TCB-VIO: Tightly-Coupled Focal-Plane Binary-Enhanced Visual Inertial Odometry

Anonymous Authors

***Abstract*—Vision algorithms can be executed directly on the image sensor when implemented on the next-generation sensors known as focal-plane sensor-processor arrays (FPSP)s, where every pixel has a processor. FPSPs greatly improve latency, reducing the problems associated with the bottleneck of data transfer from a vision sensor to a processor. FPSPs accelerate vision-based algorithms such as visual-inertial odometry (VIO). However, VIO frameworks suffer from spatial drift due to the vision-based pose estimation, whilst temporal drift arises from the inertial measurements. FPSPs circumvent the spatial drift by operating at a high frame rate to match the high-frequency output of the inertial measurements. In this paper, we present TCB-VIO, a tightly-coupled 6 degrees-of-freedom VIO by a Multi-State Constraint Kalman Filter (MSCKF), operating at a high frame-rate of 250 FPS and from IMU measurements obtained at 400 Hz. TCB-VIO outperforms state-of-the-art methods: ROVIO, VINS-Mono, and ORB-SLAM3.**

I. Introduction

Agile and mobile robotic systems often operate under strict power and processing constraints. As a result, there is an increasing demand for low-latency and low-power camera technologies [1]. State-of-the-art algorithms utilizing conventional cameras typically achieve frame rates of 40–80 frames per second (FPS) [2]–[4]. These cameras rely on an architectural design where data is captured, digitized, and transferred to separate digital processing hardware for further computation. This traditional architecture, characterized by the modular separation of sensors and processors, can introduce significant latency and increase power consumption. On-sensor vision processing presents a novel paradigm that addresses these challenges by co-locating sensors and processors on the same chip. Focal-plane sensor processor arrays (FPSPs) [5] exemplify this approach, with each camera pixel equipped with a small processor capable of processing and sharing data with adjacent pixels (See Fig. 1). This tight integration of sensing and processing reduces power consumption and minimizes delays caused by communication overhead, making FPSPs highly suitable for real-world robotic applications [6].

Due to the limited space on FPSPs, per-pixel memory is highly constrained, and processing is typically performed in analog. Analog computation has inherent limitations, including reduced numerical precision, circuit inaccuracies, thermal effects, and noise leakage [7]. These constraints challenge implementing computer vision algorithms and can lead to noisy results. Integrating additional sensing modalities, such as inertial measurement units (IMU)s, using visual-inertial odometry (VIO) algorithms [8] can reduce the impact of the noise. However, adapting VIO algorithms to FPSPs requires a redesign of the processing pipeline to preserve the high-speed and low-power advantages of FPSPs.

Early VIO algorithms were filtering-based, using the Extended Kalman Filter (EKF) [8], [9], [2], Unscented Kalman Filter (UKF) [10], and the popular MSCKF [11], where landmark positions are marginalized from the state vector to reduce the computational complexity. Later, smoothing approaches emerged, optimizing states over a window [12], [13] with tools like GTSAM [14], leading to variants such as VINS-Mono [3], SVO 2.0 [15], Kimera VIO [16], and ORB-SLAM3 [4]. Despite improved accuracy, they can suffer from inconsistencies and linearization errors [17], [18], [19]. VIO methods can also be categorized as loosely coupled or tightly coupled [8]. Loosely coupled methods, such as MSF [20], process the two modalities independently and fuse them later. In contrast, tightly coupled algorithms, such as OKVIS [12], jointly optimize visual and inertial data to achieve higher accuracy and robustness. For conventional cameras, however, the additional computation required by tightly coupled VIO often reduces update rates and increases the latency between image capture and state update [21]. FPSPs can mitigate this trade-off: their in-pixel parallel processing enables fast on-sensor computations, while their ultra-high frame rate shortens the delay between image captures. For FPSPs, several visual odometry algorithms have been proposed [22], [23]. To date, the only VIO algorithm is BIT-VIO [24], which is a loosely coupled method; no tightly coupled VIO approaches exist. This gap is addressed in this paper.

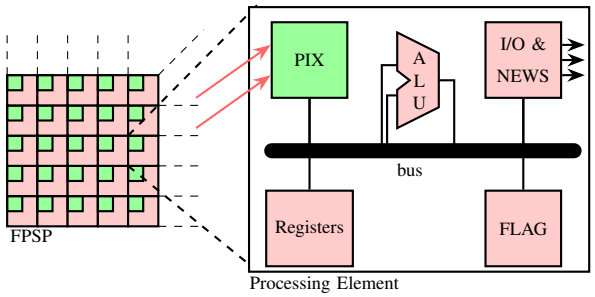

Figure 1: In a focal-plane sensor-processor array (FPSP), each pixel, referred to as a processing element, combines a photosensor circuit (PIX) with on-pixel compute resources, including an arithmetic-logic unit (ALU), input/output (I/O) circuits, local communication links (NEWS), local memory (Registers), and activity control (FLAG). This architecture enables image processing to be performed directly on the sensor.

This paper presents the first tightly-coupled 6 degrees-of-freedom (DoF) VIO algorithm, designed for FPSPs. The algorithm coined, TCB-VIO, is a **T**ightly-**C**oupled VIO, utilizing **BI**T-ehanced features [22]. TCB-VIO processes high-speed, on-sensor binary edge images and feature maps using a novel binary-enhanced Kanade-Lucas-Tomasi (KLT) tracker. It is an extended and modified adaptation of the tightly coupled framework, OpenVINS [25]. The contributions of this paper are: 1) First high frame-rate tightly-coupled VIO for FPSPs achinving a high frame rate of 250 FPS. 2) The framework processes binary edge data and feature coordinates directly on-sensor, reducing computational cost. These outputs are then processed by the novel binary-enhanced KLT tracker at high frame rates. 3) Real-world evaluations comparing against ROVIO [2], VINS-Mono [3], and ORB-SLAM3 [4].

In the rest of the paper, Sec. II describes the background of FPSP. Sec. III explains the TCB-VIO algorithm. Sec. IV details our experimental results. Sec. V concludes the work.

II. Background: SCAMP-5 FPSP

A traditional camera consists of a 2D array of light-sensitive pixels. In contrast, FPSPs integrate a processor within each pixel on the same chip [5] (See Fig. 1). FPSPs are also known as processor-per-pixel arrays (PPA) or cellular-processor arrays (CPA). One example of FPSP is the SCAMP-5 system [26], which features a 256×256-pixel FPSP. Each pixel, or Processing Element (PE), includes a photosensor circuit (PIX) and compute resources. These PEs operate in parallel, executing identical instructions on local data in a Single Instruction, Multiple Data (SIMD) architecture. Each PE can also communicate with its north, east, west, and south neighbors, shown as NEWS in Fig 1. Each PE contains 7 analog registers, 13 1-bit registers, and an ALU composed of 176 transistors, which means they can perform logical and arithmetic operations on the sensor. Arithmetic operations are performed directly in the analog domain using analog registers. Avoiding digitization improves the computational speed but introduces computational noise [26]. The computational results can be outputted in various formats, e.g. events, binary frames, or analog frames.

The SCAMP-5 FPSP has been used in some robotics applications. Examples are target tracking [27], agile drones [28], fast inference [29], [30], obstacle avoidance [31], and localization [32]. It is worth noting that the emergence of event cameras has led to a new generation of VIO algorithms operating at high effective frame rates [33]–[36]. Event cameras are promising for VIO in high-speed and low-light scenarios. However, unlike FPSPs, they lack the flexibility of user-programmable processing directly on the focal plane, which enables customized computation.

III. Proposed Method

Fig. 2 shows an overview of the TCB-VIO pipeline. Edges and corner features are computed on the FPSP, then transferred to a host for further processing with a novel binary-enhanced Kanade-Lucas Tomasi (KLT) tracker. The results are fused with IMU measurements. The division of the workload between the FPSP and the host is dictated by the computational characteristics of each stage. Feature extraction is highly parallelizable across pixels and is well-suited to the SIMD architecture of the FPSP. In contrast, the data fusion and the optimization required for the tracker involve iterative and global computations that are not well-suited to the current FPSP architecture, and are therefore performed on the host.

A. Visual Processing

This section details visual processing steps, shown as green-filled block in Fig. 2.

1) Image Formation and Feature Extraction: The front-end visual processing of corner and binary features occurs directly on the SCAMP-5 FPSP. We use a modified version of FAST-16 implemented on the FPSP [37] for the front-end corner feature-part, where the processing of all pixels is done in parallel on FPSP [38], allowing for high frame-rate outputting. The same is done for the binary edge feature detection on the FPSP [22], where the FPSP analog registers were used,

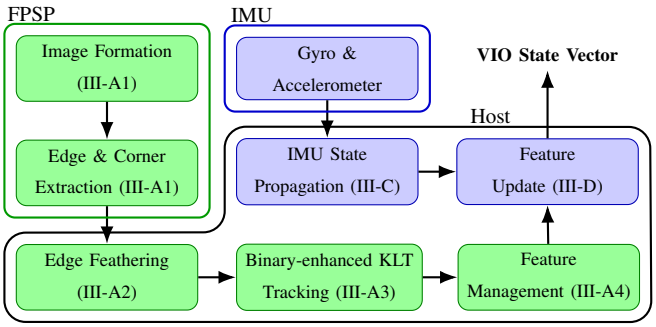

Figure 2: TCB-VIO processing pipeline. Numbers inside each block correspond to the paper sections describing them. Green blocks denote visual processing: the FPSP performs on-sensor corner and edge extraction once an image is formed, and the extracted features are transferred to the host where a novel binary-enhanced KLT tracker ensures fast and robust tracking. Blue blocks denote inertial propagation and updates via MSCKF.

computed in parallel, like with the corner features. The rest of the processing steps are described as follows.

The set of N_p detected corner (see Table I for the size) and edge features acquired on the FPSP is stored in two binary feature maps: the corner map $\mathbf{M}_c$ and the edge map $\mathbf{M}_e$. These maps are transmitted to the host device at a high frame rate of 250 FPS and are formally defined as $\mathbf{M}_c, \mathbf{M}_e \in \{0,1\}^{256\times256}$, where $\mathbf{M}_c(i,j) = 1$ indicates that a corner feature is present at pixel location (i,j), and $\mathbf{M}_e(i,j) = 1$ denotes the presence of an edge feature at that pixel.

2) Edge Feathering: Once a binary edge image is received on the host, it is converted to an 8-bit grayscale image, where all edges take the value of 128. To enhance edge continuity and introduce smooth intensity transitions for improved feature tracking, a Gaussian-based feathering operation is applied to the grayscale image, generating $\mathbf{M}_{\text{feathered}}$. The goal is to produce softly graded edge intensities that improve robustness in gradient-based feature processing. We empirically determine a suitable value of variance of the Gaussian kernel σ_e (see Table I for the value) that produces desirable feathering, preserving high intensity at the center of edges while introducing sufficient gradient smoothness for tracking stability.

3) Binary-enhanced KLT Tracking: To enable efficient, high frame-rate visual tracking on binary data from the FPSP, we adapt the traditional KLT tracking algorithm [39], [40], [41] to operate on the feathered grayscale image. Unlike conventional KLT pipelines, which rely on dense intensity derivatives from natural images, our approach utilizes minimal yet structured gradients derived from binary edge maps to maintain visual tracking robustness at significantly reduced computational cost.

The binary-enhanced KLT utilizes both corners and feathered edges. Each corner pixel (x_i, y_i) for which $\mathbf{M}_c(x_i, y_i) = 1$ is selected as a tracking point, forming the set $\mathbf{C} = \{(x_i, y_i)\}$.

While the corner locations come from the binary corner map $\mathbf{M}_c$, the actual intensity information used for tracking is drawn from $\mathbf{M}_{\text{feathered}}$, which encodes smooth gradients generated from the edge map $\mathbf{M}_e$. For each corner, a spatial window $\mathbf{W}_i$ (see Table I for the size) is centered on (x_i, y_i), and optical flow is computed over these windows. The feathering process introduces local intensity uniqueness between adjacent edges, ensuring that similar binary edge structures become distinguishable for gradient-based motion estimation.

For each feature, the displacement $\Delta\mathbf{u}$ is computed by

minimizing the photometric error between consecutive frames:

$$\sum_{(x,y)\in\mathbf{W}_i} \left(I_t(x,y) - I_{t+1}(x+\Delta u_x, y+\Delta u_y)\right)^2, \quad (1)$$

where I_t and I_{t+1} are the feathered edge images at times t and $t+1$, respectively. The optimization is solved iteratively, leading to $\Delta\mathbf{u}^k = \mathbf{H}^{-1}\mathbf{g}$, where [39]:

$$\mathbf{H} = \sum_{(x,y)\in\mathbf{W}_i} \nabla I(x,y)\nabla I(x,y)^\top, \quad (2)$$

$$\mathbf{g} = \sum_{(x,y)\in\mathbf{W}_i} \nabla I(x,y)\left(I_t(x,y) - I_{t+1}(x,y)\right). \quad (3)$$

The updated feature locations are given by $\mathbf{C}_{t+1} = \mathbf{C}_t + \Delta\mathbf{u}^k$. Tracking continues until convergence or the maximum iteration count: $\|\Delta\mathbf{u}^k\| < \epsilon, \forall(x,y) \in \mathbf{C}$.

The binary-enhanced KLT tracker outputs a sequence of high-frequency 2D features with observations $\mathbf{z}_i^k \in \mathbb{R}^2$, where feature i is tracked across a sliding window of camera frames indexed by $k \in \{t_i, \ldots, t_{i-N_c}\}$. These features serve as visual measurements, $\mathbf{z}_{\text{meas}}(\cdot)$, in the MSCKF backend to update the state (Sec. III-D). Beforehand, a decision is made whether to include them in the state vector, as discussed next.

4) Feature Management: The MSCKF uses features to constrain camera poses during state estimation [11]. These features are not explicitly maintained in the state vector and are therefore referred to as out-of-state (or MSCKF) features. Following Li and Mourikis [42], to improve robustness, a subset of features can be explicitly included in the filter state. These are called in-state (or SLAM) features, as they provide long-horizon geometric constraints. In TCB-VIO, the in-state features are stored in $\mathbf{x}_{BIT}$, described in Sec. III-B, and updated as described in Sec. III-D.

Despite the high frame rate of the FPSP, careful management of SLAM (in-state) features is required, since considering all detected features as in-state would significantly increase both the state dimension and the computational cost of updates. To balance accuracy and efficiency, only a small, reliable subset of features is promoted to the state vector. This classification is based on spatiotemporal observability and track length: features with sufficiently long and stable tracks are designated as in-state, whereas shorter-lived tracks remain out-of-state [25]. Features tracked for fewer than N_c frames, defined in Sec. III-B, remain out-of-state; those tracked for at least N_c are promoted to in-state, with $N_c = 15$ in our experiment (Table I).

B. Visual-Inertial State Vector

Sec. III-A described how features are extracted and processed. This section describes the inertial/visual state vectors, followed by inertial/visual fusion in the remaining sections. The notation used here is adopted from OpenVINS [25]. The state vector $\mathbf{x}_i$ consists of all current estimates of the filter up to time t_i. It has four parts: the current navigation state $\mathbf{x}_{NAV}$, a set of $N_c \in \mathbb{N}$ IMU clones of a sliding window $\mathbf{x}_{IMU}$, a set of $N_r \in \mathbb{N}$ reprojected from 2D-to-3D mapped BIT-features $\mathbf{x}_{BIT}$ (explained in more detail in Sec. III-A4) and, a set of camera calibration intrinsics and extrinsics parameters $\mathbf{x}_C$:

$$\mathbf{x}_i = \begin{bmatrix}\mathbf{x}_{NAV}^\top & \mathbf{x}_{IMU}^\top & \mathbf{x}_{BIT}^\top & \mathbf{x}_C^\top\end{bmatrix}^\top. \quad (4)$$

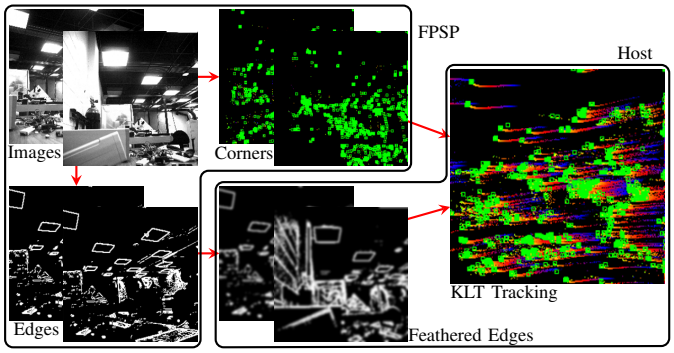

Figure 3: Overview of the proposed Binary-enhanced KLT Tracking for TCB-VIO. The SCAMP-5 FPSP generates binary corners and edges at 250 FPS. On the host, binary edges are feathered, and the KLT tracker operates in spatial windows centered on each corner feature to achieve robust displacement estimation.

The current IMU navigation state x_{NAV} is defined as

$$\mathbf{x}_{NAV} = \begin{bmatrix}{}^{I_i}_{G}\mathbf{q}^\top & {}^{G}\mathbf{p}_{I_i}^\top & {}^{G}\mathbf{v}_{I_i}^\top & \mathbf{b}_{\omega,i}^\top & \mathbf{b}_{a,i}^\top\end{bmatrix}^\top, \quad (5)$$

where ${}^{I_i}_{G}\mathbf{q}^\top, {}^{G}\mathbf{p}_{I_i}^\top, \mathbf{v}_{I_i}^\top$ represents the quaternion orientation, position, velocity from the IMU-frame (I_i) at time t_i to the global world-frame G, respectively. The $\mathbf{b}_{\omega,i}^\top, \mathbf{b}_{a,i}^\top$ are the gyroscopic and accelerometer biases.

The IMU clones $\mathbf{x}_{IMU}$ is defined as

$$\mathbf{x}_{IMU} = \begin{bmatrix}\mathbf{x}_{IMU_{t_i}}^\top & \cdots & \mathbf{x}_{IMU_{t_i-N_c}}^\top\end{bmatrix}^\top, \quad (6)$$

where $\mathbf{x}_{IMU_{t_i}}^\top = \begin{bmatrix}{}^{I_i}_{G}\mathbf{q}^\top & {}^{G}\mathbf{p}_{I_i}^\top\end{bmatrix}$. Note that the indexing is from t_i to $t_i - N_c$, forming a sliding window of states, from current to a past window of N_c prior.

The extracted BIT features are reprojected in the global frame as 3D key features,

$$\mathbf{x}_{BIT} = \begin{bmatrix}{}^{G}\mathbf{p}_{BIT,1}^\top & \cdots & {}^{G}\mathbf{p}_{BIT,N_r}^\top\end{bmatrix}^\top, \quad (7)$$

where ${}^{G}\mathbf{p}_{BIT,i}^\top = (x_i, y_i, z_i)$ is in global frame G. The frame-to-frame tracking is processed at 250 FPS using a novel binary-enhanced Kanade-Lucas-Tomasi (KLT) tracker, in which the binary edge images and feature maps are computed on-sensor.

Finally, between the camera-frame C with respect to the IMU-frame I_i at time t_i, the camera intrinsics and extrinsics,

$$\mathbf{x}_C = \begin{bmatrix}{}^{I_i}_{C}\mathbf{q_i}^\top & {}^{C}\mathbf{p_i}_{I_i}^\top & {}^{G}\mathbf{\Phi_i}_{0}^\top\end{bmatrix}^\top \quad (8)$$

are continually optimized, where ${}^{I_i}_{C}\mathbf{q_i}^\top, {}^{C}\mathbf{p_i}_{I_i}^\top$ represents the quaternion orientation and position from the IMU-to-camera frame. ${}^{G}\mathbf{\Phi_i}_{0}^\top$ is the camera intrinsic vector that updates in time, in the global frame G.

C. Propagation of State by IMU

The IMU is taken as a 6-axis input, where measurements in the local IMU frame are

$${}^{I}\boldsymbol{\omega}_{local}(t) = {}^{I}\boldsymbol{\omega}_T(t) + \mathbf{b}_G(t) + \boldsymbol{n}_G(t), \quad (9)$$

$${}^{I}\boldsymbol{a}_{local}(t) = {}^{I}\boldsymbol{a}_T(t) + {}^{I}_{G}\mathbf{R}(t){}^{G}\boldsymbol{g} + \mathbf{b}_A(t) + \boldsymbol{n}_A(t), \quad (10)$$

where ${}^{I}\boldsymbol{\omega}_T(t), {}^{I}\boldsymbol{a}_T(t)$ are the true and un-noised gyroscope and accelerometer measurements. $\boldsymbol{b}_G(t), \boldsymbol{b}_A(t)$ represent the gyroscopic and accelerometer biases, respectively. ${}^{I}_{G}\mathbf{R}(t)$ is the rotation from global frame G to IMU-frame, and is used

in transforming the approximate estimated gravity vector $\boldsymbol{g}$ to the local IMU frame. $\boldsymbol{n}_G(t), \boldsymbol{n}_A(t)$ are white Gaussian noises.

The IMU state evolves over time with notations adopted from [43] and kinematic propagation model from [25], [11]:

$$ {}^{I}_{G}\dot{\mathbf{q}}(t) = \frac{1}{2}\begin{bmatrix} -\left[{}^{I}\boldsymbol{\omega}_T(t)\times\right] & {}^{I}\boldsymbol{\omega}_T(t) \\ -{}^{I}\boldsymbol{\omega}_T(t)^\top & 0 \end{bmatrix}\begin{bmatrix} {}^{I}\boldsymbol{\omega}_T(t) \\ \boldsymbol{I}^{G}_{t}\boldsymbol{q} \end{bmatrix}, \tag{11} $$

$$ {}^{G}\dot{\mathbf{p}}_I(t) = {}^{G}\mathbf{v}_I(t), \qquad {}^{G}\dot{\mathbf{v}}_I(t) = \boldsymbol{I}^{G}_{t}\mathbf{R}^\top {}^{I}\boldsymbol{a}_T(t), \tag{12} $$

$$ \dot{\mathbf{b}}_G(t) = \boldsymbol{n}_G(t), \qquad \dot{\mathbf{b}}_A(t) = \boldsymbol{n}_A(t). \tag{13} $$

The true un-noised gyroscope and accelerometer IMU measurements $\omega_T(t), \boldsymbol{a}_T(t)$ can be obtained by rearranging Eq. (9)-(10), respectively with raw input IMU measurements $\boldsymbol{\omega}_{local}(t), \boldsymbol{a}_{local}(t)$ in the local-frame.

The solutions of the kinematic equations start with [25]

$$ {}^{I_{i+1}}_{G}\mathbf{R} = \exp\left(-\int_{t_i}^{t_{i+1}} {}^{I}\boldsymbol{\omega}_T(Z)\,dZ\right){}^{I_i}_{G}\mathbf{R} \tag{14} $$

Here the rotation from global frame G to IMU-frame from last estimate ${}^{I_i}_{G}\mathbf{R}$ is updated incrementally. Then the position is acquired from the last position, velocity ${}^{G}\mathbf{p}_{I_i}, {}^{G}\mathbf{v}$, respectively. involving a double-integral on the last acceleration $\mathbf{a}_T(Z)$:

$$ \begin{aligned} {}^{G}\mathbf{p}_{I_{i+1}} =& {}^{G}\mathbf{p}_{I_i} + {}^{G}\mathbf{v}_{I_i}(t_{i+1}-t_i) - \frac{1}{2}{}^{G}\mathbf{g}(t_{i+1}-t_i)^2 \\ &+ {}^{I_i}_{G}\mathbf{R}^\top \int_{t_i}^{t_{i+1}}\int_{t_i}^{s} {}^{I_i}_{I_Z}\mathbf{R}^\top {}^{I}\mathbf{a}_T(Z)\,dZ\,ds. \end{aligned} \tag{15} $$

The velocity is then propagated incrementally at t_{i+1} integrating the last acceleration $\mathbf{a}_T(Z)$:

$$ {}^{G}\mathbf{v}_{I_{i+1}} = {}^{G}\mathbf{v}_{I_i} + {}^{I_i}_{G}\mathbf{R}^\top \int_{t_i}^{t_{i+1}} {}^{I_i}_{I_Z}\mathbf{R}^\top {}^{I}\mathbf{a}_T(Z)dZ - {}^{G}\mathbf{g}(t_{i+1}-t_i). \tag{16} $$

The biases are also propagated integrating the random walk noises $\boldsymbol{n}_G, \boldsymbol{n}_A$. They are zero-mean Gaussians:

$$ \mathbf{b}_{G_{i+1}} = \mathbf{b}_{G_i} + \int_{t_i}^{t_{i+1}} \boldsymbol{n}_G(Z)dZ, \;\; \mathbf{b}_{A_{i+1}} = \mathbf{b}_{A_i} + \int_{t_i}^{t_{i+1}} \boldsymbol{n}_A(Z)dZ. \tag{17} $$

The MSCKF backend of TCB-VIO linearizes the nonlinear IMU kinematic model and incrementally succeeds the state along with propagating the covariance, as explained in [11].

D. Update of 3D Point Features

At each measurement time-step t_m, a tracked corner feature, described in Sec. III-A3, is reprojected as a 3D bearing landmark, based on the following measurement model $\mathbf{h}(\cdot)$:

$$ \begin{aligned} \mathbf{z}(t_m) &= \mathbf{h}(\mathbf{x}(t_m)) + \mathbf{n}(t_m) \\ &= \mathbf{T}_d(\mathbf{T}_p(\mathbf{T}_t(\mathbf{T}_r(\boldsymbol{\lambda}, \boldsymbol{K}), \mathbf{x}_{CG})), \zeta) + \mathbf{n}(t_m), \end{aligned} \tag{18} $$

where $n(t_m)$ is zero-mean Gaussian noise, and $\mathbf{h}(\cdot)$ is a chain of functions following the standard projection pipeline [25]:

- $\mathbf{T}_r(\cdot)$ converts the landmark parameterization λ (inverse-depth or 3D position, with any context such as an anchor frame/extrinsics denoted by $\mathbf{K}$) into global coordinates.
- $\mathbf{T}_t(\cdot, \mathbf{x}_{CG})$ transforms the global point to the current camera frame using the camera pose in the global frame G, $\mathbf{x}_{CG}$.
- $\mathbf{T}_p(\cdot)$ orthogonally projects onto the normalized image plane.
- $\mathbf{T}_d(\cdot, \zeta)$ maps normalized coordinates to distorted pixel coordinates using intrinsic distortion parameters ζ.

The predicted feature measurement is $\mathbf{z}_{\text{pred}}(t_m) = \mathbf{h}(\mathbf{x}(t_m))$. For the update, the residual is $r(t_m) = \mathbf{z}_{\text{meas}}(t_m) - \mathbf{h}(\mathbf{x}(t_m))$. For MSCKF features, residuals accumulated over the clone window are projected into the left null-space of the landmark Jacobian. This operation removes the landmarks analytically and allows for updating only the cloned poses.

IV. Experimental Results

This section details the experiments along with discussions.

A. Experimental Setup

TCB-VIO operates with the SCAMP-5 FPSP, processing at 250. IMU measurements, at 400 Hz, and grayscale images for benchmarking are sourced from an Intel D435i RealSense. These two sensors are rigidly attached to a fixture. (See [44]) Benchmarks methods are ROVIO [2], VINS-Mono [3], and ORB-SLAM3 [4] using grayscale and IMU data. TCB-VIO cannot be evaluated on standard benchmark datasets due to on-sensor computation. For indoor experiments, the ground truth is provided by a Vicon motion capture system running at 300 Hz. To synchronize and calibrate the FPSP and IMU, Kalibr [45] was used. For metrics, the absolute trajectory error (ATE) and relative trajectory error (RTE) are stated as a median and root mean squared error (RMSE) [46]. 13 trajectories were used for benchmarking (10 indoors and 3 outdoors). The trajectories are created by rapidly shaking the fixture in all directions to simulate violent motions. Trajectories #1-3 are mostly rotational. Trajectories #4-7 are mostly translational. Trajectories #8-10 are also mostly rotational, but more challenging than trajectories #1-3, based on their average angular velocities (see Table II, top row). Trajectories #11-13 (see Table V) are similar to trajectories #1-3, but occurring outdoors. The length and duration of each trajectory are also provided in the tables.

B. Parameters of TCB-VIO

Table I lists the key parameters used in the experiments. An exhaustive search over a wide range tuned these. Graphs of the parameter search are on the project webpage [44]. While N_r, the size of $\mathbf{x}_{BIT}$ in Eq. (7), may grow over time, the number of landmarks used for updating SLAM and MSCKF is limited, denoted with $N_{\text{SLAM, update}}$ and $N_{\text{MSCKF, update}}$.

Table I: Key parameters for TCB-VIO configuration.

Description (Symbol)	Value
Number of points extracted and tracked (N_p)	800
Max IMU clones in the sliding window, (N_c at Eq. 6)	15
Max SLAM-represented BIT feat in update ($N_{\text{SLAM, update}}$)	30
Max MSCKF-represented BIT feat in update ($N_{\text{MSCKF, update}}$)	60
Gaussian feathering variance for edge smoothing (σ_e)	2.5
Spatial window size in pixels ($\mathbf{W}_i$ at Eq. (1))	21×21

Table II: ATE (m), first block, and RTE (m), second block, for 10 trajectories. Each cell shows RMSE/median. Lowest of all errors are in **bold**; lowest of TCB-VIO, ROVIO, VINS-Mono are underlined. "F" = fail to initialize, "PF" = partial failure. Each trajectory was designed with distinct motion characteristics to vary the difficulty of visual-inertial tracking. Numbers after each trajectory ID indicate the rounded average angular velocity, length, and duration. Trajectories #1–3 involve relatively small but fast motions. Trajectories #4–7 have extended paths spanning the large motion-capture room. Trajectories #8–10 feature highly dynamic motions with aggressive rates.

Traj. / Alg.	**1**: 11 rad/s, 25 m, 19 s	**2**: 11 rad/s, 33 m, 26 s	**3**: 8 rad/s 24 m, 29 s	**4**: 10 rad/s, 20 m, 24 s	**5**: 9 rad/s, 21 m, 26 s	**6**: 9 rad/s, 29 m, 24 s	**7**: 9 rad/s, 31 m, 31 s	**8**: 13 rad/s, 26 m, 16 s	**9**: 14 rad/s, 28 m, 13 s	**10**: 15 rad/s, 25 m, 12 s	
TCB-VIO	0.199/0.202	0.223/0.156	0.158/0.113	**0.218/0.192**	**0.140/0.121**	**0.255/0.234**	**0.315/0.168**	**0.210/0.136**	**0.601/0.469**	**0.379/0.013**	A
Ablation 1	0.354/0.207	10.056/5.666	**0.107/0.073**	F/F	PF/PF	F/F	PF/PF	F/F	PF/PF	F/F	T
Ablation 2	PF/PF	PF/PF	F/F	PF/PF	5.778/3.386	PF/PF	1.375/1.152	PF/PF	F/F	F/F	E
ROVIO	0.342/0.220	0.693/0.433	0.309/0.167	2.548/1.854	9.241/6.126	3.832/3.378	PF/PF	0.648/0.629	6.501/5.415	1.405/1.245	
VINS-Mono	0.113/0.083	0.563/0.416	2.798/1.203	PF/PF	PF/PF	PF/PF	PF/PF	PF/PF	PF/PF	PF/PF	
ORB-SLAM3	**0.091/0.053**	**0.151/0.096**	PF/PF	PF/PF	PF/PF	F/F	F/F	F/F	F/F	F/F	
TCB-VIO	**0.021/0.008**	**0.019/0.010**	**0.010/0.003**	**0.009/0.004**	**0.009/0.005**	**0.016/0.009**	**0.011/0.008**	**0.024/0.013**	**0.015/0.011**	**0.028/0.009**	R
Ablation 1	0.052/0.036	0.085/0.051	0.035/0.024	F/F	PF/PF	F/F	PF/PF	F/F	PF/PF	F/F	T
Ablation 2	PF/PF	PF/PF	F/F	PF/PF	0.013/0.007	PF/PF	0.013/0.010	PF/PF	F/F	F/F	E
ROVIO	0.085/0.062	0.078/0.067	0.054/0.040	0.051/0.042	0.079/0.056	0.076/0.058	PF/PF	0.089/0.083	0.149/0.118	0.112/0.105	
VINS-Mono	0.171/0.149	0.178/0.140	0.202/0.093	PF/PF	PF/PF	PF/PF	PF/PF	PF/PF	PF/PF	PF/PF	
ORB-SLAM3	0.091/0.072	0.098/0.071	PF/PF	F/F	PF/PF	F/F	F/F	F/F	F/F	F/F	

C. Comparisons with ROVIO and VINS-Mono

As shown in Table II, TCB-VIO demonstrates superior robustness against fast and hostile motions, achieving the lowest ATE and RTE across all experiments, compared with ROVIO and VINS-Mono. VINS-Mono suffers from partial failures. ROVIO exhibits reasonable robustness, avoiding large errors throughout all experiments. However, its ATE and RTE values are still up to 4 to 10 times higher than those of TCB-VIO. Error values for partial failures are not reported, as they are at least 10× larger than the lowest error.

As an example, Fig. 5 shows the RMSE of trajectory #10 along all axes. TCB-VIO achieves the lowest total translational RMSE (Fig. 5 top-right)). ROVIO diverge over time, especially in the X and Z directions, with errors exceeding 2.7 meters. VINS-Mono has its own axes for RMSE in red on the right of subplots because of its much larger error ranges, particularly in the X direction, resulting in higher errors compared to TCB-VIO, with errors exceeding 30 meters in the X. These trends align with the quantitative data in Table II, further emphasizing TCB-VIO's robustness during fast and hostile motions. In terms of rotational RMSE (Fig. 5 bottom-right), all methods perform similarly, but TCB-VIO maintains the lowest overall error. This highlights its ability to handle rapid rotational dynamics effectively, unlike ROVIO and VINS-Mono, which experience occasional spikes due to tracking loss.

Fig. 6 shows the tracking error, color-coded, for trajectory #10. The ground truth is shown in gray. TCB-VIO demonstrates significantly lower error mapping compared to ROVIO and VINS-Mono. TCB-VIO consistently maintains errors close to the ground-truth trajectory, with minimal deviations. In contrast, both ROVIO and VINS-MONO exhibit large error clusters, particularly at sharp turns and high dynamic regions, as evident from the red and orange segments on the map. Notably, VINS-MONO diverged to such an extent that it jittered and failed to maintain local consistency.

D. Trajectory Failures Present in ORB-SLAM3

As shown in Table II, while the metrics of TCB-VIO and ORB-SLAM3 for trajectories #1 and #2 are close, ORB-SLAM3 fails in the rest of the trajectories. In the table, an 'F" indicates a failure to initialize, and "PF" is a partial failure.

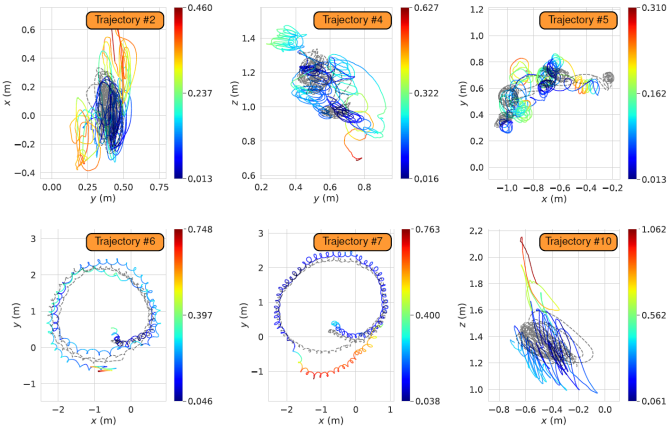

Figure 4: Overview of representative testing trajectories used in our evaluation, aligned with the performance metrics reported in Table II. All trajectories were executed under fast and hostile motions, as reflected by the high angular velocities (10–33 rad/s) reported in Table II. Ground truth is shown in gray, with error mapping done for TCB-VIO.

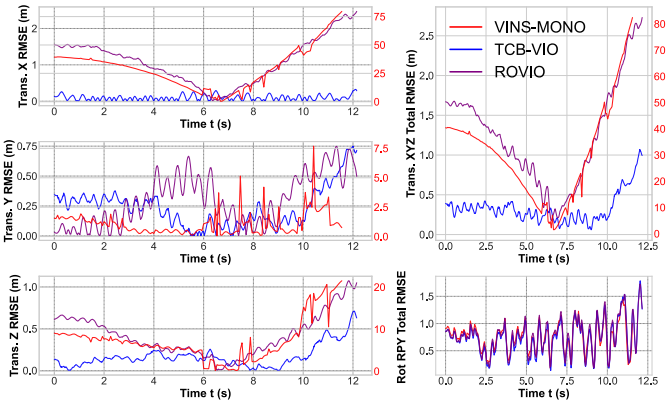

Figure 5: Estimated RMSE for trajectory #10: (left) translation for X, Y, and Z axes, (top right) total translation, and (bottom right) total rotation. VINS-Mono has its own axes, in red, due to large error ranges. While rotational RMSE values for all three methods are close, TCB-VIO maintains the lowest overall translational RMSE.

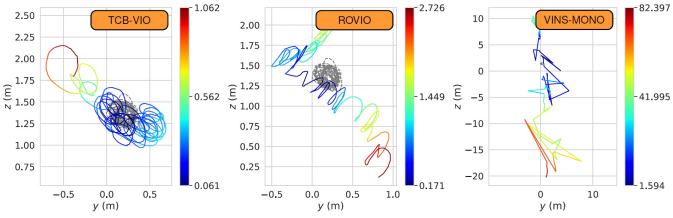

Figure 6: Error mapping for trajectory #10, comparing TCB-VIO (left), ROVIO (middle), and VINS-Mono (right). TCB-VIO maintains low error, closely following the ground-truth path (in gray) with minimal deviations. ROVIO shows significant drift, while VINS-Mono diverges. Note that the colorbar scales differ across the graphs.

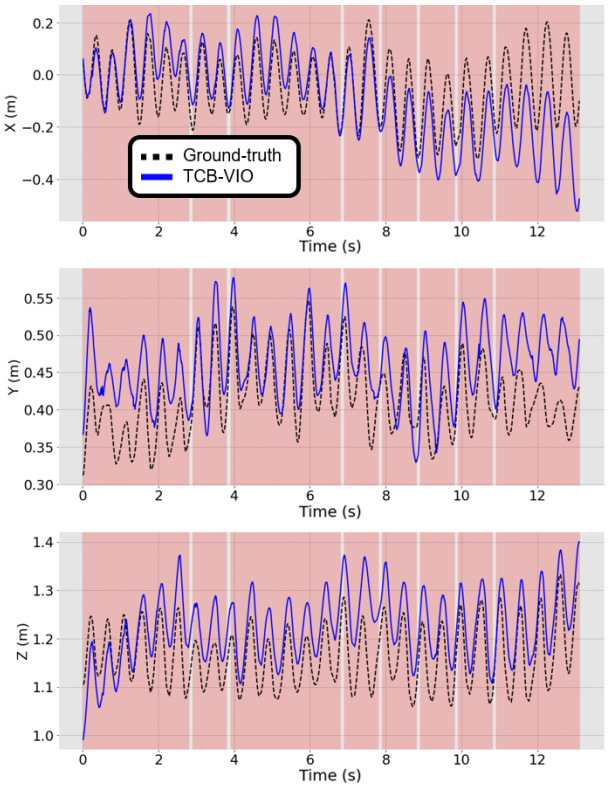

Figure 7: Trajectory #8 (F-type failure) showing complete failure during pink regions, while TCB-VIO tracks consistently despite occasional deviations from ground-truth data.

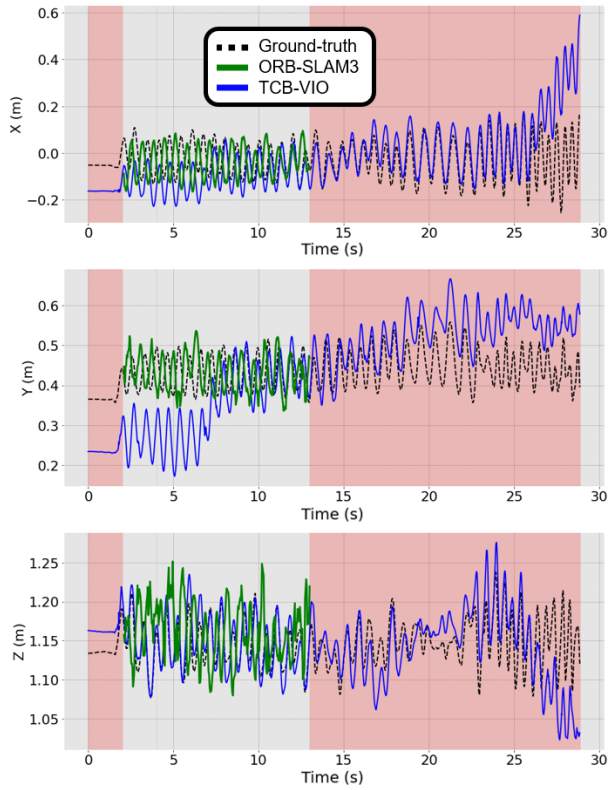

Figure 8: Trajectory #3 (PF-type failure) illustrating ORB-SLAM3 reinitializing but quickly diverging, highlighted in pink,leading to significant ATE and RTE errors.

Fig. 7 corresponds to trajectory #8, where ORB-SLAM3 fails. This type of failure is characterized by ORB-SLAM3's inability to initialize in certain regions, specifically those marked in pink. In the short non-pink regions, ORB-SLAM3 performs well, demonstrating its potential under favorable conditions. However, during pink intervals, ORB-SLAM3 crashes, losing tracking completely. In contrast, TCB-VIO, while deviating slightly from the ground truth in some areas, maintains reasonable tracking accuracy and does not suffer from the catastrophic failures observed in ORB-SLAM3.

Fig. 8 illustrates a PF-type failure, as seen in trajectory #3 from Table II. In this particular trajectory, compared to trajectories #1-2, the scene was designed to contain fewer trackable features, making successful tracking possible primarily in high-exposure conditions where the binarized image remained stable. Under these constraints, ORB-SLAM3 repeatedly attempts to reinitialize within the pink regions but diverges rapidly, leading to large ATE and RTE values. Although the pink regions are less prominent compared to F-type failures, their impact is still significant due to the rapid loss of tracking. The limited feature availability, coupled with faster motions, caused ORB-SLAM3's failure to recover. In contrast, TCB-VIO demonstrates its robustness by maintaining a consistent trajectory, mitigating errors, and maintaining reliable performance throughout the trajectory, despite the challenging conditions.

E. *Discussions*

This section discusses five aspects of TCB-VIO's performance and design, including its 1) time complexity, 2) computational load, 3) the impact of in-state (SLAM) features, 4) outdoor performance, and 5) ablation studies.

1) Time Complexity Analysis: To quantify the computational benefits of running TCB-VIO on the FPSP, the execution times of edge and corner detection on the SCAMP-5 FPSP were compared against CPU and GPU. The CPU used was a For the CPU/GPU measurements, 1000 images were captured by the SCAMP-5 FPSP and processed to perform edge and corner detection. Average per-frame results were reported using system utility tools. For the SCAMP-5 FPSP, execution times were calculated from the number of instructions in the code and the execution time per instruction.

As shown in Table III, for edge detection, the SCAMP-5 FPSP achieves $16.4\times$ speedup over the CPU and $17.0\times$ over the GPU. For feature extraction (1000 keypoints), SCAMP-5 FPSP requires 292 μs, while CPU and GPU require 76.67 μs and 20.23 μs. On SCAMP, the corner detection runtime is constant regardless of keypoint count since all 256×256 processing elements execute neighbour comparisons in parallel. Thus, the cost per frame is fixed. In contrast, CPU runtimes scale with keypoints. SCAMP-5 may appear slower in Table III, but this reflects its fixed cost model rather than inefficiency. SCAMP-5 executes operations at 5-10 MHz, far below conventional processors (CPU $\approx$ 2.7 GHz, GPU $\approx$ 400 MHz). This frequency gap explains SCAMP's longer absolute runtimes despite its parallel architecture. Future FPSP designs with higher clock rates would narrow this gap while preserving parallelism. Note that the intensity image transfer

time, significant for CPU/GPU, was not included. By avoiding intensity image transfer, host processor saturation is prevented.

Table III: Runtime comparison for Sobel edge detection (kernel size $k = 3$) and corner extraction (1000 keypoints), performed SCAMP-5 FPSP, CPU, and GPU.

Process	FPSP (SCAMP-5)	CPU (i7- 12650H)	GPU (RTX 3070)
Edge Detection	9.0 μs	147.818 μs	152.63 μs
Corner Detection	292.0 μs	76.671 μs	20.23 μs

2) Computational Load Analysis: The efficiency of TCB-VIO was evaluated in terms of energy consumption. Edge and corner detection were executed directly on-sensor by the SCAMP-5 FPSP using its dedicated processing elements, resulting in per-frame energy consumption of only tens of millijoules (mJ). In contrast, substantially higher CPU/GPU utilization was required when the same operations were offloaded to conventional processors. Per-frame energy consumption was measured for edge and corner detection on the FPSP, CPU, and GPU. For CPU/GPU evaluation, 1000 images were captured by the SCAMP-5 FPSP while the sensor was moved along a known trajectory. For the FPSP evaluation, the same trajectory was repeated twice, once with edge detection and once with corner detection enabled. Energy measurements were obtained using a USB power meter for the SCAMP-5 FPSP, while Intel RAPL and Nvidia SMI were employed to monitor CPU and GPU energy consumption, respectively.

Table IV reports the per-frame energy consumption for the Sobel edge detection (kernel size $k = 3$) and BIT corner detection, averaged on 1000 256×256 images. For edge detection, SCAMP-5 consumes only $\sim$0.019 mJ per frame, whereas the CPU/GPU requires 994.23 mJ/6.46 mJ per frame. For corner detection, the FPSP consumes 1.15 mJ per frame, compared to 987.22 mJ/1.65 mJ on the CPU/GPU. This computational load analysis confirms that SCAMP-5 reduces energy consumption relative to a CPU/GPU-based pipeline, underscoring the lightweight and embedded nature of on-sensor SIMD computation.

Table IV: Energy consumption comparison for edge detection and feature extraction (1000 keypoints) on FPSP/CPU/GPU. Note: Energy consumption is calculated per frame.

Process	FPSP (SCAMP-5)	CPU (i7- 12650H)	GPU (RTX 3070)
Edge Detection	0.019 mJ	994.23 mJ	6.46 mJ
Corner Detection	1.15 mJ	987.22 mJ	1.65 mJ

3) Impact of in-state Features: TCB-VIO consistently achieves low ATE and RTE, but deviations from ground truth occur under specific conditions. Fig. 9 shows RMSE translation error (deviation) and in-state feature count, for trajectory #2, overlaid. In yellow regions, due to poor texture, feature count is low and RMSE is high, the converse is true in green regions. The negative correlation (corr $= -0.42$) shows that having persistent features helps constrain long-term drift. This trend was seen in other trajectories as well.

4) Outdoor Trajectories: In addition to the 10 indoor experiments, three additional experiments were conducted outdoors. Lacking ground truth in the outdoor environment, the performance was evaluated by starting and ending the trajectories from a single fixture, to ensure the start and end poses were identical. The VIO performance was then measured by the endpoint drift (distance) between the estimated start and end poses. As indicated in Table V, TCB-VIO exhibits the lowest error compared to the other algorithms. Videos of these experiments are on the project website [44].

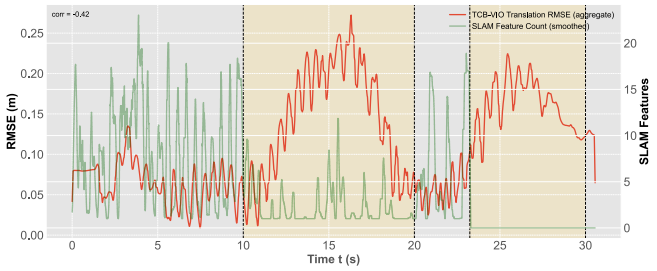

Figure 9: Correlation between in-state feature and translational RMSE for trajectory #2. When feature count is <10 (yellow regions), TCB-VIO has higher translational RMSE.

Table V: Start-to-end error (m) for outdoor experiments, performed in sun/shade contrast. Lowest error is in **bold**. "F" and "PF" are for 'fail to initialize' and 'partial failure'. The last column shows the average angular velocity and trajectory length/duration. The velocity and length have been computed using the TCB-VIO pose estimates.

Traj.	TCB-VIO	ROVIO	VINS-Mono	ORB-SLAM3	Description of Trajectory
11	**1.713**	21.6 (PF)	F	F	7 rad/s, 21 m, 20.3 s
12	**0.130**	11.8 (PF)	695.5 (PF)	F	9 rad/s, 47 m, 49.9 s
13	**2.736**	5.1	F	F	8 rad/s, 31 m, 15.9 s

5) Ablation Studies: To evaluate the design of the algorithm, two ablation studies were conducted, as summarized in Table II. The corresponding ATE and RTE results are reported in the rows labelled Ablation 1 and Ablation 2. In Ablation 1, edge images were transmitted to the host, and Shi–Tomasi features (extracted from the edges) were used for KLT tracking instead of BIT features. The only difference from the TCB-VIO pipeline is the type of features. In Ablation 2, the feathering block (Sec. III-A2) was removed from the TCB-VIO pipeline.

Table II shows that across all trajectories, TCB-VIO achieves lower ATE and RTE than the ablation studies, with the exception of trajectory #3, where its performance is nearly identical to Ablation 1. For instance, in trajectory #1, the median ATE decreases from 0.207m (Ablation 1) to 0.202m (TCB-VIO). More importantly, in challenging sequences such as trajectory #9, Ablation 1 fails to track, while Ablation 2 fails to initialize. These results highlight that both the choice of features and the inclusion of the feathering process are essential for the effectiveness of the TCB-VIO pipeline.

V. Conclusion

In this work, we presented TCB-VIO, a tightly-coupled 6-DoF VIO framework using the on-sensor compute capabilities of SCAMP5 FPSP. By integrating visual and inertial measurements at 250 FPS and 400 Hz, respectively, and processing binary edge images and feature maps with a novel binary-enhanced Kanade-Lucas-Tomasi (KLT) tracker, TCB-VIO achieves robust and accurate tracking under fast motions. The evaluations demonstrated that TCB-VIO outperforms state-of-the-art methods, including ROVIO, VINS-Mono, and ORB-SLAM3 in terms of both translational and rotational accuracy. While ROVIO exhibits drift over time, and VINS-Mono fails to maintain tracking under extreme conditions, TCB-VIO maintains accurate trajectory estimation. This highlights the

importance of high frame-rate synchronization between visual and inertial data, as well as the computational advantages provided by FPSPs. The proposed framework addresses the long-standing trade-off between tracking performance and latency by combining high-frequency visual-inertial data processing with efficient on-sensor computation. This work demonstrates the potential of FPSPs in advancing the capabilities of mobile robotic systems, paving the way for more reliable visual-inertial odometry in applications such as high-speed navigation, drone racing, and autonomous driving.

The proposed method has partial on-sensor computation due to hardware limitations. Future hardware with more flexible compute units could enable all computations to be migrated entirely to the FPSP. In particular, while the feathering process is highly parallelizable, it demands significant bandwidth to transfer the results to the host. An optimal solution would shift this computational load from the host to the FPSP. Another approach is to utilize lower-dimensional and more computationally efficient features for tracking; the corner and edge features in the binary-enhanced KLT algorithm in this work show promise for further investigation.

REFERENCES

[1] D. Falanga, K. Kleber, and D. Scaramuzza, "Dynamic obstacle avoidance for quadrotors with event cameras," *Science Robotics*, vol. 5, no. 40, p. eaaz9712, 2020.

[2] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *IJRR*, vol. 36, no. 10, pp. 1053–1072, 2017.

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

[4] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "ORN-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE TRO*, vol. 37, no. 6, pp. 1874–1890, 2021.

[5] A. Zarandy, *Focal-Plane Sensor-Processor Chips*. Springer NY, 2011.

[6] P. Dudek, T. Richardson, L. Bose, S. Carey, J. Chen, C. Greatwood, Y. Liu, and W. Mayol-Cuevas, "Sensor-level computer vision with pixel processor arrays for agile robots," *Science robotics*, vol. 7, p. eabl7755, 06 2022.

[7] R. S. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," in *ISCA*, 2014, pp. 505–516.

[8] D. Scaramuzza and Z. Zhang, *Visual-Inertial Odometry of Aerial Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–9.

[9] P. Tanskanen, T. Naegeli, M. Pollefeys, and O. Hilliges, "Semi-direct EKF-based monocular visual-inertial odometry," in *IROS*, 2015, pp. 6073–6078.

[10] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive systems*, 2000, pp. 153–158.

[11] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *ICRA*, 2007, pp. 3565–3572.

[12] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, "Keyframe-based visual-inertial SLAM using nonlinear optimization," *RSS*, 2013.

[13] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE TRO*, vol. 33, no. 1, pp. 1–21, 2016.

[14] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," *Georgia Institute of Technology, Tech. Rep*, vol. 2, p. 4, 2012.

[15] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "SVO: Semidirect visual odometry for monocular and multicamera systems," *IEEE TRO*, vol. 33, no. 2, pp. 249–265, 2016.

[16] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *ICRA*, 2020, pp. 1689–1696.

[17] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, "Camera-IMU-based localization: Observability analysis and consistency improvement," *IJRR*, vol. 33, no. 1, pp. 182–201, 2014.

[18] T.-C. Dong-Si and A. I. Mourikis, "Motion tracking with fixed-lag smoothing: Algorithm and consistency analysis," in *ICRA*, 2011, pp. 5655–5662.

[19] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis, "An observability-constrained sliding window filter for SLAM," in *IROS*, 2011, pp. 65–72.

[20] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to mav navigation," in *IROS*, 2013, pp. 3923–3929.

[21] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *ICRA*, 2018, pp. 2502–2509.

[22] R. Murai, S. Saeedi, and P. H. J. Kelly, "High-frame rate homography and visual odometry by tracking binary features from the focal plane," *Autonomous Robots*, Jul 2023.

[23] L. Bose, J. Chen, S. J. Carey, P. Dudek, and W. Mayol-Cuevas, "Visual Odometry for Pixel Processor Arrays," in *ICCV*, 2017, pp. 4614–4622.

[24] M. Lisondra, J. Kim, R. Murai, K. Zareinia, and S. Saeedi, "Visual inertial odometry using focal plane binary features (bit-vio)," in *ICRA*, 2024, pp. 1661–1668.

[25] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "OpenVINS: A research platform for visual-inertial estimation," in *ICRA*, 2020.

[26] S. J. Carey, A. Lopich, D. R. Barr, B. Wang, and P. Dudek, "A 100,000 FPS vision sensor with embedded 535GOPS/W 256× 256 SIMD processor array," in *Symposium on VLSI Circuits*, 2013, pp. 182–183.

[27] C. Greatwood, L. Bose, T. Richardson, W. Mayol-Cuevas, J. Chen, S. J. Carey, and P. Dudek, "Tracking control of a UAV with a parallel visual processor," in *IROS*, 2017, pp. 4248–4254.

[28] A. McConville, L. Bose, R. Clarke, W. Mayol-Cuevas, J. Chen, C. Greatwood, S. Carey, P. Dudek, and T. Richardson, "Visual odometry using pixel processor arrays for unmanned aerial systems in GPS denied environments," *Frontiers in Robotics and AI*, vol. 7, p. 126, 2020.

[29] Y. Liu, J. Chen, L. Bose, P. Dudek, and W. Mayol-Cuevas, "Direct servo control from in-sensor CNN inference with a pixel processor array," *arXiv preprint arXiv:2106.07561*, 2021.

[30] H. M. So, L. Bose, P. Dudek, and G. Wetzstein, "PixelRNN: in-pixel recurrent neural networks for end-to-end-optimized perception with neural sensors," in *CVPR*, 2024, pp. 25 233–25 244.

[31] J. Chen, Y. Liu, S. J. Carey, and P. Dudek, "Proximity estimation using vision features computed on sensor," in *ICRA*, 2020, pp. 2689–2695.

[32] H. Castillo-Elizalde, Y. Liu, L. Bose, and W. Mayol-Cuevas, "Weighted node mapping and localisation on a pixel processor array," in *ICRA*, 2021, pp. 6702–6708.

[33] A. Zihao Zhu, N. Atanasov, and K. Daniilidis, "Event-based visual inertial odometry," in *CVPR*, 2017, pp. 5391–5399.

[34] H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization," 2017.

[35] A. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios," *RAL*, vol. 3, no. 2, pp. 994–1001, 2018.

[36] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza, "Continuous-time visual-inertial odometry for event cameras," *IEEE TRO*, vol. 34, no. 6, pp. 1425–1440, 2018.

[37] J. Chen, S. J. Carey, and P. Dudek, "Feature extraction using a portable vision system," in *IROS-Workshop*, vol. 2, 2017, p. 3.

[38] ——, "Scamp5d vision system and development framework," in *Distributed Smart Cameras*, 2018, pp. 1–2.

[39] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI*, vol. 2, 1981, pp. 674–679.

[40] C. Tomasi and T. Kanade, "Detection and tracking of point," *Int J Comput Vis*, vol. 9, no. 137-154, p. 3, 1991.

[41] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*. IEEE, 1994, pp. 593–600.

[42] M. Li and A. I. Mourikis, "Optimization-based estimator design for vision-aided inertial navigation," in *RSS*, vol. 8, 2013, pp. 241–248.

[43] N. Trawny and S. I. Roumeliotis, "Indirect Kalman filter for 3D attitude estimation," *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep*, vol. 2, p. 2005, 2005.

[44] A. Author, "TCB-VIO," https://sites.google.com/view/tcb-vio, accessed: 2025-09-04.

[45] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *IROS*, Tokyo, Japan, 2013.

[46] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *IROS*, 2012, pp. 573–580.