Navion: A 2mW Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones

Amr Suleiman, *Member, IEEE,* Zhengdong Zhang, *Student Member, IEEE,* Luca Carlone, *Member, IEEE*
Sertac Karaman, *Member, IEEE* and Vivienne Sze, *Senior Member, IEEE*

***Abstract*—This paper presents Navion, an energy-efficient accelerator for visual-inertial odometry (VIO) that enables autonomous navigation of miniaturized robots (e.g., nano drones), and virtual/augmented reality on portable devices. The chip uses inertial measurements and mono/stereo images to estimate the drone's trajectory and a 3D map of the environment. This estimate is obtained by running a state-of-the-art VIO algorithm based on non-linear factor graph optimization, which requires large irregularly structured memories and heterogeneous computation flow. To reduce the energy consumption and footprint, the entire VIO system is fully integrated on chip to eliminate costly off-chip processing and storage. This work uses compression and exploits both structured and unstructured sparsity to reduce on-chip memory size by 4.1×. Parallelism is used under tight area constraints to increase throughput by 43%. The chip is fabricated in 65nm CMOS, and can process 752×480 stereo images from EuRoC dataset in real-time at 20 frames per second (fps) consuming only an average power of 2mW. At its peak performance, Navion can process stereo images at up to 171 fps and inertial measurements at up to 52 kHz, while consuming an average of 24mW. The chip is configurable to maximize accuracy, throughput and energy-efficiency trade-offs and to adapt to different environments. To the best of our knowledge, this is the first fully-integrated VIO system in an ASIC.**

***Index Terms*—visual inertial odometry, VIO, localization, mapping, SLAM, nano drones, navigation**

I. Introduction

Technologies for autonomous navigation has attracted a lot of attention in the recent years, motivated by a myriad of consumer products that become available in the market. In particular, estimating the 3D-motion of a vehicle within an environment, referred to as estimating its ego-motion, is of crucial importance. Many other autonomy tasks, including motion planning and obstacle avoidance, often require an understanding of the location of the vehicle, which is provided by ego-motion estimation [1–3]. Ego-motion estimation is also critical for various other applications, such as virtual reality (VR) and augmented reality (AR) [4].

The problem of ego-motion estimation is well-known in the robotics community. A widely-studied relevant class of problems is called Simultaneous Localization And Mapping (SLAM), in which the goal is to build a map of the robot's surrounding and estimate the location of the robot in this map, simultaneously. Visual-Inertial Odometry (VIO) is often considered a special instance of SLAM, where camera and inertial measurement unit (IMU) data is used to estimate their location (with respect to their initial location) and a map the surroundings. Fig. 1 illustrates the inputs and outputs of a VIO pipeline. Compared to a full-SLAM system [5, 6], VIO does not have loop closure, which happens at much lower frequency and can be off-loaded to the cloud. As a result, a VIO system can output the estimated motion at significantly higher throughput, which is critical for autonomous navigation of fast moving robots/drones/vehicles, and also critical to reduce the motion sickness of a consumer using AR/VR devices.

Many VIO algorithms have been proposed [7–9]. However, running these algorithms in real-time requires relatively powerful CPUs and/or GPUs [10]. Mounting such CPUs and GPUs on a big drone such as Skydio R1 is feasible [11]. However, these solutions cannot be applied to nano and pico drones/unmanned aerial vehicles (UAVs) as those presented in [12, 13]. This is because of the constraints on the form factor as well as the extremely limited power budget available on these miniature vehicles. For example, the budget for a stable flight in nano/pico drone is around 100mW [13], which is an order of magnitude lower than the power dissipation of embedded CPUs. To the best of our knowledge, the smallest commercially-available UAV that is using VIO to estimate its own location was announced by Qualcomm using their machine vision SDK [14]. However, the drone is relatively large and weighs 250 grams, and it uses Qualcomm Snapdragon 801 processor, utilized in smart phones, which consumes around 3 Watts of power [15].

This motivates us to build a VIO accelerator for miniature drones. Potential utilization of custom accelerators for VIO was mentioned in a review article on pico drones, which appeared recently in Nature [16]. There has been a lot of work on building energy-efficient accelerators for these applications. A unified graphics and vision processor for pose estimation is presented in [17]. However, the problem is simplified by using known markers in images. Hong et al. [18] implements a marker-less camera pose estimation for a practical AR application, but it depends on off-chip image processing and external storage, which increase the overall system power consumption. Li et al. [19] implements parts of the image processing needed in a VIO/SLAM system. One known downside of custom accelerators, compared to CPU/GPU, is

A. Suleiman, Z. Zhang and V. Sze are with the Department of Electrical Engineering and Computer Science, MIT, Cambridge MA, 02139. L. Carlone and S. Karaman are with the Department of Aeronautics and Astronautics, MIT, Cambridge MA, 02139. (Project website: http://navion.mit.edu/). This work was partially funded by the AFOSR YIP FA9550-16-1-0228 and by the NSF CAREER 1350685.

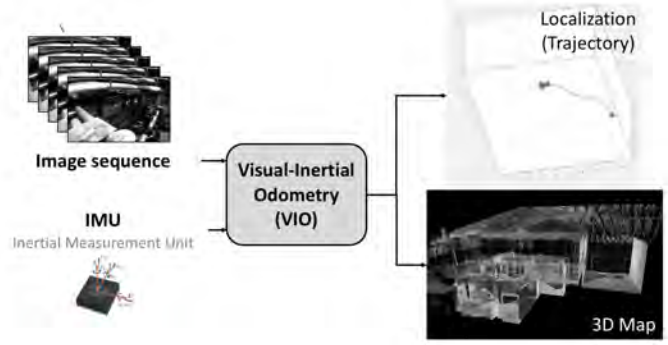


Fig. 1. Visual-Inertial Odometry (VIO) processing image and IMU measurements streams for trajectory and 3D map output.

the lack of flexibility, which comes as a direct trade-off with the low power and fast processing that accelerators deliver. Some accelerators take it to an extreme by hard-wiring the design to perform a specific task in a specific environment [20], which severely limits its practical applications.

In all of these works, the hardware design is separated from the algorithmic choices; however, it is shown in our work in [21] with an FPGA prototype implementation that a hardware and algorithm co-design strategy can provide significant benefits. This work builds on [21] and carries out more optimization for an efficient ASIC implementation. Here is a summary of the contributions and findings of this work:

- We propose Navion, an energy-efficient and fully integrated VIO implementation that runs in real-time to enable autonomous navigation in miniaturized robots/UAVs. Compared to an optimized software VIO implementation, Navion is 1582× more energy-efficient than Xeon desktop CPU, and 684× more energy-efficient than a low power embedded ARM CPU. To the best of our knowledge, this is the first fully integrated VIO system in an ASIC.
- Multiple algorithmic and architectural optimizations are carried out in Navion. Efficient memory hierarchy and data movement enable a 9× reduction of the external DRAM bandwidth. Data compression reduces on-chip memory size by 4.4×, while taking advantage of fixed and dynamic data sparsity enables 5.2× and 5.4× smaller memory size, respectively. Rescheduling and parallelism enable 43% faster processing with minimal to no overhead.
- We find that adding sufficient adaptability in Navion's architecture can improve accuracy and increase throughput based on the environment and camera movement. Adapting Navion to the different sequences results in an additional 2.5× improvement in energy efficiency.
- While Navion is built as a fully custom VIO accelerator to target autonomous navigation in miniaturized robots/UAVs, several blocks in Navion can be easily used as accelerators for other SLAM/VIO algorithms. This includes IMU preintegration, linear solver, and all image processing blocks (i.e., feature detection, feature tracking, Undistortion & rectification and stereo block matching).

The rest of the paper is organized as follows: Section II gives an overview of the VIO algorithm. In Section III, Navion architecture is presented showing the degree of flexibility in the architecture. Section IV discusses some of the hardware optimizations for memory size reduction. Section V shows the evaluation of Navion's accuracy and an analysis of the effect of changing the chip's configuration parameters on overall performance. Finally, Section VI concludes the paper.

II. Overview of Visual-Inertial Odometry (VIO)

VIO is used to estimate the trajectory of a sensing device (e.g., sensors mounted on a drone) while reconstructing a map of the environment as shown in Fig. 2. The trajectory is the collection of the drone's state x, specifically its position P and orientation R, over time. These states are estimated based on measurements of the environment. Navion implements the keyframe-based VIO pipeline described in [9]. Keyframes (KFs) are a subset of the incoming frames at which the state estimation is performed. Different approaches for VIO are able to attain highly accurate state estimation via nonlinear optimization using cameras and IMUs. However, real-time optimization quickly becomes infeasible as the trajectory grows over time. This problem is further emphasized by the fact that inertial measurements come at high rate (100 Hz to 1 kHz), hence, leading to the fast growth of the number of variables in the optimization problem. In this work, we use the state-of-the-art IMU preintegration approach to significantly reduce the problem size and enable real-time implementation [9].

Navion's pipeline consists of three main components: Vision frontend (VFE), IMU frontend (IFE), and Backend (BE) following the standard terminology in [22]. The VFE tracks 3D landmarks (L_i) in the scene by extracting their corresponding 2D features (i.e., corners) from a camera frame, and tracks them between consecutive frames to create *feature tracks*. The IFE summarizes the IMU sensor data between two camera KFs into one measurement. The BE fuses the summarized inputs from the two sensors through a non-linear graph optimization, and then it outputs the position and orientation of the drone.

A. Vision Frontend (VFE)

The VFE detects and tracks 2D features that correspond to the 3D landmarks in the scene across camera frames. It supports both mono and stereo modes. VFE implements five main functions on the input images: feature tracking (FT), feature detection (FD), undistort & rectify (UR), stereo matching (SM), and geometric verification (GV).

- **Feature Tracking (FT)** uses the Pyramidal Lucas-Kanade optical flow [24] to track features corresponding to the same landmark across all frames, including non KFs.
- **Feature Detection (FD)** extracts features in the left stereo frame using the lightweight Shi-Tomasi corners [25]. Features are detected on a grid to keep the number of features in different regions of a KF constant.
- **Undistort & Rectify (UR)** processes the left and right stereo frames only at KFs to prepare them for disparity calculations.
- **Stereo Matching (SM)** is only active in stereo mode and it only runs on KFs. It gets the 3D coordinate of a feature using template matching between left and right

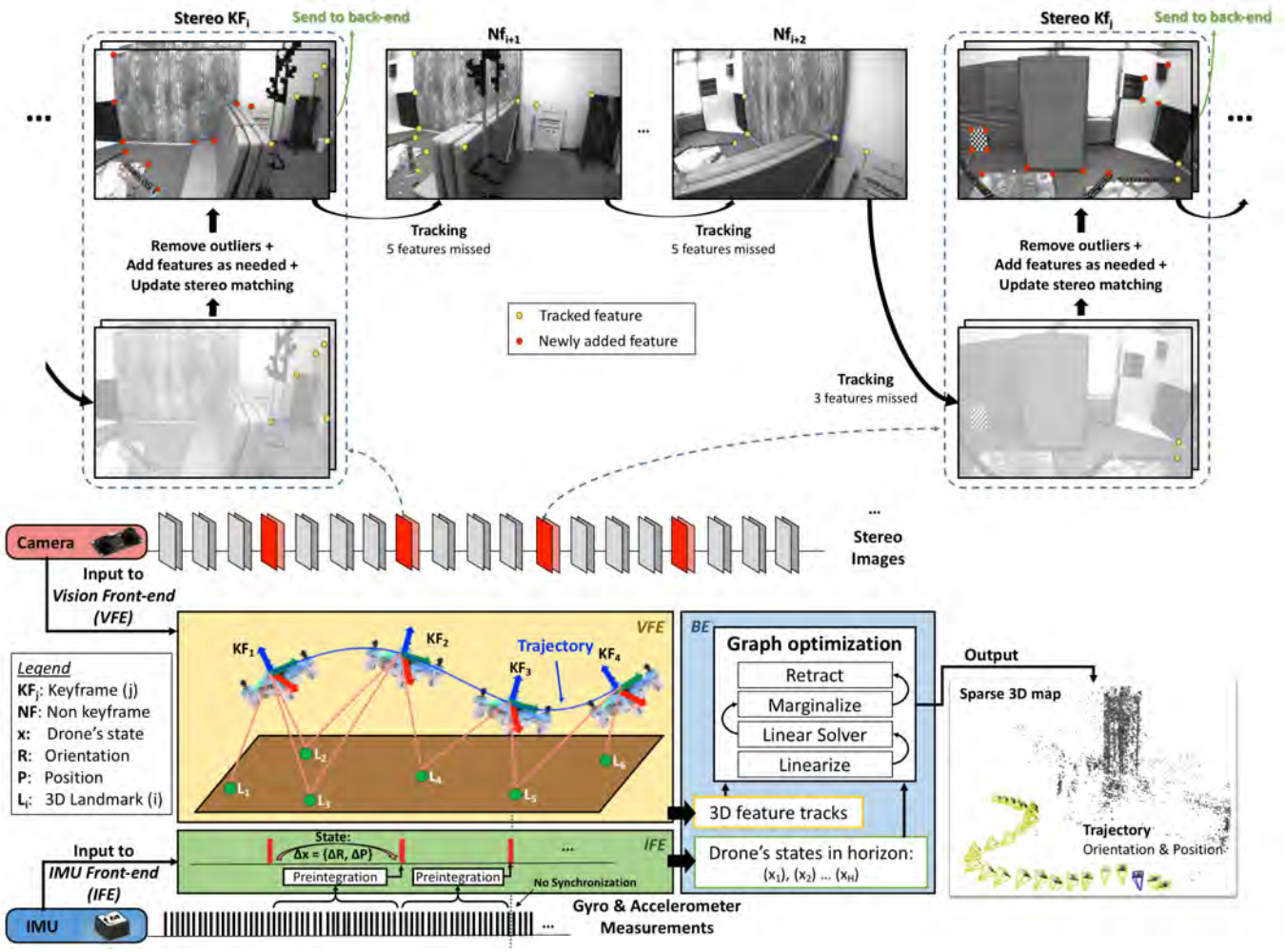


Fig. 2. VIO pipeline based on [23]. *IFE* summarizes the IMU measurements between *KF*s, while *VFE* tracks features in input images that correspond to important landmarks in the scene. *VFE* is *KF* based, i.e., it tracks features in all frames, but removes outliers, adds new features to compensate lost ones, and performs stereo matching only at *KF*s. *BE* fuses *VFE* and *IFE* outputs to perform state estimation at *KF*s using factor graph optimization.

stereo frames [26]. With undistorted and rectified frames, template matching happens on a horizontal search line.

- **Geometric Verification** (*GV*) removes any incorrect matches (i.e., outliers) resulting from *FT* and/or *SM*, and it runs only at *KF*s. Random sample consensus (*RANSAC*) is used in this pipeline. In Navion, *RANSAC* is informed by the rotation estimate produced by gyroscope integration. This enables using the simpler 2-point method [27] for *FT* (i.e., mono *RANSAC*) and 1-point method [28] for *SM* (i.e., stereo *RANSAC*), instead of the more complicated 5-point *RANSAC* method.

In summary, *VFE* processes the input mono/stereo frames to track several landmarks (L_i) based on their 2D projections in the images (i.e., features f_i) as shown in Fig. 2. *VFE* tracks features in all frames, and outputs landmark IDs, and their corresponding feature coordinates in left and right frames at *KF*s only. This information is used to generate feature tracks in the *BE*.

B. IMU Frontend (IFE)

IFE summarizes all the IMU measurements between two consecutive *KF*s into a single measurement using the preintegration theory [9]. Preintegration is a novel way to reduce the computation needed to include the inertial measurements in the estimation problem by two methods:

- With the inertial measurements coming at a relatively high output rate, preintegration combines all measurements between *KF*s into a single measurement, as shown in the bottom left part of Fig. 2. This reduces the number of variables in the optimization problem.
- *IMU* preintegration gives relative measurements between *KF*s, unlike standard *IMU* integration that uses the state estimate at the first frame as an initial condition [29]. This avoids repeating the *IMU* integration every time the state estimation is updated.

The mathematical details of the preintegration theory are outside the scope of this paper and can be found in [9]. The preintegration between *KF*s i and j gives the relative rotation ($\Delta\tilde{\boldsymbol{R}}_{ij}$), velocity ($\Delta\tilde{\boldsymbol{v}}_{ij}$) and position ($\Delta\tilde{\boldsymbol{p}}_{ij}$). At each *KF*, *IFE*

outputs the preintegration measurements to *BE*.

C. Graph Optimization Backend (BE)

The *BE* fuses the feature tracks from the VFE output and the IMU data from the IFE by solving a non-linear optimization problem whose solution is the *KF* state estimation, which describes the drone's trajectory and a sparse 3D landmarks map. The optimization is performed on multiple states within a sliding time window, referred to as the *horizon*, using a fixed-lag smoother to compute the maximum a posterior (*MAP*) estimate [23]. It formulates the problem as a *factor graph* [30], which describes the relationship and thus constraints[1] between all states in the horizon. These relationships between states defined by the factors are non-linear. There are three main types of factors in *BE*: *vision factors* are the feature tracks that describes the constraints between the 3D position and orientation of all the *KF*s that observe the same landmark; *IMU factors* describes the difference between the relative motion between a pair of *KF*s given by the *IMU* preintegrated measurements and the estimated relative motion from the state variables, and *marginalization* factor the describes the constraints between the states of the *KF*s that were connected by any factor that fall outside the current horizon. The *BE* must perform an optimization to find the *KF* states that best satisfies these factors and minimize any discrepancies.

BE solves the minimization problem using an on manifold Gauss-Newton method [23]. The four main functions are *linearization*, *linear solver*, *marginalization*, and *retract* as shown on the right side of Fig. 2. Specifically, the factors are linearized and accumulated into a large system of linear equations ($H\Delta x=\varepsilon$), where the Hessian matrix H and the error vector ε describe how the frontend measurements affect each *KF* state in the horizon, and the vector Δx is the state update.

A linear solver uses Cholesky factorization and back-substitution to solve this linear system of equations [31]. Cholesky factorization decomposes matrix H into the product of a lower triangular matrix and its transpose such that ($H=LL^T$), then back-substitution is used to solve for Δx. Marginalization summarizes the information of the states that fall outside of the current horizon. All the factors that constrain these states are removed from the graph. As a result of marginalization and limited horizon, the structure of the factor graph is dynamically updated over time. Finally, the solution of the linear system (Δx) is used to update the remaining *KF* states in the horizon using retract such that the updated variable is also on the manifold. Generally, the Gauss-Newton method is iterative but our BE runs only one iteration because the IMU information is used to provide a good initialization.

Navion's *BE* supports a very large and continuously varying factor graph problem, with a maximum of 20 *KF*s in the horizon and 4000 vision factors (i.e., landmarks); the selection of these parameters are described next in Section II-D. This poses challenges in storing and maintaining the graph on-chip efficiently.

[1]Here, we don't use the formal definition of constraints for optimization. Instead, these factors are included in the objective function in the optimization.

D. Design Considerations

a) Parameters: In Navion, a few important parameters can have significant impact on the trade-off in memory, throughput and accuracy of the design.

- **Horizon size** determines how many *KF*s are used in the *BE*. Let N be the number of *KF*s in a horizon. The size of the linear system that *BE* solves after linearization is $O(N^2)$. The time complexity for solving this linear system is $O(N^3)$. Therefore, reducing N can significantly reduce the memory requirements and increase the throughput of the system, but it will hurt the accuracy of the estimated trajectory [21]. Navion has a maximum horizon size of 20.
- **Number of features tracked per frame** affects all the major components of VFE linearly in terms of power and throughput. Navion tracks up to 200 features per frame.
- **Feature track age** is the maximal length allowed for a feature track. This parameter affects how many feature tracks will be included in the factor graph based optimization. Since each feature corresponds to a vision factor, scaling the feature track age affects the *BE* linearly. Note that a larger feature track age is generally preferred for high accuracy. Navion has a maximum feature track age of 10.

In applications where the tolerance of localization error is higher, it is possible to tune these parameters to reduce the memory and increase the throughput of the design significantly.

b) Feature Tracking vs. Descriptors Matching: VFE uses tracking to find features correspondence between frames, rather than matching descriptors, such as SIFT [32] and ORB [33], etc. In descriptors matching approach, such as ORB-SLAM [34], feature descriptors are generated for all detected features, and then descriptors are matched between frames to find correspondences. This requires memory and computation overhead to calculate, store, and match descriptors. Our approach avoids this overhead. Additionally, the performance of descriptors matching is greatly affected by the accuracy and the repeatability of feature detection in every frame, so that descriptors can be matched successfully.

c) Optimization Method: We employ a non-linear factor-graph based optimizer in our *BE* rather than an extended Kalman Filter (*EKF*). Extended Kalman Filter essentially summarizes all the feature tracks between two adjacent *KF*s into one factor, hence the maximal feature age can be considered to be 2. Therefore, extended Kalman Filter has lower complexity but its average motion estimation error can be 3× higher when benchmarked on the EuRoC dataset [35]. For instance, 0.52 m absolute translation error is achieved by ROVIO [36], which uses extended Kalman Filter, while 0.16 m error is achieved by our factor-graph method for the same dataset [9].

Bundle adjustment is also a very well known technique to estimate the camera positions from the visual features [26]. The goal is to minimize the total reprojection error from the landmark to the features. It is identical to a special case of the factor graph optimizer presented in this work, where only vision factors and prior factors are used.

d) VIO vs Visual Odometry (VO): Including inertial measurements increases the robustness of a SLAM system, especially in the indoor environments where insufficient visual features can be detected and tracked (e.g., a drone facing a white wall). It can also provide a good initialization for the non-linear optimization in *BE*, reducing the number of iterations needed for *BE* to converge to a good state estimation [9].

e) Stereo vs Mono: Using stereo frames avoids the scale ambiguity of a mono VIO system [26]. However, a stereo camera has higher power consumption than a mono camera. Therefore, Navion supports both mono and stereo modes for the user to choose depending on the target application scenario.

III. Navion: Architecture

Fig. 3 shows the overall architecture of the proposed VIO hardware accelerator. The VIO pipeline is fully integrated in Navion with no external storage or computation required. The architecture consists of the frontend (i.e., VFE, IFE), which processes the high-dimensional sensor data, and the backend (i.e., *BE*) which performs the state estimation following the algorithm discussed in Section II. The frontend and the backend are decoupled and running at two different frequencies. The data is transferred from the frontend to the backend through a simple memory interface. Navion is a standalone accelerator that takes images (mono or stereo) and inertial measurements as inputs, and outputs the trajectory and the sparse 3D map of the surrounding environment.

With the optimizations carried out in this work, no external storage is required (e.g., DRAM), which lowers the overall system power consumption. Navion has more than 250 programmable parameters (a subset of them is shown later in Table II). Some of these parameters are used to define the camera and IMU calibration and settings, such that a wide range of sensors can be used with Navion. The remaining parameters are used to control the VIO pipeline's complexity and trade-off accuracy, throughput and power consumption across different environments as shown later in Section V-D.

A. IMU Frontend (IFE)

Inertial measurements, with 6 values per measurement (i.e., 3 values from the accelerometer and 3 values the gyroscope) are streamed into the chip through a 32-bit input bus in single precision. IFE (lower left box in Fig. 3) processes them using double precision arithmetic to perform the IMU preintegration based on [23]. It contains a small shared register file and a 2kB SRAM to store intermediate data and the preintegrated results. IFE uses a dedicated double precision arithmetic compute unit that runs in parallel with the rest of the chip. The number of operations in the preintegration process is relatively small, hence IFE's complexity is low compared to the rest of the chip. It accounts for only 2.4% of the chip area, while consuming around 1% of the average total power.

B. Vision Frontend (VFE)

The input images, with a maximum resolution of 752×480 pixels, are streamed into the chip through two 8-bit buses for left and right stereo images, which are processed by the VFE (upper left box in Fig. 3). The input pixels are stored on-chip in line and/or frame buffers to lower the external bandwidth by up to 9× to 0.34 MB/frame only, which is the minimum bandwidth required to stream in a single frame (i.e., no pixel is read more than once). VFE uses fixed point arithmetic for lower power and smaller area, and can be divided into two different dataflows: parallel image processing and serial geometric validation.

The image processing dataflow (i.e., FT, FD, UR, and SM) processes pixels in parallel to increase the throughput. The geometric validation, highlighted by the dashed box inside VFE in Fig. 3, implements mono and stereo *RANSAC* using finite state machines (FSM), shared memory and shared arithmetic compute unit. VFE accounts for 53% of the overall chip area, and contains 439.4kB SRAM which is dominated by four frame buffers: original frames from the camera are stored in *Frame (1)* and *Frame (2)* memories to support FT. Undistorted and rectified frames are stored in *Left Frame* and *Right Frame* memories to support *SM*. VFE detects a total of 1824 features per frame, and can track up to 200 features per frame.

C. Backend

BE (right box in Fig. 3) solves the factor graph optimization problem. Due to the serial nature of this process, a dataflow similar to the geometric validation in VFE is used with a complex FSM. To reduce area and increase resource sharing, *BE*'s FSM is divided into a hierarchy of smaller FSMs. The small FMSs include: 1) Matrix operations that are used all over the factor graph optimization problem. 2) Cholesky factorization and back-substitution used in both linear solver and marginalization processes. 3) Rodrigues operations that are used in the linearization and retract processes.

A shared register file is used, with 85 double precision registers, as a memory hierarchy for intermediate data storage similar to IFE, along with 412.6kB of SRAM. The *Factor Graph* memory stores the VFE and IFE outputs in the horizon, which are then linearized into the *Linear Solver Matrix* memory. *BE* includes more than 4000 factors to support a maximum horizon size of 20 KFs in the optimization problem. *Horizon States* memory stores the state of the KFs in the horizon, while the *Shared Memory* stores large intermediate data. *BE* outputs the trajectory and the sparse 3D map of the surroundings at the KF rate.

Navion uses double precision arithmetic in the *BE* and IFE to ensure robustness of the VIO system. This doubles the size of all memories in *BE* and IFE. Using single precision does not provide sufficient numerical precision for the underlying VIO algorithm from [23]. This is due to the fact that the VIO pipeline is an open-loop system. Therefore, as time goes on, the uncertainty of the state estimation keeps increasing and the condition number of the linear system increases; this causes the problem to be ill-defined and thus cannot be properly solved.

D. Processing Modes

Navion has two modes of operations since the VIO is KF-based, as shown earlier in Fig. 2. A detailed timing diagram is shown later in Fig. 12. IFE is active in both modes.

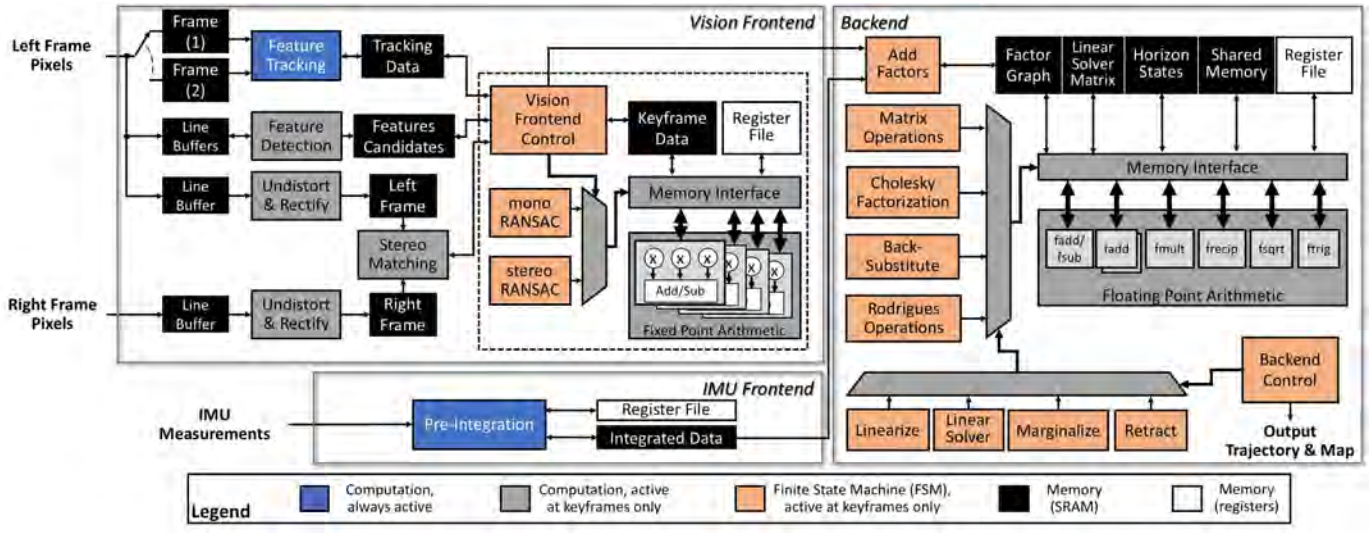


Fig. 3. Overall architecture of Navion chip. All components of the VIO pipeline are integrated on-chip. Frame buffers, Factor Graph, and Linear Solver memories account for 95% of the on-chip memory. Memory optimizations, along with rescheduling and parallelism, are carried out to enable full integration.

KF processing (stereo images labeled red in Fig. 2): In this mode, stereo frames are streamed in and all components are active. FT, FD, UR and SM modules process the incoming frames in parallel. The control FSM then starts the mono and stereo *RANSAC* in series to perform the geometric verification and remove outliers before adding new features as needed. BE then starts solving the factor graph optimization problem. The trajectory and sparse map outputs are updated after BE is done.

Non-KF processing (all other stereo images in Fig. 2): In this mode, FT is the only active component, while the rest of VFE and BE are off and clock-gated to reduce their power consumption. Right frames are not streamed in to reduce the off-chip memory bandwidth. The previously tracked features are stored in the *Tracking Data* memory, and FT updates them in-place if successfully tracked, or removes them from the memory if tracking fails. On average, processing non-FK is 3.8× faster than processing KF.

IV. Architecture Optimizations

This section presents the main optimizations carried out to enable energy-efficient full VIO integration in Navion. Table I, shown in next section, summarizes the resulting size reduction of the various on-chip memories, which is important to reduce power consumption and area cost. Additionally, we exploit parallelism and rescheduling to increase the overall throughput.

A. Image Compression

VFE has four frame buffers as shown in Section III-B. These buffers are needed because of the random reading patterns in FT, as a feature can move to anywhere between frames depending on the camera movement. Additionally, multiple frame readings happens over time in *SM* since it runs twice at different time slots. As a result, frames are stored on-chip to avoid re-streaming. Lossy image compression is used to reduce the size of these frame buffers.

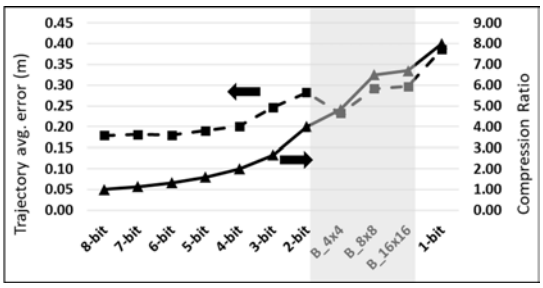


Fig. 4. Memory savings vs. VIO error with lossy image compression.

Fig. 4 shows the trade-off between the compression ratio and the VIO error. For a minimal overhead cost, two compression methods are analyzed. Truncating the least significant bits (*LSB*) is the simplest method to reduce the bitwidth with no overhead. Fig. 4 shows that VIO error increases by just 6% going from 8-bit to 5-bit per pixel while achieving 38% memory size reduction. However, the error increases much faster after that to more than double at the extreme of 1-bit per pixel.

Another lossy compression technique is block-wise quantization. The frame is divided into blocks of $N \times N$ pixels, and the dynamic range of each block is quantized into 2 levels for a 1-bit per pixel representation. This technique's overhead includes a line buffer storing $N-1$ rows and some logic to calculate the dynamic range of each block. The shaded part in Fig. 4 shows the numbers for three different block sizes: 4×4, 8×8, and 16×16. The 4×4 block quantization achieved less error than the 3-bit case, while increasing the memory savings from 2.7× to 4.4×.

Fig. 5 shows the image compression architecture used in Navion, which combines both lossy compression techniques described above. The pixels are quantized to 5-bit by simple *LSB* truncation, then the image is divided into blocks of 4×4 pixels. The pixel intensity dynamic range within each block is found, and a threshold divides this range in half. Every pixel is

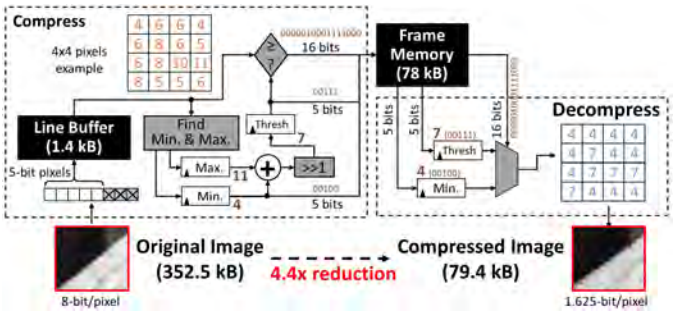


Fig. 5. Block-wise image compression architecture.

then represented by 1-bit, which is the result of comparing the pixel intensity to the threshold. Accordingly, each 4×4 block of pixels uses 26 bits to store its dynamic range (1-bit/pixel), threshold (5-bit) and minimum value (5-bit). With a compute overhead of 4 kgates (0.8% of the total kgates) and a 1.4kB line buffer, compression reduces the frame memory size by 4.4× and power by 4.9×. Compressed frames are used in *SM* and FT, but they are not use in FD because it is more sensitive to blocking and quantization artifacts as shown in Fig. 5.

B. Feature Tracks Unstructured Sparsity

Feature tracks account for 88% of the factor graph memory in BE. They contain all observed features in the current horizon, and are used by BE to solve the non-linear optimization problem. Each feature track stores the KF IDs where the landmark is observed in, and the 3D coordinates of the corresponding feature in each KF. Fig. 6-a shows an example of some feature tracks. Although a maximum feature age is defined (e.g., 10 in Navion), feature tracks have variable length depending on the image sequence. A feature track can be short because its landmark gets out of the field of view while the camera is moving (e.g., L_2), or because the tracking algorithm fails to track a landmark over time (e.g., L_5).

Fig. 6-b shows the feature tracks stored in one memory. Designing for the worst case, the memory has to store all observations in 20 KFs (i.e., maximum number of KFs in a horizon) each having 200 features (i.e., maximum number of features tracked per frame) and 10 observations per landmark (i.e., maximum feature age). This results in a big 962kB SRAM, with 40,000 entries, each containing a 5-bit KF ID and 3 64-bit double precision numbers for the 3D coordinates per observation. This memory is populated with measurements from VFE, and it is continuously changing with old features being removed and new features being added.

To reduce the size of this large memory, we noticed that it is sparsely populated with a maximum of 4,000 observations from VFE (200 features tracked per frame, 20 KFs in a horizon). However, the distribution of these non-zero entries is unknown and depends on the feature track length (Fig. 6-a). As a result, a two-stage memory architecture is used as shown in Fig. 6-c. The first sparse memory still has 40,000 entries, but rather than storing the 3D double precision coordinates, it stores 12-bit pointers to the second dense memory, which stores the 3D values of the 4,000 observations. This reduces the graph

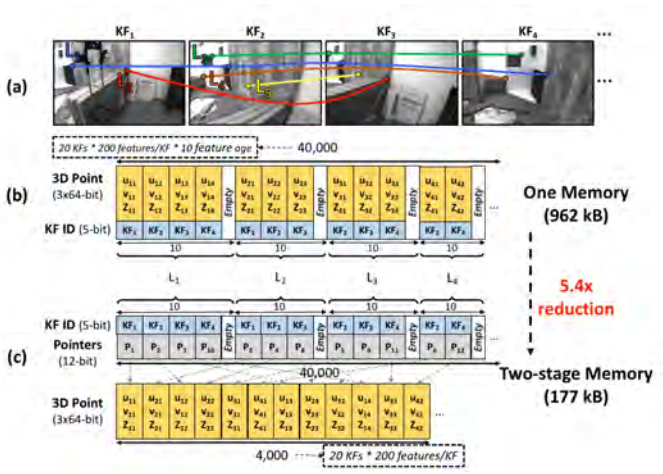


Fig. 6. Memory size reduction in the vision factors memory. (a) Examples of feature tracks with variable length. (b) Storing feature tracks in one memory designed for worst case. (c) Dividing the memory into two stages with pointers.

memory size by 5.4×, with an overhead of increasing access latency by only one cycle.

C. Linear Solver Structured Sparsity

The first step in BE to solve the non-linear factor graph optimization problem is to linearize all factors in the horizon. The result is a linear system of equations ($H\Delta x = \varepsilon$), where H is the Hessian matrix and ε is the residual error resulting from the linearization process. The top left part of Fig. 7 shows the linear solver process using Cholesky factorization and back-substitution, happening in-place in the linear solver memory that originally stores the H matrix. With a maximum of 20 KFs in the horizon, the size of H matrix is 300×300, where each KF's state contains 15 variables: 3 for position, 3 for orientation, 3 for velocity, and 6 for IMU bias. The H matrix stored in the linear solver memory and is updated every KF.

The H matrix has some characteristics that enable memory size reduction, shown in the bottom left part of Fig. 7. For instance, H is symmetric, which directly results in 2× memory size reduction by only storing the upper (or lower) triangle. Additionally, based on the ratio between the feature track age and the horizon size, only 38% of each of the matrix's triangles have non-zero values, labeled in black in Fig. 7, and their positions are fixed. By storing only the non-zero values, a total 5.2× memory size reduction is achieved. The top right part of Fig. 7 shows the linear solver memory wrapper. A sparse-based control unit takes the read/write requests with the row and column addresses, and it controls whether to perform the read/write operation on the small 134kB memory or to mask it according to the fixed sparsity pattern.

Solving the linear system using Cholesky factorization and back-substitution involve traversing the matrix row by row and column by column. Taking into account the fixed sparsity pattern of H matrix, the linear solver processing time can be reduced by skipping processing the zero locations. With the maximum horizon size of 20 KFs, a 7.2× speed-up is achieved by exploiting the H matrix structured sparsity. Fig. 8

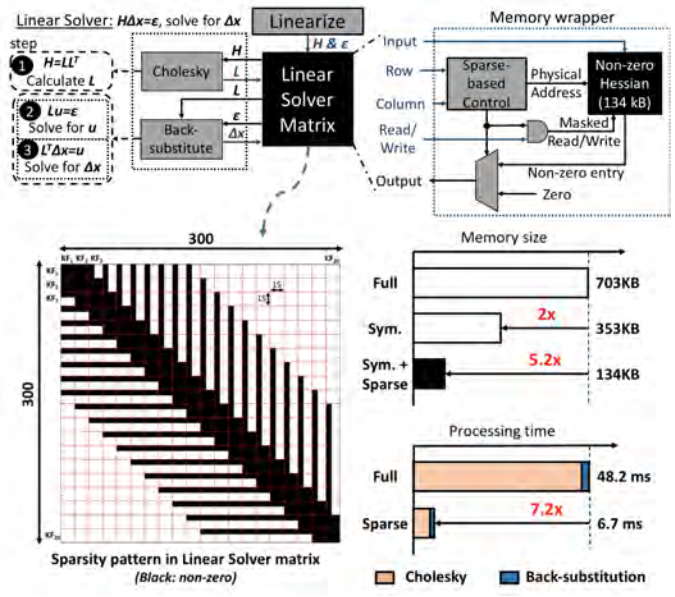


Fig. 7. Fixed sparsity visualization in linear solver matrix, with memory size reduction and throughput increase.

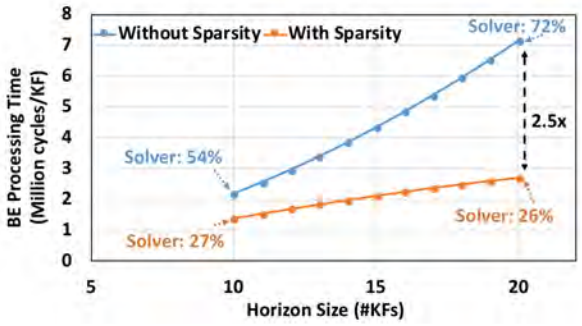


Fig. 8. BE processing time savings by exploiting sparsity in the linear solver matrix. The horizon size (in number of KFs) defines the size of the linear system. Percentages show how much time is spent in the linear solver relative to BE processing time.

shows the *BE*'s overall processing time savings with different horizon sizes. The linear solver processing time increases with $O(N^3)$ without sparsity, compared to an increase much slower than $O(N)$ when exploiting sparsity. This results in more time savings as the number of *KF*s in the horizon increases. At 20 *KF*s in the horizon, a maximum of 2.5× speed-up of *BE* processing time is achieved.

D. Rescheduling and Parallelism

Parallelism is carefully used to increase the throughput with minimal overhead. *VFE* in particular is suitable for parallelism due its nature of running image processing computation. The main advantage of rescheduling is that it achieves processing time savings without any effect on the overall system accuracy.

Fig. 9-a shows the rescheduling of *VFE*'s pipeline to make use of parallel processing in hardware. Algorithmically, the image processing parts of the *VFE* can run in parallel. Although the algorithm has some dependency where *FD* waits for *FT* and *RANSAC* to know how many new features are needed to be detected (Fig. 9-a top), this dependency can be broken by pre-detecting features and selecting the relevant features later (Fig. 9-a bottom). This parallelism can be also exploited in software, but firing up multiple cores can significantly increase power consumption, which is already high with one CPU core. The rescheduling results in *VFE*'s processing time saving between 43% to 55% depending on the environment. This comes with a 77kB (10%) memory overhead with line buffers to support the required bandwidth of the parallel components.

Another rescheduling is carried out in *BE*'s pipeline to enable parallelism between *BE* and *VFE*. At *KF*s, *BE* waits for *VFE* and *IFE* to process their outputs. However, *IFE* is much faster and its output is ready earlier than *VFE* as shown in Fig. 9-b. *BE* processing can be rescheduled such that the initialization and all IMU factors linearization start immediately after *IFE*'s output is available. This results in *BE*'s processing time saving between 4% to 19% depending on the scene, with no overhead.

TABLE I
THE SIZE OF DIFFERENT MEMORY BLOCKS IN NAVION, SHOWING NUMBERS BEFORE AND AFTER OPTIMIZATIONS IN SECTION IV.

	Memory Blocks	Size (kB)		Savings
		Before	After	
VFE	Frame Buffers	1410	317.6	4.4×
	Line Buffers	78.2		–
	Tracking Data	18.2		–
	Features Candidates	16		–
	Keyframe Data	11.7		–
IFE	Integrated Data	2		–
BE	Factor Graph	1012	205.6	4.9×
	Linear Solver Data	882.4	163.5	5.4×
	Horizon States	5.2		–
	Shared Memory	36		–
Total		**3471.7**	**854**	**4.1×**

V. IMPLEMENTATION AND RESULTS

A. Chip Implementation Results

Fig. 10 shows the die photo of Navion chip with a summary of the chip specifications and the memory optimization results. Navion is implemented in a 65nm CMOS technology with 2 million NAND2 equivalent logic gate count and 854kB on-chip SRAM. The chip contains two clock domains, one for *VFE* and the other for both *IFE* and *BE*. By using image compression and exploiting structured and unstructured sparsity, an overall 4.1× memory saving is achieved, which enables full *VIO* pipeline integration on-chip. Table I shows different memory sizes before and after optimizations.

Navion can process stereo images with a maximum resolution of 752×480 at a rate of 28–171 fps in real-time across the different sequences in EuRoC dataset [37] used for evaluation; this is referred to as the tracking rate. The chip can also process inertial measurements at up to 52kHz. Navion updates the states and the sparse 3D map at *KF* rate of 16–90 fps, which is the *BE* rate, also depending on the sequence. It consumes an average power consumption of 24mW when operating at 1V. These numbers are measured when Navion's programmable parameters are set to their maximum values;

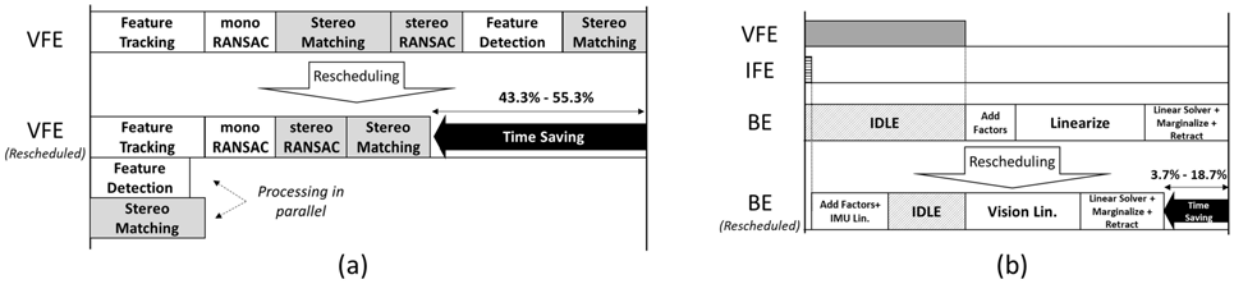
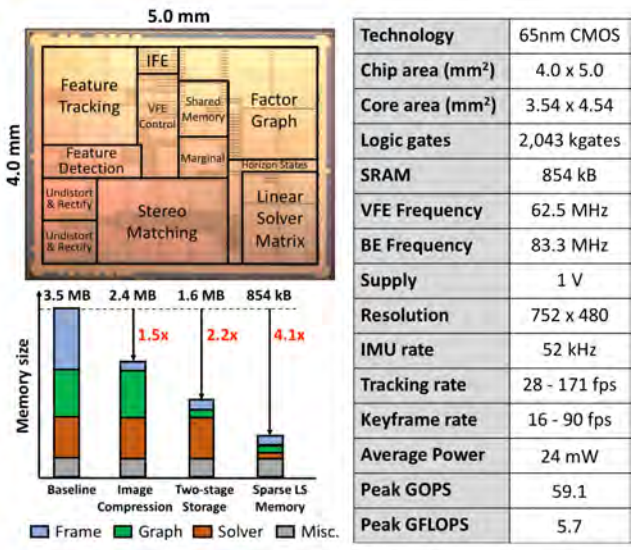


Fig. 9. (a) Rescheduling VFE's pipeline processing KFs. (b) Rescheduling BE's pipeline.



Technology	65nm CMOS
Chip area (mm²)	4.0 x 5.0
Core area (mm²)	3.54 x 4.54
Logic gates	2,043 kgates
SRAM	854 kB
VFE Frequency	62.5 MHz
BE Frequency	83.3 MHz
Supply	1 V
Resolution	752 x 480
IMU rate	52 kHz
Tracking rate	28 - 171 fps
Keyframe rate	16 - 90 fps
Average Power	24 mW
Peak GOPS	59.1
Peak GFLOPS	5.7

Fig. 10. Die photo and summary of the chip specifications.

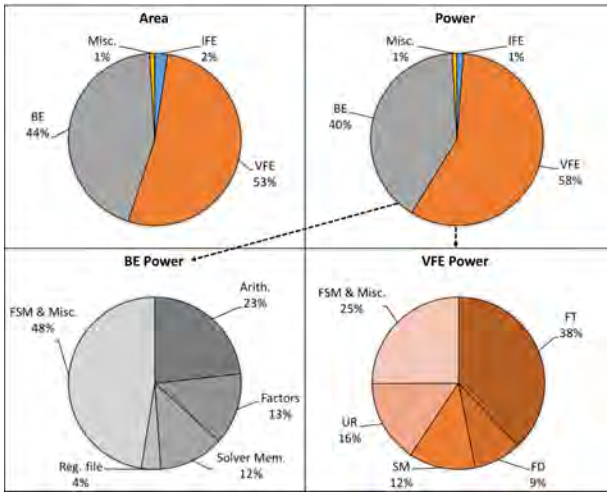


Fig. 11. Area and power breakdown.

TABLE II
MAIN PROGRAMMABLE PARAMETERS WITH THEIR MAXIMUM VALUES.

Parameter		Maximum Value
Frame resolution		752×480
Number of features per frame		200
Feature Tracking	Pyramid (P_{FT})	3 levels
	Cell size (C_{FT})	15× 15 pixels
	Iterations (I_{FT})	30 per level
Stereo	Template size (T_S)	51× 5 pixels
	Search region (R_S)	421× 5 pixels
Backend	Horizon size	20 Keyframes
	Feature age	10 Keyframes
	Feature tracks	4000 tracks

the main parameters are shown in Table II. Changing these parameters values can trade-off accuracy, throughput and power consumption across different environments, as shown in Section V-D.

Fig. 11 shows Navion's area and power breakdowns. IFE has a very low cost from both area and power points of view, while VFE and BE have relatively similar area. The average power consumption of BE is 40% of the total power since it is active only at KFs. It's worth-mentioning that the large on-chip SRAM power is only 21% of the total power consumption, although it accounts for 77% of the chip area. This is a result of dividing the large memories (i.e., frame buffers, vision factors memory, etc.) into small banks, and clock-gating the ones that are not used. This results in 5–9× power savings, depending on the memory size.

Fig. 11 also shows a detailed power breakdown of VFE and BE, where different dataflows result in different power distribution. In VFE, parallel image processing modules (i.e., FT, FD, UR, and SM) consume the majority of the power compared to control and FSM. FT consumes the largest amount of power in VFE because it is always active regardless of the frame type (i.e., KF or not). However, in BE, half of the power is consumed by the control and FSM because of the serial nature of the BE's architecture. Additionally, almost one quarter of BE's power is consumed by the shared arithmetic unit.

Fig. 12 shows a detailed timing breakdown for the different VIO modules, in both non-KF and KF, measured on average over different sequences in EuRoC dataset (see Section V-C). Based on the rescheduling optimization discussed in Section IV-D, FD, UR, and SM modules all run in parallel with FT at KFs, where their processing time is hidden. For BE, the majority of the processing time (65%) is consumed in linearizing the vision factors. Linearizing IMU and the other factors (not shown in Fig. 12) is carried out in parallel with VFE processing, and it takes less than 2 ms to finish. Additionally, the linear solver is relatively fast (i.e., only 25%

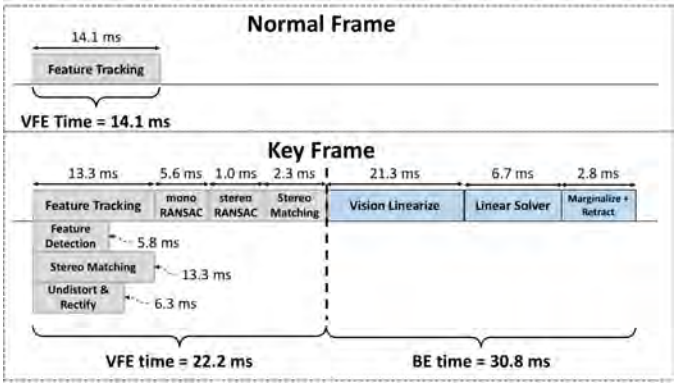


Fig. 12. Navion's timing breakdown at maximum configuration (62.5 MHz VFE clock and 83.3 MHz BE clock). Numbers are averages over EuRoC dataset.

of *BE*'s processing time) as it exploits sparsity by skipping processing of zeros.

B. Demonstration System

To validate the fabricated chip, a demonstration system is developed for real-time localization and mapping as shown in Fig. 13. It is composed of the custom test chip board and a Xilinx ZC-706 evaluation board. The FPGA board is used to stream images and IMU measurements to the chip, and read the output results for verification and visualization. The Xilinx Zynq-7000 FPGA has 2 ARM Cortex-A9 embedded cores. A C API is developed on an Ubuntu operating system in the ARM core to control the dataflow through several buses using AXI protocol. Fig. 13 shows Navion's output trajectory drawn on the monitor in real-time. A video of this demo system can be found on the Navion project website [38].

C. Evaluation Results

To evaluate Navion's accuracy, we use the EuRoC dataset [37] which is one of the most challenging and widely used datasets for UAVs flying indoors. EuRoC dataset contains 11 different sequences, each representing the UAV flying in one of three different rooms: *Machine Hall* (*MH*), *Vicon room 1* (*V*1), and *Vicon room 2* (*V*2). Some of the sequences are quite challenging as they have relatively fast and unstable motion and strong brightness change, which lead to dark and/or blurred images. The sequences are divided into easy, medium and difficult sequences accordingly.

Table III shows the average error, throughput and power numbers of the VIO comparing Navion to software implementations on a desktop Intel Xeon E5-2667 CPU and an embedded ARM Cortex-A15 CPU. The trajectory error is defined as the difference between the VIO output and the ground truth along the whole flight path (Fig. 14). To account for different sequence lengths, the trajectory error is normalized to the flight length. The reported CPU power numbers are the average compute power, not including idle power or the external DRAM. Navion consumes 684× and 1582× less energy than ARM A15 and Intel Xeon CPUs respectively. In

TABLE III
VIO AVERAGE ERROR, THROUGHPUT AND POWER PROCESSING EUROC DATASET.

Platform		Xeon	ARM	Navion
Trajectory error (cm)		16.98		23.25
Trajectory error (%)		0.22%		0.28%
Average tracking throughput (fps)		63	19	71
Average *KF* throughput (fps)	**Frontend**	19	3	45
	Backend	35	6	32
	Total	12	2	19
Average power (W)		27.9	2.4	0.024
Energy (mJ/KF)		3638.6	1573.2	2.3

this experiment, all sequences are processed with the same configuration parameters in both the software implementation and Navion. Due to randomness in *RANSAC*, all accuracy numbers (i.e., trajectory error) are the average of 5 runs per sequence.

Navion's average trajectory error increases by 6.27 cm, over an average flight length of 83 m in EuRoC dataset. Hence, the normalized error increases from 0.22% to 0.28%. This 0.06 percentage point error increase is mainly due to lossy image compression and fixed point arithmetic in VFE. Based on the odometry survey in [39], different algorithms have relative trajectory errors ranging from 0.1 to 2%. Thus, Navion has a relatively low average trajectory error compared to these works. Additionally, Navion achieves three orders of magnitude less energy consumption compared to both Xeon and ARM cores.

Fig. 15 shows the VIO trajectory error detailed for the 11 sequences in EuRoC dataset, in both Xeon/ARM CPU and in Navion. As expected, with slow motion, bright scene, and highly textured regions in the easy sequences (i.e., MH_1, MH_2, V1_1 and V2_1), low trajectory error are achieved in both CPU and Navion. However, the error increases for the blurry, dark, and fast moving medium and difficult sequences. Note that for this experiment, the VIO parameters are set to their maximum settings, in both Navion and the software implementations, for all 11 sequences.

D. Adapting to the environment

The flexibility that Navion presents with its programmable parameters gives an opportunity to trade-off throughput, accuracy and power consumption. This is done by configuring the chip differently based on the environment and/or the UAV movement. Accuracy numbers for each sequence, previously shown in Fig. 15, show a large gap between the trajectory error of easy and difficult sequences when all have the same configuration. Particularly, difficult sequences MH_4, MH_5 and V2_3 have relatively large trajectory error when using the same configuration as other sequences. Additionally, adapting to the environment also means that Navion's workload can change based on the environment, resulting in energy savings.

To show the potential savings that can be achieved with adaptation, we set a target normalized trajectory error of 0.35% for all sequences, and find the optimum configuration for each one independently to meet this target. Table IV shows the main adapted parameters for each sequence and the resulting normalized trajectory error. Small number of features and small

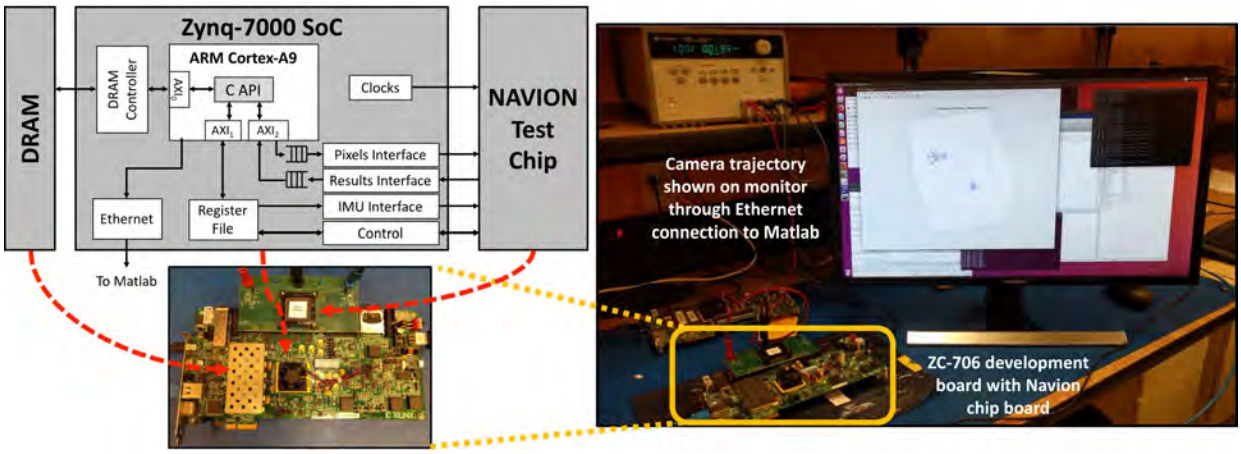


Fig. 13. Demonstration system. ARM core in Zynq-7000 SoC is connected to the on-board DRAM thought AXI0 bus. AXI1 and AXI2 buses are used to connect the ARM core to the control and the interface logic on the FPGA fabric. The two clock signals needed to run the chip are generated on the FPGA.

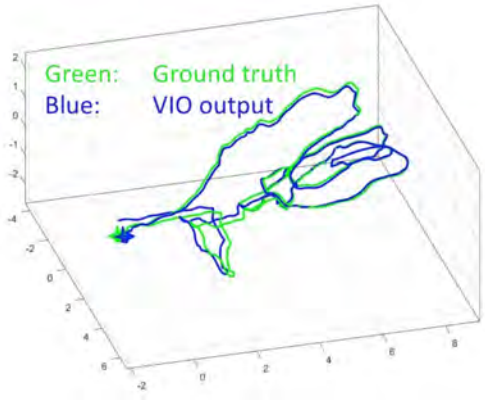


Fig. 14. VIO output example with ground truth, processing a sequence from EuRoC dataset.

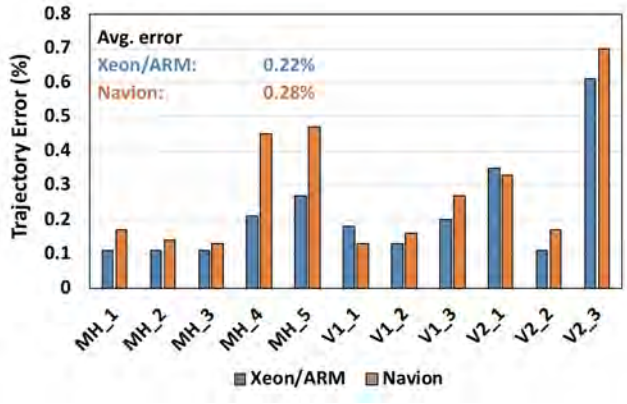


Fig. 15. VIO average error for the 11 sequences in EuRoC dataset. VIO parameters are set to their maximum values for all sequences.

TABLE IV
VIO INDEPENDENT PARAMETERS PER SEQUENCE FOR 0.35% TARGET TRAJECTORY ERROR.

Name	Parameters		Error
	Features/frame	Horizon size	
MH_1	35	10	0.33%
MH_2	35	10	0.22%
MH_3	35	10	0.22%
MH_4	150	10	0.29%
MH_5	100	15	0.30%
V1_1	35	10	0.21%
V1_2	35	10	0.19%
V1_3	35	10	0.33%
V2_1	50	10	0.33%
V2_2	35	10	0.16%
V2_3	50	15	0.34%

horizon size are sufficient in easy sequences because it is easy to track them. This reduces the workload of easy sequences, which reduces the energy consumption. More features and longer horizons are used in difficult sequences to account for the short feature tracks.

Fig. 16 shows a comparison between Navion's measurements with and without adaptation. Fig. 16-a shows the trajectory error for each sequence. Although the average trajectory error remains the same for both cases, the individual error numbers for the easy sequences (i.e., MH_1, MH_2, V1_1 and V2_1) increased slightly but are still within the 0.35% target, and the error for difficult sequences (i.e., MH_4, MH_5 and V2_3) decreased significantly. Fig. 16-b shows the energy consumption for each sequence. The energy numbers for all sequences decreased significantly by an average of 2.5×.

Table V shows the average throughput and energy numbers when adapting Navion for each EuRoC dataset sequence. Different throughput modes are shown such that: 1) Maximum rate means that Navion is running at its maximum frequency for all sequences. 2) Fixed distance means that the throughput of each sequence is set for a constant distance between KFs (5 cm/KF). 3) Fixed rate means that Navion is running at the rate of which EuRoC dataset is captured (i.e, 20 fps). In this third configuration, and with adaptation, Navion can process

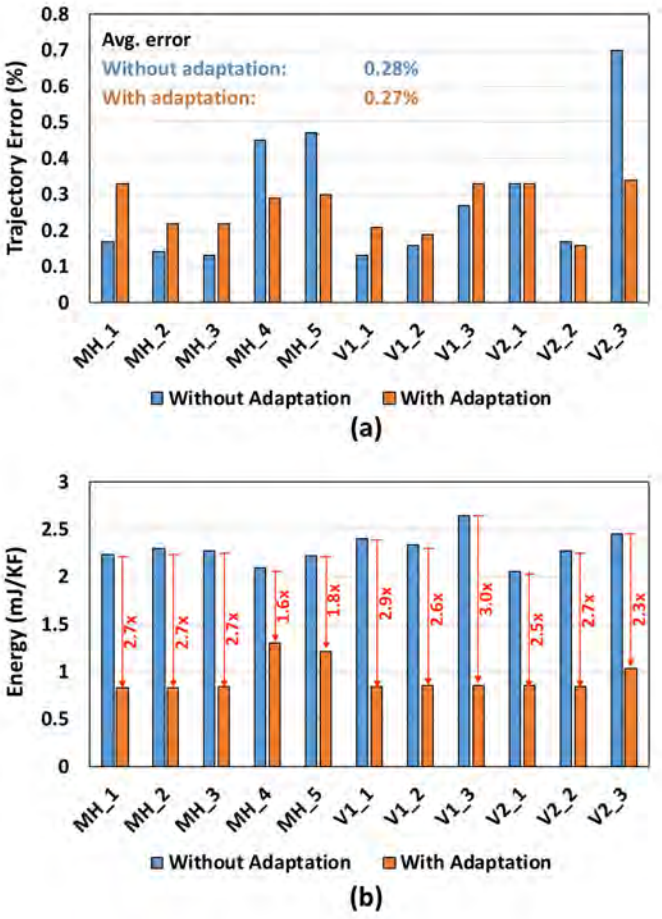


Fig. 16. With and without adaptation comparisons for the 11 sequences in EuRoC dataset. (a) Trajectory error. (b) Energy consumption.

TABLE V
NAVION'S ERROR, THROUGHPUT, CLOCK FREQUENCIES AND POWER NUMBERS FOR DIFFERENT CONFIGURATIONS, WITH AND WITHOUT ADAPTATION. NUMBERS ARE AVERAGES OVER EUROC DATASET.

	Without Adaptation			With Adaptation		
Configuration	Max Rate	Fixed Dist	Fixed Rate	Max Rate	Fixed Dist.	Fixed Rate
Traj. Error	0.28%			0.27%		
VFE Clock (MHz)	62.5	45.7	16.8	62.5	18.6	6.6
BE Clock (MHz)	83.3	60.9	22.5	83.3	24.8	8.8
Tracking (fps)	71	41	20	142	41	20
KF (fps)	19	14	5	49	14	5
Power (mW)	**24**	**18**	**6**	**22**	**7**	**2**

the EuRoC dataset while consuming only an average power of 2mW [40]. This experiment is clearly showing the importance of having a configurable VIO accelerator that can adapt to the environment, not only for an improved accuracy, but also for large energy and power savings. In all these experiments, the power supply was set to 1V. Additional power and energy savings can be achieved by lowering down the supply voltage.

VI. CONCLUSIONS

Scaling down localization and mapping to nano and pico drones/robots requires hardware and algorithm co-design. In this paper, we improve our initial co-designed visual-inertial odometry on FPGA and propose the first fully integrated ASIC solution, Navion. A simple yet effective on-chip image compression technique is developed to reduce the size of the memory. A specialized memory architecture is proposed to efficiently store the mono/stereo feature tracks within the whole horizon. Both structured and unstructured sparsity patterns of the memory for the *BE* are exploited to further reduce the memory and increase the throughput.

Navion is fabricated in a 65nm CMOS technology. Several important parameters can have significant impact on the trade-off in throughput, accuracy and energy efficiency of Navion. These parameters include, for example, keyframe rate, horizon size, and number of feature tracks. They are designed to be programmable, allowing the chip to prioritize different goals under different scenarios. At its peak performance, Navion can process 752×480 stereo image at up to 171 fps and inertial measurements at up to 52 kHz while consuming an average of 24mW at 1V. When configured to process EuRoC dataset at the sensor rate of 20 fps, the chip consumes only an average of 2mW at 1V. Thus, Navion can adapt to handle different environments while being energy efficient and processing in real-time. Additionally, Several blocks in Navion can be easily used in other SLAM/VIO algorithms. This includes IMU preintegration, Linear solver and all of the frontend (i.e., feature detection, feature tracking, etc.), which makes Navion suitable for different applications such as autonomous navigation, mapping, and portable AR/VR.

REFERENCES

[1] F. Nex and F. Remondino, "UAV for 3D mapping applications: a review," *Applied Geomatics*, vol. 6, no. 1, pp. 1–15, 2014.
[2] F. Mohammed, A. Idries, N. Mohamed, K. Al-Jaroodi, and I. Jawhar, "UAVs for smart cities: Opportunities and challenges," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014.
[3] K. P. Valavanis and G. J. Vachtsevanos, "UAV Applications: Introduction," in *Handbook of Unmanned Aerial Vehicles*. Dordrecht: Springer Netherlands, 2015, pp. 2639–2641.
[4] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.
[5] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *Computer Vision – ECCV*, 2014, pp. 834–849.
[6] J. Engel, J. Stckler, and D. Cremers, "Large-scale direct slam with stereo cameras," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 1935–1942.
[7] A. Mourikis and S. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *ICRA*, April 2007, pp. 3565–3572.
[8] G. Sibley, L. Matthies, and G. Sukhatme, "Sliding window filter with application to planetary landing," *JFR*, vol. 27, no. 5, pp. 587–608, 2010.
[9] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2017.
[10] K. P. Valavanis, "Classification of UAVs," in *Handbook of Unmanned Aerial Vehicles*, K. P. Valavanis and G. J. Vachtsevanos, Eds. Dordrecht: Springer Netherlands, 2015.
[11] Skydio, https://www.skydio.com/2018/02/introducing-r1/.
[12] R. J. Wood, B. Finio, M. Karpelson, K. Ma, N. O. Pérez-Arancibia, P. S. Sreetharan, H. Tanaka, and J. P. Whitney, "Progress on 'pico' air vehicles," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1292–1302, 2012.

[13] J. Bonnet, P. Yin, M. E. Ortiz, P. Subsoontorn, and D. Endy, "Controlled Flight of a Biologically Inpsired, Insect-Scale Robot," *Science*, vol. 340, no. 6132, pp. 599–603, 2013.

[14] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2017.

[15] https://www.qualcomm.com/products/snapdragon/processors/410e.

[16] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, 2015.

[17] J. S. Yoon, J. H. Kim, H. E. Kim, W. Y. Lee, S. H. Kim, K. Chung, J. S. Park, and L. S. Kim, "A unified graphics and vision processor with a 0.89 /spl mu/w/fps pose estimation engine for augmented reality," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, pp. 206–216, 2013.

[18] I. Hong, G. Kim, Y. Kim, D. Kim, B. G. Nam, and H. J. Yoo, "A 27 mw reconfigurable marker-less logarithmic camera pose estimation engine for mobile augmented reality processor," *IEEE JSSC*, vol. 50, no. 11, pp. 2513–2523, 2015.

[19] Z. Li, Q. Dong, M. Saligane, B. Kempke, L. Gong, Z. Zhang, R. Dreslinski, D. Sylvester, D. Blaauw, and H. S. Kim, "A 1920 $\times$ 1080 30-frames/s 2.3 tops/w stereo-depth processor for energy-efficient autonomous navigation of micro aerial vehicles," *IEEE JSSC*, vol. 53, no. 1, pp. 76–90, 2018.

[20] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, "The microarchitecture of a real-time robot motion planning accelerator," in *IEEE/ACM MICRO*, 2016, pp. 1–12.

[21] Z. Zhang, A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach," in *RSS*, 2017.

[22] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[23] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *RSS*, 2015.

[24] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *The International Joint Conference on Artificial Intelligence*, vol. 2, 1981, pp. 674–679.

[25] J. Shi and C. Tomasi, "Good features to track," in *IEEE CVPR*, 1994, pp. 593–600.

[26] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[27] L. Kneip, M. Chli, and R. Siegwart, "Robust real-time visual odometry with a single camera and an imu," in *BMVC*, 2011, pp. 1–11.

[28] J. Civera, O. G. Grasa, A. J. Davison, and J. M. M. Montiel, "1-point ransac for ekf-based structure from motion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 3498–3504.

[29] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, , and P. Furgale, "Keyframe-based visual-inertial slam using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, 2015.

[30] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[31] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[32] D. G. Lowe, "Object recognition from local scale-invariant features," in *IEEE International Conference on Computer Vision*, 1999, pp. 1150–1157.

[33] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *IEEE International Conference on Computer Vision*, 2011, pp. 2564–2571.

[34] R. Mur-Artal, J. M. M. Montiel, and J. D. Tards, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[35] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," *Memory*, vol. 10, p. 20, 2018.

[36] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *IEEE/RSJ IROS*, 2015, pp. 298–304.

[37] 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*, 2016.

[38] Navion website, http://navion.mit.edu/.

[39] F. Fraundorfer and D. Scaramuzza, "Visual odometry: Part ii matching, robustness, optimization, and applications," *IEEE Robot. Autom. Mag.*, vol. 19, no. 2, pp. 78–90, 2012.

[40] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: An energy-efficient visual-inertial odometry accelerator for micro robotics and beyond," in *IEEE Hot Chips: A Symposium for High-Performance Chips*, 2018.

Amr Suleiman (S'08–M'18) received the B.S. and M.S. degrees in Electronics and Electrical Communications Engineering from Cairo University, Egypt in 2008 and 2011 respectively. He received the M.S. and Ph.D. degrees in Electrical Engineering and computer science from MIT in 2013 and 2018 respectively. He is currently working as a research scientist in Facebook Reality Labs. Amr's research work focuses on developing new energy-efficient implementations for machine vision algorithms (e.g. detection, recognition, and tracking). Amr is a recipient of the Endowed fellowship of the Arab Republic of Egypt, and the 2015 Broadcom Foundation University Research Competition.

Zhengdong Zhang (S'15) received the B.S. in Computer Science in 2011 from Tsinghua University, Beijing, China. He received the M.S. degree in Computer Science from Massachusetts Institute of Technology, Cambridge in 2014. Between 2011 and 2012 he worked in Microsoft Research Asia, Beijing, China, as an Assistant Researcher. He is pursuing the Ph.D. degree under the supervision of Prof. Vivienne Sze. His research interest spans the area of sparsity, low-rank matrix recovery, symmetry/regularity of textures, 3D computer vision, computational photography and vision systems. His current research focuses on the design of energy-efficient vision systems.

Luca Carlone is the Charles Stark Draper Assistant Professor in the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology, and a Principal Investigator in the Laboratory for Information & Decision Systems (LIDS). He received his PhD from the Polytechnic University of Turin in 2012. He joined LIDS as a postdoctoral associate (2015) and later as a Research Scientist (2016), after spending two years as a postdoctoral fellow at the Georgia Institute of Technology (2013-2015). His research interests include nonlinear estimation, numerical and distributed optimization, and probabilistic inference, applied to sensing, perception, and decision-making in single and multi-robot systems. His work includes seminal results on certifiably-correct algorithms for localization and mapping, as well as approaches for visual-inertial navigation and distributed mapping. He is the recipient of the 2017 Transactions on Robotics King-Sun Fu Memorial Best Paper Award, and the best paper award at WAFR 2016.

Sertac Karaman is an Associate Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology (since Fall 2012). He has obtained B.S. degrees in mechanical engineering and in computer engineering from the Istanbul Technical University, Turkey, in 2007; an S.M. degree in mechanical engineering from MIT in 2009; and a Ph.D. degree in electrical engineering and computer science also from MIT in 2012. His research interests lie in the broad areas of robotics and control theory. In particular, he studies the applications of probability theory, stochastic processes, stochastic geometry, formal methods, and optimization for the design and analysis of high-performance cyber-physical systems. The application areas of his research include driverless cars, unmanned aerial vehicles, distributed aerial surveillance systems, air traffic control, certification and verification of control systems software, and many others. He delivered the Robotics: Science and Systems Early Career Spotlight Talk in 2017. He is the recipient of an IEEE Robotics and Automation Society Early Career Award in 2017, an Office of Naval Research Young Investigator Award in 2017, Army Research Office Young Investigator Award in 2015, National Science Foundation Faculty Career Development (CAREER) Award in 2014, AIAA Wright Brothers Graduate Award in 2012, and an NVIDIA Fellowship in 2011. He serves as a board member for the Robotics: Science and Systems (RSS) Foundation, as the technical area chair for the robotics area for the IEEE Transactions on Aerospace Electronic Systems, and a co-chair of the IEEE Robotics and Automation Society Technical Committee of Algorithms for the Planning and Control of Robot Motion.

Vivienne Sze (S'04–M'10–SM'16) received the B.A.Sc. (Hons) degree in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 2004, and the S.M. and Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2006 and 2010 respectively. In 2011, she received the Jin-Au Kong Outstanding Doctoral Thesis Prize in Electrical Engineering at MIT.

She is an Associate Professor at MIT in the Electrical Engineering and Computer Science Department. Her research interests include energy-aware signal processing algorithms, and low-power circuit and system design for portable multimedia applications including computer vision, deep learning, autonomous navigation, image processing and video coding. Prior to joining MIT, she was a Member of Technical Staff in the Systems and Applications R&D Center at Texas Instruments (TI), Dallas, TX, where she designed low-power algorithms and architectures for video coding. She also represented TI in the JCT-VC committee of ITU-T and ISO/IEC standards body during the development of High Efficiency Video Coding (HEVC), which received a Primetime Emmy Engineering Award. Within the committee, she was the primary coordinator of the core experiment on coefficient scanning and coding, and has chaired/vice-chaired several ad hoc groups on entropy coding. She is a co-editor of High Efficiency Video Coding (HEVC): Algorithms and Architectures (Springer, 2014).

Prof. Sze is a recipient of the 2018 & 2017 Qualcomm Faculty Award, the 2018 Facebook Faculty Award, the 2016 Google Faculty Research Award, the 2016 AFOSR Young Investigator Research Program Award, the 2016 3M Non-Tenured Faculty Award, the 2014 DARPA Young Faculty Award, the 2007 DAC/ISSCC Student Design Contest Award and a co-recipient of the 2017 CICC Outstanding Invited Paper Award, the 2016 IEEE Micro Top Picks Award and the 2008 A-SSCC Outstanding Design Award. Prof. Sze is a Distinguished Lecturer of the IEEE Solid-State Circuits Society (SSCS), and currently serves on SSCS AdCom and the technical program committees for VLSI Symposium, SysML and MICRO.