_p \overline{\boldsymbol{p}}_{c_p k}^{c_p 0} \tag{2}$$

Now we triangulate all the feature points observed in the subordinate cameras.

Once the principal camera initializes successfully, the initialization is regarded as a success. All the values are rotated to the world frame $(\cdot)^w$ and fed into the tightly coupled multi-camera VIO.

D. Tightly Coupled Multi-Camera VIO

After estimator initialization, a sliding window based tightly coupled multi-camera VIO proceeds. We incorporate the inverse distances of features in multiple cameras into the optimization. The state vector is defined as

$$\begin{aligned} \mathcal{X} &= [\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_n, \boldsymbol{x}_{c_1}^b, \boldsymbol{x}_{c_2}^b, ..., \boldsymbol{x}_{c_t}^b, \boldsymbol{\lambda}] \\ \boldsymbol{\lambda} &= [\lambda_{c_1,0}, ...\lambda_{c_1,m_{c_1}}, ..., \lambda_{c_2,0}, ...\lambda_{c_t,m_{c_t}}] \\ \boldsymbol{x}_k &= [\boldsymbol{p}_{b_k}^w, \boldsymbol{v}_{b_k}^w, \boldsymbol{q}_{b_k}^w, \boldsymbol{b}_a, \boldsymbol{b}_g] \\ \boldsymbol{x}_{c_t}^b &= [\boldsymbol{p}_{c_t}^b, \boldsymbol{q}_{c_t}^b] \end{aligned} \tag{3}$$

where $\boldsymbol{x}_k$ is the IMU state at the time when the *k*-th image is captured. It contains position, velocity, and orientation of the IMU in the world frame, acceleration bias $\boldsymbol{b}_a$ and gyroscope bias $\boldsymbol{b}_g$ in the IMU body frame. n is the total number of keyframes, t is the total number of initialized cameras, and m_i is the total number of features of the *i*-th camera in the sliding window. $\lambda_{i,j}$ is the inverse distance of the *j*-th feature in the *i*-th camera from its first observation.

A visual-inertial bundle adjustment is performed to estimate the states. The formulation is

$$\begin{aligned} \min_{\mathcal{X}} \Bigg\{ &\|\boldsymbol{r}_p - \boldsymbol{H}_p \mathcal{X}\|^2 + \sum_{k \in \mathcal{K}} \|\boldsymbol{r}_{\mathcal{K}}(\hat{\boldsymbol{z}}_{b_k}^{b_{k+1}}, \mathcal{X})\|_{\boldsymbol{p}_{b_{k+1}}^{b_k}}^2 \\ &+ \sum_{i \in \mathcal{C}} \sum_{(l,j) \in \mathcal{B}_i} \alpha_i \rho(\|\boldsymbol{r}_{\mathcal{B}_i}(\hat{\boldsymbol{z}}_l^{c_{ij}}, \mathcal{X})\|_{\boldsymbol{p}_l^{c_{ij}}}^2) \Bigg\} \end{aligned} \tag{4}$$

$\rho(s)$ is the cauchy robust function used to suppress outliers. $\mathcal{K}$ is the set of all IMU measurements. $\mathcal{C}$ is the set of all initialized cameras. $\mathcal{B}_i$ is the set of features that have been observed at least twice in the current sliding window in the *i*-th camera. $\{\boldsymbol{r}_p, \boldsymbol{H}_p\}$ is the prior information from marginalization. $\boldsymbol{r}_{\mathcal{K}}(\hat{\boldsymbol{z}}_{b_k}^{b_{k+1}}, \mathcal{X})$ is the residual for IMU. $\boldsymbol{r}_{\mathcal{B}_i}(\hat{\boldsymbol{z}}_l^{c_{ij}}, \mathcal{X})$ is the residual for visual measurements. The detailed definitions are presented in [1]. For the third term of equation (4), the double summation indicates that the feature information in each camera is fused together into the optimization. α_i is defined as

$$\alpha_i = (\frac{N_i}{N_{max}})^2 \tag{5}$$

where N_i, N_{max} are the number of features in the *i*-th camera and the maximum number of features observed in one typical camera, respectively. When one camera faces challenging scenes with low textures, they will detect less features and many of them are unreliable. We introduce α_i to reduce the influence of unreliable visual features.

E. Re-Localization

The re-localization generally follows the scheme in [1]. The only difference is that the frames from different cameras are all stored in the previous map for loop detection and feature retrieval. A loop match detected in any camera will trigger the re-localization.

IV. Experimental Results

In this section, we present the results of the VPI-enhanced VINS-Mono and the final multi-camera VIO.

A. VINS-Mono with VPI

To demonstrate the effectiveness of the proposed new frontend in terms of CPU usage and computation latency, we test different VIO methods on EuRoC dataset [20]. The used CPU for testing is *AMD Ryzen 5800H with Radeon Graphics 3.20GHz*. For each video sequence, we conduct five trials for both the original VINS-Mono (VINS) and our VPI enhanced VINS with Harris corner (VINS-VPI-Harris). In addition, we deploy another GPU enhanced frontend from VILIB [18] into VINS-Mono for comparison. In VILIB, we test two feature detectors, Harris and FAST [21]. Therefore, we add two versions for comparison: VINS-VILIB-FAST and VINS-VILIB-Harris. We use the modules provided in rpg_trajectory_repository [22] to evaluate and compare the results quantitatively. The main comparison metrics consist of 1) the CPU usage and 2) the time of different VIO frontends, and 3) the average absolute trajectory errors (RMSE) on translation and rotation of 5 trials. Table I shows the CPU usage and time cost of different VIO frontends on the EuRoC dataset.

TABLE I
Frontend CPU usage and feature extraction time for different VIO methods on EuRoC dataset

Method	CPU usage (%)	Time (ms)
VINS	52	15.4
VINS-VPI-Harris	**31**	**7.8**
VINS-VILIB-FAST	49	8.5
VINS-VILIB-Harris	51	8.7

TABLE II
Frontend CPU usage and feature extraction time for different VIO methods with a fisheye camera

Method	CPU usage (%) (single core)	Time (ms) (frontend)
VINS	91	20.6
VINS-VPI-Harris	43	11.7
VINS-VILIB-FAST	82	28.7
VINS-VILIB-Harris	91	16.2

Table I shows VINS-VPI-Harris has the most significant reduction on CPU usage and time cost, achieving a reduction of 40.4% and 50.6% compared with the original VINS, respectively. Modules from VILIB (including VINS-VILIB-FAST and VINS-VILIB-Harris) reduce the frontend time cost, but they still have very high CPU usages. According to Table II, when the fisheye is used, the CPU usage and time cost increase compared with those on EuRoC dataset. This increase mainly comes from the larger image size and more feature points for undistortion. It is noticed that except VINS-VPI-Harris, all the other three methods have CPU usages close to 100%. For VINS-Mono with FAST feature detector of VILIB, the time cost is even larger than that of VINS.

Fig. 5 and Fig. 6 show the trajectory evaluation results on 7 relatively difficult sequences of EuRoC dataset. Compared with the original VINS, VINS-VPI-Harris has the most competitive performance in terms of translation and rotation errors. In some sequences, like V1_02, V1_03, and V2_02, VINS-VPI-Harris obtains better accuracy. On the other hand, VINS with VILIB loses accuracy to some extent, especially for the FAST feature when testing on V1_02 and V2_02 sequences. These results demonstrate that the VPI-enhanced VINS can reduce the CPU usage and time cost and maintain stable state estimation.

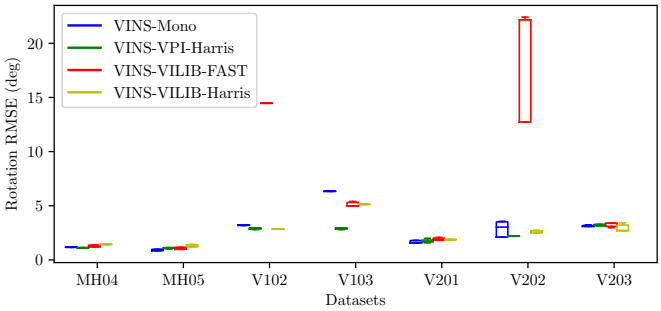

Fig. 5. Rotation RMSE for different VIO methods.

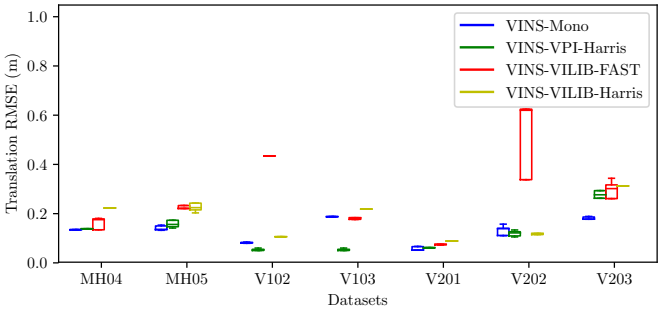

Fig. 6. Translation RMSE for different VIO methods.

B. Multi-Camera VIO

To test the performance of the proposed multi-camera VIO in challenging environments, we collect our data using a pedestrian carried system (as shown in the first row of Fig. 7) with two non-overlapping fisheye cameras. The two Leopard IMX264 cameras capture fisheye images (816×686 pixels images at 24.5Hz) with a synchronized Epson g365 IMU (200Hz). We collect two datasets to evaluate the performance and compare it with monocular VIO results. The first dataset consists of five different trajectories: Outdoors, Indoors, Corridor, NSH wall, and Smith Hall. The outdoor and indoor trajectories are mainly used to test the VIO initialization performance. Since it is hard to obtain the ground truth trajectories for both indoor and outdoor datasets, The other three trajectories are provided with accurate 3D point cloud maps [23] for result evaluation. The second dataset consists of six trajectories. These six trajectories explore the same environment with aggressive motions and low textures: 3 of them are short (short 1, 2, 3), and 3 of them are long (long 1, 2, 3), with numbers indicating the level of difficulties.

Table III compares the time cost and the CPU usage of the entire algorithm. We notice that the CPU usages on both frontend and backend do not increase too much

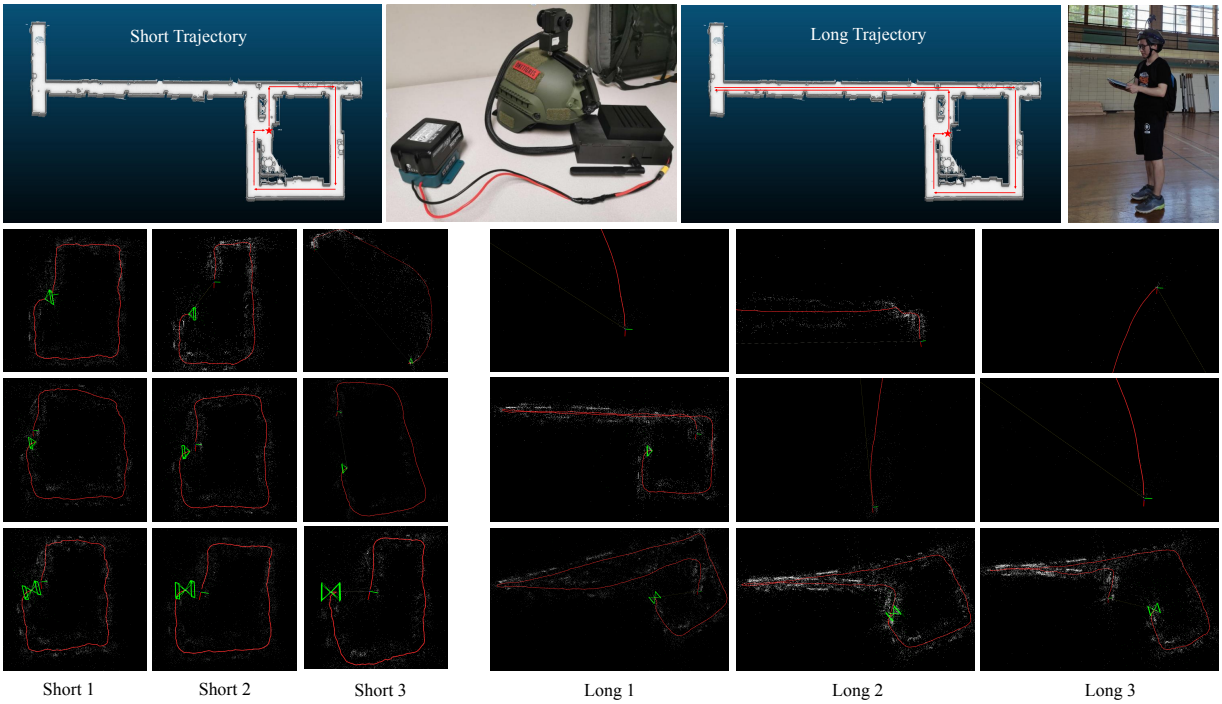

Fig. 7. Our hardware setup and Trajectories of VINS-Mono and multi-camera VIO from the second dataset. First row (left to right): short trajectory, Our VIO setup (From left to right, the devices are: battery, helmet with fisheye cameras, and Xavier. Camera 1 and camera 2 are the cameras facing in the front and back), long trajectory, pedestrian carrying the hardware for experiments. In the trajectories, the star indicates the starts and endings of the trajectories. Second row: results from VINS-Mono using camera 1. Third row: results from VINS-Mono using camera 2. Fourth row: results using multi-camera VIO. For camera 1 in long 1, 2 and 3, and camera 2 in long 2 and 3, the abnormal trajectories indicate a estimator failure.

TABLE III
CPU usage and computation time on optimization of multi-camera VIO and VINS

VIO	VINS	multi-cam VIO
Time (backend)	40 ms	40 ms
CPU usage (frontend)	91 %	43%(each cam)
CPU usage (backend)	101%	105%

TABLE IV
Analysis of the point clouds reconstructed using different VIO methods

Scene	multi-cam VIO		VINS cam-1		VINS cam-2	
	RMSE	*num*	*RMSE*	*num*	*RMSE*	*num*
Corridor	0.1059	14743	0.1143	5410	0.1049	10860
Smith Hall	0.1107	6751	0.1123	5843	0.1124	4642
NSH wall	0.1130	4523	0.1167	3006	0.1164	1343

compared with VINS. Although multiple cameras introduce more features in the optimization, the computation time and CPU usage on optimization are roughly the same as that of VINS. The CPU usage on backend does not increase too much mainly because the optimization function is not sensitive to the number of feature points. Therefore, it further shows the potential of using the VPI-enhanced frontend on the multi-camera VIO system.

To qualitatively analyze the state estimation, the point clouds obtained by VIO are compared with the corresponding accurate 3D maps. We use the mean registration error ($RMSE$) and the number of registered points (num) as the metrics to measure the VIO performance. From Table IV, we can observe that the point clouds collected by multi-camera VIO have more registered points compared with other methods. With richer point clouds, the RMSE of multi-camera VIO is still competitive to the original VINS-mono.

TABLE V
Initialization success rate and robustness of different VIO methods

Scene	multi-cam VIO	VINS cam-1	VINS cam-2
Outdoor	**96.67%**	83.33%	66.67%
Corridor	**98.57%**	25.71%	81.43%
Indoor	**98.00%**	46.00%	62.00%
NSH wall	**100.00%**	**100.00%**	**100.00%**
Smith Hall	**100.00%**	91.40%	**100.00%**

To analyze the robustness, we have two matrics: 1) initialization success rate on the first dataset, and 2) algorithms stability on the second dataset. In terms of the initialization success rate, We uniformly select several time points on each to initialize the VIO. If the VIO can initialize within a limited time (usually 4 seconds), it is regarded as a success. For each sequence, the experiment is repeated 10 times for each VIO. The results are shown in Table V. Overall, the multi-camera VIO has a remarkably higher initialization success rate compared with monocular VIO on one camera for all datasets. Especially for the Corridor dataset, the multi-camera setup improves the success rate significantly. Fig. 7 shows the evaluated trajectories on the second dataset. For all the short and long trajectories, the monocular VINS becomes less robust as the difficulty level increases. For most of the long trajectories, the trajectories of monocular VINS drift away because of unreliable feature constraints. On the other hand, the multi-camera VIO is resilient to challenge motion patterns and low textures. It gives good state estimation because the frontend feature selection from multiple non-overlapping cameras gives reliable landmarks on whatever the level of aggressiveness is. These results show that the

multi-camera VIO is much more robust than the monocular VINS.

Notably, the state estimation accuracy of multi-cam VIO does not increase significantly compared to VINS-Mono. Several observations during our experiments can explain this phenomenon. First, We notice that the feature points detected by VPI do not distribute as uniform as those using OpenCV deployed in VINS. The tracking quality is also less stable. These two drawbacks contribute to relatively poor visual landmarks, and thus the state estimation accuracy is not significantly improved. Secondly, reliable features on a single camera are enough to constrain the backend optimization for some easy cases. Thus, adding more features from multiple cameras will not improve the accuracy. The multi-camera VIO shows robustness and better state estimation on challenging scenarios with aggressive motions and low textures (as shown in Fig. 7). We conclude that the benefit of a multi-camera setup is redundancy, e.g., if one of the cameras suffers from unreliable or limited features, the feature selection from the other cameras can still provide enough landmarks for state estimation.

V. CONCLUSION

In this paper, we propose a robust VIO algorithm with multiple non-overlapping monocular cameras. To reduce the CPU burden introduced by multiple cameras, we design the frontend feature extraction and tracking on GPU with the VPI tool. The reliable features from multiple cameras are fed into the backend for pose optimization. The feature tracking on multiple non-overlapping cameras provides redundancy for backend pose estimation. Thus, robust state estimation can be obtained in challenging scenarios with aggressive motions and low textures. Experiments on public and our own collected challenging datasets demonstrate that the multi-camera VIO system obtains higher robustness on system initialization and state estimation than the monocular VINS without causing an extra burden on the CPU. In the future, reliable feature selection from multiple cameras for backend optimization will be studied to further improve estimation accuracy.

VI. ACKNOWLEDGEMENT

The work was partially supported by the Fundamental Research Funds for the Central Universities under Grant 2042022kf1010 and the RISS 2021 Program, Carnegie Mellon University. We thank Yaoyu Hu for revising this paper and Ruohai Ge for collecting the datasets.

REFERENCES

[1] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, 2018.

[2] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filer based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research (IJRR)*, vol. 33, no. 10, pp. 1255–1262, 2017.

[3] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[4] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer, "Tartanair: A dataset to push the limits of visual slam," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4909–4916.

[5] T. Oskiper, Z. Zhu, S. Samarasekera, and R. Kumar, "Visual odometry system using multiple stereo cameras and inertial measurement unit," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007, pp. 1–8.

[6] T. Kazik, L. Kneip, J. Nikolic, M. Pollefeys, and R. Siegwart, "Real-time 6d stereo visual odometry with non-overlapping fields of view," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 1529–1536.

[7] L. Zhang, D. Wisth, M. Camurri, and M. Fallon, "Balancing the budget: Feature selection and tracking for multi-camera visual-inertial odometry," in *IEEE Robotics and Automation Letters*, vol. 7, no. 2, 2022, pp. 1182–1189.

[8] J. Joshua, G. M. Joshua, S. Sebastian, and K. Michael, "A robust multi-stereo visual-inertial odometry pipeline," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4623–4630.

[9] Z. Xiang, J. Yu, J. Li, and J. Su, "Vilivo: Virtual lidar-visual odometry for an autonomous vehicle with a multi-camera system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2486–2492.

[10] J. Jeon, S. Jung, E. Lee, D. Choi, and H. Myung, "Run your visual-inertial odometry on nvidia jetson: Benchmark tests on a micro aerial vehicle," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5332–5339, 2021.

[11] P. Liu, M. Geppert, L. Heng, T. Sattler, A. Geiger, and M. Pollefeys, "Towards robust visual odometry with a multi-camera system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1154–1161.

[12] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 3565–3572.

[13] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM," *IEEE Transactions on Robotics*, 2021.

[14] H. Strasdat, J. M. Montiel, and A. Davison, "Visual SLAM: Why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[15] J. Shi *et al.*, "Good features to track," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1994, pp. 593–600.

[16] C. Harris, M. Stephens, *et al.*, "A combined corner and edge detector," in *Alvey Vision Conference*, vol. 15, no. 50, 1988, pp. 10–5244.

[17] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision (ECCV)*, 2006, pp. 430–443.

[18] B. Nagy, P. Foehn, and D. Scaramuzza, "Faster than FAST: GPU-accelerated frontend for high-speed VIO," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[19] NVIDIA. (2021) Vpi-vision programming interface documentation.

[20] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research (IJRR)*, vol. 35, no. 10, pp. 1157–1163, 2016.

[21] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, 2008.

[22] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[23] H. Yu, W. Zhen, W. Yang, J. Zhang, and S. Scherer, "Monocular camera localization in prior lidar maps with 2d-3d line correspondences," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 4588–4594.

[24] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.