Accelerated Visual Inertial Navigation via Fragmented Structure Updates

Yehonathan Litman, Ya Wang, Ji Liu

***Abstract*— Tightly coupled Visual-Inertial Navigation System (VINS) implementations have proven their superiority due to their ability to jointly optimize all state variables. However, this joint optimization is considered as a computational bottleneck within the system, and thus many traditional VINS can only be implemented on platforms containing powerful processors. In this work, we show a significant reduction in the computational burden of optimization can be achieved through the use of *fragments*; a set of co-visible feature points from across several different cameras efficiently split, categorized into groups, and then analyzed using a machine-learning inspired *frequent-pattern growth* algorithm. Furthermore, we use a reduced continuous representation during preintegration for better accuracy while requiring less computational resources. We validate our algorithm in datasets, showing that the derivation is not only more accurate, but requires significantly less computational resources. When tested on a Raspberry Pi, our implementation was able to track the system's state in nearly 20 frames per second using only a single CPU core. Testing on another low power module shows a power draw of around 800 mW. Due to run-time considerations the Raspberry Pi was only competitive in regards to other optimization methodologies, but our algorithm displays remarkable accuracy on a more powerful platform.**

I. Introduction

The Simultaneous Localization and Mapping (SLAM) challenge has been approached from multiple directions with multiple methodologies, but for small platforms such as mobile phones and Micro Aerial Vehicles (MAVs), Visual Inertial Navigation Systems (VINS) have been deemed particularly attractive due to their low physical requirements. Using a synchronized monocular camera with an inexpensive IMU, VINS allow for generating very accurate odometry in GPS-denied environments with a minimal yet highly practical setup. In traditional vision-based systems, a monocular (or stereo) camera setup tracked its position and orientation in space through either optimization or filtering. Since these traditional vision sensors are fundamentally very sensitive to rotations and fast movements, the addition of an IMU is highly effective, enabling the system to improve tracking of the position because it allows for recovering the global roll, pitch and the metric scale (for monocular setups). This is done by addressing the gaps where the camera cannot supply enough information for the system to reliably estimate its state.

Various methodologies have been developed for inertial navigation with varying degrees of success. Initial implementations were of loosely coupled approaches [1], [2] where the IMU was considered an independent unit from vision-based SLAM. The IMU was used for propagating the state while pose tracking was done using the camera. Quickly, tightly coupled implementations [3]–[5] proved their superiority, as they use only one estimator to find the optimal state estimate by fusing raw measurements from the camera and IMU directly. Many current visual inertial approaches are based on a simple yet effective Extended Kalman Filter (EKF) to process a measurement only once whenever the state changes. Despite their successes, it has been shown that EKF based VINS suffer from errors linked to an ill estimated state uncertainty. For these systems, their advantages ultimately give rise to theoretical weaknesses, as the EKF based optimization leads to an underestimated state uncertainty as well as an impaired handling of consistency due to their partial observability [6], [7]. Observability constraint aware filters have been expanded upon in additional works [8]–[10] due to their ability to enforce the correct observability of the EKF of the system along specific directions, but at the cost of additional computations and ignoring older states.

To alleviate the weaknesses of filtering-based approaches, formulations of VINS that are based on graph optimization arose. Optimization approaches are very effective given their ability to consider the entire state history of the system. This is done by solving a least-squares formulation or a Bundle Adjustment (BA) problem over a set of state measurements. While optimization approaches are very accurate, the computational complexity grows steadily in the long term, as each batch of measurements must be re-linearized when the system optimizes its state. Incremental optimization strategies were developed to address this, with full smoothing that does not ignore early measurements [11]–[13]. Despite this, optimization-based approaches, even incremental ones, are typically not used on computationally constrained platforms. Instead filtering-based approaches are more common due to their aforementioned computational simplicity as even the traditional incremental approach suffers extensively (as much as non-incremental solvers) in situations where many of the variables have to be re-linearized.

Our first contribution in this work is modifying the incremental properties of optimization approaches resulting in a much simpler process we nickname *fragmentation*. We do not take into account all the variables, but instead group

Yehonathan Litman is with the Department of Computer Science at Stony Brook University (yehonathan.litman@stonybrook.edu).

Ya Wang is with the J. Mike Walker '66 Department of Mechanical Engineering at Texas A&M University (ya.wang@tamu.edu).

Ji Liu is with the Department of Electrical and Computer Engineering at Stony Brook University (ji.liu@stonybrook.edu).

This work was partially supported by the U.S. Department of Energy ARPA-E [DE-AR0000531 and DE-AR0000945].

the cameras that share the most points into *fragments*. By splitting up the problem into smaller bits of information the system enjoys several benefits of sparsification and thus the computational cost goes down. It also allows us to track many more frames using fixed lag smoothing. Coupled with coding our system's visual frontend in NEON intrinsics, the estimator can naturally estimate the state dramatically faster and work with many more features on ARM processors.

Our second contribution in this paper is employing continuous linear dynamics during the IMU preintegration [14], making the process more accurate and less computationally expensive compared to discrete based preintegration. By incorporating our continuous approach into a full BA optimization problem, we succeed in better capturing the IMU states and biases, thus improving the positional accuracy of our system.

We also adopt the approach described in [15] for our marginalization strategy, but only for more computationally capable platforms. In the experimental section we show the system's robustness in both datasets and a real-world test, and display its exceptional performance on both a laptop processor and a Raspberry Pi.

II. Fragmented Bundle Adjustment

A. Bundle Adjustment

To optimally track the system's current state we minimize the errors of both the feature reprojection and IMU measurements within a set of recent camera frames. This optimization can be described as a maximum likelihood estimation problem over a set of structure landmarks denoted as $\mathcal{L}_k$ and relative IMU states denoted as $\mathcal{C}_k = \{\mathbf{R}_k, \mathbf{p}_k, \mathbf{v}_k, \mathbf{b}_k^g, \mathbf{b}_k^a\}$, where $(\mathbf{R}_k, \mathbf{p}_k) \in \text{SE(3)}$ is the pose, $\mathbf{v}_k \in \mathbb{R}^3$ is the velocity, and $\mathbf{b}_k^g \in \mathbb{R}^3, \mathbf{b}_k^a \in \mathbb{R}^3$ are both the gyro and accelerometer biases, respectively. We do full smoothing in our VINS, but present a fixed-lag formulation where the optimization is done in relation to the recent history of keyframes $\mathcal{K}$ up until the current timeframe k. For fixed-lag optimization, a sliding window containing the most recent measurements is used. Thus, we denote the complete state $\theta_k = \{\mathcal{C}_k, \mathcal{L}_k\}$ as a set of all states from the beginning of our sliding window up until the current timeframe k (the most recent measurement of the sliding window). Lastly, we denote the set of raw IMU and feature measurements collected up until the current timeframe as $\mathcal{Z}_k$. The optimal solution θ^* can now be formulated as the following

$$\theta_k^* = \underset{\theta_k}{\text{argmax}}\, P(\theta_k|\mathcal{Z}_k) = \underset{\theta_k}{\text{argmin}} -\log\, P(\theta_k|\mathcal{Z}_k) \tag{1}$$

For a current frame i, each feature measurement $\mathbf{l}_j \in \mathcal{L}_i$ is projected into the camera frame to become a two dimensional measurement in $\mathbb{R}^2$. We then get the reprojection error by comparing with the actual matching image measurement $\mathbf{z}_j$:

$$\mathbf{r}_{\mathbf{l}_j} = \pi(\mathbf{R}_i, \mathbf{p}_i, \mathbf{l}_j) - \mathbf{z}_j \tag{2}$$

$$\mathbf{r}_{\mathbf{l}_j} \sim \mathcal{N}(0, \Sigma_{\mathbf{l}_j}) \tag{3}$$

where $\Sigma_{\mathbf{l}_j}$ is the covariance of the matching image measurement $\mathbf{l}_j$ and $\pi(\cdot)$ is the projection function, which in our work is implemented via an inverse depth representation [16]. We also assume a pinhole-camera model and incorporate a previously determined IMU-camera transformation.

The second residual around the IMU state is represented as $\mathbf{r}_{\mathcal{C}}$ with covariance $\Sigma_{\mathcal{C}}$. Note that $\mathcal{C}_i$ is the relative IMU measurement between two keyframes i and $i+1$. The third residual $\mathbf{r}_{\mathcal{C}_0}$ is a prior on the first state $\mathcal{C}_0$ with covariance $\Sigma_{\mathcal{C}_0}$. The reason we need to implement both the prior and IMU residuals is due to the unobservability of both position and angle around the gravitational axis in a visual inertial system. Note that the prior and IMU error functions will be discussed in detail in section III. Then (1) can be simplified into the following fixed-lag formulation

$$\theta_k^* = \underset{\theta_k}{\text{argmin}} \|\mathbf{r}_{\mathcal{C}_0}\|^2_{\Sigma_{\mathcal{C}_0}} + \sum_{\mathcal{C}_i \in \mathcal{K}_k} \|\mathbf{r}_{\mathcal{C}_i}\|^2_{\Sigma_{\mathcal{C}_i}} + \sum_{\mathcal{L}_i \in \mathcal{K}_k} \sum_{\mathbf{l}_j \in \mathcal{L}_i} \|\mathbf{r}_{\mathbf{l}_j}\|^2_{\Sigma_{\mathbf{l}_j}} \tag{4}$$

where $\|\mathbf{r}\|^2_\Sigma = \mathbf{r}^\top \Sigma^{-1} \mathbf{r}$ is the squared Mahalanobis distance.

B. Incremental Formulation

By utilizing a sparse information matrix $\boldsymbol{\mathcal{I}}$ that encodes the change in state variables within a window of measurements, we do not need to refactorize the entire history of the measurements every time camera and IMU measurements arrive. As such, we modify the information matrix only as measurements arrive, i.e. incrementally.

In this section we focus on efficiently recovering θ's reprojection error $\mathbf{r}_\mathbf{l}$, and covariance $\Sigma_\mathbf{l}$ in an incremental fashion. This can be done using the Gauss-Newton iterative algorithm in order to find a linear least squares solution. We start with an initial estimate θ_0 and intial error $\mathbf{r}_0$. At each i-th iteration, we compute the small correction Δ_{i+1} towards the approximate solution in the neighborhood of θ_i

$$\mathbf{r}_i(\theta_i + \Delta_{i+1}) \approx \mathbf{r}_i\theta_i + \mathbf{J}_i\Delta_{i+1} \tag{5}$$

where $\mathbf{J}_i = \frac{\partial \mathbf{r}}{\partial \theta}|_{\theta=\theta_i}$. (5) is ultimately a linear least-squares problem that also preemptively encorporates the covariances [12]. When the optimizer solves for Δ_{i+1}, we can update the next state θ_{i+1} via addition for the vectors and an exponential map operation on the matrices [11].

For simplicity, from now on Δ_{i+1} will be denoted as Δ. To solve for Δ, we can employ the Schur trick on the Hessian information matrix $\boldsymbol{\mathcal{I}}$. We marginalize $\mathbf{L}$ from $\mathbf{C}$ in order to get the following block matrix

$$\boldsymbol{\mathcal{I}} = \begin{bmatrix} \mathbf{C} & \mathbf{B} \\ \mathbf{B}^\top & \mathbf{L} \end{bmatrix} \quad \boldsymbol{\sigma} = \begin{bmatrix} \sigma_\mathbf{C} \\ \sigma_\mathbf{L} \end{bmatrix} \tag{6}$$

where $\mathbf{C} = \mathbf{J}_\mathbf{C}^\top \mathbf{J}_\mathbf{C}$, $\mathbf{L} = \mathbf{J}_\mathbf{L}^\top \mathbf{J}_\mathbf{L}$, $\mathbf{B} = \mathbf{J}_\mathbf{C}^\top \mathbf{J}_\mathbf{L}$, and $\boldsymbol{\sigma}$ is the residual change where $\sigma_\mathbf{C} = \mathbf{J}_\mathbf{C}\mathbf{r}$, $\sigma_\mathbf{L} = \mathbf{J}_\mathbf{L}\mathbf{r}$ and we solve for $\boldsymbol{\mathcal{I}}\Delta = \boldsymbol{\sigma}$. Now we are able to compute the Schur complement

$$\mathbf{S}_\mathbf{C} = \mathbf{C} - \mathbf{B}\mathbf{L}^{-1}\mathbf{B}^\top \tag{7}$$

$$\mathbf{S}_\mathbf{C}\Delta_\mathbf{C} = \sigma_\mathbf{C} - \mathbf{B}\mathbf{L}^{-1}\sigma_\mathbf{L} \tag{8}$$

where $\mathbf{S_C}\Delta_\mathbf{C}$ is the conjugate gradient primitive. Note that $\Delta_\mathbf{L}$ is solved through back-substitution. The key to solving the above is making the products $\mathbf{BL}^{-1}\mathbf{B}^\top$ and $\mathbf{BL}^{-1}\sigma_\mathbf{L}$ as sparse as possible. We can efficiently solve for these equations through naive incremental updates $\delta\mathbf{L}^{-1}$ to the structure, but for cases where too many points are being modified $\delta\mathbf{L}^{-1}$ can be almost as dense as it would be through a non-incremental step, defeating the purpose of incremental simplification. We present an incremental approach that is capable of incrementally solving the Schur complement in cases where $\delta\mathbf{L}^{-1}$ is both sparse and dense.

C. Fragmented Acceleration

To dramatically accelerate the iterative solution for (8), the solver must be able to handle both cases where the update $\delta\mathbf{L}^{-1}$ is either sparse or dense. Other incremental formulations [12], [13], [17] consider only sparse cases and thus have to resort to expensive batch updates when the frames in the sliding window share the same features (since the state of all variables have to be updated). For this we modify the incremental update step of (8) by changing the set of structure points used for optimization.

Since $\mathbf{S_C}$ is composed from the Schur complement of individual points, we can treat them separately. However, this doesn't mean we have to construct the Schur complement for every single feature point in an *explicit* fashion. This is favorable because some points that are observable for a few cameras hold minimal information content, and only serve to make the Schur complement more dense. If we consider the case where $\mathcal{C}_k$ are seeing $\mathcal{L}_k$ points, then the complexity of an explicit method would be $\mathcal{O}(\mathcal{C}_k^2)$ since the Schur complement matrix is dense and the vector is full, but for an implicit method the complexity would be the number of measurements that we see, or $\mathcal{O}(\mathcal{C}_k\mathcal{L}_k)$ [18]. In the case of incremental VINS there are many more points shared by cameras than there are isolated points, so we can simply omit the isolated points in $\delta\mathbf{L}^{-1}$ from the dense part of the system for further efficiency and opt for additional iterations with our optimizer to ensure completeness. Ultimately this operation does not corrupt our information matrix as much and in the long run makes the problem sparser.

Now that we devised a way to keep our information matrix sparse, the question shifts to how we can make an incremental scheme that makes use of most information while remaining efficient. When we analyzed complexity, we considered the case for when all points are seen by all the cameras, but this is not the case for our VINS. This is where groups or factors, or *fragments* can come into use. It is known that $\mathbf{S_C}\Delta_\mathbf{C}$ is a sum of the contributions from each point observed by the cameras, or $\sum_{\mathbf{l}_j\in\mathcal{L}}\mathbf{S}_j\Delta_\mathbf{C}$. In that case, if we consider a single factor, calculating the Schur complement would be unfavorable, since it would bring little information to $\mathbf{S_C}$ given that it adds one image measurement yet many unknowns. Obviously, in such a case an implicit method would be best, but it is precisely what we would like to avoid. By the use of fragmentation, we can sum multiple $\mathbf{S}_j$ into a single resulting matrix, a fragment of $\mathbf{S_C}$.

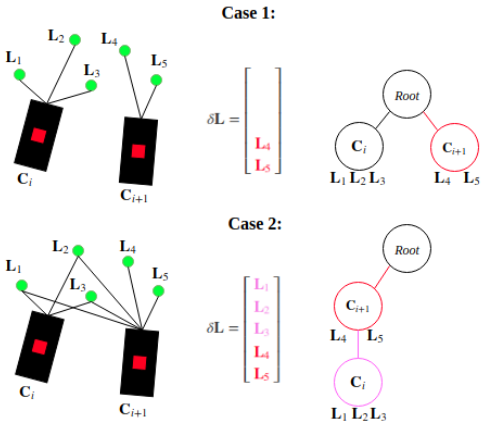

Fig. 1: A graphical representation of an FP tree construction. Note that $\delta\mathbf{L}$ is only for illusory purposes and does not represent the true dimensions. The green dots are observed features while the red-black rectangles are the camera-IMU system. We can construct a FP tree from an arbitrary root node through addition (a "red" operation) of camera-feature associations, or modifying (a "pink" operation) pre-existing nodes.

Finally, we must find a way to efficiently determine the points that are shared by many cameras and those the points that are not. [15] checks for smaller segments of cameras that observe common points, but it employs a less efficient brute-force approach that explicitly searches for the set of cameras that share the most points. It is also less accurate since it may not include cameras that share a high number of points but are out of range. In data mining problems, a *frequent pattern tree* (FP tree) [19], [20] is sometimes used in machine-learning techniques to categorize the database by priority for very efficiently finding associations between variables. We can similarly utilize this for our BA problem [21] by constructing a FP tree to efficiently associate the number of points with the number of cameras, where cameras are represented as nodes that observe a number of feature points.

Cameras associated with more feature points are of lower depth (i.e. closer to the root node). However, the FP tree has only been applied to generic BA problems [21]; as such, we utilize our FP tree in an incremental manner by adding incoming camera measurements as new nodes as well as removing old camera nodes that are at the back of the sliding window, and an example of our FP tree can be seen in Figure 1. This is also referred to as the FP-growth algorithm [22]. Using the FP tree we can easily find fragments that share the most common points and solve the Schur complement update step. Points that are too isolated are not considered. To represent the degree of isolation, we can choose a limit to the points-to-cameras ratio below which the points are not included in the Schur complement calculation. For example, if we choose a limit of 4 points-to-cameras ratio, we do not include 3 points seen by a single

camera or 4 points seen by 2 cameras. Since we do not include all the points in our Schur complement calculation, error optimization becomes significantly cheaper and many frames can be utilized in our sliding window for a much more robust solution.

III. Continuous Preintegration

Our state formulation for the IMU state and noise rests upon the foundations described in [14]. An IMU measuring the inertial state of a robot outputs measurements $\widetilde{\boldsymbol{\omega}}_k$ and $\widetilde{\mathbf{a}}_k$ affected by noise from both its internal gyroscope and the accelerometer, with the true measurements being $\boldsymbol{\omega}_k$ and $\mathbf{a}_k$. As such, the standard IMU kinematic model of our system can be described as the following [23]

$$\dot{\mathbf{R}} = \mathbf{R}_k \boldsymbol{\omega}_k^{\wedge} \quad \mathbf{a}_k = \mathbf{R}_k \mathbf{a}_k + \mathbf{g}_w \quad \dot{\mathbf{p}}_k = \mathbf{v}_k \tag{9}$$

where $\mathbf{p}_k$ and $\mathbf{v}_k$ are the position and velocity at time frame k, and the $\wedge$ operator represents the skew-symmetric operation. Given a state $\mathbf{x}_k$ at keyframe k, we can use the process of preintegration (i.e. combining a number of IMU measurements from two consecutive keyframes into one motion contraint) [14] to predict the future state $\mathbf{x}_{k+1}$ of our system. For our preintegration formulation, we first define the change in IMU velocity and position using (9) as follows

$$\mathbf{R}_{k+1} = \mathbf{R}_k \Delta\mathbf{R} \exp(\mathbf{J}^g_{\Delta\mathbf{R}} \mathbf{b}^g_k) \tag{10}$$

$$\begin{aligned}\mathbf{v}_{k+1} &= \mathbf{v}_k + \mathbf{g}_w \Delta t \\ &\quad + \mathbf{R}_k(\Delta\mathbf{v} + \mathbf{J}^g_{\Delta\mathbf{v}} \mathbf{b}^g_k + \mathbf{J}^a_{\Delta\mathbf{v}} \mathbf{b}^a_k)\end{aligned} \tag{11}$$

$$\begin{aligned}\mathbf{p}_{k+1} &= \mathbf{p}_k + \mathbf{v}_k \Delta t + \frac{1}{2}\mathbf{g}_w \Delta t^2 \\ &\quad + \mathbf{R}_k(\Delta\mathbf{p} + \mathbf{J}^g_{\Delta\mathbf{p}} \mathbf{b}^g_k + \mathbf{J}^a_{\Delta\mathbf{p}} \mathbf{b}^a_k)\end{aligned} \tag{12}$$

where $\Delta\mathbf{R}, \Delta\mathbf{p}$, and $\Delta\mathbf{v}$ are the preintegration measurements that are linearized with respect to the biases between two consecutive image time frames k and $k+1$ unless specified otherwise, and $\Delta t = t_{k+1} - t_k$. The Jacobian terms are defined in [14]. They refer to the first order approximations, e.g. $\mathbf{J}^g_{\Delta\mathbf{R}_k} = \frac{\partial \Delta\mathbf{R}_k}{\partial \mathbf{b}^g_k}$, and can be calculated incrementally as the IMU measurements arrive.

Following [23], we use a *local* representation when calculating the next velocity preintegration mean in the IMU measurement time frame τ

$$\Delta\mathbf{v}^k_\tau = \int_{t_k}^{t_\tau} \mathbf{R}^k_\tau (\widetilde{\mathbf{a}}_k - \mathbf{b}^a_k - \mathbf{n}^{ba}_k - \mathbf{R}^k_\tau \mathbf{R}_k \mathbf{g}_w)\, dt \tag{13}$$

where $\mathbf{R}^k_\tau$ is the rotation from time frame k to the IMU time frame τ and $\mathbf{n}^{ba}_k$ is the slowly accelerometer's varying Brownian random motion bias at k. The effect of gravity is integrated into the acceleration estimation to account for cases where the IMU is rotating against the gravitational direction, hence we estimate acceleration in its own local frame. This is because unlike the traditional model, we choose not to treat IMU measurements as strictly fixed in their value over the measurement time, as small as it is. Going off of the fact that $\Delta\dot{\mathbf{p}} = \Delta\mathbf{v}$, we can simplify the preintegration mean calculation to a continuous formulation done successively over the IMU measurements

$$\Delta\mathbf{R}^k_{\tau+1} = \Delta\mathbf{R}^k_\tau \exp(\boldsymbol{\omega}_k \Delta t) \tag{14}$$

$$\Delta\mathbf{v}^k_{\tau+1} = \Delta\mathbf{v}^k_\tau + \mathbf{R}^k_{\tau+1} \int_{t_\tau}^{t_{\tau+1}} \exp(\boldsymbol{\omega}_k \delta t)\mathbf{a}_k \; d\delta t \tag{15}$$

$$\begin{aligned}\Delta\mathbf{p}^k_{\tau+1} &= \Delta\mathbf{p}^k_\tau + \Delta\mathbf{v}^k_\tau \Delta t \\ &\quad + \mathbf{R}^k_{\tau+1} \int_{t_\tau}^{t_{\tau+1}} \delta t \exp(\boldsymbol{\omega}_k \delta t)\mathbf{a}_k \; d\delta t\end{aligned} \tag{16}$$

where $\Delta t = t_{\tau+1} - t_\tau$ between IMU measurements, $\delta t = t_{\tau+1} - u$ with u as an integration constant, and exp is an exponential map operation from $\mathbb{R}^3$ to SO(3). We can apply Rodrigues' equation to remove the integration notation [24]. Overall, the continuous preintegration mean update step better models the dynamics while remaining simpler than discrete methods.

Since we defined everything needed to describe the motion constraint, the complete error state uncertainty can be defined by manipulating (10), (11), (12)

$$\mathbf{r}_{\mathbf{R}} = \log((\Delta\mathbf{R} \exp(\mathbf{J}^g_{\Delta\mathbf{R}} \delta\mathbf{b}^g_i))^\top \mathbf{R}_i^\top \mathbf{R}_j) \tag{17}$$

$$\begin{aligned}\mathbf{r}_{\mathbf{p}} &= \mathbf{R}_i^\top(\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}\Delta t - \frac{1}{2}\mathbf{g}_w \Delta t^2) \\ &\quad - (\Delta\mathbf{p} + \mathbf{J}^g_{\Delta\mathbf{p}} \delta\mathbf{b}^g_i + \mathbf{J}^a_{\Delta\mathbf{p}} \delta\mathbf{b}^a_i)\end{aligned} \tag{18}$$

$$\begin{aligned}\mathbf{r}_{\mathbf{v}} &= \mathbf{R}_i^\top(\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}_w \Delta t) \\ &\quad - (\Delta\mathbf{v} + \mathbf{J}^g_{\Delta\mathbf{v}} \delta\mathbf{b}^g_i + \mathbf{J}^a_{\Delta\mathbf{v}} \delta\mathbf{b}^a_i)\end{aligned} \tag{19}$$

$$\mathbf{r}_{\mathbf{b}} = \mathbf{b}_j - \mathbf{b}_i \tag{20}$$

$$\mathbf{r}_{\mathcal{C}_0} = \begin{bmatrix} \log(\hat{\mathbf{R}}_j^\top \mathbf{R}_j) \\ \mathbf{R}_j^\top(\hat{\mathbf{p}}_j - \mathbf{p}_j) \\ \mathbf{R}_j^\top(\hat{\mathbf{v}}_j - \mathbf{v}_j) \\ \hat{\mathbf{b}}_j - \mathbf{b}_j \end{bmatrix} \tag{21}$$

where $[\hat{\mathbf{R}}_j, \hat{\mathbf{v}}_j, \hat{\mathbf{p}}_j, \hat{\mathbf{b}}_j]$ is the estimated state at a time frame j, $\mathbf{r}_{\mathcal{C}_0}$ is the prior residual, $\mathbf{r}_{\mathbf{C}_i} = [\mathbf{r}_{\mathbf{R}}, \mathbf{r}_{\mathbf{p}}, \mathbf{r}_{\mathbf{v}}, \mathbf{r}_{\mathbf{b}}]^\top$ is the IMU residual, $\delta\mathbf{b}$ is the difference between the estimated bias and the real bias, and $\Delta t = t_j - t_i$. Note that the above error definitions are the IMU error and prior optimized upon in section II. For accurately estimating the noise in the preintegrated measurements, the discrete covariance $\Sigma_{\mathbf{C}}$ (which we denote as $\mathbf{P}_k$ in this section) of $\mathbf{r}_{\mathbf{C}}$ needs to be obtained. When the program starts, the covariance is initialized to $\mathbf{0}_{15\times15}$. We define $\mathbf{F}_k$ as the jacobian transition matrix, and given that it changes between two consecutive IMU measurements, $\mathbf{P}_k$ can be calculated with a first degree Taylor expansion and incrementing with a predetermined IMU diagonal noise covariance matrix $\mathbf{Q}_0$ [24]

$$\mathbf{P}_{\tau+1} = (\mathbf{I} + \mathbf{F}_\tau \Delta t)\mathbf{P}_\tau(\mathbf{I} + \mathbf{F}_\tau \Delta t)^\top + \mathbf{Q}_0 \tag{22}$$

IV. Experimental Validation

A. EuRoC Datasets

Apart from the formulation we described in the previous sections, our VINS is implemented in two different ways;

for more computationally capable devices the visual frontend utilizes stereo images and employs the marginalization strategy described in [15], though we also recommend [25]. However, for less powerful platforms the visual frontend utilized images from a monocular camera and older frames were simply dropped out of the sliding window. Feature detection, LK optical flow tracking, and pyramid construction for our visual frontend was implemented using NEON intrinsics when the algorithm was executed on ARM processors. We adapted the code base of ICE-BA [15] but replaced the segment searching portion of it with our FP-tree formulation. We tested the two versions of our algorithm on the EuRoC datasets [26] with a laptop equipped with an Intel i7-8650U and a Raspberry Pi 3B+ slightly overclocked to 1.5GHz. The EuRoC datasets were recorded with a VI-Sensor, which synchronized a high quality IMU at 200Hz with two global shutter image sensors at 20Hz, but for the Raspberry Pi we only use the data from the IMU and left camera. We evaluate our two VINS versions with respect to the groundtruth and provide the RMSE in Table I. Cases where the algorithm failed are denoted with an "X".

TABLE I: EuRoC Absolute Translation RMSE (Meters)

	Ours	Ours (Raspberry Pi)	OKVIS	MSCKF	ROVIO
MH_01	0.126	0.217	0.179	**0.120**	0.236
MH_02	**0.091**	0.249	0.137	0.146	0.170
MH_03	**0.161**	0.367	0.268	0.305	0.419
MH_04	**0.198**	0.302	0.342	0.317	0.541
MH_05	**0.232**	0.421	0.313	0.452	0.607
V1_01	**0.068**	0.175	0.090	0.118	0.221
V1_02	**0.101**	0.213	0.112	0.158	0.198
V1_03	**0.157**	0.368	0.203	0.281	0.345
V2_01	**0.080**	0.233	0.188	0.179	0.294
V2_02	0.182	0.288	**0.155**	0.269	0.337
V2_03	0.223	0.359	**0.219**	X	0.256

We compare our algorithm against MSCKF-VIO [9], OKVIS [3], and ROVIO [27], all state-of-the-art VINS with implementations based on both optimization and filtering, and it can be easily seen that our implementation performs remarkably well on all datasets against the other algorithms. The Raspberry Pi had performance comparable to that of ROVIO, but due to a large window size it was fairly robust, and even managed to beat out the other algorithms in MH_04_difficult. Note that loop closure was disabled for OKVIS and ROVIO for a fair comparison with our approach. In addition, we attempted an accuracy comparison with ICE-BA [15], but we found that the accuracy was very similar to our algorithm. This is likely attributed to the fact that our code is based upon ICA-BA.

We also compare the CPU resource usage of all the VINS in Figure 2. For our laptop, we used a sliding window size of 40 frames and on the Raspberry Pi we used a sliding window size of 12 frames. While we compared the accuracy of our algorithm on both the laptop and the Raspberry Pi in respect to other VINS formulations, we do not include the Raspberry Pi in the computational resource usage comparison test since the only other algorithm that could run on the Raspberry Pi was ICE-BA, albeit at around 7 frames per second. As for our VINS, the Raspberry Pi tracked the state using a single ARM core with near real-time performance as the frontend received images at their original 752 by 480 pixel resolution. Our visual frontend needed only 18 ms on average when using NEON intrinsics as opposed to 50 ms when using OpenCV. The Raspberry Pi did experience intermittent buffering due to updates in global map updates, but we chose to keep it enabled since it made the estimator significantly more robust.

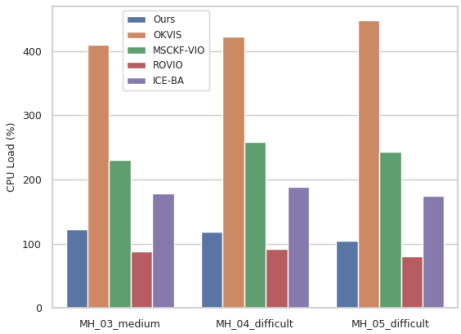

Fig. 2: Comparison of Laptop CPU utilization on three datasets. Note that the results correlate when testing on other datasets, so three datasets are enough to represent the true CPU utilization.

Interestingly, both MSCKF-VIO and OKVIS showed very high resource usage. This can be attributed to the fact that the i7-8650U processor inside the laptop is composed of eight low-power cores running at 1.9GHz. MSCKF-VIO and OKVIS are obviously designed for processors with high-power cores as opposed to low-powered ones. Our proposed VINS, however, uses slightly more than one core and utilizes almost as little resources as ROVIO. We also include a comparison of ICE-BA's resource usage on the laptop, though we don't see a great difference in resource usage, likely since our algorithm and ICE-BA both use the same marginalization technique on laptops. It is likely that the noticeable difference in performance on the Raspberry Pi was due to the fact that the computational bottleneck of both our algorithm and ICE-BA was the fragmentation process.

B. Ultra-Low Power Test

Since the algorithm ran fairly well on the Raspberry Pi, we decided to do an additional test using a NanoPi Neo Core2-LTS computer module which sports an Allwinner H5, an ultra-low power processor that is computationally less capable than the Raspberry Pi. The benefit of the computer module is that it is very small and light, measuring at 40mm by 40mm and weighing around 7 grams when the heatsink and headers are removed.

When tested on the NanoPi, our VINS showed an average throughput of 12 FPS with original images from MH_01_easy, and required around 800mW of power. As can be seen in Figure 3, powerful computers can be mounted on

250mm size MAVs, but not on smaller quadrotor platforms due to weight and power constraints. However, given the small size and weight of the NanoPi module, there is great promise in implementing our fragmented VINS algorithm on nano quadrotors of sizes as small as 75mm.

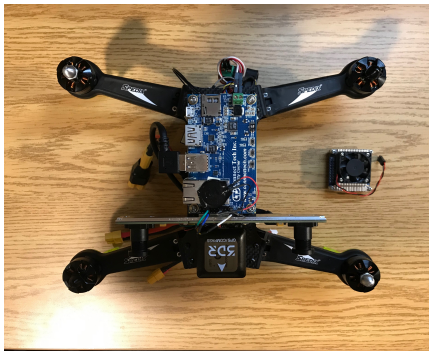

Fig. 3: Size comparison between the NanoPi module and a Jetson TX2 mounted on a 250mm size quadrotor.

V. Conclusion

In this work, we showed that it is possible to significantly accelerate the process of bundle adjustment via a fragmentation process. To further speed up our algorithm we utilized a continuous representation for preintegration and used NEON intrinsics to efficiently extract and track image features. When tested in the EuRoC datasets, our algorithm required little computational resources yet exhibited highly accurate state estimation on a laptop, a Raspberry Pi, and an ultra-low power computer module. Overall, our algorithm consists of the "best of both worlds"; on the one hand it's as accurate as optimization-based approaches and on the other it's as computationally light as filtering-based approaches. We see great promise in implementing our VINS in nano drones for indoor autonomous navigation without any external input. In the future, we plan to build an integrated system consisting of a monocular camera, IMU, and a low-power processor to place in a nano quadrotor for onboard autonomous estimation and navigation.

References

[1] S. Weiss, M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart. Realtime onboard visual-inertial state estimation and self-calibration of MAVs in unknown environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 957–964, 2012.

[2] S. Lynen, M. Achtelik, S. Weiss, M. Chli, and R. Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2013.

[3] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visualinertial odometry using nonlinear optimization. *International Journal of Robotics Research*, 34:314–334, 2015.

[4] A. Concha, G. Loianno, V. Kumar, and J. Civera. Visual-inertial direct slam. In *International Conference on Robotics and Automation (ICRA)*, pages 1331–1338, May 2016.

[5] M. Li and A. Mourikis. High-precision, consistent ekf-based visual inertial odometry. *Int. J. Robot. Research*, 32(6):690–711, May 2013.

[6] S. Huang and G. Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. In *IEEE Transactions on Robotics*, volume 23, pages 1036–1049, October 2017.

[7] T. Zhang, K. Wu, D. Su, S. Huang, and G. Dissanayake. An invariant-ekf vins algorithm for improving consistency. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[8] Joel A. Hesch, Dimitrios G. Kottas, Sean L. Bowman, and Stergios I. Roumeliotis. Camera-imu-based localization: Observability analysis and consistency improvement. In *The International Journal of Robotics Research*, volume 33, pages 182–201, 2014.

[9] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J. Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[10] Alex Zihao Zhu, Nikolay Atanasov, and Kostas Daniilidis. Event-based visual inertial odometry. In *CVPR*, pages 5816–5824. IEEE Computer Society, 2017.

[11] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31:217–236, February 2012.

[12] Frank Dellaert and Michael Kaess. Square Root SAM: Simultaneous location and mapping via square root information smoothing. *IJRR*, 25(12):1181, 2006. Special issue on RSS 2006.

[13] Viorela Ila, Lukás Polok, Marek Solony, and Klemen Istenic. Fast incremental bundle adjustment with covariance recovery. In *2017 International Conference on 3D Vision, 3DV 2017, Qingdao, China, October 10-12, 2017*, pages 175–184, 2017.

[14] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual-inertial odometry. *IEEE Trans. Robotics*, 33(1):1–21, 2017.

[15] Haomin Liu, Mingyu Chen, Guofeng Zhang, Hujun Bao, and Yingze Bao. ICE-BA: incremental, consistent and efficient bundle adjustment for visual-inertial SLAM. pages 1974–1982, 2018.

[16] Javier Civera, Andrew J. Davison, and J. M. Martnez Montiel. Unified inverse depth parametrization for monocular slam. In *In Proceedings of Robotics: Science and Systems*, 2006.

[17] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO)*, 24(6):1365–1378, December 2008.

[18] Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. Bundle adjustment in the large. In *Proceedings of the 11th European Conference on Computer Vision: Part II*, ECCV'10, pages 29–42, 2010.

[19] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, Aug 2007.

[20] Akshita Bhandari, Ashutosh Gupta, and Debasis Das. Improvised apriori algorithm using frequent pattern tree for real time applications in data mining. *CoRR*, abs/1411.6224, 2015.

[21] Luca Carlone, Pablo Fernández Alcantarilla, Han-Pang Chiu, Zsolt Kira, and Frank Dellaert. Mining structure fragments for smart bundle adjustment. 2014.

[22] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition*. Morgan Kaufmann, 2017.

[23] Manon Kok, Jeroen D. Hol, and Thomas B. Schn. Using inertial sensors for position and orientation estimation. *Foundations and Trends in Signal Processing*, 11(1-2):1–153, 2017.

[24] Kevin Eckenhoff, Patrick Geneva, and Guoquan Huang. Continuous preintegration theory for graph-based visual-inertial navigation. *CoRR*, abs/1805.02774, 2018.

[25] Jerry Hsiung, Ming Hsiao, Eric Westman, Rafael Valencia, and Michael Kaess. Information sparsification in visual-inertial odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 1146–1153, 2018.

[26] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.

[27] Michael Blösch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*, pages 298–304, 2015.