SPARTAN/SEXTANT/COMPASS: Advancing Space Rover Vision via Reconfigurable Platforms

George Lentaris[1], Ioannis Stamoulias[1], Dionysios Diamantopoulos[1], Konstantinos Maragos[1], Kostas Siozios[1], Dimitrios Soudris[1], Marcos Aviles Rodrigalvarez[2], Manolis Lourakis[3], Xenophon Zabulis[3], Ioannis Kostavelis[4], Lazaros Nalpantidis[4], Evangelos Boukas[4], and Antonios Gasteratos[4⋆]

[1]School of Electrical and Computer Engineering, NTUA, Greece.
[2]Advanced Space Systems and Technologies, GMV, Spain.
[3]Institute of Computer Science, FORTH, Greece.
[4]Dpt. of Production and Management Engineering, DUTH, Greece.
{glentaris,dsoudris}@microlab.ntua.gr
maaviles@gmv.com
{lourakis,zabulis}@ics.forth.gr
{gkostave,agaster}@pme.duth.gr

Abstract. Targeting enhanced navigational speed and autonomy for the space exploration rovers, researchers are gradually turning to reconfigurable computing and FPGAs. High-density space-grade FPGAs will enable the acceleration of high-complexity computer vision algorithms for improving the localization and mapping functions of the future Mars rovers. In the projects SPARTAN/SEXTANT/COMPASS of the European Space Agency, we study the potential use of FPGAs for implementing a variety of stereo correspondence, feature extraction, and visual odometry algorithms, all with distinct cost-performance tradeoffs. The most efficient of the developed accelerators will assist the slow space-grade CPU in completing the visual tasks of the rover faster, by one order of magnitude, and thus, will allow the future missions to visit larger areas on Mars. Our work bases on a custom HW/SW co-design methodology, parallel architecture design, optimization techniques, tradeoff analysis, and system tuning with Martian-like scenarios.

Keywords: Space rovers, computer vision, stereo correspondence, feature extraction, visual odometry, HW/SW co-design, FPGA acceleration

1 Introduction

Planetary exploration relies heavily on the navigational speed, accuracy, and autonomy of the robotic vehicle employed in each space mission. Autonomy is

⋆ Projects funded by the European Space Agency (reference numbers AO/1-6512/10/NL/EK, 4000103357/11/NL/EK, 4000111213/14/NL/PA).

important for overcoming the difficulty of tele-operation due to poor communication conditions. Accuracy is important for the safety of the billion-dollar cost vehicle and for performing precision scientific experiments. Speed, together with accuracy and autonomy, will enable the vehicle to explore larger areas on the planet during the lifespan of its mission and bring even more results/discoveries to the scientific community.

One major factor slowing down today's Mars rovers is their very slow onboard space-grade CPU. Due to physical constraints and the error mitigation techniques employed in radiation environments, the space-grade CPUs suffer from extremely low performance, i.e., achieving only 22 to 400 MIPS. When combined with the increased complexity of the sophisticated computer vision algorithms required for rover navigation, the space-grade CPUs become a serious bottleneck consuming several seconds for each step of the vehicle and decreasing its velocity to only 10-20m/h [1][2][3]. The most promising solution being considered today for the rovers of the future is reconfigurable computing and, more specifically, co-processing with space-grade FPGAs.

Space-grade FPGAs are already being used in several space missions, however, not for performing highly-complex Computer Vision (CV) tasks. These rad-hardened devices of Xilinx, Microsemi/Actel, and Atmel, are built on Flash, Antifuse or SRAM technology. Similar to their earthbound counterparts, they can outperform CPUs by allowing parallel computation, power-efficient architectures, and reconfiguration at run-time. The latter is of utmost importance in space missions, because it allows for multi-purpose use and repairing of hardware, which is located millions of miles away from the engineers and would otherwise be rendered useless. The benefits of remote programming and reconfiguration have already been demonstrated in practice for the Mars rovers (e.g., changing between flight/wake/dream modes, repairing high-energy particle glitches, etc). Energy efficiency is important due to the limited power supply of these rovers (e.g., power in the area of 100 Watts, during daytime). Parallel circuits are important for managing numerous sensors/controls concurrently and, in the future, for accelerating the calculation of demanding CV functions.

FPGA acceleration of CV algorithms for rover navigation on Mars is the primary objective of the projects SPARTAN (SPAring Robotics Technologies for Autonomous Navigation), SEXTANT (Spartan EXTension Activity), and COMPASS (Code Optimisation and Modification for Partitioning of Algorithms developed in SPARTAN/SEXTANT), of the European Space agency (ESA). The second objective, of equal significance with acceleration, is to improve the accuracy of the CV algorithms and assemble CV pipelines of increased reliability. The advent of high-density space-grade FPGA devices allows us to explore various HW/SW co-design possibilities for implementing and optimizing CV pipelines tailored to the needs of future Mars rovers. The options increase by considering single- or multi-FPGA approaches, on-chip softcores or off-chip CPUs, altogether with a variety of algorithms offering distinct cost-performance tradeoffs.

The algorithms considered in SPARTAN/SEXTANT/COMPASS relate to two very basic functions of the rover: *mapping* and *localization*, essentially 3D

reconstruction and visual odometry, which are used to solve the more general SLAM problem. Simultaneous Localization and Mapping (SLAM, or in our case VSLAM, because we use Vision to tackle it), is the computational problem of constructing a map of an unknown environment while simultaneously keeping track of the rover's location within the environment. Typically, the rover is equipped with two stereo cameras of distinct characteristics (resolution, field of view, etc) to feed the *mapping* and *localization* functions. The 3D reconstruction part bases on stereo correspondence algorithms, which proceed by comparing the pixels recorded from the two views of the stereo camera to perform triangulation and deduce the depth of the scene. The visual odometry part proceeds by detecting and matching salient features between the images of the stereo pair, as well as between successive stereo pairs, the apparent motion of which is used to deduce the actual motion of the rover on the Martian surface. This feature-based approach to visual odometry breaks down to the sub-problems of accurate feature detection, sufficient feature description, correct feature matching, and robust egomotion estimation. The bibliography includes a plethora of computer vision algorithms for each one of the above sub-problems. The purpose of SPARTAN/SEXTANT/COMPASS is to explore a number of these algorithms and combine them in distinct visual odometry and 3D reconstruction pipelines, which we then partition into HW and SW functions, accelerate using FPGA(s), optimize, tune, and validate, for selecting the most efficient HW/SW solution for the future space rovers. In the remaining of the paper we give an overview of the work being performed in these projects, starting with the system specifications.

2 System Overview

2.1 Specifications and System Configuration

The Automation and Robotics group of the European Space Agency (ESA) determines certain specifications for the systems under development and provides valuable guidelines. In our case, the specifications include certain limits on the type and size of the HW to be used, on the execution time of the algorithms, as well as on the accuracy of the algorithmic results.

For the localization algorithms (visual odometry, VSLAM), the developed pipeline must process one stereo image per second. The images are input from a dedicated stereo camera, namely the "localization" camera. The distance between two successive stereo images during the motion of the rover will be 6cm. That is, the system must be able to sustain a rover velocity of 6cm/sec and travel 100m in 1667 rover steps. At the end of each 100m path, the output of the localization algorithm must have an error of less than 2m in position and 5 degrees in attitude. Regarding the 3D reconstruction (stereo correspondence algorithms), the developed pipeline must generate one 3D local map in 20sec. The map must cover a circular area of 120 degrees and 4m radius in front of the rover. For such an increased coverage, the "navigation" camera will acquire three high-definition stereo images in three distinct directions (left, middle, right) in

front of the rover and, subsequently, we will generate and stitch three partial maps. The accuracy of the 3D maps must be better than 2cm even at 4m depth.

The arrangement of the rover cameras are depicted in fig. 1. The localization camera is placed close to the ground and is intended mainly for feeding the visual odometry pipeline. It has 1Hz sampling rate, a baseline distance of 12cm between its two monocular cameras, focal lengths of 3.8mm (pin-hole model), and image resolution of 512x384 with 8-bit pixels. Notice that, the algorithms will operate on greyscale values. The localization camera is mounted 30cm above ground level and is tilted by 31.55^o with respect to the horizon. The monocular cameras are parallel to each other. Their field of view is 66^o horizontally and 49.5^o vertically. The navigation camera is placed higher, at 1m above ground, and has the ability to rotate left and right to acquire images at distinct directions. The two monocular cameras have a baseline distance of 20cm, focal length of 6.6mm (pin-hole model), are parallel to each other, tilted by 39^o with respect to the horizon, and output images of 1120x1120 resolution with 8-bit pixels. The above stereo setups help us overcome the depth/scale ambiguity, while the given parameters help us tune the algorithms, and also, to generate synthetic image sequences for system testing.

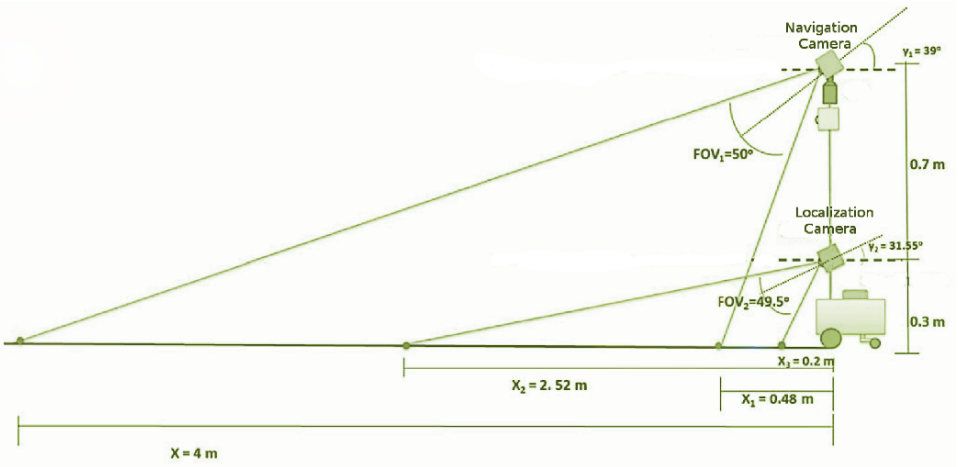

Fig. 1. Camera arrangement of the rover (2 stereo cameras).

The HW specifications limit the amount of processing power that can be employed by the system due to the nature of such space applications. At the same time, the HW specifications consider the potential capabilities of future HW components (these research projects refer to future space missions of 2018 and 2020+). Overall, the HW used in SPARTAN/SEXTANT/COMPASS is a hypothetical LEON-based space-grade CPU and a hypothetical space-grade FPGA of near-future technology; as a working hypothesis, based on current technology trends, the limits are set to 150 MIPS for the CPU (32-bit ISA, 512MB RAM)

and to roughly 100K LUTs and 2MB on-chip RAM for the FPGA. Notice that, in the SPARTAN/SEXTANT/COMPASS projects, we avoid the increased price of actual space-grade devices by emulating them on conventional off-the-self components; to emulate the FPGA, we use a Xilinx XC6VLX240t-2 Virtex6 device, whereas to emulate the low-performance CPU, we use a Intel Core2-Duo CPU (E8400 at 3.00GHz), the time results of which are scaled by a factor of 18.4x (based on CPU benchmarking). Additionally to the single-FPGA solution, we examine multi-FPGA approaches and HW-HW partitioning on 2, 3 or 4 devices of lower performance (more conservative approach of future space-grade technology, e.g., 65nm process, 4-input LUTs, 0.8MB on-chip RAM).

Following the above specifications and recommendations of ESA, we built a proof-of-concept system consisting of a general purpose PC (scaled to 150 MIPS) and an actual Virtex6 evaluation board or a HAPS multi-FPGA board (four Virtex5 devices with predefined constraints on their resources). We develop a custom 100Mbps Ethernet communication scheme for the FPGA and the CPU to exchange data. We connect the cameras to the CPU and we also store a variety of test sequences on a hard disk drive for off-line use. We design a high-level software architecture in a three-level hierarchy. Level-0 interacts with the OS of the CPU and includes the kernel drivers (cameras, custom Ethernet) and the wrappers of the three most basic functions: imaging, mapping, and localization. Hypothetically, Level-0 also interacts with other parts/modules of the rover, e.g., the IMU sensor, the wheel encoders, navigation algorithm, etc. On top of Level-0, we build Level-1 to include all the C/C++ procedures of our vision algorithms. Finally, Level-2 includes a subset of the algorithms, i.e., the most computationally intensive, which are implemented on the FPGA.

2.2 Datasets

To evaluate and fine-tune the developed system, we perform several tests based on pre-recorded videos and stereo images resembling the Martian surface. ESA specifies a number of qualitative characteristics for the test images to comply with, i.e., they depict an arbitrary mixture of fine grained sand, rock outcrops and surface rocks of various sizes, with the percentage of each component ranging from 0 to 100% among frames. Visually, the images have rather diffuse lighting and low contrast, as is expected to be happening on the “red” planet.

The test sequences are recorded in natural Earth terrains of Martian appearance and in synthetic Mars-like environments. The former provide the highest degree of realism when it comes to image features, lighting conditions, content diversity, etc., which prove to be absolutely necessary for reliable system tuning and pipeline validation. The latter offer the advantage of 100% certainty when measuring errors (in a synthetic model, the position of every element in the environment is known a priori, with maximum accuracy), which proves extremely useful for detailed fine-tuning and accurate trade-off evaluations. Our tests combine both type of environments to exploit both of their advantages, and also, we use distinct sequences from each environment to exploit their diversity and examine as many situations as possible. Hence, the selected datasets:

– help us improve the developed HW/SW pipelines by exposing every potential error/problem (e.g., photometric/blurring variations between the stereo pair, intensity changes between frames, highly-varying amount of features, etc),
– they allow us to perform word length optimization and cover the dynamic range of all variables, safely, without overestimating their HW cost,
– they facilitate design space and/or algorithmic exploration to select the most efficient cost-performance pipeline by making detailed measurements
– customize the system to the visual content expected on the surface of Mars

The synthetic image test sequences are generated by 3DROV (fig. 2). 3DROV is a complete virtual simulator based on SimSat [4]. We feed the simulator with precise models of the rover (camera positions) and samples of known Martian environment. By simulating the actuation operations and data acquisition functions, we record thousands of successive stereo frames showing arbitrary 100m paths of the rover, as well as still-images of high resolution for testing our 3D reconstruction algorithms. Specifically, to test the localization algorithms (visual odometry, VSLAM), we generate three sequences of 1667 stereo frames each, with 512x384 resolution and 6cm between successive frames (fig. 3, upper row). To test the mapping algorithms (stereo correspondence, 3D reconstruction), we generate two sequences of 34 stereo frames each, with 1120x1120 resolution and depths ranging from 1 to 4m (similar content with fig. 3, upper row).

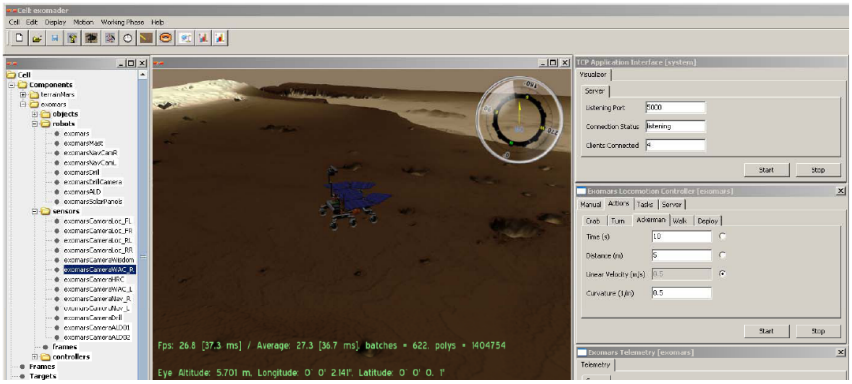

Fig. 2. 3DROV virtual simulator used to create synthetic Mars-like datasets.

The natural image test sequences originate from three distinct places on Earth, which were carefully selected to resemble as much as possible the Martian landscape (fig. 3, bottom row). First, we use the video recorded on the Atacama desert of Chile, which was specifically designed to recreate Mars-like scenarios [5]. We use 1999 stereo images of 512x384 resolution and approx. 6cm distance between frames (the 334 are stationary). To make measurements with 100% certainty, we set the first 1000 frames to depict a forward rover motion and the remaining 999 to depict a backward rover motion (the previous frames played

in reverse order), such that the rover ends up at its initial position. Second, we use the Devon island sequences [6], which are more challenging than Atacama and depict rover motion with approx. 19cm distance per frames on rough terrains with increased vibration. For Devon, we use the GPS measurements as groundtruth. Third, we use a custom video recorded in Ksanthi, DUTH, Greece, by using a stereo rig with the Bumblebee2 camera.It depicts a rocky and sandy landscape and it has been acquired under low lighting conditions with alow elevation angle of the sun. Also for DUTH, we use as groundtruth the data acquired with the Promark 500 Magellan Differential GPS.

Fig. 3. Sample frames from six distinct test sequences. Above: synthetic image datasets showing sand and rocks. Below: Natural image datasets form the Atacama desert in Chile (left), Devon Island in Canada (middle), Ksanthi-DUTH in Greece (right).

3 Related work and Overview of the Algorithms

The majority of the algorithms explored and accelerated by FPGA in SPARTAN/SEXTANT/COMPASS fall in two categories of Computer Vision: stereo correspondence and feature extraction/matching. The former algorithms are used for reconstructing the 3D scene in-front of the rover, i.e., for its *mapping* function. The latter are used to develop visual odometry pipelines and estimate the motion of the rover in a frame-by-frame basis during its path within the environment, i.e., for its *localization* function. For both categories, we perform algorithmic exploration and HW/SW implementation of multiple pipelines to evaluate their cost-performance and select the most efficient solution.

For stereo correspondence, we examine and implement two of the most widely used algorithms in the field: disparity and spacesweep [7][8]. Essentially in both cases, from an implementation point of view, the process boils down to comparing

a region of left-image pixels to a region of right-image pixels (e.g., of size 13x13). The comparison will occur for all such regions within the two images in a full-search fashion, however, by taking into account the epipolar geometry of the stereo pair to decrease the search space of the correspondence problem. The details for fetching and comparing each one of the areas are determined by the algorithm. The distance of the areas matched between the two images (their disparity) determines the depth of the scene (via algorithm-specific calculations, e.g., triangulation) at all points of the image (dense stereo).

Feature detection, description, and matching are the tree algorithmic phases preceding the actual egomotion estimation of a feature-based visual odometry pipeline [9]. The egomotion estimation will input two point-clouds (set of 3D or 2D points of the environment, as captured by the camera and reconstructed by the CV algorithms) from two distinct positions/frames of the rover and will try to align them for deducing the motion of the rover (pose estimation of the translation/rotation parameters). For egomotion, we consider approaches such as the *absolute orientation* of Horn and the $SO(3)$ based solution of Lu, Hager and Mjolsness. Before employing egomotion, first, we detect salient features on each image, most often by examining its intensity gradients.We explore and implement well-known detection algorithms such as SURF [10], Harris [11], FAST [12]. Second, we describe each detected feature by processing its surrounding pixels to collect information in a succinct and resilient vector, most often a histogram based on the gradients' magnitude/orientation. We explore and implement well-known description algorithms such as SURF [10], SIFT [13], BRIEF [14]. Third, we match the descriptors between stereo images for triangulating and deducing their 3D coordinates, as well as between successive frames to feed the egomotion with the correspondences of the two point-clouds. We explore and implement distinct matching methods based on epipolar geometry and on Euclidean distance, Hamming distance, and the χ^2 distance (from the χ^2 test statistic).

4 FPGA acceleration

The algorithms described in the previous section are used individually or in combination to form distinct pipelines for *mapping* and *localization*. For instance, we can use spacesweep to construct a dense 3D point cloud of the environment (mapping); or, we can put SURF detection/description, Euclidean matching and Horn's egomotion to estimate the visual odometry of the rover (localization). The execution time of the algorithms varies greately with respect to their complexity and image content. In some cases, it exceeds by far the 20sec constraint of mapping (e.g., spacesweep requires approx. 1 hour on the space-grade CPU for processing 3 high-definition stereo images) or the 1sec constraint of localization (e.g., feature detection alone requires up to 2sec for one stereo pair). In other cases, it requires around 10 msec (e.g., Horn's egomotion). By analyzing their execution time, complexity, communication requirements, arithmetic precision, and the platform's characteristics, we develop a custom HW/SW co-design methodology to determine those kernels and/or sub-kernels that should be exe-

cuted on CPU and those that must be accelerated by the FPGA for the pipelines to meet all time/accuracy/cost specifications of ESA.

To accelerate the most demanding kernels and at the same time avoid over-utilizing the FPGA resources, we propose HW architectures based on a number of diverse design techniques. First of all, to overcome the FPGA memory bottleneck, we perform decomposition of the input data on the CPU side and download each set to the FPGA, separately. Each set (e.g., a stripe of the image) is downloaded, processed, and then uploaded to the CPU, before the following set can be processed by re-using the same FPGA resources. Second, we perform pipelining at pixel basis, which accounts for one of the main reasons for achieving high speed-up factors. The nature of the image processing algorithms (repetitive calculations on successive pixels) allows for efficient design and increased pipeline utilization. Third, we design parallel memories allowing multiple data to be fetched at a single cycle and support the throughput of the pixel-based pipeline. Fourth, we parallelize the calculation of the mathematical formulas and support the throughput of the pipeline. Next, we explain the techniques by describing a representative example of our HW kernels: the SURF descriptor.

4.1 Architecture of the SURF Descriptor

The proposed architecture accelerates the Haar wavelet based processing part of SURF [10]. It utilizes both HW and SW modules (Fig. 4) to divide the area of each feature (interest point, ipts) in 16 smaller areas, which are then processed by the FPGA sequentially starting from the top and moving to the bottom of the image. Breaking down the problem of describing an interest point to describing several smaller boxes has a twofold purpose. First, we decrease the area required to be cached on the FPGA by almost $1/16^{th}$. Second, we reuse our HW resources by developing a custom sliding window: the on-chip memory of the FPGA stores only a horizontal stripe of the integral image, instead of 512x384 values, which is updated iteratively until the entire image is scanned downwards. Therefore, given that we detect ipts in 13 scales of SURF and that, in the worst case, the orientation of an interest point will be at 45^o, the maximum height of an interest point's box is reduced to 130 rows (instead of almost the entire image height). Hence, our sliding window has a size of 512×130 integral values.

To support the FPGA process, we perform certain SW modifications on OpenSURF. We develop the *Ipts Disassembler* component (Fig. 4) to divide each ipt to its 16 main squares, namely to its "boxes" (1 ipt $= 4 \times 4$ boxes). The set of all boxes of the image (up to 1600) is sorted according to their y coordinate. This sorting allows the HW descriptor to process the boxes in order, i.e., to use a sliding window, which will always move downwards on the image unloading old data and reusing memory space. We implement a custom data structure on SW to sort and store the boxes in linear time: we use one 1-D matrix of lists containing stacks of boxes, which are grouped according to their y coordinate. The data used to identify each box are transmitted to the FPGA starting from the left-most list. Upon completion of the FPGA, the *Ipts Assembler* receives

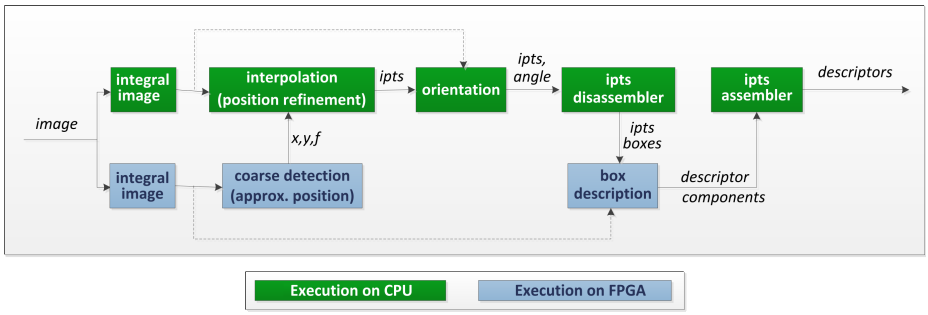

Fig. 4. Proposed HW/SW Partitioning of the SURF algorithm.

the box descriptors and stores them within our custom data structure. Finally, it normalizes the HW results and forms the 64-value descriptor of each ipt.

The HW accelerator of SURF description consists of 7 components (Fig. 5): the *Boxes Memory* (BM), the *SampleX & SampleY Computation* (SSC), the *Component of Memories* (CM), the *HaarX & HaarY Calculator* (HHC), the *Gauss Computation* (GC), the *Box Descriptor Computation* (BDC) and the *Descriptors Memory* (DM). It processes the received boxes iteratively by moving its sliding window downwards on the integral image (by 1 row at each iteration) and describing all possible boxes lying on the central row of the window.

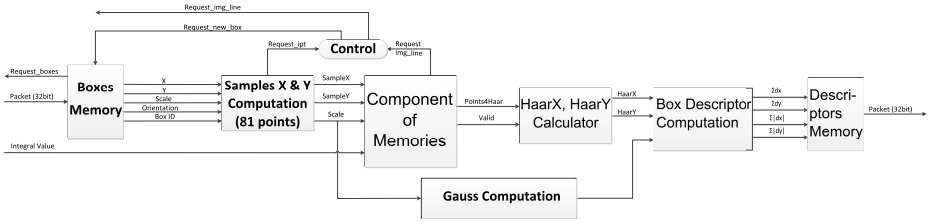

Fig. 5. Proposed HW architecture of SURF's Box Descriptor

The Box Memory stores all the information regarding the boxes to be described ($\{x$, y, scale, sin, cos, ID$\}$). Each 6-tuple is received in two 32-bit words and is unpacked and placed in a local FIFO memory. During execution, whenever a box description is completed, the CU will pop a new box from the FIFO to initiate the next box description, until the FIFO is empty.

The SampleX & SampleY Computation generates 81 coordinate pairs specifying the 81 neighboring "samples" required for the description of a single box [10]. It uses the information associated with each box (the 6-tuple) to compute $X = \text{round}(x + (-j \cdot scale \cdot sin + i \cdot scale \cdot cos))$, where (i, j) is used to identify each sample and X is its actual image coordinate (Y is computed analogously).

The Component of Memories acts as a local cache of the integral values, i.e., it implements our sliding window on the integral image. During the description of a box, it inputs the aforementioned 81 sample coordinate pairs in a pipeline fashion and fetches 8 integral values per sample. The parallel memory will provide all 8 values with a single cycle access facilitating the high throughput computation of the HaarX-HaarY characterization of each sample (1 per cycle). The parallel memory storing the 512×130 stripe of the integral image utilizes 16 true dual port banks and is organized based on the non-linear mapping $BANK(x, y) = (x+y\times3) \bmod 16$ and $ADDR(x, y) = x \text{ div } 16+(y \bmod 130)\times(512/16)$. The above organization allows for a single cycle access of the 8 integral values required for the computation of any Haar wavelet used by SURF (i.e., dx and dy responses), for up to 13 scales (besides scale 8), and from anywhere on the image. Note this this requirement implies random access to multiple rectangle-shaped patterns, and hence, increases the complexity of the organization.

The HaarX & HaarY Calculator inputs the 8 integrals coming from the parallel memory to calculate (add/sub) the HaarX and HaarY for each sample [10].

The Gauss Computation (GC) generates the values used to weight each one of the 81 samples. The generation bases on the distance of the sample from the center of its box. We decrease its hardware complexity by simplifying the rounding of the scale value and the coordinate pair used to calculate the exponential terms. As a result, we avoid implementing exponentiation and division circuits, we reduce the set of possible inputs for each sample to 13 or 25 numbers stored in LUTs , and we use only one multiplication for their final combination.

Finally, the Box Descriptor Computation inputs the HaarX & HaarY of the samples and weights them to produce the descriptor of the box. The box descriptor consists of 4 values, i.e. Σdx, Σdy, $\Sigma|dx|$, and $\Sigma|dy|$, which are computed via the summation of the 81 samples. The responses dx and dy are computed separately for each sample using 20 bit fixed-point multipliers. Specifically, for dx (and similarly for dy), it computes $dx = \text{gauss\_weight}\cdot(-\text{HaarX}\cdot\sin+\text{HaarY}\cdot\cos)$.

4.2 Results

The accelerators are implemented on the XC6VLX240T-2 FPGA and the results show significant speed-up factors compared to the SW execution on the 150MIPS CPU. The speed-up ranges from 60x for feature detection to over 1000x for stereo correspondence. Such speed-up is necessary for meeting the time specifications of future rovers. The quality/accuracy of the FPGA output proves to be sufficient for the rover needs; by careful customization, it becomes comparable to the full floating point accuracy of the CPU, even though we tend to employ fixed-point arithmetic and do mild simplifications on the FPGA. The FPGA resource utilization per accelerator, due to our efficient architecture design with hand-written VHDL and customization/tuning, decreases to 7-21% of the FPGA slices, and to less than 1/3 of the on-chip memory. Specifically for the SURF Descriptor presented above, the processing of the boxes completes in 5ms for 200 features (ipts) by utilizing 4.3K LUTs, 5.7K registers, 13 DSPs, and 136 RAMB36.

5 Conclusion

The current paper presented an overview of the projects SPARTAN, SEXTANT, and COMPASS of the European Space Agency, which are part of the ongoing research to improve the autonomous planetary exploration rovers. Our work bases on single- and/or multi-FPGA acceleration and optimization of selected computer vision algorithms for advancing the localization and mapping skills of the future rovers. Ongoing results show significant speedup factors with improved algorithmic accuracy and quantify the benefits of using space-grade FPGAs for 3D reconstruction and visual odometry.

References

1. L. Matthies, M. Maimone, A. Johnson, Y. Cheng, R. Willson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein, and A. Angelova, "Computer Vision on Mars," *International Journal of Computer Vision, Springer*, vol. 75, no. 1, pp. 67–92, 2007.
2. T. M. Howard, A. Morfopoulos, J. Morrison, Y. Kuwata, C. Villalpando, L. Matthies, and M. McHenry, "Enabling continuous planetary rover navigation through FPGA stereo and visual odometry," in *IEEE Aerospace Conference*, 2012.
3. A. Johnson, S. Goldberg, Y. Cheng, and L. Matthies, "Robust and efficient stereo feature tracking for visual odometry," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, May 2008, pp. 39–46.
4. Poulakis, P., L. Joudrier, S. Wailliez, and K. Kapellos, "3DROV: A planetary rover system design, simulation and verification tool," in *Proc. of the 10th Int'l Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-08)*. 2008.
5. M. Woods, A. Shaw, E. Tidey, B. V. Pham, U. Artan, B. Maddison, and G. Cross, "SEEKER-autonomous long range rover navigation for remote exploration," in *Artificial Intelligence, Robotics and Automation in Space, Int'l Symp. on, Italy*, 2012.
6. P. Furgale, P. Carle, J. Enright, and T. D. Barfoot, "The Devon Island Rover Navigation Dataset," *Int'l Journal Robotics Research*, vol. 31, no. 6, pp. 707–713, 2012.
7. Lentaris George, Dionysios Diamantopoulos, Kostas Siozios, Dimitrios Soudris, and Marcos Avils Rodrigalvarez. "Hardware implementation of stereo correspondence algorithm for the exomars mission." In Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on, pp. 667-670. IEEE, 2012.
8. Richard Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010
9. D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robot. Automat. Mag.*, vol. 18, no. 4, pp. 80–92, 2011.
10. Bay, H., Ess, E., Tuytelaars, T., Van Gool, L.: Speeded-Up Robust Features (SURF). In Comp. Vision and Image Understanding, Vol. 110, Iss. 3, 346–359, 2008
11. C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, 1988, pp. 147–151.
12. 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, 2010.
13. D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int'l Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
14. M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *ECCV (4)*, 2010, pp. 778–792.