HW/SW Codesign and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars

George Lentaris, Ioannis Stamoulias, Dimitrios Soudris, and Manolis Lourakis

Abstract— **Future Mars exploration missions rely heavily on high-mobility autonomous rovers equipped with sophisticated scientific instruments and possessing advanced navigational capabilities. Increasing their navigation velocity and localization accuracy is essential for enabling these rovers to explore large areas on Mars. Contemporary Mars rovers move slowly, partially due to the long execution time of complex computer vision algorithms running on their slow space-grade CPUs. This paper exploits the advent of high-performance space-grade field-programmable gate arrays (FPGAs) to accelerate the navigation of future rovers. Specifically, it focuses on visual odometry (VO) and performs HW/SW codesign to achieve one order of magnitude faster execution and improved accuracy. Conforming to the specifications of the European Space Agency, we build a proof-of-concept system on an HW/SW platform with processing power resembling that to be available onboard future rovers. We develop a codesign methodology adapted to the rover's specifications, design parallel architectures, and customize several feature extraction, matching, and motion estimation algorithms. We implement and evaluate five distinct HW/SW pipelines on a Virtex6 FPGA and a 150 MIPS CPU. We provide a detailed analysis of their cost–time–accuracy tradeoffs and quantify the benefits of employing FPGAs for implementing VO. Our solution achieves a speedup factor of 16× over a CPU-only implementation, handling a stereo image pair in less than 1 s, with a 1.25% mean positional error after a 100 m traverse and an FPGA cost of 54 K LUTs and 1.46-MB RAM.**

Index Terms— **Binary robust independent elementary features (BRIEF), egomotion, features from accelerated segment test (FAST), field-programmable gate array (FPGA), Harris, matching, rover navigation, scale-invariant feature transform (SIFT), speeded-up robust features (SURF), stereo vision, visual odometry (VO).**

I. Introduction

COMPUTER vision is constantly gaining ground in space exploration, providing increased flexibility and autonomy to a variety of robotic applications such as spacecraft docking, safe landing, terrain mapping, localization, and navigation. The last objective is of utmost importance to the Mars exploration programs pursued by various space agencies. In this context, a so-called visual odometry (VO) capability can endow a camera-equipped rover with accurate estimates of its location, despite any wheel slippage that renders traditional dead-reckoning ineffective. VO refers to the process of incrementally determining the pose (position and orientation) of a camera rig by analyzing the images it acquires over time [1].

VO on Mars involves a binocular camera capturing stereo image pairs and onboard hardware for processing these images to estimate the six pose parameters of the rover. The process combines various computer vision (CV) algorithms and is repeated at regular time intervals during the rover's movement. The most effective approaches proceed by identifying visual landmarks in the form of natural image features, matching them across images and eventually using their apparent displacement to infer the 3D motion of the rover relative to its environment. Despite its merits, this strategy relies on computationally demanding algorithms for almost every stage of the process. As a result, feature-based VO algorithms require an increased amount of time to complete, especially when executed on space-grade CPUs with limited processing power. Besides, VO suffers from drift caused by error accumulation over time. To overcome these limitations, this paper exploits the advent of high-performance space-grade field-programmable gate arrays (FPGAs) to accelerate prominent CV algorithms so as to improve both the speed and accuracy of rover localization.

FPGAs are already being used in space missions to implement switching tasks, power management, motor and pyrotechnic control, communications, etc. They are rad-hardened devices (from Xilinx, Microsemi/Actel, and Atmel), built with Antifuse, Flash, or SRAM technology, and offer all of their conventional advantages over space-grade ASIC and CPUs. Currently on Mars, the Opportunity Mars exploration rover (MER) and the Curiosity Mars Science Laboratory (MSL) vehicles utilize near-100 FPGA devices. However, these devices are not used for VO. Instead, VO is executed on rad-hard CPUs, i.e., the PowerPC-based RAD6000 and RAD750, which have a processing power of only 22–400 MIPS. As a result [2]–[4], a single VO iteration requires 150 s to complete on MER when executed together with the remaining flight software (or 21 s on its own, whereas MSL's VO is 3–4× faster). Hence, MER seldom uses VO in practice (only in 11% of its traverse, mostly on terrains

Manuscript received December 18, 2014; revised April 2, 2015 and May 15, 2015; accepted June 26, 2015. Date of publication July 2, 2015; date of current version August 5, 2016. This work was supported by the European Space Agency through the SEXTANT Project within the ETP-MREP Research Programme under Grant ESTEC 4000103357/11/NL/EK. This paper was recommended by Associate Editor S. Shirani.

G. Lentaris, I. Stamoulias, and D. Soudris are with the Department of Electrical and Computer Engineering, National Technical University of Athens, Athens 15780, Greece (e-mail: glentaris@microlab.ntua.gr).

M. Lourakis is with the Institute of Computer Science, Foundation for Research and Technology–Hellas, Crete 71110, Greece (e-mail: lourakis@ics.forth.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2015.2452781

with slope or sand drifts), as it limits the maximum average navigation velocity to only 10 m/h. With our work, we achieve to increase the above speed limit by an order of magnitude by offloading the CPU and parallelizing the most demanding algorithms on FPGA. State-of-the-art space-grade FPGAs are now capable of supporting very complex computations due to their increased density, and thus, they enable the HW/SW codesign of VO algorithms for use on high-mobility rovers.

The contribution of this paper is to propose an HW/SW codesign tailored to the localization needs of future Mars rovers, along with HW architectures achieving increased VO accuracy and speed. Specifically, we present our algorithmic exploration, HW/SW partitioning approach, parallelization techniques, FPGA designs, CPU-FPGA system development, and particularly, our customization/tuning of VO to Martian scenarios and demands. Our work involves the acceleration of algorithms that result in high localization accuracy without resorting to mechanical sensor fusion (e.g., inertial measurement unit (IMU)) or other techniques that build on top of VO (e.g., bundle adjustment or simultaneous localization and mapping). That is, we optimize the core of the VO process independently, and assume that any additional refinement can be applied *a posteriori*. Thus, we consider a number of diverse CV algorithms and combine them to develop multiple VO pipelines with various cost–time–accuracy tradeoffs. We provide a complete tradeoff analysis and select the most efficient VO pipeline. Overall, the proposed methodology inputs algorithms and system specifications to produce HW/SW pipelines meeting certain requirements. The requirements are set by the European Space Agency (ESA) in its ongoing research programmes for improving Mars exploration.

ESA specifies certain HW cost, execution time, and accuracy bounds for the developed localization solutions. The VO pipeline should input a stereo image pair and estimate the rover's pose with a 1-s delay at the most. The rover's velocity should be 6 cm/s (distance between two successive images). The positional error of the VO estimates should be less than 2% of the total distance traveled, i.e., a less than 2-m error at a 100-m distance, while the error in attitude/orientation should be less than 5°. The HW should be a space-grade CPU like LEON (a 32-bit embedded processor based on SPARC architecture, from ESA and Gaisler) and a hypothetical space-grade FPGA of near-future technology; as a working hypothesis (after extrapolation of technology trends), the HW specifications are set to 150 MIPS for the CPU and to an FPGA of roughly 100-K LUTs and 2-MB ON-chip RAM, e.g., to a mid-range Xilinx Virtex6. Notice that this heterogeneous HW serves more as an example platform for evaluating the speedup gained when using future FPGAs to assist the slow onboard CPUs (which are seemingly even slower when multitasking), rather than for comparing dissimilar devices. The exact CV algorithms to be used, their HW/SW partitioning, their architecture, as well as their customization were all subjects of investigation. In the two-year long ESA SEXTANT project [5], we developed solutions conforming to the above specifications and demonstrated their effectiveness by utilizing FPGAs and CPUs that emulate space-grade devices. The proposed VO pipelines outperform similar works in the recent literature and quantify the benefits of employing FPGAs to enhance the mobility of future rovers.

The remaining sections are organized as follows: Related previous approaches are briefly reviewed in Section II. Section III presents the system architecture and the CV algorithms being considered. Section IV describes our HW/SW methodology and the HW/SW partitioning of five VO pipelines. Section V describes the HW architecture of our accelerators and their FPGA implementation. Section VI describes the system integration and testing and provides a detailed evaluation of the five VO pipelines. Finally, Section VII concludes this paper.

II. Previous Work

Estimating the camera motion from images is a long-studied problem in CV, often referred to as egomotion, visual motion, or VO to stress its analogy to wheel odometry. Due to several theoretical and practical reasons, the most successful approaches to VO adopt the feature-based paradigm [6], which suggests that VO should proceed by first extracting features, then matching them across images, and finally, estimating sparse 3D structure and egomotion. This is in contrast to the so-called direct methods that are plagued by a limited range of convergence in their quest to simultaneously estimate the camera motion and determine the correspondence of each and every pixel [7].

Related work in the literature includes algorithms and FPGA implementations for various components of feature-based VO, as well as few HW/SW solutions for complete VO pipelines. Relevant algorithms range from feature detection with Harris [8], FAST (features from accelerated segment test) [9], SURF (speeded up robust features) [10], SIFT (scale-invariant feature transform) [11], to feature description with SIFT [11], BRIEF (binary robust independent elementary features) [12], SURF [10], BASIS (basis sparse-coding inspired similarity) [13], to feature matching [11], [14], and to motion estimation techniques [15]–[17]. Completion of feature detection on commercial CPUs (e.g., Pentium4@3 GHz) requires 2–1800 ms, depending on the image size and the complexity of the algorithm [9], [10]. Similarly, description requires 8–355 ms for 512 features per image [18]. Matching is in the order of 100 ms, whereas motion estimation requires only a few milliseconds. A typical example of a feature extraction pipeline executes on a commercial CPU on the order of 1 s for 800×640 images with 1500 features from each [10]. Therefore, the execution time of the most prominent/complex pipelines on space-grade CPUs increases to tens of seconds per frame. FPGAs can accelerate individually each one of their computationally demanding stages to around 7–70 ms, at a cost of 13–65 K LUTs and 0.5–1.2-MB ON-chip RAM [13], [19]–[25]. The algorithms are evaluated with respect to the amount of features detected and correctly matched between images. Their resilience to changes in illumination/scaling/rotation is measured via *repeatability, distinctiveness, correctness*, etc. For moderate image deformations, e.g., 20° viewpoint change, the literature reports a repeatability of 78%–90% and a correctness of 50%–90% for

sets of 100–200 features (for small deformations, the accuracy approaches 99%) [10], [26]. Overall, the feature extraction algorithms show significant cost–performance tradeoffs, which have a direct impact on the entire VO. More insights into these algorithms are provided in the remainder of this paper.

As regards complete VO pipelines on HW/SW platforms, Howard *et al.* [3] and Kostavelis *et al.* [27] propose solutions for Martian rovers, while [28]–[32] target terrestrial applications, either for ground robots or for aerial vehicles. Howard *et al.* [3] implement on FPGA the Harris feature detector and a sum-of-differences (SAD) criterion for matching, whereas they delegate RANSAC and motion estimation to a commercial CPU. Their system utilizes Virtex4 and PC104 to process images of 512×384 pixels and output VO estimates at 5 Hz with rover velocity in the range 5–7 cm/s. [28] implements via high level synthesis tools the SIFT extraction algorithm on a Virtex2 and utilizes a Pentium-M to match features against a database of $2 \cdot 10^5$ landmarks; the system estimates VO at 3 Hz with a rover velocity of 5 cm/s. Kostavelis *et al.* [27] and Krajník *et al.* [29] perform HW/SW codesign based on the SURF algorithm and report VO tests on various 100-m paths. Schmid and Hirschmüller [30] and Schmid *et al.* [31] develop a micro air vehicle by computing stereo on a Spartan6 FPGA and executing VO at more than 14 Hz on a Core2duo CPU, which implements Harris (or AGAST for an almost $10\times$ faster detection) and retrieves the depth of features directly from the stereo output. Bakambu *et al*. [33] report field test results of rover localization in a Mars-like environment. The accuracy of the above HW/SW VO pipelines, i.e., the positional error over the traveled distance, ranges from 2% to 3.3% in Mars-like scenarios and from 2% to 8.15% in terrestrial scenarios (becomes less than 0.5% when using VO in conjunction with external sensors, i.e., an Inertial Measurement Unit [30]). In MER, the reported SW VO accuracy ranges from 1.3% to 2.5% or 4% for 25-m traverses with a 95.5% probability to converge to a final estimate [2]. In MSL, the SW VO has a mean positional error of 2.9% for 100-m traverses [4].

III. System Architecture and Vision Algorithms

The high-level architecture of our system reflects the design of the concept rovers for the 2018 and 2020+ planned Mars missions (e.g., Exomars, MSR). We thus assume a rover equipped with two distinct pairs of cameras: the wide-angle, high-definition navigation cameras, which are mounted on a mast 1m above ground and used mostly for terrain mapping, and the lower resolution localization cameras, which are placed lower on the rover to provide a closer view of the ground and feed the VO pipeline. The localization cameras have a baseline distance of 12 cm, focal lengths of 3.8 mm, and an image resolution of 512×384 with 8-bit pixels each. They are mounted 30 cm above the ground, parallel to each other and in a forward-looking configuration. Compared with a single camera, such a stereo setup resolves the depth/scale ambiguity [34] and permits the estimation of motion even from near-stationary stereo frames. The cameras are tilted by 31.55° with respect to the horizon and their fields of view span 66° horizontally and 49.5° vertically, i.e., they each capture an area in front of the rover that spans from 0.2 m to 2.52 m away (when on level ground). The above parameters play an important role in tuning the VO algorithms and generating realistic synthetic test sequences.

We built a proof-of-concept system consisting of an actual evaluation board with an FPGA and a general purpose PC emulating a LEON CPU of 150 MIPS. The FPGA and the CPU communicate over our custom 100 Mbits/s Ethernet. The localization camera connects to the CPU, together with a hard disk drive for image storage. From a software architecture point of view, the system is structured hierarchically in three levels. Level-0 interacts with the operating system and includes the kernel drivers for the cameras and custom Ethernet as well as the wrappers of imaging, mapping, and localization. Moreover, Level-0 facilitates the interaction with other parts/modules of the rover, e.g., IMU sensor, wheel encoders, and navigation module. Level-1 builds on top of Level-0 and includes the C/C++ functions implementing the CV algorithms. Level-2 includes the most computationally intensive subset of these functions, which are accelerated by the FPGA. The current work focuses on Level-1 and Level-2 and, more specifically, on localization (i.e., VO).

From an algorithmic point of view, feature-based solutions to VO need to address three distinct subproblems, namely feature detection, feature matching, and motion estimation. Feature detection concerns the extraction of sparse point features from a scene, feature matching involves tracking them across successive image frames, and motion estimation concerns the recovery of the incremental pose of the camera(s) as well as the recovery of some partial scene 3D information using structure from motion algorithms [34]. Sparse distinctive points, also called features, interest points or corners in the CV literature, refer to salient image locations with sufficient local variation [26]. Image features correspond to points that can be accurately localized, whereas the local geometric properties of the image patches surrounding them can be captured by suitable feature descriptors, which can be robustly matched from different viewpoints. In the context of our work, several algorithms addressing the above VO subproblems were investigated and are briefly reviewed next.

A. Interest Point Detectors and Descriptors

1) Harris Detector: The Harris corner detector is a popular detector that responds strongly to significant local intensity variations occurring in multiple directions [8]. It employs Gaussian-smoothed products of image derivatives to define the *autocorrelation matrix*, whose eigenvalues capture the principal intensity changes in a point's neighborhood. Various corner strength measures (i.e., *cornerness*) have been proposed to avoid the costly computation of the eigenvalues. The original publication suggests using $S_x^2 S_y^2 - (S_x S_y)^2 - 0.04 \cdot (S_x^2 + S_y^2)^2$, with S_x^2, S_y^2 and $S_x S_y$ being the smoothed products of image derivatives. Corners are detected on pixels whose cornerness is sufficiently high and exceeds that of its neighboring pixels in a 3×3 region (via *nonmaximum suppression*). Subpixel accuracy is achieved by finding the minimum of a

quadratic surface fitted to the cornerness of the 3×3 region around the detected corner. Harris has a high repeatability rate and is widely used in practice. It is not scale invariant; however, this is not an issue for VO as successive images differ only moderately.

2) FAST Detector: FAST operates by considering a circle of pixels around a corner candidate p [9]. Pixel p is classified as a corner if there exists a set of n contiguous pixels in a circle centered on p which are all either brighter than the intensity of the candidate pixel $I(p)$ plus a threshold t, or all darker than $I(p) - t$. The number n is chosen to be equal to 12 because it admits of a high-speed test that can rapidly exclude a very large number of noncorner pixels. FAST is much faster compared with Harris but also more fragile as it employs a very simple criterion.

3) SURF Detector: SURF is a scale- and rotation-invariant interest point detector and descriptor developed with the aim of being accurate and fast to compute [10]. The SURF detector employs an integer approximation of the Hessian matrix that locates interest points at blob-like structures. Computational complexity is kept low by relying on integral images for implementing box-type image convolutions. Specifically, SURF applies box-filters to approximate the second-order partial derivatives Dxx, Dyy, and Dxy, and evaluate the determinant $V_{ijs} = Dxx \cdot Dyy - w \cdot Dxy^2$ with $w = 0.81$, at distinct locations i, j and scales s. Subsequently, the determinant values are nonmaximally suppressed over the resulting scale-space to detect all image blobs. Retained values are interpolated in a $3 \times 3 \times 3$ scale-space cube to increase the detection accuracy. Our experiments use the OpenSURF library [35] configured with three octaves (eight layers) to detect blobs sized approximately up to 33×33 pixels. Overall, SURF offers high repeatability and, compared with SIFT, it is faster and can be implemented with lower HW cost. However, it might suffer from loss of accuracy in certain situations.

4) SIFT Descriptor: SIFT is a combined detector-descriptor [11]. It determines feature locations as the local extrema of a difference of Gaussians function applied in scale space to a series of smoothed and resampled versions of an image. Following this, an orientation is assigned to each SIFT feature by computing the maximum of an orientation histogram weighted by the magnitude of local image gradients. Finally, feature descriptors are determined by rotating the region surrounding each keypoint according to its dominant orientation (a step known as *orientation normalization*) and then dividing it into square patches. A histogram of image gradient directions quantized to a finite set of orientations is extracted from each such patch. The histogram bins are arranged into a 128D vector (the SIFT descriptor). SIFT is a proven descriptor that exhibits good robustness to common transformations and has been successfully employed in a variety of vision-related tasks. Its primary shortcoming is related to its costly computation.

In this paper, the computationally expensive scale-space analysis for determining SIFT feature locations is avoided and only the descriptor part of SIFT is employed. More precisely, features are obtained with the Harris detector described above. For each of these features, a SIFT descriptor capturing the distribution of orientations in a region around the feature is computed as in standard SIFT. In doing so, it is noted that since no scale selection is performed, the scale of the feature is determined arbitrarily (customized to 43×43 in our case). This is acceptable by considering that in the context of VO, successive images are not expected to differ considerably. For the same reason, the orientation normalization step can also be omitted to reduce the computational cost. When the SIFT descriptor is used in conjunction with Harris, the required image gradients can be obtained at no extra cost by reusing the derivatives generated during the computation of cornerness.

5) BRIEF Descriptor: BRIEF is an efficient feature point descriptor based on binary strings extracted from image patches [18]. It is based on performing several pair-wise intensity comparisons on a Gaussian-smoothed image patch and encoding the comparison outcomes using a bit vector. BRIEF compares pixels at a set of random spatial locations and creates a bit vector from the test responses. Despite not being designed to be rotationally invariant, BRIEF can tolerate small amounts of rotation. Compared with SIFT, BRIEF is less discriminating but more compact and faster to compute and match.

B. Interest Point Matching

Prior to estimating 3D motion, the 2D displacement of image points has to be determined by matching the corresponding features between images. Point matching employs a nearest neighbor search among the computed descriptors and proceeds according to the standard distance ratio test [11]. Specifically, for each feature descriptor from one image, its two nearest neighbors from another image are found. The distance to the nearest neighbor is then divided by the distance to the second nearest and the pair is accepted as a match only if the ratio is smaller than a threshold (typically 0.8). Distances among SIFT descriptors are often computed with the Euclidean (L_2) norm. Higher quality matches are obtained via the χ^2 distance [14], which is a histogram metric taking into account that in natural histograms, the differences between large bins is less important than those for small bins. The χ^2 is defined by the sum of ratios of squared bin count differences to bin count sums. This normalization increases the number of correct matches at the cost of some extra computational time or FPGA logic (e.g., 363 LUTs for a 16-bit divider), which we are willing to pay in the most accurate of the VO solutions being explored. BRIEF descriptors, which do not amount to histograms, are compared using the much cheaper Hamming distance which counts their differing bits.

C. Motion Estimation

For the final stage of VO, we explore two prominent egomotion estimation algorithms, namely *absolute orientation* and *LHM*. However, prior to applying either of the two, we employ a preprocessing step to remove mismatched interest points (i.e., outliers) via the robust estimation of the underlying stereo epipolar geometry [34]. Furthermore, the 3D points giving rise to inlying matched point pairs are

reconstructed via triangulation [36]. Triangulation recovers 3D points as the intersections of back-projected rays by the matched image projections and the camera centers. Since there is no guarantee that these rays will actually intersect in space (e.g., they might be skewed), matched image points should be refined prior to triangulation so as to exactly satisfy the estimated epipolar geometry. This is achieved by computing the points on the epipolar lines that are closest to the original ones. The computation involves minimizing the distances of points to epipolar lines with a noniterative scheme that boils down to solving a sixth degree polynomial [36]. Since this is rather costly in terms of computation, we employ a much cheaper alternative relying on the Sampson approximation of the epipolar distance function. In this paper, the operations for eliminating outliers via the epipolar geometry and triangulating the inliers will be collectively referred to as *filtering*.

1) Absolute Orientation: Using the feature detection and matching techniques described above, a number of points are reconstructed from the two stereo views at a certain instant in time t. As the cameras move, detected interest points are tracked over time in both stereo views from time t to $t+1$. By triangulating the tracked points in time $t+1$, the camera motion can be computed as the rigid transformation bringing the 3D points of time t in agreement with those obtained at time $t+1$. Determining the rigid body transformation that aligns two matched 3D point sets is known as the absolute orientation problem and can be solved analytically with the aid of unit quaternions [15]. Since incorrectly matched points cannot be completely avoided in descriptor-based matching, care must be taken to suppress their influence on motion estimation. This is achieved by embedding the estimation in a RANSAC robust regression framework that eliminates mismatches [17].

2) LHM: Lu, Hager, and Mjolsness (LHM) refers to the iterative approach developed by Lu *et al.* [16] to estimate camera pose from 3D to 2D correspondences by operating on the 3D rotation group SO(3). It combines a constraint on the 3D points, effectively incorporating depth, with an optimal update step in the iteration. The LHM algorithm picks a starting rotation and then alternates between translation and rotation refinement until convergence. LHM is one of the most efficient algorithms in the general case and converges from very broad geometrical configurations. It is combined with robust regression by RANSAC to safeguard against mismatched points. In our case, LHM is applied to 3D points that have been reconstructed via triangulation at time t and their 2D counterparts that have been tracked at time $t+1$. Due to its iterative nature, LHM is expected to be less precise compared with absolute orientation.

D. Visual Odometry Pipelines

The above algorithms for feature detection, description, matching, and motion estimation are interchangeable and can be suitably combined to realize several distinct VO pipelines. In this paper, we develop five such pipelines: *Pipe1*, *Pipe2*, *Pipe3*, *Pipe1no*, and *Pipe1c2*. *Pipe1* employs Harris to detect features, SIFT to describe their appearance, χ^2 distance to match them, and absolute orientation to estimate motion. *Pipe2* combines FAST features, BRIEF descriptors, Hamming distance, and LHM motion estimation. *Pipe3* consists of SURF detection, BRIEF description, and absolute orientation. The *Pipe1no* is a variation of *Pipe1* that omits the orientation normalization of the SIFT descriptors. *Pipe1c2* is a hybrid of *Pipe1* and *Pipe2* that consists of Harris, BRIEF, and absolute orientation. These pipelines were coded in optimized *C* and then analyzed using the methodology of Section IV.

IV. HW/SW Codesign Methodology

To determine the most suitable HW/SW solution for VO, we propose an HW/SW codesign methodology to systematically analyze multiple VO pipelines and map their components to HW and SW modules whose combination meets all specifications. These are 1 s per localization step, maximum 2-m positional and 5° attitude error after a 100-m path, running on HW resources limited to a Virtex-6 FPGA and a 150-MIPS CPU.

The proposed design methodology consists of six basic steps:

1) selection of algorithms and datasets;
2) algorithmic analysis and automated SW profiling;
3) partitioning of the pipeline to SW and HW modules;
4) HW architecture design and parametric Very High Speed Integrated Circuit Hardware description language (VHDL) coding;
5) HW/SW system integration;
6) tuning and evaluation (return to three until specs are met).

The first step involves bibliographic search, preliminary complexity-performance estimations, and SW coding. The outcome of this process (already outlined in Section III) facilitates step 2, which takes into account the pipeline structure and dataset runs to assess the corresponding time, accuracy, memory, communication, etc. Step 3 takes into account every platform-dependent constraint/characteristic and the given specifications (i.e., time budget, HW resources, VO error) to determine the components that must be accelerated by HW. Step 4 involves the design of HW architectures tailored to each module/pipeline, as well as their development in parametric VHDL. Parameterizing the HW modules will facilitate the device adaptation and fine-tuning performed at steps 5 and 6. More precisely, at step 5 we select a suitable FPGA board to integrate all HW modules. Moreover, we integrate the entire HW/SW system by developing a custom raw Ethernet communication scheme; the motivation behind the Ethernet customization is to reduce the packet overhead (e.g., use only eight octets/bytes per frame by avoiding the source address and CRC) and provide synchronization at algorithmic level via simple CPU-FPGA handshaking. At step 6, we evaluate the system's performance on various Mars-like datasets, while we methodically tune all parameters striving to optimize both accuracy and execution time. When the resulting HW/SW pipeline meets all given specifications, we successfully complete the development cycle; otherwise, we return to step 3 to refine our partitioning decisions and

optimize the design (in the worst case scenario, after many unsuccessful iterations, one has to return to step 1 and select new algorithms).

A. Algorithmic Analysis and SW Profiling

The SW analysis stage of the proposed methodology is a critical prerequisite for designing efficient HW/SW partitions and HW architectures. Our work relies both on automated SW profilers and on human analysis of the algorithmic complexity/structure. Experience shows that these two approaches are complementary to each other and must be combined in order to derive safe conclusions. Our automated approach involves the widely used Valgrind profiler, which reports detailed information about time and memory usage of every program function, as well as the recording of CPU timestamps at the beginning and end of major algorithmic tasks. The automated results are combined with an in-depth study of the algorithms that aims to enrich and customize the final reports for the given problem and derive platform-independent conclusions. Overall, our analysis includes: 1) the execution time of each function; 2) the amount of platform-independent calculations (e.g., total arithmetic operations); 3) memory utilization; 4) communication (I/O data); 5) parallelization amenability; and 6) type of operations/variables (e.g., memory access patterns, float or integer values, dynamic range). In this paper, we first seek to identify those functions that need to be accelerated by FPGA. Therefore, we are mainly interested in processor-intensive functions performing several arithmetic operations (overburdening the space-grade CPU), mostly doing fixed-point calculations (avoiding floating point units), with relatively low I/O requirements (limiting the CPU-FPGA communication), and amenable to parallelization (efficient acceleration). The remainder of the analysis (e.g., memory utilization, dynamic ranges) becomes useful in later stages for designing the architecture, refining the pipelines, or even for fine-tuning.

The most representative results are shown in Table I, which refers to the processing of one 512 × 384 image with 1000 detected features. The execution time of each function is measured on an Intel Core2-Duo CPU (E8400 at 3.00 GHz) and scaled to the 150 MIPS space-grade CPU with multiplication by 18.4× (a factor derived after benchmarking). Column 3 reports the workload of each function relative to the total pipeline calculations; the percentage varies depending on the detection-description algorithms combined in the pipeline (e.g., Harris-BRIEF or SURF-BRIEF). Columns 4 and 5 report the I/O and memory requirements of each function assuming that it is separated/partitioned from the remaining pipeline. Notice that the I/O of certain subfunctions is greater than the I/O of the main function due to the data generated internally. Also, subfunctions reuse memory whenever possible (e.g., when not partitioned, Harris stores pixels to overwrite them with *cornerness* values reused later by interpolation, SIFT subfunctions overwrite derivatives and reuse gradients; however, matching must store all descriptors and corners without reuse).

The SIFT functions are the most computationally intensive, accounting for up to 88% of the Harris-SIFT pipeline (they

TABLE I
PIPELINE ANALYSIS (TIME PROJECTED TO SPACE-GRADE CPU, % REFERS TO TOTAL PIPELINE TIME, RANGES DUE TO DISTINCT PIPELINES/TESTS)

function	time (ms)	time (%)	I/O (Kb)	RAM (Mb)
Harris detector	886	10.0	1605	19.0
−derivatives	190	2.1	4719	4.7
−main part	686	7.7	8289	19.0
−subpixel	10	0.2	5176	6.5
SURF detector	717	39.0	1605	14.2
−integral image	11	0.6	7864	6.3
−main part	498	27.0	6612	14.2
−interpolation	208	11.3	1637	6.3
FAST detector	39	4	1605	1.9
SIFT descriptor	3478	38.5	5386	26.0
−gradients	110	1.3	15729	15.7
−orientation	175	2.1	12903	12.8
−main part	3193	35.1	14887	23.0
BRIEF descriptor	180	10−17	2149	5.3
SIFT matching	5244	49.0	4416	4.7
−distance calc.	5139	48.0	4224	4.2
−match check	105	1.0	448	0.5
BRIEF matching	755	42−66	1154	1.3
−distance calc.	653	36.3−57	1088	1.0
−match check	102	5.7−9	224	0.3
filtering	350−500	3−13	207	0.4
egomotion abs.or.	6−8	0.04−0.2	173	0.4
egomotion LHM	8	0.3	263	0.5

are executed twice for each stereo pair; however, the spatial matching is an order of magnitude faster than the temporal matching reported in Table I due to exploiting epipolar geometry constraints). Harris, SURF, and BRIEF matching are also too slow to process two images in less than 1 s. Filtering is performed once per localization step and its execution time alone will not violate our 1-s constraint. The increased I/O of certain subfunctions mandates their integration/coupling with each other to avoid increasing the CPU-FPGA communication: e.g., SURF's main part should be coupled with the integral image computation, whereas Harris' main part with subpixel refinement. In other cases, the reduced workload but increased HW cost of certain subfunctions set their HW implementation at low-priority: e.g., the SIFT orientation normalization and SURF interpolation need to process only 1000 corners instead of the entire image, but involve complex trigonometric and algebraic calculations. Finally, we note that the memory requirements of most pipelines exceed the capacity of most FPGA devices (e.g., RAM size 15 Mbits), and hence, our architecture will prioritize solving this serious memory problem.

B. HW/SW Partitioning and Pipeline Scheduling

Based on the aforementioned analysis, the characteristics of our HW platform, and the performance requirements specified by ESA, we partition the VO pipelines as shown in Fig. 1. We present three distinct dataflows and distinguish between SW functions executed on the space-grade CPU (hexagons) and HW functions accelerated by the FPGA (boxes).

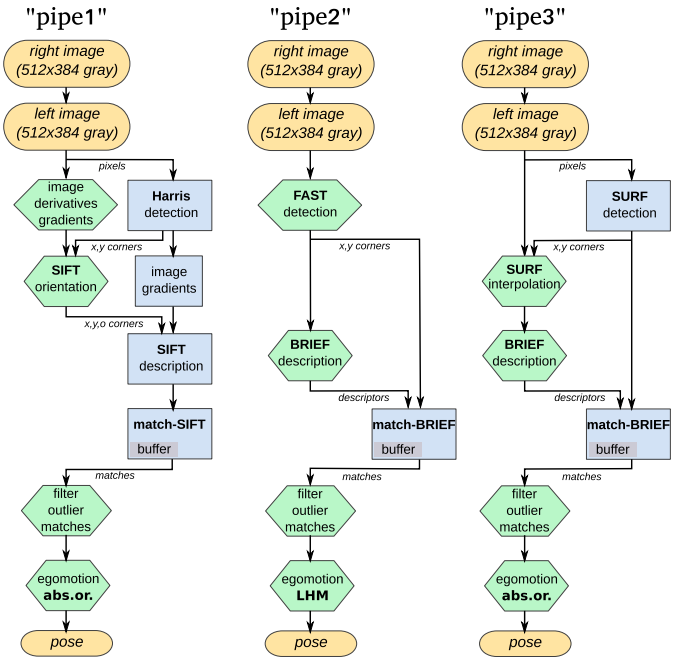

Fig. 1. HW/SW partitioning of three pipelines: hexagon ↦ CPU, box ↦ FPGA.

In *pipe1*, almost all functions relating to feature extraction/matching are executed on HW. The only exception is the orientation normalization of SIFT (including the necessary image gradients calculation), which is executed on SW together with filtering and egomotion. On the FPGA side, the Harris module outputs corners plus image derivatives (to be reused during HW-description), while the HW-match module inputs descriptors plus corners. The CPU-FPGA communication link transfers successively transfers image pixels, corner coordinates, and matching results. The *pipe1no* variation omits the SIFT normalization on SW, and thus, executes the entire feature extraction/matching on FPGA. In *pipe2* we only need to accelerate matching, because FAST and BRIEF are already 20× faster than Harris and SIFT on SW (also, we avoid the transfer of images to the FPGA). In *pipe3*, we accelerate detection and matching. Finally, *pipe1c2* is partitioned to execute Harris on FPGA, BRIEF on SW, matching on FPGA, and egomotion with *abs.or.* on SW. Notice that *pipe1no* and *pipe1c2* are derived from the second iteration of the proposed methodology, where we perform pipeline refinement to meet all ESA specifications. The proposed partitioning lifts the majority of the computations from SW to HW: for 1000 features per image, the *pipe1* HW is exclusively assigned 92% of the total computations, the *pipe1no* HW 96%, the *pipe1c2* HW 79%, the *pipe2* HW 66%, and the *pipe3* HW 69%.

The scheduling of operations is common to all pipelines. At each step of the rover, a new stereo image is input to the CPU; first, we process the left image to detect features (either on CPU or FPGA), second, we describe the features, third, we temporally match the features of the current left image to those of the previous left image (already buffered on HW, Fig. 1), fourth, we store the features of the current left image on HW for use in the two upcoming spatial and temporal matching procedures, fifth, we detect features on the right image, sixth, we describe the new features, seventh, we conduct a spatial matching of the left and right image features, eighth, we filter out the mismatched features, and ninth, we perform motion estimation to obtain the pose of the rover. All pipelines performing detection on HW divide the image in horizontal stripes, which are downloaded and processed successively by the FPGA to reuse space and tackle the memory bottleneck (Section V). In *pipe1* and *pipe1no*, which also do description on HW, each image is downloaded twice to the FPGA (the second for description, with Harris regenerating the image derivatives) increasing the communication overhead but reducing the FPGA memory utilization.

The proposed partitioning strikes the right balance among various design constraints. To begin with, it is critical to accelerate each one of the above functions because, otherwise, the CPU execution time of each function alone would violate the 1 s constraint. Accelerating the remaining SW functions will not add considerable gain to the system, given the available time budget of each localization step. In contrast, extra HW functions would utilize extra FPGA resources, especially ON-chip RAM, which are greatly valued in space-grade devices. Besides, execution on CPU has the advantages of increased floating-point accuracy and easier algorithmic modification. Finally, we avoid further partitioning of the HW functions to avoid increasing the CPU-FPGA communication overhead.

The precision of the arithmetic calculations on HW, i.e., the bits and the representation of each variable, is selected according to the dynamic ranges recorded during profiling. Later, during the tuning phase, we refine the datapath widths to reduce the HW cost by paying attention to preserving the accuracy of VO. Generally, we select fixed-point representations in all HW components except the gradients' computation and descriptor normalization in SIFT, which use floating-point units compatible to IEEE-754. As expected, the finite precision leads to results slightly different from the SW output (e.g., 4% of the low-strength corners not the same and 0.2% of the SIFT descriptors not identical). However, the nature of the CV algorithms allows such minor differences without affecting the final VO outcome. The selected precision is mentioned, among others, in the following section.

V. Proposed Architecture of HW Accelerators and Development on FPGA

A. Harris Corner Detector

The most computationally intensive part of the Harris corner detection algorithm (see Section III-A1) is the 2D convolution generating the *partial image derivatives* and the *Gaussian smoothing* of the squared derivatives. The former uses a 5×5 integer coefficient mask combining the Gauss kernel ($\sigma = 0.9$) with the derivative kernel ($-1, -3, 0, +3, +1$) and is applied twice on a 512×384 image (one for dx and one for dy). The smoothing convolution uses a 7×7 Gauss kernel ($\sigma = 1$) and is performed three times, i.e., once for each squared derivative image, to output S_x^2, S_y^2, and $S_x S_y$. Accelerating these five convolutions is essential to our design approach,

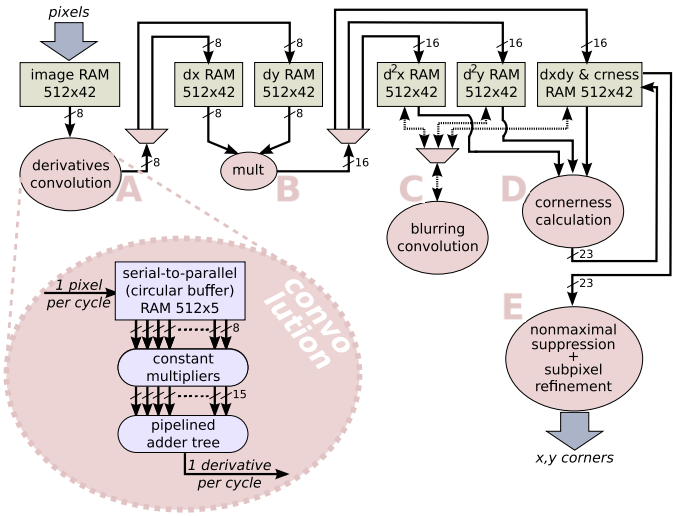

Fig. 2. Proposed architecture of the Harris module with pipelining on pixel basis and parallel arithmetic calculations (A–E denotes the processing order).

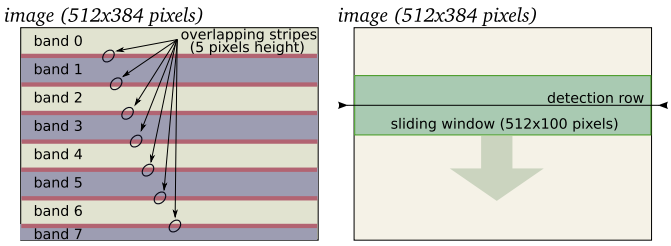

Fig. 3. Division of image in bands (Harris) and sliding window (SURF).

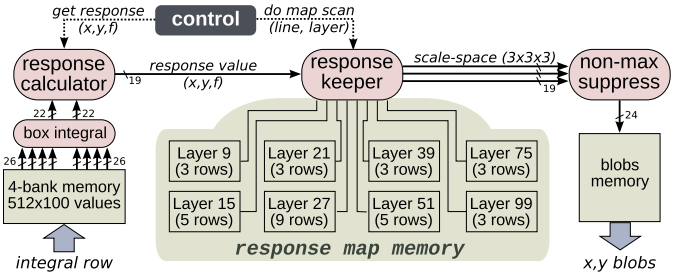

Fig. 4. Proposed architecture of SURF detector based on parallel memories.

which overall relies on fast array transformations and pixel pipelining techniques.

The proposed architecture (Fig. 2) fully parallelizes the multiplication with convolution masks and pipelines the accumulation circuitry to achieve a throughput of one derivative value per cycle. That is, the pixels are forwarded one by one in raster-scan order into a serial-to-parallel buffer, which outputs one 5×5 sliding window per cycle. Each window is multiplied in parallel with the integer coefficient mask using 5×5 constant multipliers and a pipelined adder-tree. Effectively, our architecture completes one 512×384 image convolution in 512×384 cycles (derivatives and blurring use similar convolution components). To accelerate the computation of each pixel's *cornerness* (see Section III-A1), we store the squared derivatives in three parallel memory banks, we parallelize the cornerness formula calculation, and we pipeline the circuit to complete all cornerness values within a burst read of the smoothed squares. Overall, the entire Harris computation is completed in 10 serial steps of 512×384 cycles each (Fig. 2): two convolutions for derivatives (A), three element-wise multiplications (B), three convolutions for blurring (C), calculation of cornerness (D), and finally, (E) nonmaximum suppression of cornerness and subpixel refinement (parabola fitting on a 3×3 cornerness window using a 1-to-3×3 converter similar to that of convolution) to localize corners with increased accuracy.

The proposed architecture achieves an efficient balance of resources and read-cycles (i.e., execution time) tailored to the requirements. On the one hand, the convolution components employ 2D kernels so that a single image/array scan suffices for each convolution (two separate 1D kernels in succession would require two 512×384 scans per convolution). On the other hand, to limit resource utilization, we avoid performing all five convolutions in parallel (e.g., scan the image once and use five distinct convolution components). We also avoid utilizing variable multipliers: we use two dedicated convolution components, each fixed with constant multipliers (for dy, the scan order of the image is reversed so that we can use the same multipliers with dx) and we reuse one variable multiplier for all squares. Our design utilizes intermediate buffers for all derivatives/squares to facilitate: 1) pipelining on an image basis and 2) parallel calculation of each arithmetic formula.

Besides cost–performance balancing, our design tackles the memory bottleneck caused by the increased storage requirements of vision algorithms and the limited RAM of the FPGAs. Specifically, we avoid downloading the entire image on the FPGA by dividing the image in horizontal bands and sequentially processing them [Fig. 3 (left)]. As a result, we decrease the FPGA memory utilization by an order of magnitude. We make the process equivalent to the original SW algorithm by overlapping the bands at their top and bottom rows to allow the convolution kernels to seamlessly slide between successive bands (the overlapping stripes provide a $2 + 3$ pixel margin for the two kernels, of half-size 2 and 3, to slide beyond the band limits). The band's height is configurable allowing adaptation on diverse project requirements (in SEXTANT we set height $= 32$ resulting in the RAM size shown in Fig. 2).

B. SURF Blob Detector

The SURF detector avoids the computationally intensive convolutions of Harris; however, it poses greater implementation challenges with respect to memory. The main reason for this is the increased size of the examined features: effectively, Harris examines a local area of 13×13 pixels to determine whether a location on the image resembles a corner, whereas SURF examines areas of size up to 99×99 pixels to detect scale-invariant blobs (in three octaves). Hence, even though SURF issues considerably fewer memory requests than Harris, the requested locations are far more distant on the image requiring a large part of the image to be cached on the FPGA. To minimize the FPGA RAM, the CPU-FPGA communication, and the pipeline stalls, our design caches on the FPGA a 512×100 sliding window [Fig. 3 (right)]. We restructure the search order of SURF so that at a single iteration, it will scan the entire central row of the cached window to detect any blob

at any octave. At the next iteration, two new image rows are downloaded to the FPGA (overwriting the two oldest rows in a circular buffer) and the window slides downward allowing the next central row to be processed. The iterations continue until all rows are scanned by reusing the same resources.

In the proposed architecture (Fig. 4), we store our sliding window in a four-bank memory interleaved both in the horizontal and the vertical directions of the image plane. This organization allows parallel access to the four value-corners of any required box filter (any size and image location). Thus, in a pipelined fashion, we calculate one box integral per cycle. By using dual port banks, we double the throughput and we compute each pixel's response only in four cycles (involving all approximate second-order partial derivatives Dxx, Dyy, and Dxy). During execution, the control unit requests successively all response values lying on the central row of the cached window. The results are stored in the response map memory, which consists of eight circular buffers (distinct sliding windows), one for each of the eight layers of the three examined octaves. Upon completion of the central row, we scan the map to forward triplets of responses to the nonmaximum suppressor for detecting blobs. The suppressor collects the triplets to form $3 \times 3 \times 3$ scale-space cubes and detect maxima at the center of each cube. Notice that successive cubes overlap each other by $3 \times 3 \times 2$ values. Based on that, and by developing a pipelined comparator tree, we achieve a throughput of one cube examination per three cycles. The acceleration in the proposed architecture comes, mainly, from the parallelization of the integral memory and the response calculation (CPUs would require 1–2 orders of magnitude more cycles), as well as from the parallel reading and pipelining of the scale-space cubes.

C. SIFT Descriptor

The hardware design of our SIFT module focuses on the acceleration of the most computationally intensive part of description, which inputs orientation-normalized detected features and outputs 128-element description vectors. The orientation normalization is performed in SW (only 5% of the total description time), while the input gradient values are computed both in SW and HW, in parallel.

The HW component computing the magnitude of the image gradients $\text{mag} = (dx^2 + dy^2)^{1/2}$ and the orientation of the gradients $\text{orien} = \arctan(dx, dy)$ inputs the values dx and dy already produced in the HW Harris module. We use 32-bit arithmetic to develop a floating-point square-root unit, two fixed-point multipliers, one fixed-point arctan unit, and one fixed-point modulo 2π unit. We fully pipeline the design to generate a magnitude-orientation pair per cycle (Q9.23 and Q5.23 bits) and decrease the CPU-FPGA communication overhead from 1.4 MB to only 0.2 MB (=image size).

The main part of HW SIFT (Fig. 5) utilizes two distinct ON-chip memories to store the aforementioned image gradients and the detected corners. The corners are described iteratively, one by one. At each iteration, an finite state machine (FSM) successively generates 43×43 image

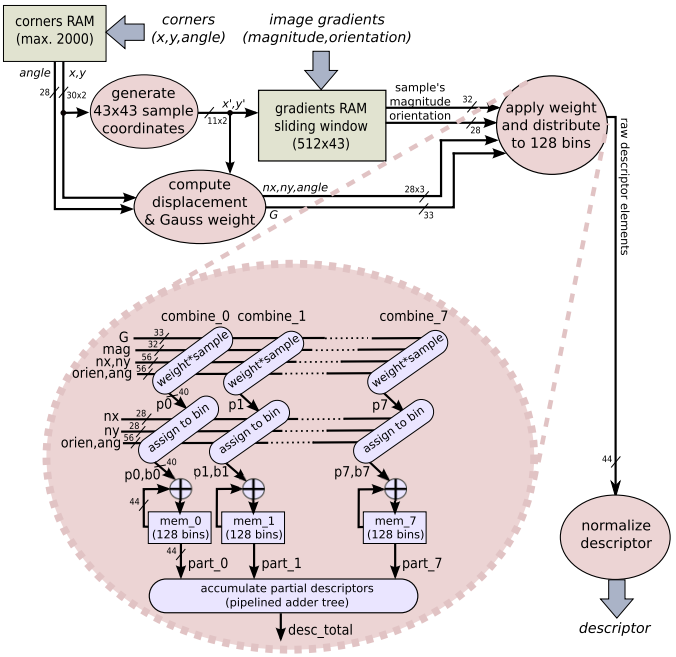

Fig. 5. Proposed architecture of SIFT description with parallel bin loading.

locations to address the samples surrounding a particular corner. For each sample, we calculate its normalized distance from the corner (nx, ny), as well as its Gauss weight $G = e^{-(nx^2+ny^2)/8}$ by interpolating two prestored values from a 256-word LUT. The nx, ny, G are combined with the sample's magnitude and orientation fetched from memory to distribute the sample in eight histogram bins as shown in the distribution component of Fig. 5: first, we apply the Gaussian weight to the sample in eight distinct modes according to SIFT [11]. Second, we assign each one of the eight products to a distinct bin out of 128 bins in total. Third, we accumulate each product to the contents of its assigned bin. Notice that in the proposed architecture, the eight-way weighted sample is distributed to the bins in parallel using eight distinct pipes, which terminate to eight local RAMs. Each local RAM stores a partial description vector with 128 bins. Upon completion of all 43×43 samples, the eight partial vectors are summed to form the 128-element raw descriptor of the corner. The raw descriptor is normalized, truncated at 0.2, and renormalized by custom-developed 32-bit floating-point units to enhance the precision of our results: HW SIFT outputs 99.8% identical descriptors with that of the original SW SIFT.

The proposed architecture is pipelined to sustain a throughput of one sample process per cycle. Moreover, to reduce the increased ON-chip memory requirements of SIFT, it employs a sliding window (similar to that of our SURF detector) to store temporarily only a 512×43 stripe of the image gradients; during execution, we describe all corners lying on the middle row of the stripe and we advance to the next row by sliding the window downward in a circular buffer fashion (notice that each descriptor requires gradients from only a 43×43 window). As a result, we decrease the ON-chip memory requirements by almost $9\times$. In terms

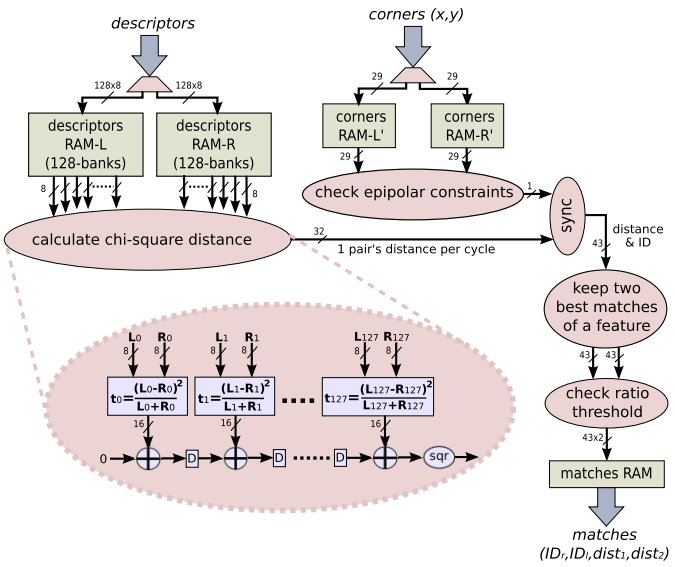

Fig. 6. Proposed architecture of SIFT matching with linear systolic array.

of arithmetic complexity, the most demanding components of the proposed architecture are the distribution, due to its eight parallel pipelines (15 32-bit multipliers in total), and the normalization due to its floating point units (MULT, DIV, sq. root, 2 floating point converters).

D. Feature Matching With SIFT

SIFT matching is the most time-consuming operation in our pipelines, and hence, is implemented entirely on the FPGA. It inputs data from SIFT and Harris, i.e., descriptors and corners, to match features based both on their similarity (χ^2 distance) and their image location (subject to epipolar constraints).

The developed matching module stores temporarily the entire feature set of the image entering the pipeline. The set of corners and descriptors remains cached until the image following the current image has exited the pipeline. Buffering the entire sets of two consecutive images allows pair-wise examination of all features, as well as spatial matching (left versus right image) and temporal matching (left versus previous left image) with minimum CPU-FPGA communication. Our buffer consists of two components, one storing corners and one storing descriptors (Fig. 6). We use two distinct components so that we can examine in parallel the similarity metric and the epipolar validity of any candidate match. Each component consists of two memories, one storing left image data and one storing right image data. Notice that the two memories alternate between left and right image storage depending on which image enters the pipeline: the current left image overwrites the previous right image, while the current right image overwrites the previous left image. Therefore, at any instant, our matching module buffers one stereo pair (left–right image features) or one temporal pair (left–left image features).

After initialization, an FSM fetches all the descriptors stored in memory, one by one, to successively form every feature pair between images. Each descriptor pair is forwarded to the calculate distance component, which performs element-wise comparison of the two vectors to calculate the similarity metric between the two given features (Fig. 6). Specifically, we calculate the χ^2 distance of two 128-bin descriptors R and L by evaluating 128 individual terms, $t_i = (R_i - L_i)^2/(R_i + L_i)$, one for each bin i. To speed up matching, we compute the t_i terms in parallel as follows. We divide the memory of descriptors in 128 banks to store each element of the input descriptor to a distinct bank. The output of each bank is connected to a private t_i calculator utilizing its own multiplier and divider (16-bit). The outputs of the 128 calculators are summed and then squared to produce the final 32-bit distance value. We perform the summation by using the linear systolic array shown in Fig. 6 (for correct operation, the 128 t_i calculators are scheduled to begin with one cycle latency each). The entire structure is pipelined and sustains a throughput of one 128-bin χ^2 distance calculation per cycle.

In parallel to the distance calculator, an FSM fetches the coordinates of the features being examined and compares them against epipolar line limits. It provides a 0/1 output and is synchronized with the distance calculator to validate the match under examination (Fig. 6). A new validation is provided in every cycle until all candidates are examined for a certain feature. During this time window, a monitoring unit compares the produced distance metrics to detect on-the-fly the two best matches of the certain feature. The distance ratio of these two matches is compared with a fixed threshold to accept or reject the best match (by utilizing a 32-bit multiplier and a comparator).

E. Feature Matching With BRIEF and Parallelization Factor

The matching of BRIEF descriptors on HW uses almost the same architecture with SIFT matching (Fig. 6). The only structural differences are due to the length of each descriptor (64 bytes instead of 128) and the similarity metric. The former allows smaller RAM-L and RAM-R memories. The latter allows developing a less complex distance component: we calculate the Hamming distance by replacing the t_i units with $t_i = |L_i \oplus R_i|_1$, where $\oplus$ denotes the bit-wise XOR operator and $|\cdot|_1$ the population count, i.e., the number of 1s in a binary word (implemented with a lookup table). Also, in the linear systolic array, we omit the squaring unit. The remaining parts of Fig. 6 are common for both SIFT and BRIEF matching.

The proposed architecture supports a variable number of t_i calculators, up to 128 for SIFT and up to 64 for BRIEF matching. Section V-D referred to the fully parallel design for ease of presentation. However, in the general case, we equip the systolic array with one accumulator following the adders, which allows us to sum the distance value in multiple iterations. We parameterize the VHDL code with respect to the number of t_i calculators and we refer to this number as parallelization factor f. Accordingly, we divide RAM-L and RAM-R in f memory banks each, and we program the FSM to fetch the descriptor in $128/f$ memory access cycles (or $64/f$ for BRIEF). To avoid under-utilization of our systolic array, we select f to be a power of 2.

The throughput becomes one pair examination every $128/f$ cycles (or $64/f$ for BRIEF). The logic cells of the proposed modular architecture increase almost proportionally with f as the distance calculator becomes far more complex than the remaining components.

F. CPU-FPGA Communication

During HW/SW coprocessing, the data are transferred between the FPGA and CPU over a custom communication scheme based on raw Ethernet packets. On the CPU side, we developed a device driver to exploit the functionality of today's Ethernet network interface cards (NIC) and at the same time to abstract implementation details away from the user space. Specifically, in user's space, the VO algorithm performs read/write operations to a Linux character device file invoking our driver for receiving/sending data to the FPGA. The driver is implemented as a loadable kernel module handling NIC interrupts and providing the communication data to the NIC.

Before exchanging chunks of data (frames of 1500 bytes maximum transmission unit), the HW–SW modules involved perform a handshaking to ensure synchronization at algorithmic level.

On FPGA, an off-the-shelf Ethernet PHY chip performs character encoding, transmission, reception, and decoding. On top of PHY, we employ an Rx/Tx controller (based on the Ethernet MAC IP core from OpenCores) to implement the CSMA/CD LAN in compliance with IEEE 802.3. On top of the Rx/Tx controller, we build a wrapper to provide a Wishbone interface and divide large packets (e.g., image bands) into frames of 1500 bytes. Finally, a custom arbiter is developed between the Rx/Tx wrapper and the various HW accelerators to route traffic with designated packet headers and to cache I/O data.

G. FPGA Implementation and Evaluation

The accelerators described are implemented on a Xilinx Virtex-6 FPGA (XC6VLX240t-2). Table II reports the resource utilization of each accelerator together with the time required to process one 512×384 image (with a clock frequency at 172 MHz). Notice that this time does not include the CPU-FPGA communication (achieved speed is 81 Mbits/s, cost in bottom row) and that description and matching is measured with 900 features per image (with common configuration at 2000 maximum features and $64\times$ parallelization for matching).

The SURF detector (in row 3) consumes considerably fewer resources compared with Harris: $5\times$ fewer FPGA slices with almost 40% less time and memory (Harris can be configured to use less memory at the cost of 10%–30% more execution time depending on band height). This result is primarily due to the nature of SURF, which was designed to decrease the complexity of convolution-based detectors via its use of integral images and box filters. Also, our SURF detector avoids doing interpolation on FPGA (for Harris, the subpixel component accounts for one-third of all Harris logic resources). The increased logic of the SIFT descriptor (in row 4) is due to the gradients' computation (7-K LUTs), the complexity of the 128-bin vector calculation (e.g., 60 DSPs for parallel multiplications), and the final floating-point normalization (5-K LUTs). The most expensive kernel is SIFT matching (in row 5) due to the $f = 64$ elements calculating the distance metric in parallel, which require multiplications and divisions. However, the logic of SIFT matching can be significantly reduced at compile time by decreasing the parallelization factor (e.g., to 5 K slices and 56 DSPs for $f = 16\times$) and increasing the execution time (inversely proportional to f). BRIEF is less expensive due to the smaller description vector and simplicity of the distance metric (no multiplications/divisions). For communication, most of the FPGA logic is consumed by the Ethernet MAC IP core (3-K LUTs) and most of the memory by the arbiter (48 RAMBs).

TABLE II
KERNELS IMPLEMENTED ON FPGA (XC6VLX240t-2, 172 MHz)

kernel	LUTs	DSPs	RAMB36	slices	msec
Harris-d	11477 (08%)	10 (01%)	82 (20%)	4045	14
SURF-d	1646 (01%)	4 (01%)	54 (13%)	788	8
SIFT-d	22814 (15%)	100 (13%)	101 (24%)	7804	25
SIFT-m	42662 (28%)	104 (14%)	142 (34%)	13652	10
BRIEF-m	16695 (11%)	37 (05%)	116 (28%)	6092	5
comm.	3889 (03%)	0 (00%)	52 (13%)	1662	-

Compared with other works in the literature, the accelerators proposed in this paper prove to be particularly cost efficient and, in some cases, make reasonable compromises in terms of time. Our Harris detector consumes less than half the FPGA logic of the component included in [3] (10 K slices) to operate at roughly half the speed, but accounting for only 1% of the 1-s budget of SEXTANT (a negligible compromise). Our SURF detector utilizes almost half the LUTs of the component presented in [21] (2.5-K LUTs excluding their interpolation and integral units) to process the image in double speed. Moreover, our integral unit is more efficient in performing on-the-fly calculation of 1 integral value per cycle (inputs 1 pixel per cycle from the communication module) to feed the SURF detector by utilizing only 1 RAMB and 170 LUTs. Our SIFT descriptor consumes $10\times$ less memory than [25] to process the same amount of features in almost the same amount of time (assuming a 100-MHz clock frequency), whereas it consumes $3\times$ fewer LUTs compared with [19] at the cost of 18 ms more time (small considering our 1-s budget). The proposed SIFT matching with parallelization $= 64\times$ processes 4000 features per image in approximately 39% less time than the matching module of [20] with 42% fewer LUTs (even though our χ^2 distance requires 128 extra divisions).

The developed accelerators achieve significant speedup compared with a space-grade CPU of 150 MIPS. Specifically, the Harris detector achieves a speedup of $63\times$, the SURF detector $62\times$, the SIFT descriptor $100\times$, and the BRIEF matching $125\times$, while the SIFT matching achieves a speedup of up to $425\times$ depending on image content (for parallelization $= 64\times$). Such speedups are essential for

Fig. 7. Sample frames from three 100-m long test sequences depicting synthetic Martian surface (left, middle) and Atacama desert landscape (right).

TABLE III
COMPLETE EVALUATION OF FIVE PIPELINES
(ON FPGA + SPACE-GRADE CPU)

pipeline	basic components	FPGA cost (LUT−RAM)	time (step)	accuracy (at 100m)
pipe1	Harris-SIFT-abs.or.	54% − 90%	1.7sec	1.60m
pipe1no	Harris-SIFT′-abs.or.	36% − 78%	1.0sec	1.25m
pipe2	FAST-BRIEF-LHM	14% − 72%	0.8sec	4.50m
pipe1c2	Harris-BRIEF-abs.or.	21% − 81%	1.0sec	1.65m
pipe3	SURF-BRIEF-abs.or.	15% − 75%	1.1sec	2.83m

Cost = XC6VLX240t utilization (150,720 LUTs and 416 RAMB36 in total)
Time = seconds per stereo image process (1 localization step on HW/SW)
Accuracy = positional error after 100m paths (averaged on synthetic tests)

implementing VO algorithms that meet the specifications of future high-mobility rovers.

VI. SYSTEM INTEGRATION AND EVALUATION

The proposed HW/SW system is integrated in a modular fashion allowing a variety of HW and SW components to be selected at compile time and assemble distinct VO pipelines. The HW components are placed on the Xilinx Virtex-6 FPGA (XC6VLX240t-2 at 172 MHz), while the SW components are executed on the Intel Core2-Duo CPU (E8400 at 3.00 GHz).[1] The SW execution time is scaled by a factor of 18.4× (derived from benchmarking) to emulate execution on a space-grade CPU with only 150 MIPS processing power [5]. The components are selected from our collection of SW *C* functions (Section III) and VHDL functions (Section V). The current section evaluates and compares the five pipelines considered in Section IV with respect to their HW cost, execution time, and VO accuracy.

To evaluate the pipelines in Mars-like scenarios, we use two distinct synthetic sequences depicting a Mars-like environment and one real sequence captured from a Mars analog terrain in the Atacama desert [37], all with 512 × 384 resolution (Fig. 7). The synthetic sequences include 1667 stereo images recording rover trajectories with 6 cm per step. The Atacama sequence consists of 1999 stereo frames, for 334 of which the rover is stationary. The first 1000 frames correspond to a forward rover motion, whereas the remaining 999 are the forward ones in reverse order. Thus, the Atacama sequence simulates a motion where the rover ends up where it started. In all cases, the trajectories have arbitrary shape and are 100 m long each. Following the ESA specifications, the images are characterized visually by diffuse lighting and low contrast and have an unpredictable mixture of fine grained sand, rock outcrops, and surface rocks. In addition, as a stress-test, we use the Devon island sequences [38] in which a pushcart moves at about 19 cm/s (much faster than the Mars rovers) on quite rough terrain. We perform extensive tests to measure the positional error and fine-tune parameters regarding the number of detected corners, the amount of matched features, the width of the HW datapaths, the HW parallelization factor, and the maximum size of the HW buffers. Our parametric VHDL facilitates this optimization phase, which ultimately leads to a system configuration with reduced HW cost, sufficient execution time, and increased VO accuracy. We note that due to the robustness of each pipeline (especially those based on Harris), its final configuration remains the same in all of the test sequences. Table III summarizes the results and the following paragraphs analyze them.

[1]Video available online at: users.uoa.gr/~glentaris/video/tcsvt2015.html.

TABLE IV
TIME ANALYSIS OF A LOCALIZATION STEP
(ONE STEREO PAIR OF ATACAMA)

pipeline	features per 512x384 image	execution time per function (msec)						
		detection (×2)	sw-support (×2)	description (×2)	matching (×2)	filtering	egomotion	Eth.comm.
pipe1	800	$\widetilde{14}$	490	$\widetilde{38}$	$\widetilde{17}$	498	6	88
pipe1no	1200	$\widetilde{14}$	–	$\widetilde{41}$	$\widetilde{67}$	655	8	90
pipe2	1190	48	–	193	$\widetilde{10}$	307	8	18
pipe1c2	1000	$\widetilde{14}$	–	180	$\widetilde{8}$	489	8	54
pipe3	870	$\widetilde{8}$	192	174	$\widetilde{5}$	320	6	52

The most demanding pipeline is *pipe1*, in terms of both FPGA resources and execution time. *pipe1no* improves *pipe1* with respect to the orientation normalization of SIFT (avoids it), the parallelization factor of matching (16× for *pipe1no* but 64× for *pipe1*), and the buffers storing corners (20% smaller for *pipe1no*). Specifically, our tests show that the SIFT orientation normalization requires almost 1 s on SW without enhancing the odometry accuracy in Mars-like scenarios; by avoiding normalization, *pipe1no* saves enough time to process more corners (1200 per single image versus 800 for *pipe1*), do slower/cheaper matching on HW (18% less FPGA logic than *pipe1*), and meet the 1-s real-time constraint. Besides cost efficiency, *pipe1no* leads to the best odometry accuracy. The least computationally demanding pipeline is *pipe2*, which, however, less reliably estimates odometry. *pipe1c2* considerably improves the accuracy of *pipe2* at the additional cost of only 7% FPGA logic, 9% FPGA memory, and 0.2 s cycle time. *pipe3* improves the accuracy of *pipe2* with negligible increase in the FPGA cost. Overall, note that the low logic utilization of *pipe1no* and *pipe1c2* leaves enough room for applying fault tolerance methods, e.g., triple modular redundancy. Also note that *pipe1no* and *pipe1c2* can already fit in today's space-grade FPGAs, i.e., in Virtex-5QV (the only 5QV resource over-utilized by the configurations of Table III is memory, by 39 RAMBs, which, however, can be avoided by simple customization, e.g., by decreasing the 48 RAMBs of our proof-of-concept arbiter).

The execution time of each pipeline is analyzed in Table IV, which presents the average amount of time required by each

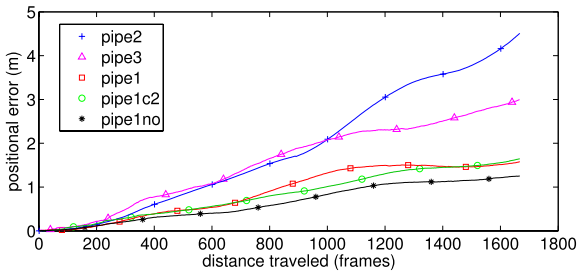

Fig. 8. Mean positional error during 100-m rover traverses (6 cm/frame).

function of the HW/SW system when processing 1 localization step in Atacama (in columns 3–6, tilde "~" denotes execution on HW and ×2 denotes double execution, one for each image of the stereo pair). Time depends on the average numbers of features detected (see column 2) and matched in images: 870 for *pipe1* (which examines up to 2 orientations per feature), 1047 for *pipe1no*, 610 for *pipe2*, 690 for *pipe1c2*, and 500 for *pipe3*. Around 60%–70% of these matches are actually used for egomotion estimation, whereas the rest are rejected as outliers. Note that the above matching results originate from real image datasets; for synthetic datasets, we obtain approx. 50% fewer matches (with only 40% of them used for egomotion, e.g., 200 in *pipe1no*, or even fewer in pipes using BRIEF) and the total execution time decreases by 10%–15%. As shown in Table IV, the FPGA significantly accelerates each function to only 5–67 ms. Despite their low complexity, the SW functions require increased time to execute on the space-grade CPU. Specifically, the SW filtering consumes about 0.5 s, the SIFT orientation normalization consumes 490 × 2 ms (sw-support of *pipe1*), and the interpolation of SURF consumes 192 × 2 ms (sw-support of *pipe3*). The HW matching of *pipe1* completes almost 4× faster than that of *pipe1no*, however, at the cost of 18% extra LUTs (all pipelines except *pipe1no* use HW matching at 64× parallelization). Compared with *pipe1c2*, *pipe1no* is able to process 200 extra corners per 512 × 384 image per second due to its accelerated HW descriptor (the HW SIFT includes 3 ms for image derivatives on the FPGA). The CPU-FPGA communication consumes up to 9% of the total execution time primarily due to the transferring of images over our custom 81 Mbits/s Ethernet link. Overall, compared with the all-software execution on the space-grade CPU, the developed HW/SW system achieves a speedup of 16× for *pipe1no*, 8× for *pipe1*, 4.5× for *pipe1c2*, 4× for *pipe2*, and 3.5× for *pipe3*.

The VO accuracy is evaluated by comparing the output of the pipelines at each localization step to the known ground truth of each synthetic sequence. For Atacama, we measure only the final position of the rover, which is the only position known with absolute certainty. Table III reports the mean error measured at the 100-m final positions of our synthetic sequences. Fig. 8 analyzes the mean error at each intermediate step of these 100-m paths. In all steps, *pipe1no* generates very accurate results limiting the positional error to less than 1.8% of the traveled distance and the attitude error to less than 3.2°. Similarly, *pipe1c2* and *pipe1* have errors always less than 2.6% in position and 5.2° in attitude. For the first 30 m, *pipe2* and *pipe3* are almost equally accurate to the *pipe1* variations. Generally, all pipelines follow gracefully the ground truth path without notable fluctuations of the accumulated error. In the Atacama sequence, the return point error is only 0.42 m for *pipe1no*, 0.71 m for *pipe1*, 1.2 m for *pipe1c2*, and around 10 m for *pipe2* and *pipe3*. Even under the stress-test on the Devon island (100-m sequences #13, #17), the pipelines proved to be quite resilient, especially those based on Harris: *pipe1no*, *pipe1*, and *pipe1c2* terminated with about 4.0% positional error (they differ by 20–50 cm, *pipe1no* still being the best), while *pipe2* and *pipe3* produce errors up to 10% (*pipe3* was configured to use a denser detection grid to reduce the probability of completely losing track).

Besides pipeline validation and tuning, our tests provide useful insight into various individual parts of VO. First, for the 512 × 384 image resolution, the subpixel refinement of detected corner locations is of utmost importance, as it decreases the odometry error from, e.g., 16% to 2% (SURF without subpixel refinement yields error up to 33%). Second, the orientation normalization of SIFT introduces errors in the pipeline (besides cost) making the output less reliable, statistically, in 2 out of 3 tests; hence, it is preferable to avoid such normalization in Mars-like scenarios of 6 cm per step and relatively smooth rover motion. Third, the absolute orientation' egomotion is more reliable than LHM and both have negligible cost. Fourth, the image diversity of distinct datasets, or even between the stereo pair itself (e.g., due to photometric/blurring variations), mandates in-depth study and word-length optimization of the fixed-point datapaths in HW to support the dynamic-range of all internal variables and adapt to every possible image content. Fifth, it is important to use corner selection techniques to detect a constant amount of features per image; this facilitates tuning and adaptation to image content, i.e., it improves the robustness and accuracy of the pipeline (in all cases, the best results were obtained for 800–1400 features per image).

In total, the developed pipelines achieve real-time processing with very accurate odometry, especially *pipe1no* and *pipe1c2*, which meet all of the constraints imposed by ESA. Compared with similar works in [3] and [29]–[31], *pipe1no* implements more advanced algorithms and improves the accuracy and robustness of VO. In particular, compared with the HW/SW codesign of [3] for Mars rovers, we apply SIFT to generate robust descriptors instead of merely using 15 × 15 pixel regions and we match the features based on χ^2 and epipolar geometry instead of merely using SAD; as a result, our VO error becomes roughly half of the 3.3% in [3] (they get a 0.5-m error near a 15-m-away target in real-world experiments with a rover velocity of 5–7 cm/s), whereas *pipe1no* consumes 42% more FPGA slices and *pipe1c2* consumes 15% fewer FPGA slices for processing than the VO of [3]. Notice that making detailed quantitative comparisons at system level is difficult due to the diversity of the datasets, the project specifications, and the HW platforms. Compared with HW/SW pipelines for earth applications, the hand-held device of [30], without IMU fusion, produces bigger error than *pipe1no* (2%–7% depending on the number of key frames and increases to 8% in actual flying tests [31]) and sometimes their VO fails completely; our pipelines process 2–4× more corners

than [30], [31] while employing sophisticated description and matching on FPGA instead of relying merely on pixel areas and correlation on CPU. Compared with the VO of the actual MER rovers [2], our HW/SW pipelines improve the VO accuracy, process one order of magnitude more features, and complete 21× faster due to the proposed FPGA acceleration.

VII. Conclusion

This paper has presented the HW/SW codesign of five distinct VO pipelines for the future Mars rovers. The pipelines were implemented on a proof-of-concept platform consisting of a Virtex6 FPGA and a 150 MIPS CPU resembling the processing power available to future missions. Testing and tuning were based on Mars-like scenarios with carefully selected datasets resembling the Martian terrain. The proposed solutions, especially *pipe1no* and *pipe1c2*, quantify the benefits of employing FPGAs for VO on Mars and meet all specifications provided by the ESA in an effort to advance planetary exploration.

The advantages of the proposed solutions come, first, from exploring and customizing a number of diverse feature-based algorithms and pipelines, second, from developing an HW/SW codesign methodology tailored to the needs of Mars rovers, and third, from designing parallel architectures with extensive pipelining on pixel-basis, parallel arithmetic calculations, parallel memory organization for multidata single-cycle access, input data decomposition for alleviating the memory size limitation, and parametric VHDL accommodating the fine-tuning of the final HW/SW pipelines. Our tests show that VO was accelerated by up to 16× to achieve a 1-s execution time for a 6-cm localization step, with only a 1.25% mean positional error after multiple 100-m paths. The FPGA cost was 54-K LUTs and 1.46-MB RAM. As a result, the proposed solution is more cost effective and more accurate than similar works in the literature, including the VO onboard the actual MER and MSL rovers. Future work involves the implementation of a ready-to-use HW/SW VO pipeline on actual flight hardware.

Acknowledgment

The authors would like to thank M. Avilés Rodrigálvarez from GMV, Spain, for the dataset preparation and technical comments. The authors would also like to thank G. Visentin and S. Vijendran from ESA-ESTEC, The Netherlands.

References

[1] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robot. Autom. Mag.*, vol. 18, no. 4, pp. 80–92, Dec. 2011.

[2] L. Matthies *et al.*, "Computer vision on Mars," *Int. J. Comput. Vis.*, vol. 75, no. 1, pp. 67–92, 2007.

[3] T. M. Howard *et al.*, "Enabling continuous planetary rover navigation through FPGA stereo and visual odometry," in *Proc. IEEE Aerosp. Conf.*, Mar. 2012, pp. 1–9.

[4] A. E. Johnson, S. B. Goldberg, Y. Cheng, and L. H. Matthies, "Robust and efficient stereo feature tracking for visual odometry," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2008, pp. 39–46.

[5] *SEXTANT Project Statement of Work, Reference E913-005MM*, European Space Agency, Paris, Île-de-France, France, 2011.

[6] P. H. S. Torr and A. Zisserman, "Feature based methods for structure and motion estimation," in *Proc. Int. Workshop Vis. Algorithms*, Sep. 1999, pp. 278–294.

[7] R. Szeliski, *Computer Vision: Algorithms and Applications*. New York, NY, USA: Springer-Verlag, 2010.

[8] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. 4th Alvey Vis. Conf.*, 1988, pp. 147–151.

[9] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 105–119, Jan. 2010.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Comput. Vis. Image Understand.*, vol. 110, no. 3, pp. 346–359, 2008.

[11] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[12] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: Computing a local binary descriptor very fast," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1281–1298, Jul. 2012.

[13] S. G. Fowers, D.-J. Lee, D. A. Ventura, and J. K. Archibald, "The nature-inspired BASIS feature descriptor for UAV imagery and its hardware implementation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 5, pp. 756–768, May 2013.

[14] Y. Rubner, J. Puzicha, C. Tomasi, and J. M. Buhmann, "Empirical evaluation of dissimilarity measures for color and texture," *Comput. Vis. Image Understand.*, vol. 84, no. 1, pp. 25–43, 2001.

[15] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *J. Opt. Soc. Amer. A, Opt. Image Sci.*, vol. 4, no. 4, pp. 629–642, 1987.

[16] C.-P. Lu, G. D. Hager, and E. Mjolsness, "Fast and globally convergent pose estimation from video images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 6, pp. 610–622, Jun. 2000.

[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[18] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary robust independent elementary features," in *Proc. 11th ECCV*, 2010, pp. 778–792.

[19] W. Deng, Y. Zhu, H. Feng, and Z. Jiang, "An efficient hardware architecture of the optimised SIFT descriptor generation," in *Proc. IEEE 22nd Int. Conf. Field Program. Logic Appl. (FPL)*, Aug. 2012, pp. 345–352.

[20] N. Battezzati, S. Colazzo, M. Maffione, and L. Senepa, "SURF algorithm in FPGA: A novel architecture for high demanding industrial applications," in *Proc. IEEE Conf. Design, Autom. Test Eur. (DATE)*, Mar. 2012, pp. 161–162.

[21] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient FPGA-based feature detection employing the SURF algorithm," in *Proc. IEEE 18th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2010, pp. 3–10.

[22] J. Jiang, X. Li, and G. Zhang, "SIFT hardware implementation for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 7, pp. 1209–1220, Jul. 2014.

[23] J. Wang, S. Zhong, L. Yan, and Z. Cao, "An embedded system-on-chip architecture for real-time visual detection and matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 3, pp. 525–538, Mar. 2014.

[24] R. Yan, Z. Cao, J. Wang, S. Zhong, D. Klein, and A. Cremers, "Horizontal velocity estimation via downward looking descent images for lunar landing," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 50, no. 2, pp. 1197–1221, Apr. 2014.

[25] F.-C. Huang, S.-Y. Huang, J.-W. Ker, and Y.-C. Chen, "High-performance SIFT hardware accelerator for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 3, pp. 340–351, Mar. 2012.

[26] K. Mikolajczyk *et al.*, "A comparison of affine region detectors," *Int. J. Comput. Vis.*, vol. 65, nos. 1–2, pp. 43–72, 2005.

[27] I. Kostavelis *et al.*, "SPARTAN: Developing a vision system for future autonomous space exploration robots," *J. Field Robot.*, vol. 31, no. 1, pp. 107–140, 2014.

[28] T. D. Barfoot, "Online visual motion estimation using FastSLAM with SIFT features," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Aug. 2005, pp. 579–585.

[29] T. Krajník, J. Šváb, S. Pedre, P. Čížek, and L. Přeučil, "FPGA-based module for SURF extraction," *Mach. Vis. Appl.*, vol. 25, no. 3, pp. 787–800, 2014.

[30] K. Schmid and H. Hirschmüller, "Stereo vision and IMU based real-time ego-motion and depth image computation on a handheld device," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2013, pp. 4671–4678.

[31] K. Schmid, P. Lutz, T. Tomić, E. Mair, and H. Hirschmüller, "Autonomous vision-based micro air vehicle for indoor and outdoor navigation," *J. Field Robot.*, vol. 31, no. 4, pp. 537–570, 2014.
[32] M. E. Angelopoulou and C.-S. Bouganis, "Vision-based egomotion estimation on FPGA for unmanned aerial vehicle navigation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 6, pp. 1070–1083, Jun. 2014.
[33] J. N. Bakambu, C. Langley, G. Pushpanathan, W. J. MacLean, R. Mukherji, and E. Dupuis, "Field trial results of planetary rover visual motion estimation in Mars analogue terrain," *J. Field Robot.*, vol. 29, no. 3, pp. 413–425, May/Jun. 2012.
[34] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2004.
[35] C. Evans, "Notes on the OpenSURF library," Dept Comput. Sci., Univ. Bristol, Bristol, U.K., Tech. Rep. CSTR-09-001, Jan. 2009.
[36] R. I. Hartley and P. Sturm, "Triangulation," *Comput. Vis. Image Understand.*, vol. 68, no. 2, pp. 146–157, 1997.
[37] M. Woods *et al.*, "Seeker—Autonomous long-range rover navigation for remote exploration," *J. Field Robot.*, vol. 31, no. 6, pp. 940–968, 2014.
[38] P. Furgale, P. Carle, J. Enright, and T. D. Barfoot, "The Devon Island rover navigation dataset," *Int. J. Robot. Res.*, vol. 31, no. 6, pp. 707–713, May 2012.

George Lentaris received the B.Sc. degree in physics, the M.Sc. degree in logic, algorithms, and computation, and electronic automation, and the Ph.D. degree in computing from the National and Kapodistrian University of Athens, Athens, Greece.

He is a Research Associate with the National Technical University of Athens, Athens, where he is involved in hardware/software codesign with single- and multi-FPGA platforms and targeting space applications for ESA. His research interests include parallel architectures and algorithms for DSP, digital circuit design, image processing, computer vision, H.264/advanced video coding and High Efficiency Video Coding encoding, and digital signal filtering.

Ioannis Stamoulias received the B.Sc. degree in computer science and telecommunications and the M.Sc. degree in microelectronics with a specialization in design of integrated circuits from the National and Kapodistrian University of Athens, Athens, Greece, in 2010 and 2007, respectively, where he is currently pursuing the Ph.D. degree with the Department of Informatics and Telecommunications.

His research interests include digital signal processing systems, embedded systems, network-on-chip, hardware/software codesign, and computer vision.

Dimitrios Soudris received the Ph.D. degree in electrical engineering from University of Patras, Patras, Greece.

He is an Associate Professor with the School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece. He is the Leader and a Principal Investigator in numerous research projects funded by the Greek government and industry, the European Commission, and the European Space Agency. His research interests include embedded systems design, reconfigurable architectures, reliability, and low power very-large-scale integration design.

Manolis Lourakis received the Ph.D. degree in computer science from University of Crete, Heraklion, Greece, in 1999.

He has been with Foundation for Research and Technology–Hellas (FORTH), Crete, Greece, since 2002, where he has been involved in several research and development projects. He is currently a Principal Researcher with the Computational Vision and Robotics Laboratory, Institute of Computer Science, FORTH. His research interests include computer vision algorithms and topics related to robotics, motion tracking and analysis, registration and matching, multiple and single view geometry, camera(s) self-calibration, 3D reconstruction, 3D shape extraction and analysis, and continuous optimization.