Two Camera System for Robot Applications; Navigation

Jörgen Lidholm, Fredrik Ekstrand and Lars Asplund
School of Innovation, Design and Technology
Mälardalen University
Sweden
{jorgen.lidholm,fredrik.ekstrand,lars.asplund}@mdh.se

Abstract

Current approaches to feature detection and matching in images strive to increase the repeatability of the detector and minimize the degree of outliers in the matching. In this paper we present a conflicting approach; we suggest that a lower performance feature detector can produce a result more than adequate for robot navigation irrespectively of the amount of outliers. By using an FPGA together with two cameras we can remove the need for descriptors by performing what we call spurious matching and the use of 3D landmarks. The approach bypasses the problem of outliers and reduces the time consuming task of data association, which slows many matching algorithms.

1. Introduction

Navigation, object detection and object recognition are fundamental problems in robotics - all which can be resolved with vision. The fundamental task in any of these applications is the reduction of the image complexity. In order to reduce the image information we need some sort of feature extraction such as line, edge or corner detection, or a combination thereof. Various approaches have been made on adapting these principle feature detectors into more complex, high-level detectors with different sets of descriptors. The idea is to increase the invariant properties of the detector and thereby increase what is by many viewed as the most important factor of the detector, the repeatability. However, in general, the more complex the detector the more computational heavy it becomes. When features have been extracted, the next step is to perform some sort of matching, either by tracking a feature in subsequent images or by matching in two cameras. The general idea is that if this is to be performed at a high frame rate (around 30 frames per second) it requires a high repeatability detector and a computationally light matching algorithm with minimal dynamic properties.

In this paper we present a somewhat divergent approach. We suggest that it is possible to produce a result adequate for navigating a mobile robot with a lower performance feature detector. We use reprogrammable hardware (FPGA) together with two cameras to generate a real-time, stereo-vision, feature detector and matching application. By using the motion of the robot we can reduce the problems associated with feature matching. The advantages of an FPGA is manifold; the parallel properties of an FPGA makes for a high-throughput, small footprint system, and the comparatively low power consumption makes it ideal for mobile applications.

2 Related work

Robot navigation is a well-explored subject with vision based navigation being where the current focus lies. The approach of using an FPGA for the system is also becoming widely adopted as it enables real-time image processing [1] [6], which is a crucial part in mobile applications [4]. For certain applications FPGAs are better suited than desktop computers due to their parallel structure. In [15] an FPGA implementation outperforms a PC by one order of magnitude for the SIFT detector [8]. The power of the FPGA is further shown in [12] where they are unable to run Harris corner detector in real-time on a computer with an Opteron processor running at 2.6 GHz. This advantage over personal computers is likely to remain as both technologies are evolving in a similar fashion performance-wise.

Navigation by vision requires matching of images, or rather features in separate images. Tracking features in real-time in subsequent images from a camera is not a trivial task, partially due to the fact that it requires a very stable feature detector with high repeatability [3]. The current feature detectors with the highest repeatability, such as SIFT [8] and SURF [2], create descriptors for each feature in order to simplify the matching task. Unfortunately, the high dimensionality of such a descriptor means that it is computationally intense [10].

All matching algorithms are faced with the correspondence problem, i.e., how to match corresponding features in two images without assigning any incorrect matches. The approaches differ, from *cross-correlation* to the *sum of squared differences*, but there will always be outliers, features not correctly matched, or not matched at all.

Many have tried to minimize the occurrence of outliers, and in [14] a comparison, and yet a new approach, is presented.

3. Experimental platform

We have designed a camera system intended to work as a general purpose research platform for FPGA based vision.

3.1. Image sensors

The system uses the MT9P031 5-megapixel CMOS camera sensor from Micron. The sensor elements are organized in a bayer pattern, i.e., the first line consists of green and red pixels and the second line consists of blue and green pixels, see figure 1.

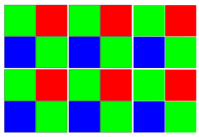

Figure 1. The Bayer pattern pixel layout with one row of green (light gray) and red (medium gray) pixels, and one row of blue (dark gray) and green pixels.

The pixels can be read in a number of ways. The readout follows that of the bayer pattern, however the order can be mirrored and pixels skipped for both the row and column. In skipping mode, a number of row-pairs and/or column-pairs are not sampled, i.e. skipped, thereby reducing the resolution and increasing the frame rate but preserving the field of view.

It is possible to combine the adjacent skipped pixels in order to reduce the effect of aliasing introduced by skipping. This is called binning and results in a more coherent/smooth image, than with solely skipping, but also in lower performance as all the pixel elements need to be sampled.

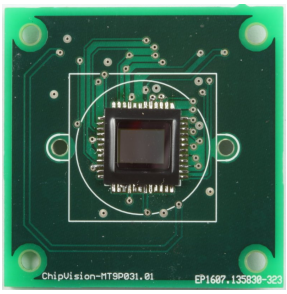

Figure 2. MT9P031 image sensor from Micron mounted on our carrier board (the lens is not mounted).

Additionally, one can specify what region of the image sensor to read, useful when only a limited field of view is needed and a higher frame rate desired. The imaging sensor is capable of running at 96 MHz, and the frame rate is dependent on the clock frequency and the frame size, i.e., the number of pixels read.

Table 1. Micron MT9P031 CMOS image sensor features [9]

MT9P031 features	
Color filter array	RGB Bayer pattern
Maximum data rate	96 Mp/s at 96 MHz
Power consumption	381mW at 14 fps full resolution
Pixel size	2.2μm×2.2μm
Maximum frame rates	
2592×1944	14 fps
1280×720 skipping	60 fps
640×480 with binning	53 fps
640×480 with skipping	123 fps

3.2. FPGA board

The FPGA board has a size of 70×55mm and is equipped with a Xilinx Virtex II XC2V8000 FPGA together with 256 Mbit flash, 512 Mbit SDRAM and a CPLD. The purpose of the flash memory is for storing FPGA configurations and it accommodates 8 different configurations. At power on the CPLD loads the FPGA according to the switch settings. The configuration selector is fitted on the Carrier Board (section 3.3) but may be overrun by for example a micro controller. See figure 3.

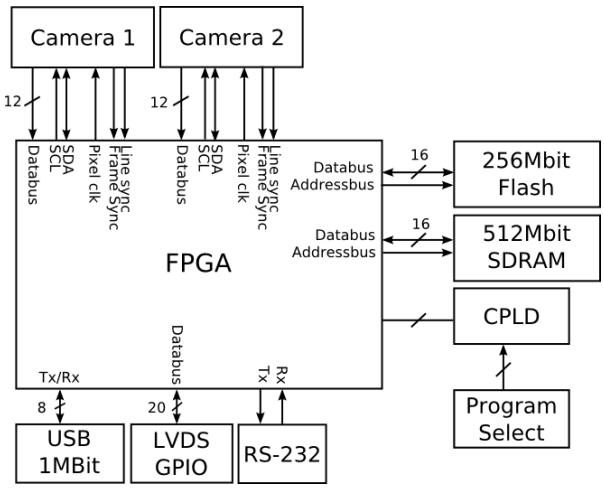

Figure 3. Block diagram of the camera system, two additional cameras are connected for a total of four cameras.

3.3. Carrier board

The carrier board has a size of 110×90mm and have four camera connections, with all signals, individual to

each camera and generated by the FPGA. Additionally, the carrier board incorporates a program selector, power supply, an USB controller, serial port and control-IO signals. It is also fitted with a FireWire connector for future extension.

4. Feature detectors

A feature can be a corner, line or any salient region which can be extracted from an image.

One of the first feature detectors was Moravec's corner detector [11], from 1977, which Harris et. al. improved in 1988 [7]. Since then, many other feature detectors have been developed with different qualities [13]. Most detectors are designed with repeatability in mind, although some are designed for other properties, such as speed [12]. Repeatability, as defined in [13], is an important property, however, for our application, localization accuracy and speed are paramount. Harris is still one of the most robust detectors available and this together with its speed when implemented on an FPGA makes it a suitable detector for this application.

4.1. Stephen and Harris combined corner and edge detector

In [13] the authors concluded that, among the tested feature detectors, (Foerstner, Cottier, Heitger, Horaud, Harris and Improved Harris), the improved version of the Harris corner detector performed best regarding repeatability and information content. The original implementation of the same detector was, however, not far behind.

Moravec's corner detector measures the variation in intensity in an image and looks for low self-similarity in a point. A corner is defined as a point with low similarity to the surrounding region in all directions, i.e., a point where the minimum change in intensity, in any direction, is large (above a certain threshold) [7].

According to Stephen and Harris, Moravec's detector, however, suffers from a number of problems which they try to correct with their combined corner and edge detector. In order to remove the anisotropy and noise of the discrete, rectangular window in which the variation is calculated, they introduce an analytic expansion about the shift origin together with smoothing with a Gaussian filter. By also taking into account the direction of shift they can produce a rotationally invariant detector that is not oversensitive to edges.

Stephen and Harris also introduce a response function in order to select isolated interest points, as opposed to simply classify the region as containing a potential feature. This response function, which includes a structure matrix calculated from image derivatives, indicates the quality of the detected feature and allows for the filtering out of less distinctive features with the use of a threshold similar to Moravec's.

4.2. FPGA implementation of Harris corner detector

We have a VHDL implementation of the Stephens and Harris combined corner and edge detector. It was originally implemented as a undergraduate thesis for an older vision system. We have adapted it to a new, larger FPGA, allowing us to increase the parallelism and thus improve the speed.

Some operations need to be performed sequentially for practical purposes. One of the most limiting factors of the FPGA is the number of multipliers available. Certain steps in the algorithm requires simultaneous multiplications, and the need for multipliers would surpass the available numbers if parallelled to the full extent. In order to save computational resources, the units needs to be "reused", i.e., not exclusive to a single task. Due to the fact that the corner detector measures the intensity in the image and not the saturation or color values, we need only measure the contribution in one point of the bayer matrix, i.e., the green pixel. We chose to use only one value per color quadrant and thus we only feed the corner detector with a new pixel every other column every other row. This leaves room for sequential operations on four clock cycles for every pixel.

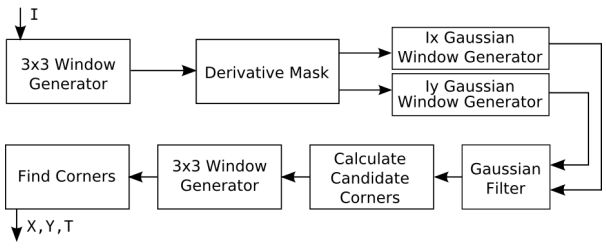

Figure 4. A block diagram of our VHDL implementation of the Harris corner detector.

The corner detector uses 3×3 and 5×5 pixel windows. This is the only buffering required, all other processing is performed as the pixel data arrives. In the block diagram in figure 4 our implementation of Harris corner detector can be seen. The process consists of 7 major internally piped blocks. The first block creates a 3×3 sliding window. When two pixel rows plus one pixel have been extracted from the camera the first window is passed on to the "Derivative Mask" block.

The derivative block calculates the intensity x- and y-gradients. These values, the first derivatives, are then passed onto the window generator for the multiplication/-Gaussian stage, which creates a 5×5 sliding window.

In the multiplication stage, the structure matrix is calculated and then run through a Gaussian filter. The Gaussian filter is constructed using shift operations, as opposed to multiplications, in order to save multipliers that can be used for either increased parallelization or multiplier-heavy postprocessing. The filter is not a true Gaussian function as the values are selected to enable shifting, but no performance degradation has been observed for the

approximation, which can be supported by [5] that shows that Gaussian weighting need not be the optimal weighting function.

The filtered value is then used in the response function and the result is fed to a new window generator. The last stage of the pipe filters the response value so that only the local maxima within the 3×3 sliding window generates a corner response, as long as it exceeds the current threshold.

5. Interest point location

An interest point is, what we call, a stereo matched feature that can be located in a coordinate system as a landmark, that a robot can use for navigation. In this section we describe how we can calculate the location of a landmark from two stereo matched features. The same procedure, in reverse order, can be followed to calculate the pixel coordinate at which a landmark should appear, given the robots current location and attitude.

We use the right-handed coordinate system with positive X to the right, positive Y in front and positive Z above.

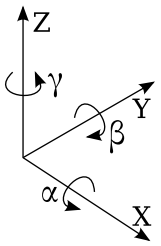

Figure 5. Right-handed coordinate system

The full definition of the robot absolute vector defines the position in three dimensions and the attitude in three dimensions (5.1). The robot center is located at the floor in the center of the robot in the x, y plane.

$$\mathbb{R} = (X_r, Y_r, Z_r, \alpha_r, \beta_r, \gamma_r) \tag{5.1}$$

Since the robot is moving in a controlled indoor environment without slopes, we can consider Z_r constant and zero. The same applies for α_r and β_r. The stereo camera rig has a fixed location on the robot and the constant relative vector of each camera is defined in (5.2), where n marks the camera, left or right. The vector is relative to the robot center. To simplify the stereo matching the $\hat{\beta}$ factor should be zero and the $\hat{\alpha}$ should be the same for both cameras, resulting in that line j in the left camera corresponds to line j in the right camera.

$$\underline{c}_n = \left(x_n, y_n, z_n, \hat{\alpha}_n, \hat{\beta}_n, \hat{\gamma}_n\right) \tag{5.2}$$

The absolute vector of each camera can be calculated by adding the relative camera vector to the absolute robot vector:

$$\begin{aligned}\mathbb{C}_n &= (X_r + x_n, Y_r + y_n, Z_r + z_n, \\ &\quad \hat{\alpha}_n, \hat{\beta}_n, \gamma_r + \hat{\gamma}_n) && (5.3)\\ &= (X_n, Y_n, Z_n, \alpha_n, \beta_n, \gamma_n) && (5.4)\end{aligned}$$

Now we have the absolute position of the cameras and the direction they are pointing in, the attitude.

Every pixel in an image corresponds to a two dimensional direction which can be calculated from the focal length of the lens f and the pixel separation on the camera chip P_{width} and P_{height}. The two angles θ and ϕ and an unknown length r form a polar vector (5.7).

(X_p, Y_p) denotes the pixel coordinate, with the camera center at $(0, 0)$, and Q is the transformation from pixel coordinates to a polar vector.

$$\theta = \arctan\left(\frac{X_p * p_{width}}{f}\right) \tag{5.5}$$

$$\phi = \arctan\left(\frac{Y_p * p_{height}}{f}\right) \tag{5.6}$$

$$Q(X_p, Y_p) = (r, \theta, \phi) \tag{5.7}$$

By using (5.5) and (5.6) we can find the angular distance between every pixel. The MT9P031 camera chip has a pixel separation of $2.2\mu m$ (table 1) but we are only sampling every second pixel column and row, thus doubling the pixel separation to $4.4\mu m$. The focal length of the lens is $4mm$. This results in approximately 1.1 milliradians per sampled pixel in both horizontal and vertical directions.

Lets consider the case where we know which feature in the left camera corresponds to which feature in the right camera. By forming a triangle with corners at the two camera centers and the third corner at the interest point with angles as seen in figure 6 we can calculate the distance of the two unknown triangle edges by using the law of sine, see equation (5.8-5.10). The camera separation is known and denoted S_c.

$$\lambda = \pi - \theta_l - (\pi - \theta_r) \tag{5.8}$$

$$\frac{\vartheta_n}{\sin(\theta_n)} = \frac{S_c}{\sin(\lambda)} \tag{5.9}$$

$$\vartheta_n = \frac{S_c * \sin(\theta_n)}{\sin(\lambda)} \tag{5.10}$$

Now we have the distance and direction to the interest point from each camera relative to the camera attitude, $(\vartheta_l, \theta_l, \phi_l)$ and $(\vartheta_r, \theta_r, \phi_r)$. By extracting the camera attitude as a polar unit-vector, and rotating it by the relative interest point vector we get the unit vector from the camera pointing at the interest point. To get the cartesian coordinate of the interest point we multiply the pointing vector by the length calculated in (5.10), convert it to a cartesian coordinate and add the absolute cartesian camera coordinate (5.14).

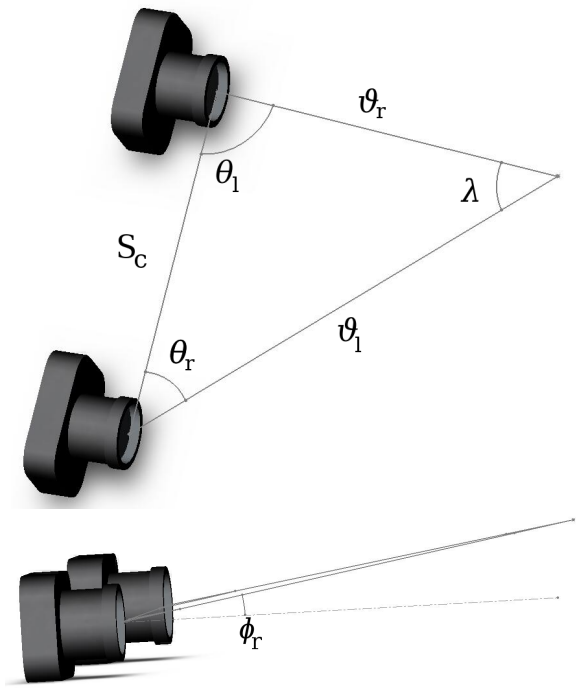

Figure 6. The angles from each camera to a feature point. θ for the left and right camera, the camera separation S_c and ϕ, which should be the same for both cameras.

C and P marks the cartesian and polar coordinate system respectively or a transformation between the two. The cartesian location of camera n.

$$C(\mathbb{C}_n) = (X_n, Y_n, Z_n) \tag{5.11}$$

The attitude of camera n as a polar unit vector.

$$P(\mathbb{C}_n) = (1, \alpha, \gamma) \tag{5.12}$$

The direction and distance to the interest point k.

$$P(\mathbb{I}_k) = (\vartheta_n, \phi_n, \theta_n) \tag{5.13}$$

The space location of the interest point.

$$C(\mathbb{I}_k) = C(\mathbb{C}_n) + C(\vartheta_n * (rot_{\phi_n,\theta_n} P(\mathbb{C}_n))) \tag{5.14}$$

The conversion from polar vector to cartesian coordinate requires the use of $\sin$ and $\cos$ as seen below. $\cos(\theta) = \sin(\frac{\pi}{2} - \theta)$ allowing a sin only implementation in the FPGA.

$$x = r * \sin\phi * \cos\theta \tag{5.15}$$
$$= r * \sin\phi * \sin(\frac{\pi}{2} - \theta) \tag{5.16}$$
$$y = r * \sin\phi * \sin\theta \tag{5.17}$$
$$z = r * \cos\phi \tag{5.18}$$
$$= r * \sin(\frac{\pi}{2} - \phi) \tag{5.19}$$

5.1. Image sequence feature tracking

To track features in an image sequence is not a trivial problem, feature extractors like Harris corner detector have minor problems with repeatability resulting in features disappearing and reappearing.

Tests has shown that a simple tracker, like nearest neighbor is not reliable enough [3]. To successfully track features in an image a more advanced algorithm is required, possibly where information of the feature neighborhood is known.

A factor which makes it even harder is that we have a resolution of 1.1milliradians per pixel which at one meters distance corresponds to approximately $1mm$, which can make minor vibrations result in large displacements of features in the image.

A common method of improving the matching performance is to use feature descriptors. Feature descriptors provide more information about a feature, by including neighborhood data. The descriptor makes the features more distinctive and unique. These descriptors simplify the tracking problem a lot, but require more computation in the feature extraction state but still does not solve the problem completely. A good example of how computationally intensive it can be is the SIFT algorithm [8], SIFT systems are often not managing more than a few frames per second.

We choose an approach to the stereo matching problem which does not require feature tracking in an image sequence.

In real-time applications, direct matching seems like the best approach, as it is desirable to match features on a real-time basis. However, if the frame rate is sufficiently high, intermediate matching can be more than adequate. Statistical approaches for handling outliers by confidence values are well-explored, but they are normally time-consuming and can be problematic when matching in a 2D environment [14]. By moving the matching part onto the 3D coordinates from a stereo-vision camera system, it is possible to eliminate the uncertainty of 2D pixel coordinates.

5.2. Spurious matching and landmark evaluation

To match a feature in the left image with a feature in the right image is known as the correspondence problem. A common approach is to use a correlation window around the features and with a statistical method calculate a matching score. The score with the highest value is the most likely to be the correct match. The matching score can be calculated with methods like, *cross-correlation*, *sum of squared differences* and χ^2 for example [14]. To successfully use statistical methods it is necessary to calculate the matching score for many different matching pairs to find the match with the highest possible score. It is also necessary to find the outliers or false matches.

Our approach is adapted for a real-time vision system where the data is processed as a stream. No image is stored as a whole, line buffers are however used.

A feature appearing at pixel row n in the left camera must appear, if existing, on row $n \pm m$, where $m = 1$ under the condition that the camera distortion is corrected and that the cameras are perfectly aligned. The horizontal limitations can be found by knowing the attitude of the cameras. The search window denoted $W_m(F_i)$ represents the maximum area in which a feature in the right image must be located to correspond to feature F_i in the left image.

By matching every feature F_i in the left image with every feature within $W_m(F_i)$ in the right image we get a set of possible landmarks $LMK(F_i)$. Within this set of 3D coordinates there can be only one that corresponds to the actual landmark, which one is unknown. We call this spurious matching. Instead of trying to find the correct stereo correspondences, we try to find which landmarks in the environment are the correct ones. While moving the robot, measuring the location of the robot using wheel based odometry, and continuously calculating the possible landmark location for every feature F_i, the reappearing landmarks are then put in a landmark database with an increasing confidence related to uniqueness and stability of the landmark location.

To rely on odometry can be risky because it is a relative measurement system with no point of calibration. Wheel slip can cause huge faults, which can be hard to recover from. For shorter distances, less than one meter, the accuracy provided by wheel based odometry should be sufficient. As soon as enough landmarks has been located with good confidence the odometry system can be used solely as a support system and is no longer required for the vision based navigation, which can be used for visual odometry.

When a number of landmarks has successfully been located it is unnecessary to try and relocate them in the manner described above. By predicting the robots location and attitude before each iteration, using for example a Kalman filter, we can find the pixel coordinate for each possibly visible landmark and exclude those features from the images. This reduces the amount of features in the images which need to be matched.

Another way of reducing the amount of possible matches is to adapt the discrimination level according to how many features was detected in the previous images in order to have a sufficient amount of features.

The camera system is a resource limited system. A navigation system like this will collect many landmarks, requiring large amounts of preferably volatile memory so that data can be retained during a power down. A robot always has a computer for controlling the high level strategy, taking actions on sensors and planning future strategies. The vision based navigation system presented here is supposed to work like an advanced sensor. The vision system can report all landmarks, confidently located in the environment, to the main computer which stores them in a database and sends them back to the vision system when they will reappear in the visual field. This approach allows the vision system to only keep a minor amount of landmarks in local storage, like block ram or SDRAM, which is available on the FPGA board.

Computational requirements.
Calculating the space location of a feature pair, as seen in (5.5-5.14), requires 25 operations. Harris extracts approximately 300 corners from a 320×480 pixel frame without being too cluttered. In average this means less than one corner per line, the maximum number of corners possible on a single line is $\frac{320}{3} = 106$, though very unlikely (see section 4.2).
A pessimistic number of matches per feature could be around 20, which would render in 6000 landmark calculations per frame. 25 operations on 6000 landmarks would result in 150'000 operations per frame, which is rather low.

6. Results

Our FPGA based stereo vision system is capable of real-time feature extraction, using the implemented Stephen and Harris combined corner and edge detector. To stereo match these features, for landmark location, is not a trivial problem. We present a novel approach which we call spurious matching allowing us to validate which matches correspond to real landmarks by moving the robot and extracting the features at different viewpoints. In the current implementation of Stephen and Harris combined corner and edge detector 75 out of 150 available multipliers are used, this could easily be reduced to 25 by sharing multipliers in the factorization step of the Harris algorithm. For performance results of the corner detector see tables 2 and 3. See table 4 for frame rates of Harris corner detector on our system.

Table 2. Computational performance of our implementation of Harris corner detector.

Op/Block	Calc of edge mask	Fact. and Gaussian filter	Calc. repsons function
Add	4	120	1
Sub	6	0	2
Shifts	0	75	0
Mul	0	75	3
Total	10	270	6

Table 3. Performance total of Harris corner detector at different frame rates

pixels/frame	fps	Instr./pix	Cameras	MIPS
148'800	27	286	2	2'298
148'800	34	286	2	2'894

Table 4. Performance of our implementation of Harris corner detector.

Frame size	Cam freq.	FPGA freq.	FPS
320×480	96MHz	100MHz	65 fps*
320×480	50MHz	100MHz	34 fps
320×480	40MHz	100MHz	27 fps

* Theoretical value which we have not been able to verify.

7. Future work

The proposed spurious matching algorithm has not been fully verified yet, there are several performance factors which need to be evaluated like, camera discrepancy, odometry precision and landmark localization accuracy.

Acknowledgements

This project is supported by Robotdalen. The authors would also like to acknowledge *Xilinx* for their kind donation of our FPGA's and design software tools, *Hectronic* for the design and manufacturing of our FPGA boards.

References

[1] P. Arribas and F.-H. Macia. FPGA board for real time vision development systems. *Devices, Circuits and Systems, 2002. Proceedings of the Fourth IEEE International Caracas Conference on*, pages T021–1–T021–6, 2002.

[2] H. Bay, T. Tuytelaars, and L. J. V. Gool. Surf: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *ECCV (1)*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer, 2006.

[3] A. Bissacco, S. Ghiasi, M. Sarrafzadeh, J. Meltzer, and S. Soatto. Fast visual feature selection and tracking in a hybrid reconfigurable architecture. In *Proceedings of the Workshop on Applications of Computer Vision (ACV)*, June 2006.

[4] D. Cardon, W. Fife, J. Archibald, and D. Lee. Fast 3d reconstruction for small autonomous robots. *Industrial Electronics Society, 2005. IECON 2005. 31st Annual Conference of IEEE*, pages 6 pp.–, 6-10 Nov. 2005.

[5] T. Cooke and R. Whatmough. Using learning algorithms to improve corner detection. In *Proceedings of the Digital Imaging Computing: Techniques and Applications (DICTA 2005)*, page 54, December 2005.

[6] T. H. Drayer, J. G. Tront, R. W. Conners, and P. A. Araman. A development system for creating real-time machine vision hardware using field programmable gate arrays. In *HICSS '99: Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences-Volume 3*, page 3046, Washington, DC, USA, 1999. IEEE Computer Society.

[7] C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[8] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[9] Micron, Boise, ID, USA. *MT9P031 Image Sensor, Product Brief*, 2006.

[10] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 27(10):1615–1630, 2005.

[11] H. Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, page 584, August 1977.

[12] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[13] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, Vol. 37(2):151–172, June 2000.

[14] P. Smith, D. Sinclair, R. Cipolla, and K. Wood. Effective corner matching. In J. N. Carter and M. S. Nixon, editors, *BMVC*. British Machine Vision Association, 1998.

[15] P. J. T.-J. M. Stephen Se, Ho-Kong Ng. Vision based modeling and localization for planetary exploration rovers. In *Proceedings of the 55th International Astronautical Congress 2004*, pages 1–11, 2004.