FPGA-Based Feature Detection

Wennie Tabib School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
wtabib@andrew.cmu.edu

Abstract

Fast, accurate, autonomous robot navigation is essential for landing and roving on planets. For landing, particularly, computing speed matters. Navigation algorithms calculate feature descriptors in stereo pairs as the basis of visual odometry and obstacle avoidance. Feature detection is understood, but requires hundred-fold speed-up which is not achievable on conventional lander and rover computers. The need is for fast, spaceworthy feature detection. This means near-real-time speed, low power consumption, and radiation tolerance. This research accomplishes this vision by developing Scale Invariant Feature Transform (SIFT) for implementation on on Field Programmable Gate Arrays (FPGA). FPGA challenges include lack of floating point limit of resources, multiple clock domains, and means for interfacing with DSPs, DRAM, and CORDIC.

1 Introduction

1.1 Compelling Motivation

Carnegie Mellon will accomplish an autonomous landing and a rover on the Moon to win the twenty million dollar Google Lunar X PRIZE. The lander and rover use Terrain-Relative Navigation (TRN) to achieve precision landing and roving, respectively. TRN uses optical registration to provide position relative to a priori map images (figure 1). During the descent orbit of the lander, the terrain relative navigation system is activated. The estimated spacecraft state from IMU, star tracking and radio is used to initialize TRN. In real time, successive camera frames are processed through SIFT feature extraction; features are corresponded using RANSAC; and an unscented Kalman filter (UKF) determines the spacecraft state. These algorithms might cycle at 10 seconds on a typical space computer, hundred-fold acceleration to 10 Hz is required for lander. This thesis implements SIFT in hardware on an FPGA to allow for real-time operations.

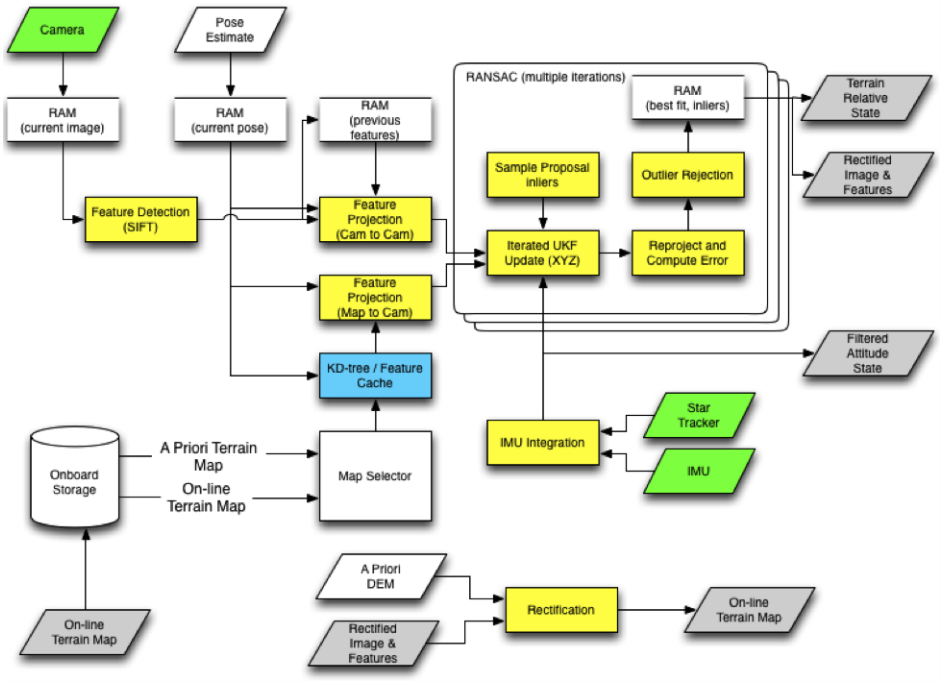

Figure 1: Navigation block diagram for TRN

SIFT is also relevant for rover navigation. The Mars Exploration Rovers (MERs) implemented low-fidelity visual odometry on standard computing to measure slip when the rover encountered a slippery surface [1]. The MER Spirit used stereo vision for obstacle detection for 28% of its total traverse and visual odometry for 16% [2]. Obstacle detection and visual odometry were used together for only 3 m because processing was too slow [2]. The MER Opportunity used stereo vision for obstacle detection for 21% of its traverse and visual odometry for 11% [2]. Opportunity never used both together [2]. A limiting factor in the use of vision on the MERs is computational speed. Use of vision makes the rovers XXX times slower. Consequently, vision is turned off, risk is accepted, and exploration progress is possible. For example, Spirit was entrapped and lost when feature detection was turned off.

Feature detection must be sufficiently accurate to produce descriptors that will provide reliable state estimation. GPS is unavailable on the Moon. Quadrature encoders suffer from wheel slip and IMUs drift. Visual odometry must be viable. SIFT provides feature descriptors invariant to uniform scaling, orientation, and partially invariant to affine distortion and illumination changes. Thus, features can be matched frame-to-frame even under variations in size, position, distortion and light. The error in measurements from vision-based odometry is far lower than quadrature encoders and IMUs—0.1% - 5% over several hundred meters [3]. Furthermore, FPGAs consume significantly less power which is essential for space computing.

1.2 Prior Work

The foundation of the algorithm implemented here is based on Scale Invariant Feature Transform (SIFT) [4]. The crux of this research is to specialize SIFT for FPGA implementation.

FPGAs have demonstrated success in computer vision applications. Computer vision algorithms are processor-intensive and slow. The risk is that CPU implementations cycle too slowly to safegaurd and navigate effectively. Speeded Up Robust Features (SURF) [5], face detection [6], features from localized principle component analysis (PCA) [7], and SIFT for small images [8] have been developed for FPGA with incredible boosts in speed. For example, SURF feature detectors

have been implemented on FPGA that can process 56 frames per second [9]. This is an 8-fold improvement over equivalent CPU-based implementations. FPGAs are clocked at around 200 MHz and consume less than 20W [9]. By comparison, a high-end GPU clocked at 1.35GHz consumes more than 200W [9]. The challenge is to achieve comparable speed-up for planetary-relevant SIFT.

2 Approach

2.1 SIFT

SIFT is a method of obtaining feature descriptors from interest points (e.g. blobs, T-junctions, corners) in an image. 'Interest points' are selected at distinctive locations, and the pixels neighboring the interest point are represented by a feature vector or *descriptor*. Feature detection is measured by:

1. repeatability: A good feature detector must reliably find the same interest points under different viewing conditions such as rotation, distortion, and illumination changes.
2. quality: Feature descriptors are robust to noise and geometric deformations while being unique for each feature. In addition, a balance between small and large feature descriptors is desirable to compute matches in real time.

2.2 Objective

The objective of this paper is to synthesize SIFT onto an FPGA and detect features in images of 2592×1944 in less than 500 milliseconds per frame.

2.3 Methodology

The methodology is partitioned into (1) the specialization to implement SIFT on FPGA and (2) method and implementation of SIFT for feature detection.

2.3.1 Specialization to Implement SIFT on FPGA

Several challenges must be overcome to design an implementation of SIFT that synthesizes on FPGA. Challenges include limitations in the specific FPGA resources; pipelining the image streaming; working without floating point numbers; multiple clock domains; and interfacing with DRAM, CORDIC, and DSPs.

Each FPGA board has different counts of resources like DSPs and block RAMs. Therefore, an implementation of SIFT designed for one FPGA board may not work on another. Resources such as Block RAMS, SDDR2 SDRAM, and DSPs must be taken into account when designing the algorithm so that these resources do not run out. In addition, pipelining is a powerful technique in FPGA design. Maximizing the utilization of this technique is imperative for an efficient and fast implementation of SIFT. Also, floating point operations must be avoided in order to speed up the FPGA. All floating point operations in the algorithm must be converted to an integer equivalent.

However, some FPGA resources are bounded by maximum clock speeds. For example, the DRAM on a Virtex-5 FPGA can only run at a maximum of 200MHz. The rest of the algorithm must work around this limitation by either slowing down the clock speed (if it is faster) or buffering data. Thus, if one resource runs at one clock speed and another resource runs at a different clock speed, the challenge of multiple clock domains is introduced. Techniques must be developed so that data is properly handled between clock domains. Furthermore, several resources on the FPGA are non-trivial to work with. DRAM, CORDIC, and DSP blocks can only be accessed by incorporating intellectual property (IP) cores into the design.

2.3.2 Discussion and Implementation of SIFT for Feature Detection

SIFT is composed of Gaussian blur, difference of Gaussians, maxima detection, magnitude and orientation calculation, dominant direction, and feature descriptor.

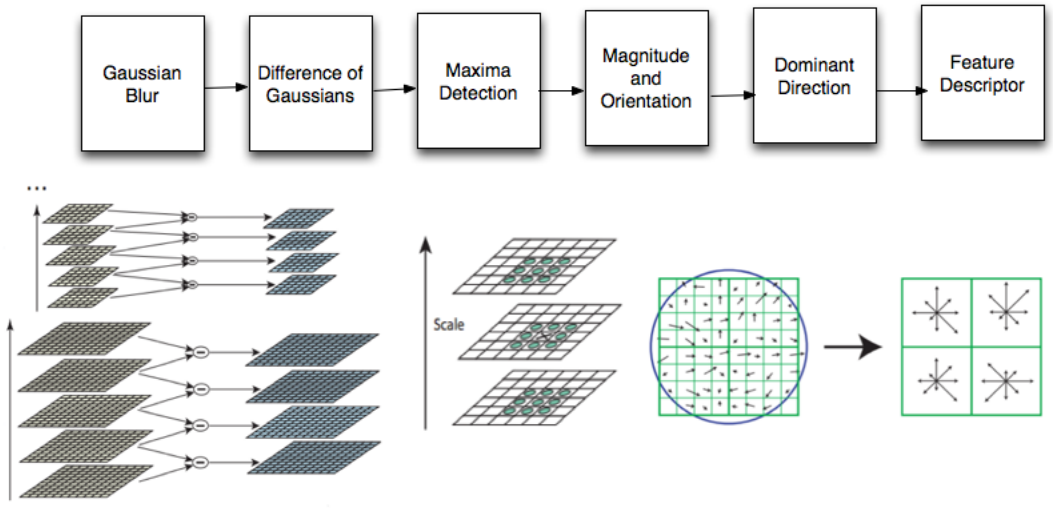

Figure 2: Block diagram of SIFT

Gaussian blur To process a pixel per clock cycle, this research implements Gaussian blur using cyclic buffers to concurrently buffer lines of image data.

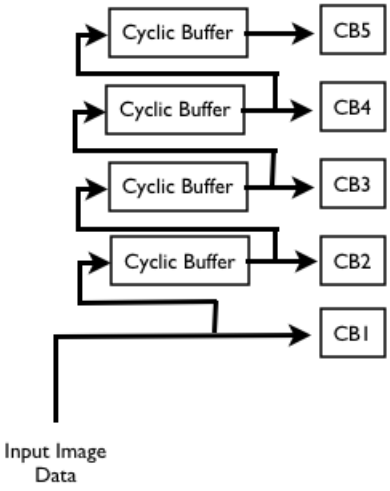

Figure 3: 4 cyclic buffers used to process 5 lines of image data simultaneously

Figure 2 displays the flow of data through the cyclic buffers. As a pixel from the first row of image data is output on the line labeled CB1, the pixel is captured in the cyclic buffer directly above the line (in this case CB2). By the time line 5 is on the CB1 wire, CB2 will contain pixel values from line 4, CB3 from line 3, and so on. Thus, columns of 5 pixels will be streamed simultaneously as they appear in the image when all 5 cyclic buffers are full.

Lowe uses five scales and four octaves [4]. Thus, the original image must be blurred at five different levels. Five blurs are performed simultaneously in figure 3.

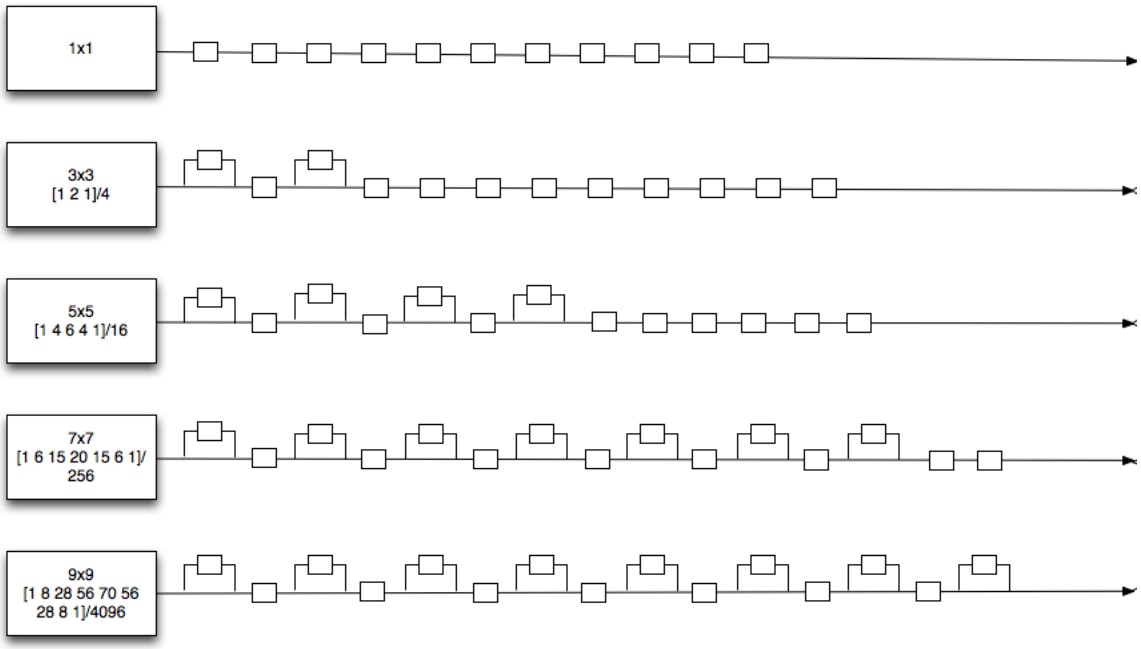

Figure 4: Gaussian blur convolution

where the empty boxes are registers (as in figure 4).

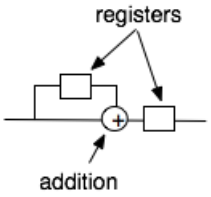

Figure 5: register/add module

The values 1×1, 3×3, 5×5, 7×7, and 9×9 specify the size of the window upon which the blur is performed. More generally, for an $n \times n$ window where $n = 2m + 1$ and $n, m \in \mathbb{N}$, the blur vector chosen for the convolution corresponds to the n^{th} row of Pascal's triangle. Thus the matrix for an $n \times n$ convolution is derived from

$$\begin{bmatrix} p_1 \\ p_2 \\ \cdot \\ \cdot \\ \cdot \\ p_n \end{bmatrix} \cdot \begin{bmatrix} p_1 & p_2 & \cdot & \cdot & \cdot & p_n \end{bmatrix} = \begin{bmatrix} p_1p_1 & p_1p_2 & \cdot & \cdot & \cdot & p_1p_n \\ p_2p_1 & p_2p_2 & \cdot & \cdot & \cdot & p_2p_n \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ p_np_1 & p_np_2 & \cdot & \cdot & \cdot & p_np_n \end{bmatrix}$$

where p_i is the i^{th} number in the n^{th} row of Pascal's triangle and $i \in \mathbb{N}$. Pixels are streamed into each module (labeled 1×1, 3×3, etc. in figure 3) and multiplied by row n of Pascal's triangle.

The rest of the matrix multiplication is performed by register/add blocks.

Difference of Gaussians Blurred images are subtracted from one another in the Difference of Gaussians (DoG) as shown below.

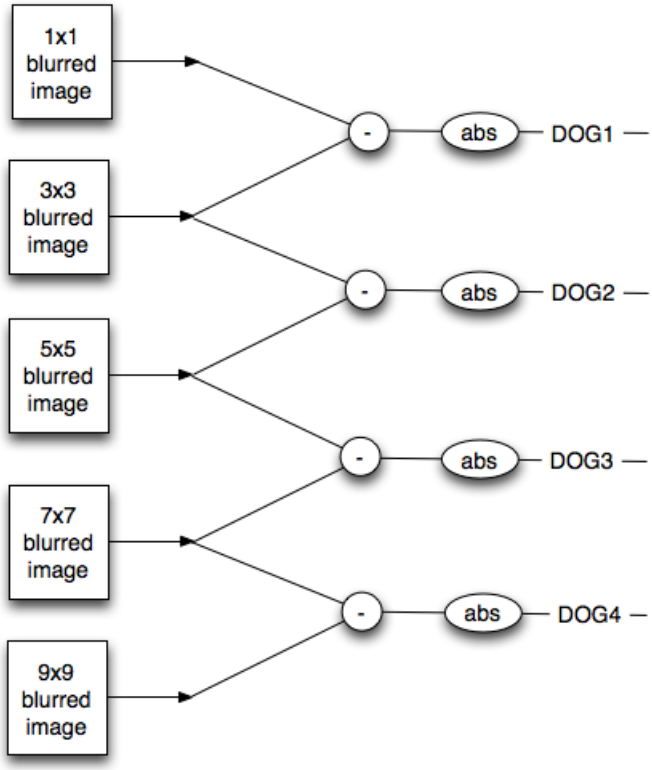

Figure 6: Difference of Gaussians

Maxima Detection The maximum is determined by comparing the center pixel of a 3×3 window in the second layer of images to all the pixels around it as well as the pixels directly below and above it (figure 5). Thus, for each maximum, 27 comparisons are made.

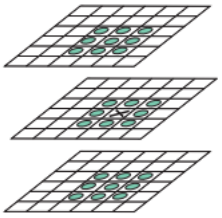

Figure 7: The pixel marked with an X is compared to the nine pixels above and below it and the 8 pixels around it.

The dataflow diagram in figure 6 streams one column of data from each of the images into the maximum detector. Maximums are obtained from DOG2 and DOG3. The pixels in DOG2 are

compared to the nine pixels in each of DOG1 and DOG3. And the pixels in DOG3 are compared to the nine pixels in each of DOG2 and DOG4.

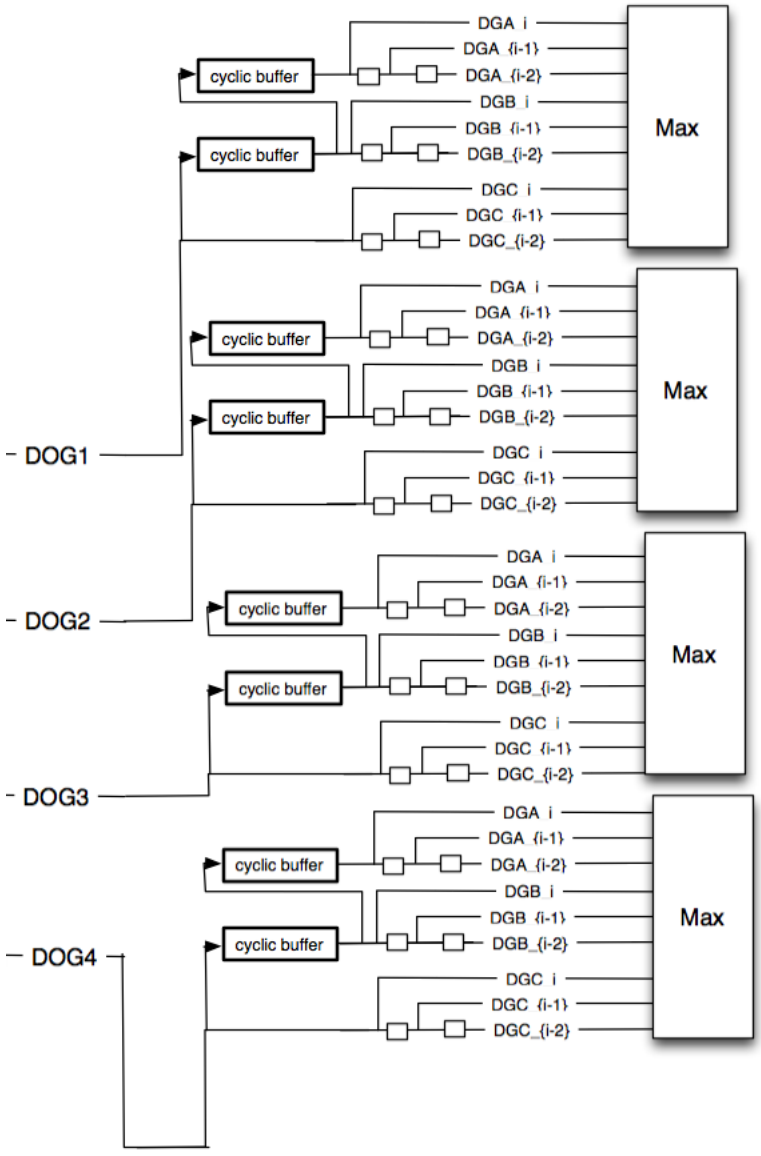

Figure 8: Data flow of Maxima Detection

Magnitude and Orientation Calculation The calculation of magnitude and orientation begins by finding the finite differences in a 16×16 window around the maximum. The difference in x, or dx, and the difference in y, or dy, are stored in block RAM (BRAM) from which CORDIC is used to generate the magnitude (m) and angle (θ) as in figure 9.

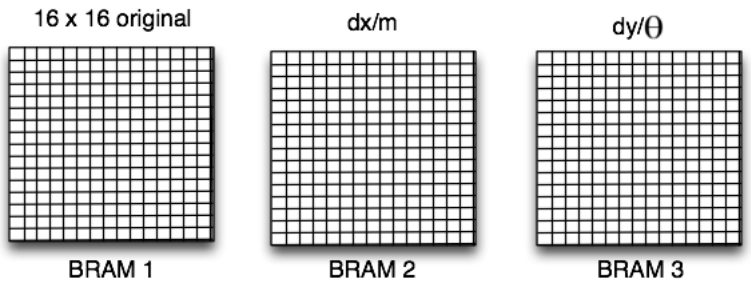

Figure 9: Maxima Detection

The finite differences (dx, dy) of a pixel are calculated by subtracting the pixel to the left from the pixel on the right and taking the absolute value. If the pixel is on the edge of the row or column, the finite differences are calculated by subtracting that pixel's value from the existing pixel on either the right or left and taking the absolute value. The figure below illustrates this for a row. Rows calculate dx values. The dy values are calculated using the same procedure but for columns.

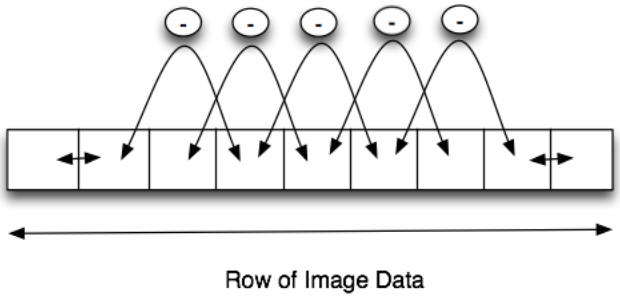

Figure 10: Maxima Detection

Once the magnitudes and orientations are calculated, the orientations are accumulated into one of 32 bins with a weight proportional to the magnitude of the pixel.

Dominant Direction Keypoints are assigned a dominant direction the bin passes the 80% mark. If a second direction passes the 80% mark, the keypoint is split into two keypoints.

Feature Descriptor Feature descriptors are generated by splitting the 16×16 window around the keypoint. The 16×16 window is broken into sixteen 4×4 windows. The gradient magnitude and orientations are calculated and put into 8 bin histograms. Once this has been done for all regions, the 16×16 window is converted into a feature descriptor by normalizing the $4 \times 4 \times 8$ feature vector.

Finally, to achieve rotation independence, the keypoint's rotation is subtracted from each orientation. The keypoint is thresholded to achieve illumination independence.

3 Results

An architecture which is synthesizable on FPGA has been developed. The architecture successfully simulates and processes an image that produces 4 octaves containing 5 scales as in [4]. Floating

point operations have been converted to fixed point and extensive pipelining has been introduced into the algorithm. Furthermore, DRAM and compact flash have successfully been interfaced with on the board. The expected timing of this architecture is 130 milliseconds per frame. Resource calculation has been determined to be 89%. The length of the critical path is thus 260,000 clock cycles at 200 MHz.

4 Future Work

The architecture will be completely synthesized on FPGA. It will be benchmarked and optimized to ensure performance. And visual odometry will be implemented on FPGA.

References

[1] Y.Cheng, M. Maimone, and L. Matthies. Visual odometry on the mars exploration rovers. *IEEE*, 13(2):54–62, June 2006.

[2] L.Matthies and et al. Computer vision on mars. *International Journal of Computer Vision*, 2007.

[3] Z. Zhu and et. al. Ten-fold improvement in visual odometry using landmark matching. *IEEE*, pages 1–8, October 2007.

[4] D.G.Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[5] T.Krajnik, J. Svab, and L. Preucil. Fpga based speeded up robust features. *IEEE*, pages 35–41, November 2009.

[6] N.Farrugia. Fast and robust face detection on a parallel optimized architecture implemented on fpga. *IEEE*, 19(4):597–602, April 2009.

[7] F.Zhong. Parallel architecture for pca image feature detection using fpga. *IEEE*, pages 4–7, May 2008.

[8] L.Yao, Y.Zhu, Z.Jiang, and D.Zhao. An architecture of optimised sift feature detection for an fpga implementation of an image matcher. *IEEE*, pages 30–37, December 2009.

[9] D.Bouris, A.Nikitakis, and I.Papaefstathiou. Fast and efficient fpga-based feature detection employing the surf algorithm. *IEEE*, pages 3–10, May 2010.