Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach

Zhengdong Zhang*, Amr Suleiman*, Luca Carlone, Vivienne Sze, Sertac Karaman
Massachusetts Institute of Technology, Cambridge, Massachusetts 02139
Emails: {zhangzd,suleiman,lcarlone,sze,sertac}@mit.edu, **Website: http://navion.mit.edu**
**These authors contributed equally to this work*

***Abstract*—Autonomous navigation of miniaturized robots (e.g., nano/pico aerial vehicles) is currently a grand challenge for robotics research, due to the need for processing a large amount of sensor data (e.g., camera frames) with limited on-board computational resources. In this paper we focus on the design of a *visual-inertial odometry* (VIO) system in which the robot estimates its ego-motion (and a landmark-based map) from on-board camera and IMU data. We argue that scaling down VIO to miniaturized platforms (without sacrificing performance) requires a paradigm shift in the design of perception algorithms, and we advocate a co-design approach in which algorithmic and hardware design choices are tightly coupled. Our contribution is four-fold. First, we discuss the *VIO co-design problem*, in which one tries to attain a desired *resource-performance* trade-off, by making suitable design choices (in terms of hardware, algorithms, implementation, and parameters). Second, we characterize the design space, by discussing how a relevant set of design choices affects the resource-performance trade-off in VIO. Third, we provide a systematic experiment-driven way to explore the design space, towards a design that meets the desired trade-off. Fourth, we demonstrate the result of the co-design process by providing a VIO implementation on specialized hardware and showing that such implementation has the same accuracy and speed of a desktop implementation, while requiring a fraction of the power.**

I. Introduction

Autonomous unmanned aerial vehicles have had tremendous societal and economic impact. They become one of the most sought-after consumer electronics products, selling millions each year. They are widely used in the industry, e.g., for mapping, inspection, surveillance, and film-making [1–3].

In almost all industrial applications of unmanned aerial vehicles (UAVs), it is key for the robot to have an estimate of its own location. In outdoors applications spanning large environments, the global positioning system (GPS) is often able to provide the necessary precision. However, in GPS-denied settings the robot needs to estimate its own ego-motion using exteroceptive sensors, such as, cameras or range finders.

Localization and mapping are widely studied and well known. However, the computing platforms that can run these algorithms in real time can be carried only on relatively large (e.g., one foot in size) *micro UAVs* [4]. Unfortunately, the same computers do not easily fit into the smaller (e.g., palm-size) *nano UAVs* [5, 6]. To the best of our knowledge, the smallest commercially-available UAV that can estimate its own ego-motion is the platform recently announced by Qualcomm, based on their own processors utilized in smart phones. It is a few hundred grams in weight and a few inches in size. It implements a visual-inertial navigation algorithm. Recently, it was demonstrated in agile flight in indoor environments [7].

One may be tempted to predict that, as the consumer electronics industry develops, smaller and more powerful general-purpose processors will be available and smaller UAVs can be powered by these processors. Unfortunately, at the pico UAV scale, the traditional approach of scaling down larger UAVs to smaller ones breaks down, mainly due to the challenges of fitting general-purpose computing elements on board.

Consider, for example, a *pico UAV* that weighs 10g. Such a UAV can be constructed using off-the-shelf actuators and sensors. Its size would be similar to that of a bottle cap.[1] We estimate that it would require roughly 1W of electrical power to lift itself, roughly 1W of power to run three cameras. However, the powerful processors that can run state-of-the-art visual-inertial navigation algorithms require up to 10W of electrical power. The electrical power differences between actuation/sensing and computing are exacerbated as the scale goes down. For example, insect-size pico aerial vehicles that weigh only 100mg have been demonstrated recently [8, 9]; their power expenditure for stable flight is around 100mW [9]. Cameras that can run on 100mW also exist. However, running visual navigation algorithms on existing general-purpose computers requires up to two orders-of-magnitude more electrical power, when compared to the pico-scale actuators and sensors!

Even though actuators and sensors that can enable pico UAV platforms are available today, their computing elements are far from even being understood. It is unlikely that the consumer electronics industry will soon develop processors that can deliver the necessary compute power within the given power budget. Thus, to enable the next-generation nano/pico UAVs, we must design their computers from the ground up, rather than the top-down approach roboticists have enjoyed for decades. In fact, the consumer electronics industry has recently moved in this direction, designing low-power specialized hardware, often implemented as Field Programmable Gate Arrays (FPGAs) [10] or Application-Specific Integrated Circuits (ASICs) [11]. Compute intensive tasks, such as video coding and face recognition in smart phones, are processed using new algorithms that are amenable to FPGA and ASIC [12].

Later in this paper, we review the essential design principles that lead to specialized hardware with low power consumption. The design is far from trivial; it requires careful evaluation of the number of memory and compute elements as well as their spatial allocation on the chip. In most cases, the algorithms

[1]To the best of our knowledge, the smallest existing consumer drone is produced by Cheerstone. The drone weighs 7.7g, and it is 3.5cm in size. Although this particular brand is not equipped with any exteroceptive sensors, we estimate that it is powerful enough to lift two smartphone cameras.

and the hardware are jointly designed to achieve the orders-of-magnitude savings in power expenditure without sacrificing performance. Going forward, it may be critical for roboticists to reconsider most compute-intensive tasks in robotics, and co-design the hardware and the algorithms together to achieve low-power (or high-throughput) implementations. In fact, the potential utilization of ASICs was mentioned in a review article on pico UAVs, which appeared recently in *Nature* [13].

To the best of our knowledge, this paper is the first to take up this challenge, and contribute the first step towards the design of low-power compute elements and accompanying algorithms for robotics applications. The design and fabrication of specialized computing hardware that can finally enable pico UAVs will be a long process. We focus on one of the most compute-intensive processes, namely the task of estimating the ego-motion of the robot with respect to its starting point. The main contributions of this paper are a co-design procedure for the low-power visual-inertial odometry and an FPGA-based evaluation with a specific design obtained via this procedure.

The visual inertial odometry (VIO) literature is vast, including approaches based on filtering [14–19], fixed-lag smoothing [20–24], full smoothing [25–32]. The algorithms considered here are related to IMU preintegration models [30–33].

There are commercial VIO implementations on embedded computing hardware. Most notably, Google Tango and the Qualcomm drone run VIO. Implementations on general-purpose embedded hardware have been recently proposed [34], including a simple low-power implementation mainly for augmented reality applications [35]. However, to the best of our knowledge, there is no specialized hardware implementation of VIO, either using FPGAs or ASICs. Visual feature detection and tracking on FPGAs have been demonstrated only recently [36, 37]. A full FPGA implementation of a sampling-based motion planning algorithm was also considered [38]. All these FPGA implementations target high-throughput applications, where various tasks are parallelized to achieve fast computation. Hence, existing FPGA implementations of robotics-related algorithms do not include full visual inertial odometry, and they do not consider power efficiency as a metric in the design process. Finally, the notion of co-design have been explored in the context of robotics [39, 40] and also control theory [41, 42]. Our work is similar to these in spirit. But, we focus on the co-design of hardware and algorithms.

This paper is structured as follows. Section II reviews algorithmic aspects of VIO. Section III contains a general statement of the VIO co-design problem, in which one is given *high-level specifications*, and has to build a VIO system that meets those specifications. Section IV translates high-level specifications into a desired *performance-resources* trade-off. Section V defines the VIO design space, and Section VI provides a systematic way to explore the design space towards a design that meets the specifications. Finally, Section VII analyzes the result of the VIO co-design: an FPGA-based low-power VIO system with performance matching a desktop implementation while consuming less than 2W of power.

II. VIO: Algorithmic Overview

The present VIO pipeline is based on [32], adopting a factor graph model to perform inference on the robot state, and using

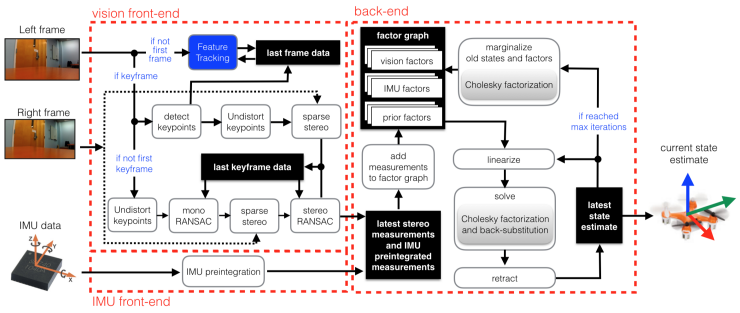

Fig. 1. VIO: algorithmic overview.

preintegrated IMU factors and structureless vision factors to model the data from the inertial measurement unit and the on-board cameras. However, contrary to [32], our approach works with both monocular and stereo camera measurements. The approach can include data from the right camera when available, and falls back to a monocular approach otherwise. Moreover, we perform fixed-lag smoothing, rather than full smoothing: as mentioned in Section VII the memory constraints for our target applications are so tight that even storing the state or the measurement history would be infeasible.

The overall VIO pipeline is shown in Fig. 1. The figure shows the information flow from the sensor data (on the left-hand side) to the VIO state estimate (on the right-hand side). The rectangles with rounded corners perform computation. The black rectangles instead represent memory storage. The blocks are grouped in estimation *front-end* and *back-end*, following standard terminology [43]. The vision front-end processes every stereo frame and is in charge of detecting, tracking, and validating the landmark observations across consecutive frames. The IMU front-end is responsible for the IMU preintegration. The back-end uses the landmark observations and the preintegrated IMU measurements to formulate and solve a maximum a-posteriori estimation problem which computes the best state estimate. Our approach is *keyframe*-based in the sense that the back-end only estimates the states at a subset of frames, named *keyframes*, while between those the state estimate is obtained via IMU integration.

Vision front-end implements four functions: *feature detection*, *feature tracking*, *stereo matching*, *geometric verification*.

The *Feature Detection* extracts N_f Shi-Tomasi corners [44], via OpenCV's *GoodFeaturesToTrack*, from the left image.

The *Feature Tracking* finds the location of the features in the k-th frame, given the corresponding pixels at frame $k-1$. We use optical flow for tracking, following OpenCV's pyramidal Lucas-Kanade method, with T_{levels} pyramidal levels.

Stereo Matching finds the pixels in the right image, corresponding to the pixels detected and tracked in the left image. We rectify and undistort left and right images to make the epipolar lines horizontal. Then, for each feature in the left frame, we consider a neighborhood of the feature of size $S_{rows} \times S_{cols}$, and we look for this template in the right image, by sweeping the template along the epipolar line and looking for the template location that minimizes the mismatch.

The Geometric verification (RANSAC) prunes incorrect matches resulting from feature tracking and stereo matching. We consider a RANSAC approach for geometric validation. In particular, for the temporal matching (feature tracking) we consider two algorithmic alternatives for RANSAC: Nister's 5-

point method [45], and the 2-point method of Kneip *et al.* [46]. The former estimates the 3D pose (up to scale) between consecutive frames, while the latter assumes the rotation to be known from gyroscope integration. For the validation of the stereo matches, we use a standard 3-point method [47].

IMU front-end is responsible for compressing the set of IMU measurements collected between two consecutive keyframes into a single preintegrated measurement and the corresponding covariance. (See [32] for details.) Considering two consecutive keyframes at time i and j, the IMU preintegration performs Euler integration of the IMU acceleration and gyroscope measurements (a_k, ω_k) for all sampling times $k = i, \dots, j$ to produce relative rotation $\Delta\tilde{\boldsymbol{R}}_{ij}$, velocity $\Delta\tilde{\boldsymbol{v}}_{ij}$, and position $\Delta\tilde{\boldsymbol{p}}_{ij}$. The integration is formulated in a local frame, such that, changes in the state estimate at time i do not alter the preintegrated measurements $(\Delta\tilde{\boldsymbol{R}}_{ij}, \Delta\tilde{\boldsymbol{v}}_{ij}, \Delta\tilde{\boldsymbol{p}}_{ij})$.

VIO back-end performs fixed-lag smoothing, and computes the maximum a posteriori (MAP) estimate of the most recent states within a given time window using the measurements produced by the front-end. Denote the state of the robot at time i by $\boldsymbol{x}_i$, which includes the pose of the robot $(\boldsymbol{R}_i, \boldsymbol{p}_i)$, the velocity $\boldsymbol{v}_i$ expressed in the global frame, and the gyroscope and accelerometer bias $\boldsymbol{b}_i$. The back-end then estimates the extended state $\boldsymbol{x} = \{\boldsymbol{x}_{i-h}\ ,\ \boldsymbol{x}_{i-h+1}\ ,\ \dots\ ,\ \boldsymbol{x}_i\}$, where h is the number of states that fall within the smoothing horizon.

The MAP estimator computes the state estimate by solving a nonlinear least squares problem, which assumes the form [32]:

$$\min_{\boldsymbol{x}} \sum_{(i,j)\in\mathcal{F}} \|r_{\text{IMU}}(\boldsymbol{x}, \Delta\tilde{\boldsymbol{R}}_{ij}, \Delta\tilde{\boldsymbol{v}}_{ij}, \Delta\tilde{\boldsymbol{p}}_{ij})\|^2 \quad + \quad (1)$$

$$\sum_{k\in\mathcal{L}} \sum_{i\in\mathcal{F}_k} \|r_{\text{CAM}}(\boldsymbol{x}, l_k, u^l_{ik}, u^r_{ik})\|^2 \quad + \quad (2)$$

$$\|r_{\text{PRIOR}}(\boldsymbol{x})\|^2 \quad (3)$$

where (1), (2), and (3) are the negative log-likelihood of the IMU measurements $(\Delta\tilde{\boldsymbol{R}}_{ij}, \Delta\tilde{\boldsymbol{v}}_{ij}, \Delta\tilde{\boldsymbol{p}}_{ij})$, vision measurements (u^l_{ik}, u^r_{ik}),[2] and the priors, generally referred to as *factors*, since the problem can be understood as the result of performing inference over a *factor graph*. In eqs. (1)-(3), $\mathcal{F}$ is the set of consecutive keyframes indices, $\mathcal{F}_k$ is the set of keyframes in which landmark l_k has been observed, $\mathcal{L}$ is the set of landmarks observed during the time horizon.

We solve the optimization problem in eq.(1) using an on-manifold Gauss-Newton (GN) method [32]. With reference to Fig. 1, at each iteration, the cost is approximated with a quadratic surrogate (*linearization*), and the quadratic approximation is minimized by solving a linear system (*linear solve*). Finally, the solution of the linear system is used to correct the current estimate (*retract*). The sequence linearize-solve-retract is then repeated until convergence or for a maximum number of iterations N_{iter}. The linear system is usually solved via Cholesky factorization, followed by back-substitution [48].

After computing the optimal estimate for the state $\boldsymbol{x}$, the back-end is responsible for marginalizing out the subset of the states that falls outside the smoothing window. The marginalization process requires linearizing the problem, similarly to what is done in GN, and performing a Cholesky factorization.

[2]The observation of landmark $l_k \in \mathbb{R}^3$ in keyframe i produces a stereo measurement (u^l_{ik}, u^r_{ik}), which is the pair of pixels corresponding to the projection of landmark l_k onto the left and right camera frame at time i.

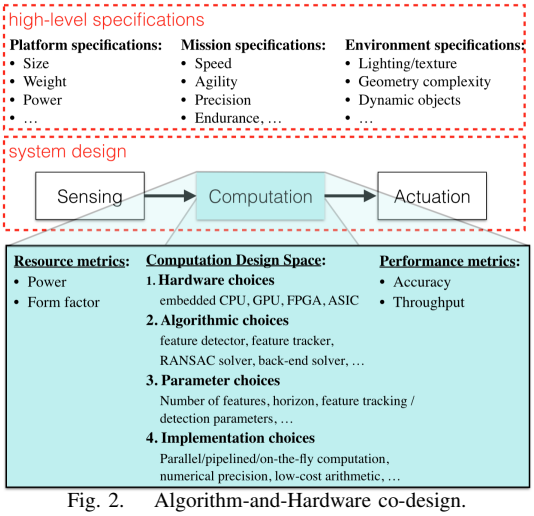

Fig. 2. Algorithm-and-Hardware co-design.

III. Algorithm & Hardware Co-design: Overview

The design process starts with the given high-level *specifications* (top of Fig. 2). We group the specifications in *platform*, *mission*, and *environment* specs. The platform specs are given constraints on the size, weight, and power of the platform on which the VIO needs to be implemented.[3] The mission specs constrain the expected performance towards the accomplishment of a given task. In most applications involving robot navigation, mission specs involve the desired (maximum) speed of the robot, its agility (defined as the capability of producing sudden angular accelerations), and precision in executing a desired maneuver. Finally, the environment specs define the type of environment in which the robot is required to operate, in terms of appearance, geometry, and dynamics.

Given these high-level constraints, the designer is responsible for the design choices regarding on-board sensing, computation, and actuation. We assume that the sensors and actuators are given, and we focus on the "Computation" block in Fig. 2.

The first step of the design process is to translate the high-level specs into constraints on the available resources for computation and on the desired level of performance for VIO. As discussed later in Section IV-A, the most relevant computational resources are the power and the form factor of the computational unit. Given the overall platform power budget, and the power consumed for sensing and actuation, one can get a power budget for the computation. Similarly, it is fairly immediate to translate platform size and weight constraints into constraints on the form factor of the computational unit. On the other hand, mission and environment specifications constrain performance. For instance, speed and agility requirements for the platform have implications on the throughput of the computation, as we discuss in Section IV-B. Resources and performance metrics, summarized in the large box in Fig. 2, are coupled, such that to improve performance one usually needs to use more computational resources.

The second step of the design process is to define a "design space", which is the set of choices the designer can make towards the creation of the VIO system. Each choice in this

[3]Clearly, the set of specifications is application-dependent. For instance, in certain applications, platform specs may include cost or complexity of design.

design space has implications in terms of the performance-resource trade-off. The design space has to be sufficiently large to ensure that there exists a feasible design (that meets the performance requirements by using the available resources). We discuss our choice of the VIO design space in Section V.

The third step is to explore the design space to find a design that attains the desired performance-resources trade-off. Section VI frames design considerations that are usually the result of the experience and judgment of an expert into a systematic sequence of design choices. We note that empirical evaluation is a necessary aspect of the design. This is typically due to the difficulty in taking reasonable assumptions on the appearance of the environment and framing those analytically.

The last step is the validation of the system resulting from the design process. We validate our design in Section VII.

IV. Step 1: From High-level Specifications to the Performance-Resources Trade-off

This section relates the high-level specifications (top of Fig. 2) to more tangible performance and resource metrics.

A. Resources: Power and Form Factor

The two main resources that characterize a VIO system are *power consumption* and *form factor*. They can be broken into factors that can be more easily related to algorithmic choices.

Power. At the hardware level, the power consumption of a computational unit is dictated by four components [49]:

$$\texttt{Power} = \alpha \cdot C \cdot f \cdot V^2 \tag{4}$$

where, C is the *capacitance*, f is the *clock frequency*, V is the *voltage*, and α is the *switching activity*. Fundamentally, power is consumed when charging the capacitance; the frequency and switching activity indicate how often the capacitance is charged, and the voltage indicates how high the capacitance needs to be charged to represent a "1" in binary format. The *voltage* V is dictated by the platform and process technology (e.g., an FPGA in 28nm operates at 1.0V [50]). The *capacitance* C is determined by the number of logic gates (e.g., AND/OR), and the memory size. Large memories that store thousands of bytes will contain long wires with high capacitance; thus, reading and writing from these memories consume more power than computation [51]. If the amount of memory exceeds what is available on-chip, off-chip memory, such as DRAM, needs to be used; however, the capacitance of the routing to go off-chip is orders of magnitude greater than on-chip and thus off-chip memory access consumes significant energy and should be avoided if possible. The *clock frequency* f is determined by how much computation needs to be performed in a given time (i.e., operations per second). A higher clock frequency means that there are more cycles per time period; assuming a fixed number of operations per cycle, this means more operations per second. The *switching activity* α is the probability that a capacitor is charged up to "1" in a given clock cycle (i.e., it captures the fact that in some cycles, certain parts of the hardware could be idle). For instance, a memory that is accessed every cycle will consume more power than a memory than is accessed sporadically.

Form Factor. The maximum size and weight available for the computational unit is another important resource to consider during the design. The form factor of the hardware is closely related to power consumption, due to the fact that high power consumption means that a larger battery is required to maintain the same operation time. In addition, if the power can be reduced to below 1W, the computational unit can operate without a fan, which significantly reduces the form factor.

B. Performance: Accuracy and Throughput

The main performance specifications the designer faces when building a VIO system: estimation error and throughput.

Estimation error. The precision in performing a desired maneuver is limited by both the control and estimation error. Therefore, the mission-level precision specification dictates an upper bound on the VIO estimation error. In this context we are interested in designing a VIO system where the drift is negligible over few minutes of operation. More precisely, our design goal is to attain an average error smaller than $25cm$ accumulated over four minutes of flight time.

Throughput. The throughput measures how fast we can process the sensor data and produce a state estimate. More precisely, since we use a keyframe-based VIO approach, we distinguish two throughput metrics, the *front-end* and the *back-end throughput*. Both throughputs are by high-level mission specifications and by sensor choice. The *front-end throughput* should be high enough to cope with the frame-rate of the on-board stereo camera. On the other hand, the *back-end throughput* should be higher than the *keyframe rate*.

The keyframe rate, in turn, is dictated by the speed and agility of the platform, and has to be high enough to ensure that landmarks are tracked across consecutive keyframes. Assuming that mission-level constraints require a maximum (linear) speed of $v_{\max}$ (in meters per second), we can use standard projective geometry to relate the speed of the robot to the maximum displacement of the pixels between keyframes. Assuming that a landmark is observed at pixel $(0, 0)$ in keyframe i, and that the robot moves at speed $v_{\max}$ for the entire time interval Δt_{ij}, then the (1D) pixel displacement at the successive keyframe j is:

$$\Delta u = (f_l \cdot v_{\max} \cdot \Delta t_{ij})/r \tag{5}$$

where we assume that the robot moves orthogonally to the landmark direction (this is the worst case for the displacement), $v_{\max}\Delta t_{ij}$ is the physical displacement of the landmark over the time interval Δt_{ij} with respect to the camera frame, f_l is the focal length, and the landmark is at a distance of r from the robot. High-level specifications on the geometry of the environment dictate the minimum distance r, while the choice of the sensor fixes the focal length f_l. Therefore, one can compute the maximum intra-keyframe time Δt_{ij} which ensures that each feature does not move more than Δu pixels (or, equivalently, that most features remain in the field of view) across keyframes. A numerical example is given at [52].

One can do similar considerations starting from the agility requirements, evaluating worse-case pixel displacements at maximum rotation rate and accelerations, obtaining another upper bound on the intra-keyframe time.

Table I summarizes our desired performance-resources trade-off, and provides examples of these bounds, assuming that the camera frame rate is 20fps, the keyframe rate is the

		Design goal	High-level specs
Resources	power	$\leq 2W$	power, endurance
	form factor	-	size, weight
Performance	estimation error	$\leq 25cm$	accuracy
	front-end throughput	≥ 20 fps	speed, agility
	back-end throughput	≥ 5 fps	speed, agility

TABLE I
PERFORMANCE-RESOURCES TRADE-OFF SPECIFICATION.

one designed in [52], the available power budget for VIO is $2W$, and we do not enforce explicit form factor constraints.

V. STEP 2: DEFINING THE DESIGN SPACE $\mathcal{D}$

Roboticists are faced with a set of design choices when designing a VIO system. Some of these choices are part of common practice in robotics, while others are less trivial and derive from a more hardware-oriented, ground-up perspective. The set of design choices spans the *design space*: each point in the design space represents a feasible design and implies a performance-resources trade-off. We group the design choices in four: *hardware*, *algorithmic*, *implementation*, and *parameter* choices. Therefore the design space is $\mathcal{D} = \mathcal{H} \times \mathcal{A} \times \mathcal{I} \times \mathcal{P}$, where $\mathcal{H}$, $\mathcal{A}$, $\mathcal{I}$, $\mathcal{P}$ are the set of hardware, algorithmic, parameter and implementation choices, respectively.

A. Hardware Choices ($\mathcal{H}$)

When designing a VIO system, the designer has to select a suitable computational unit. At a high level, the choice is between *general-purpose* (e.g., CPU or GPU) or *specialized hardware* (e.g., FPGA or ASIC). Table A1 in [52] summarizes the features of some representative platforms.

1) General-purpose processors: General-purpose platforms such as CPUs and GPUs have limited design flexibility as the amount of on-chip memory, the clock frequency, and the precision of the computation are basically fixed. The amount of parallelism is limited by the number of cores. GPUs offer more parallelism, but often at the cost of power consumption.

2) Specialized Hardware: FPGAs have more design flexibility, with opportunities to explore parallelism and pipelining, and control the numerical precision and the memory size (more in Section V-C). However, arithmetic operations are either performed using digital signal processing (DSP) hardware, which supports multiplication and addition, or Look-Up Tables (LUT) that emulate logic gates (e.g., AND/OR). Operation outside of those supported by the DSP require iterative approaches which affects both throughput and power. On-chip memory includes Block RAM (BRAM) and flip flops (FF).

Finally, ASICs provide even more design flexibility than FPGAs, as the number of processing elements and on-chip memory size are only constrained by the total area, which dictates the chip cost. Partitioning of the area between memory and compute is entirely up to the designer. In addition, logic gates are directly implemented with transistors rather than using LUTs; thus ASICs tend to consume less power than FPGAs. The major drawbacks of the ASIC is that it can only perform a specific task and the design and fabrication costs are orders of magnitude higher than the other platforms.

Specialized hardware gives extra degrees of freedom for the designer to reach a desired performance-resources trade-off, at the cost of higher design complexity. The order of hardware with increasing design flexibility, reduced power and greater design effort from left-to-right is (e = embedded, d = desktop)

$$\text{d-CPU} \rightarrow \text{e-CPU} \rightarrow \text{e-GPU} \rightarrow \text{FPGA} \rightarrow \text{ASIC} \tag{6}$$

VIO Hardware Choices: in our nano aerial vehicle application, the power budget is severely limited and we restrict our hardware choices to embedded CPU and FPGA. We excluded the ASIC for the moment, despite its low power consumption, due to the complexity of its design. However, the FPGA design considerations are important steps towards ASIC design.

B. Algorithm Choices ($\mathcal{A}$)

When designing a VIO system one is faced with a large number of algorithmic choices. Multiple choices are available for almost any block of the VIO pipeline in Fig. 1, starting from the feature tracking (e.g., use of sparse optical flow or descriptor matching [53]), to the RANSAC solver (e.g., choice of the minimal solver), and the nonlinear optimizer. However, it may be impractical to consider a large set of algorithmic choices: since the selection of the final design has to often be performed through empirical evaluation of different alternatives, a large design space implies a larger burden in terms of implementation and testing.

VIO Algorithmic Choices: in this paper, we restrict the algorithmic choices to the selection of the most convenient RANSAC method in the vision front-end.

C. Implementation Choices ($\mathcal{I}$)

The designer can choose many different ways to implement a given algorithm, and, while this aspect is sometimes overlooked in the design of VIO algorithms, it is of paramount importance towards the design of VIO systems. In the following we collect a number of implementation choices, which we classify in *accuracy-invariant and accuracy-dependent* choices. The former do not alter the numerical result, the latter trade-off throughput and power for accuracy. Most of these choices are only available on specialized hardware.

1) Accuracy-invariant choices: The following design choices have no effect on the accuracy (estimation error).

On-the-fly Computation. Since the power demand is largely influenced by the amount of on-chip memory, the designer can rethink algorithm implementations to minimize on-chip memory storage. In addition, minimizing reading from off-chip memory is also critical to reduce system power. To address these potentially conflicting challenges, an *on-the-fly* data processing approach can be used, where sensor data (e.g., pixels) is read from the sensor and used immediately and to its fullest extend, instead of storing it for later use.

Pipelined and Parallel Computation. Pipelining and parallelism are two architectural approaches that can be used to increase the number of operations per second, and thus increase the throughput. Pipelining involves breaking a computation into multiple stages such that a different piece of data can be processed in each stage simultaneous, as shown in Fig. 3.

Parallelism involves replicating the hardware for the computation, so that multiple operations can be performed at the same time to increase the throughput. However, this increase comes at the cost of increased capacitance since parallelism replicates the hardware (more logic gates and memory need

to be used). Thus parallelism alone has limited impact on reducing power, but is effective for increasing throughput.

2) Accuracy-dependent choices: The following design choices may imply a loss in accuracy (i.e., they increase the estimation error), since they induce numerical approximations.

Reduced Precision. The capacitance and consequently, power consumption, scales with the numeric precision of the computation (i.e., fixed or floating point, and number of bits). Therefore the use of fixed-point arithmetic and reduced bit-width, affects both power consumption and memory storage.

Low Cost Arithmetic. While any form of arithmetic (e.g., addition, multiplication, division, square root functions) has similar complexity on CPU, there is a significant difference in their complexity on FPGA and ASIC. Specifically, addition and multiplication require less resources (e.g., logic gates and cycles) than the other functions. In fact, FPGAs often have dedicated digital signal processing (DSP) hardware that can perform addition and multiplication within one clock cycle. However, for the other functions, iterative approaches must be used which may require multiple cycles. Therefore, an implementation choice is to limit the use of high-cost functions, and approximate those with additions and multiplications.

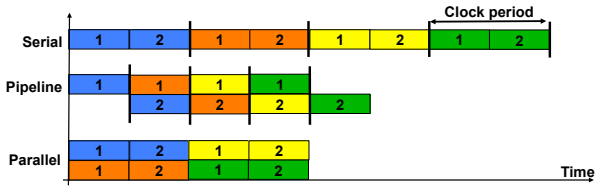

Fig. 3. Example of pipeline and parallelism, assuming that computation requires two steps and colors indicate different sets of data.

D. Parameter Choices ($\mathcal{P}$)

The VIO pipeline (Fig. 1) involves several parameters (typically between 10 and 50). A subset of these parameters has a large impact on the performance-resources trade-off. For instance, the smoothing horizon used in the back-end clearly trades-off computation for accuracy: a longer horizon usually leads to more accurate results and higher computational cost (due to the larger state space in the optimization). There are also *hardware-related* parameter choices. For instance, one may consider the clock-frequency on an FPGA as a tunable parameter to trade-off throughput for power. We denote with $p = \{p_a, p_h\}$ the choice of algorithmic parameters p_a and hardware parameters p_h.

On the other hand, a subset of the parameters has negligible influence on the performance-resource trade-off, and there is no reasons to select those parameters (later referred to as *thresholds*) differently when designing for a high-performance desktop computer or for a small chip. For instance, the thresholds appearing in RANSAC have minor implications on the use of computational resources, and can be designed on any computational unit so to minimize estimation errors.

VIO Parameter Choices: we restrict the parameter choices to the ones listed in Table III.

VI. Step 3: Exploring the Design Space via an Iterative Splitting Co-design Strategy

We provide a systematic way to explore the design space $\mathcal{D}$ towards a design that meets given constraints on the desired performance and on-board resources. Unfortunately, the space is made by a large number of (usually discrete) choices, entailing a *combinatorial explosion of potential designs*. Moreover, the *choices are intertwined*; for instance, an algorithm that on CPU leads to fast computation, may be prohibitively slow or demand too much memory on an FPGA.

Algorithm 1: Iterative Splitting Co-design (ISC)

Input: performance constraints $\bar{\mathrm{E}}, \underline{\mathrm{T}}$, resource constraints $\bar{\mathrm{P}}, \bar{\mathrm{F}}$, design space $\mathcal{D} = (\mathcal{H}, \mathcal{A}, \mathcal{I}, \mathcal{P})$ (hardware, algorithms, implementation, parameters)
Output: design $d \in \mathcal{D}$ (when feasible)

1 **begin**
2 Select the first hardware $h \in \mathcal{H}$, following the ordering in (6), whose power and form factor may be compatible with the available resources (i.e., idle power is below $\bar{\mathrm{P}}$) ;
3 Find a minimum-resources design $d = (h, a, i, p) \in \mathcal{D}$ which meets the given performance specs given h:
4 **select** algorithms a, parameters p_a to minimize power (ops and memory) while preserving the accuracy $\bar{\mathrm{E}}$;
5 **select** implementation i, hardware parameter p_h to re-establish desired throughput $\underline{\mathrm{T}}$, minimizing power ;
6 **if** *d satisfies power and form factor constraints $\bar{P}, \bar{F}$* **then**
7 **return** d;
8 **else**
9 $\mathcal{H} \leftarrow \mathcal{H} \backslash h$;
10 repeat from 2 ;

To address these challenges, we use an *iterative approach*. At each iteration, we commit to a given hardware choice, and we look for a *minimum-resources* design. This minimum-resources design sub-problem is still challenging, since throughput and estimation error may suggest antagonistic actions (e.g., we can reduce the error by increasing the computation, which in turns decreases the throughput). Therefore, we also *split* this search in two steps: the first attempts to minimize resources while preserving the desired estimation error, thought a suitable selection of algorithms a and algorithmic parameter p_a choices. The second tries to re-establish, when possible, the desired throughput by suitable choices of the implementation i and hardware parameters p_h.

With these sequence of steps, the design is guaranteed to be feasible in terms of estimation error, but it may be violating the other constraints (throughput, power, form factor). Therefore, the designer may need to iterate the algorithmic choices or the hardware choices. If we favor design simplicity, we may start iterating the hardware choices in order of simplicity of design, as in eq. (6), and explore "simpler" choices first.

This strategy, that we name *Iterative Splitting Co-design* (ISC), is summarized in Algorithm 1. In the following section we discuss how to solve the minimum-resources subproblems in lines 4 and 5 of Algorithm 1, for VIO co-design.

A. Minimum-Resources Algorithms and Parameters

Since minimizing power also reduces the form factor of the hardware, we can safely focus on minimizing power. It may not be trivial to understand how algorithm and algorithmic parameter choices impact power. Fortunately, eq. (4) rewrites power in terms of quantities that can be more easily related to algorithms. In particular, the choice of algorithms and parameters does not alter the voltage and clock frequency of

RANSAC type	Estimation error [m]	Time [s]	# iters.
5-point	0.153	$7.16 \cdot 10^{-3}$	145
2-point	0.147	$4.79 \cdot 10^{-4}$	16

TABLE II
MONOCULAR RANSAC ALGORITHM CHOICE.

the hardware, hence these values can be considered constant in this section. On the other hand, the capacitance and the switching activity depend on the amount of memory and on the number of operations (ops) required to execute the chosen algorithm. *Therefore, in this section we minimize power, by choosing algorithmic and parameters that minimize ops and memory, while preserving the desired estimation error.*

The following design choices are mostly driven by empirical evaluation, since it is hard (and unreliable) to use theoretical estimation error estimate, due to the difficulty of accounting for the appearance of the environment. Therefore, we assume the availability of a given set of *testing datasets* to verify that the algorithmic choices preserve the target estimation error. Due to our aerial robotics motivation, we use the 11 *EuRoC benchmarking datasets* [54] as testing datasets.

We also assume that a *baseline* design is available; for instance, a desktop VIO implementation that ensures the desired performance, but disregards resource-constraints. This is reasonable since the designer usually starts from a desktop-implementation and then scales down to embedded hardware.

The high-level strategy: starting from the baseline design, we change one algorithm/parameter at a time, and we find the minimum-resources setting that preserves the estimation error.

Algorithm Choices. The main algorithmic choice in our design is the selection of the monocular RANSAC approach. For minimum-resources design, we test both the 5-point and the 2-point RANSAC and we log the corresponding estimation errors. We report the estimation error, averaged over 3 runs and across all the EuRoC datasets, in the second column of Table II. The table shows that the two algorithms have practically identical performance, hence, we can prioritize the choice that minimizes the amount of ops and memory. It is possible to quantify the expected complexity of the two methods analytically. The expert reader may easily realize that the complexity of the 5-point method is definitely larger than the 2-point method, for two reasons: (i) the number of iteration in RANSAC grows with the number of points (last column in Table II, see [55]), and (ii) the complexity per iteration of the 5-point method is remarkably larger [56], compared with the 2-point method. This high-level evaluation is confirmed by empirical evaluation of the execution time (third column in Table II), which, given the hardware platform, is proportional to the ops. Therefore, the selection of the 2-point method minimizes resources, while preserving the desired accuracy.

Algorithmic Parameter Choices. The algorithmic parameter choice is still based on the idea of minimizing ops and memory while preserving the desired accuracy. An explicative example is given in Fig. 4(a): we consider a set of potential alternatives for the maximum number of features detected in the vision front-end, and we plot estimation error (solid blue line) and the computation time (dashed red line). The estimation error is the performance metric we want to preserve, while the computation time is, given the hardware, a proxy for the ops, which is the resource we want to minimize. The error has a non-monotonic trend: it is large when the number of features is too small, then it drops when increasing the number of features, then it tends to rise again (intuitively, when trying to extract more features, we also include more outliers). The key observation is that the error curve has a relatively flat region (100-300 features): this is a region in which the error is relatively invariant to the parameter choice, hence we can reduce resources at a minimum loss of accuracy. Since the computation time increases with the number of features, we should select the smallest number of features that attain the desired accuracy. In this case, a convenient choice can be to use 200 features, that corresponds to the minimum error (selecting 100 features may also be an acceptable choice).

Parameter	Type	Module	Final choice
RANSAC type	algorithm	RANSAC	2-point
nr. of features	parameter (p_a)	Feature detection	200
template size		Stereo matching	41×5px
max. tracking levels		Feature tracking	3
horizon		back-end	4s
intra-keyframe time		back-end	0.2s
nr. GN iterations		back-end	1

TABLE III
FINAL CHOICE OF ALGORITHMS AND PARAMETERS

Another example is the choice of the smoothing horizon in Fig. 4(b). We repeat the same process for each parameter in the design space. The corresponding figures are given in [52]. Table III summarizes the minimum-resources, error-preserving choices of algorithms and parameters.

B. Minimum-Resources Hardware Implementation

Given the algorithm choices, we can now focus on optimizing the implementation and the hardware parameters.

Hardware Implementation. We minimize resources in the hardware implementation by following the guidelines of Section V-C. We apply "accuracy-invariant choices" when possible, while for the "accuracy-dependent choices", we still need to make sure to operate across accuracy-preserving implementations, as done in Section VI-A.

On-the-fly Computation: To reduce the on-chip memory size, we do not store the entire image before doing feature detection, but we rather perform detection on-the-fly, while the image is read pixel by pixel. Depending on the detector "block size", we only store a small set of consecutive rows (i.e., *line buffers*), to evaluate the corner response at a pixel.

Pipelined and Parallel Computation: Various modules in the design are pipelined for increased throughput. For instance, in the feature detection, undistortion, and stereo matching, the computation is per-feature and can be easily pipelined (see Example 2 in [52]). Parallelism is also used to increase the throughput of various computationally heavy blocks. Specifically, both the feature tracking module and the linearize module are composed of two parallel engines.

Reduced Precision: The front-end uses fixed-point precision with limited bit-width. As this may affect the estimation error, we evaluate the performance-resources trade-off in Fig. 4(c). We select 16-bit, which is the minimum-resources precision in the accuracy-invariant region, reducing the number of DSPs by 40% compared to 32-bit with only a 0.019m increase in error.

Low Cost Arithmetic: To avoid the high cost arithmetic operations in the hardware, we use approximations or substitutions. For instance, in the sparse stereo block matching, rather

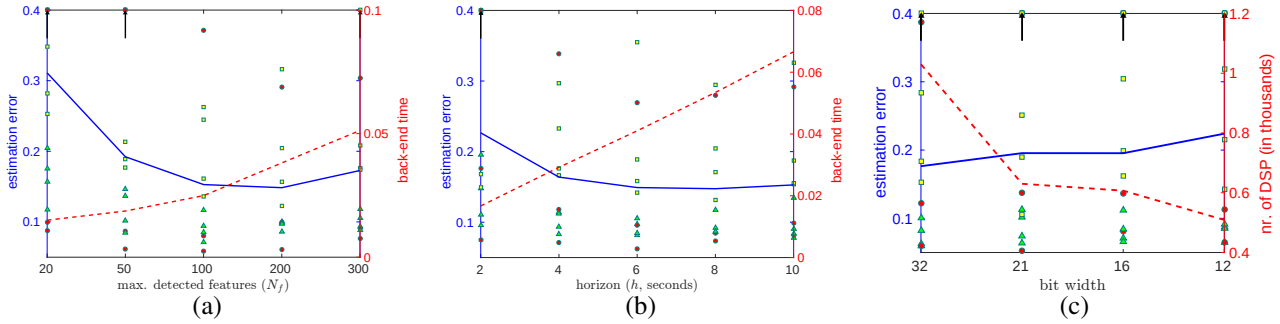

Fig. 4. Algorithmic parameter choice: error-time trade-off for different parameters. Average errors are shown as solid blue lines, while time is show as a dashed red line. The figure also shows the details of each datasets as a scatter plot: the EuRoC datasets are classified as "easy", "medium", and "difficult", and, accordingly, we show the average errors for the easy, medium, and difficult datasets as green triangles, yellow squares, and red circles.

than computing the normalized sum of squared difference and comparing it to a threshold (e.g., $\frac{s}{\sqrt{n}} \leq t$), we multiply the threshold by the normalization and square both sides, (e.g., $s^2 \leq nt^2$). This avoids expensive division and square root.

Hardware Parameters. Given the implementation, the number of ops and memory usage is fixed. The last design "knob" is the choice of the voltage and the clock frequency. Here we assume the voltage to be fixed and we only tune the clock frequency. For a fixed implementation, the throughput and the power increase linearly with the clock frequency. Therefore, we can simply select the minimum-power clock frequency that allows us to meet our throughput constraints.

VII. Step 4: Final VIO Design and Validation

This section provides an example of execution of the Iterative Splitting Co-design. We use the EuRoC dataset for testing [54]. Moreover, we compare against a *baseline* design implemented on a desktop Intel Xeon E5-4627 v2. The hardware choices are embedded ARM Cortex-A15, and an FPGA.

Iteration 1: embedded CPU. At the first iteration, we select the ARM as the target hardware. Then we follow the guidelines of Section VI-A to obtain a minimum-resources choice of algorithms a and parameters p (Table III). Since the ARM does not offer any control on hardware parameter and has a very limited set of implementation choices, we move directly to the validation step (line 6 in Algorithm 1). Table IV summarizes the performance-resources trade-off for the embedded CPU (e-CPU), compared with the baseline design (tested on both the e-CPU and on the Intel Xeon d-CPU). The algorithm and parameter design increases the throughput by 30% at the cost of a 0.014m increase in error. VIO reaches the target throughput of 20 fps and 5 keyframes per second (key fps) on the desktop platform. Unfortunately, even when only a single core is used, the desktop platform consumes 26.1W, which is an order of magnitude greater than our design goal. On the ARM, our algorithm and parameter design consumes 2.3W, which is closer to our target power consumption; however, the throughput is 2-4× lower than our design goal. Since this choice of the hardware leads to an infeasible design, we move to the next iteration.

Iteration 2: FPGA. Profiling the code on the CPU reveals that the following modules account for over 99% of the total compute cycles: front-end, linear solve, linearize and marginalize. To achieve *both* the necessary throughput and power requirements, we implemented the high complexity

Algorithm	Baseline		Design (a,p)		Design (h,a,i,p)
Accuracy	0.1501		0.1641		0.1931
Platform	d-CPU	e-CPU	d-CPU	e-CPU	FPGA
Front-end Throughput (fps)	15.38	3.91	20.83	5.19	20
Back-end Throughput (fps)	8.40	2.04	12.66	2.68	5
Power (W)	28.2	2.45	26.1	2.33	1.46

TABLE IV
Summary of results across different hardware platforms

modules on a Xilinx Kintex-7 XC7K355T FPGA. Since we used specialized hardware, the VIO algorithm can be further optimized using the hardware h and implementation i design choices, as described in Section V. These co-design choices increase the estimation error by only 0.029m.

The rightmost column of Table IV reports the performance-resources trade-off for our FPGA design. The front-end operates at a clock frequency of 23MHz to reach the design goal of 20fps. The back-end operates at a clock frequency of 100MHz to reach the design goal of 5 key fps. The overall power amounts to 1.46W. This design is feasible both in terms of resources and desired performance. In [52] we report extra details on the resource utilization for the key modules. The design requires around 2MB of on-chip storage, 771 DSPs, 144k flip flops and 192k six-input look up tables, which accounts for 32 to 86% of the available resources on the FPGA. The modules within the back-end operate serially. As a result, the marginalize module can time-share the hardware of the solver and the Cholesky factorization block in the linearize block, and only requires additional memory. The front-end can operate up to 70 MHz, increasing the frame rate to 61.2 fps and power consumption by 0.33W.

VIII. Conclusion

In order to scale down perception to nano and pico robots, we need to co-design hardware and algorithms. In this paper we take a first step to address the co-design problem, and propose a systematic, experiment-driven approach to define and explore the co-design space, in the search of a design that meets resources and performance requirements. As a result of the co-design process, we obtain a VIO system which uses specialized hardware (an FPGA) and has the same accuracy and throughput of a desktop implementation, while operating within a power budget of 2W. **Acknowledgements.** This work was partially funded by the AFOSR YIP FA9550-16-1-0228 and by the NSF CAREER 1350685.

References

[1] Francesco Nex and Fabio Remondino. UAV for 3D mapping applications: a review. *Applied Geomatics*, 6 (1):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] Kimon P Valavanis and George J Vachtsevanos. UAV Applications: Introduction. In *Handbook of Unmanned Aerial Vehicles*, pages 2639–2641. Springer Netherlands, Dordrecht, 2015.

[4] Kimon P Valavanis. Classification of UAVs. In Kimon P Valavanis and George J Vachtsevanos, editors, *Handbook of Unmanned Aerial Vehicles*. Springer Netherlands, Dordrecht, 2015.

[5] Matthew Keennon, Karl Klingebiel, and Henry Won. Development of the Nano Hummingbird: A Tailless Flapping Wing Micro Air Vehicle. In *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, Reston, Virigina, June 2012. American Institute of Aeronautics and Astronautics.

[6] Ruijie He, Sho Sato, and Mark Drela. Design of Single-Motor Nano Aerial Vehicle with a Gearless Torque-Canceling Mechanism. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, Reston, Virigina, June 2012. American Institute of Aeronautics and Astronautics.

[7] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU. *IEEE Robotics and Automation Letters*, 2(2):404–411, December 2016.

[8] 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*, 31(11):1292–1302, September 2012.

[9] J Bonnet, P Yin, M E Ortiz, P Subsoontorn, and D Endy. Controlled Flight of a Biologically Inpsired, Insect-Scale Robot. *Science*, 340(6132):599–603, May 2013.

[10] S D Brown, R J Francis, J Rose, and Z G Vranesic. Field-programmable gate arrays. Springer, 2012.

[11] M J S Smith. ***Application-Specific Integrated Circuits***. Addison-Wesley Professional, 2008.

[12] V. Sze, M. Budagavi, and G. J. Sullivan. High efficiency video coding (HEVC):Algorithms and Architectures. *Integrated Circuit and Systems, Springer*, pages 1–375, 2014.

[13] Dario Floreano and Robert J Wood. Science, technology and the future of small autonomous drones. *Nature*, 521 (7553):460–466, May 2015.

[14] A.I. Mourikis and S.I. Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3565–3572, April 2007.

[15] Dimitrios G. Kottas, Joel A. Hesch, Sean L. Bowman, and Stergios I. Roumeliotis. On the consistency of vision-aided inertial navigation. In *Intl. Sym. on Experimental Robotics (ISER)*, 2012.

[16] A.J. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Trans. Pattern Anal. Machine Intell.*, 29(6):1052–1067, Jun 2007.

[17] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct EKF-based approach. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2015.

[18] E.S. Jones and S. Soatto. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *Intl. J. of Robotics Research*, 30(4), Apr 2011.

[19] J.A. Hesch, D.G. Kottas, S.L. Bowman, and S.I. Roumeliotis. Camera-imu-based localization: Observability analysis and consistency improvement. *Intl. J. of Robotics Research*, 33(1):182–201, 2014.

[20] A.I. Mourikis and S.I. Roumeliotis. A dual-layer estimator architecture for long-term localization. In *Proc. of the Workshop on Visual Localization for Mobile Platforms at CVPR*, Anchorage, Alaska, June 2008.

[21] G. Sibley, L. Matthies, and G. Sukhatme. Sliding window filter with application to planetary landing. *J. of Field Robotics*, 27(5):587–608, 2010.

[22] T-C. Dong-Si and A.I. Mourikis. Motion tracking with fixed-lag smoothing: Algorithm consistency and analysis. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

[23] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart. Keyframe-based visual-inertial slam using nonlinear optimization. In *Robotics: Science and Systems (RSS)*, 2013.

[24] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual-inertial slam using nonlinear optimization. *Intl. J. of Robotics Research*, 2015.

[25] M. Bryson, M. Johnson-Roberson, and S. Sukkarieh. Airborne smoothing and mapping using vision and inertial sensors. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3143–3148, 2009.

[26] V. Indelman, S. Wiliams, M. Kaess, and F. Dellaert. Information fusion in navigation systems via factor graph based incremental smoothing. *Robotics and Autonomous Systems*, 61(8):721–738, August 2013.

[27] S. Shen. *Autonomous Navigation in Complex Indoor and Outdoor Environments with Micro Aerial Vehicles*. PhD Thesis, University of Pennsylvania, 2014.

[28] N. Keivan, A. Patron-Perez, and G. Sibley. Asynchronous adaptive conditioning for visual-inertial SLAM. In *Intl. Sym. on Experimental Robotics (ISER)*, 2014.

[29] A. Patron-Perez, S. Lovegrove, and G. Sibley. A spline-based trajectory representation for sensor fusion and rolling shutter cameras. *Intl. J. of Computer Vision*, February 2015.

[30] T. Lupton and S. Sukkarieh. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Trans. Robotics*, 28(1): 61–76, Feb 2012.

[31] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza.

IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems (RSS)*, 2015.

[32] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration theory for fast and accurate visual-inertial navigation. *IEEE Trans. Robotics*, 2016.

[33] K Eckenhoff, P Geneva, and G Huang. High-Accuracy Preintegration for Visual-Inertial Navigation. In *Workshop on Algorithmic Foundations of Robotics*, 2016.

[34] N de Palezieux, T Nageli, and O Hilliges. Duo-VIO: Fast, Light-weight, Stereo Inertial Odometry. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 2016.

[35] I Hong, G Kim, Y Kim, D Kim, and B G Nam. A 27 mW reconfigurable marker-less logarithmic camera pose estimation engine for mobile augmented reality processor. *IEEE Journal of Solid-State Circuits*, 50(11): 2513–2523, 2015.

[36] J Nikolic, J Rehder, M Burri, P Gohl, S Leutenegger, P T Furgale, and R Siegwart. A Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM. In *IEEE International Conference on Robotics and Automation*, 2014.

[37] G Zhou, L Fang, K Tang, and H Zhang. Guidance: A visual sensing platform for robotic applications. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015.

[38] Sean Murray, Will Floyd-Jones, Ying Qi, Daniel Sorin, and George Konidaris. Robot motion planning on a chip. In *Robotics: Science and Systems*, 2016.

[39] A Censi. Handling Uncertainty in Monotone Co-Design Problems. *arXiv*, pages 1–24, September 2016.

[40] A Censi. A Mathematical Theory of Co-Design. *arXiv*, pages 1–18, October 2016.

[41] N Matni. Communication delay co-design in H2 distributed control using atomic norm minimization. *IEEE Transactions on Control of Network Systems*, 2015.

[42] Nikolai Matni and Venkat Chandrasekaran. Regularization for Design. *IEEE Transactions on Automatic Control*, 61(12):3991–4006, November 2016.

[43] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I.D. Reid, and J.J. Leonard. Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6):1309–1332, Dec 2016.

[44] J. Shi and C. Tomasi. Good features to track. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[45] D. Nistér. An efficient solution to the five-point relative pose problem. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[46] L. Kneip, M. Chli, and R. Siegwart. Robust real-time visual odometry with a single camera and an IMU. In *British Machine Vision Conf. (BMVC)*, pages 1–11, 2011.

[47] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Machine Intell.*, 9(5):698 –700, sept. 1987.

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

[49] AP Chandrakasan, S Sheng, and RW Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992.

[50] *7 Series FPGAs Data Sheet: Overview*. Xilinx, December 2016. v2.2.

[51] M. Horowitz. Computing's energy problem (and what we can do about it). In *ISSCC*, 2014.

[52] Supplementary Material on Navion Project Website. http://navion.mit.edu.

[53] D. Scaramuzza and F. Fraundorfer. Visual Odometry: Part I The First 30 Years and Fundamentals. *Robotics Automation Magazine*, 2011.

[54] 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. doi: 10.1177/0278364915620033. URL http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[55] C. Troiani, A. Martinelli, C. Laugier, and D. Scaramuzza. 2-point-based outlier rejection for camera-IMU systems with applications to micro aerial vehicles. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[56] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Anal. Machine Intell.*, 26(6):756–770, 2004.