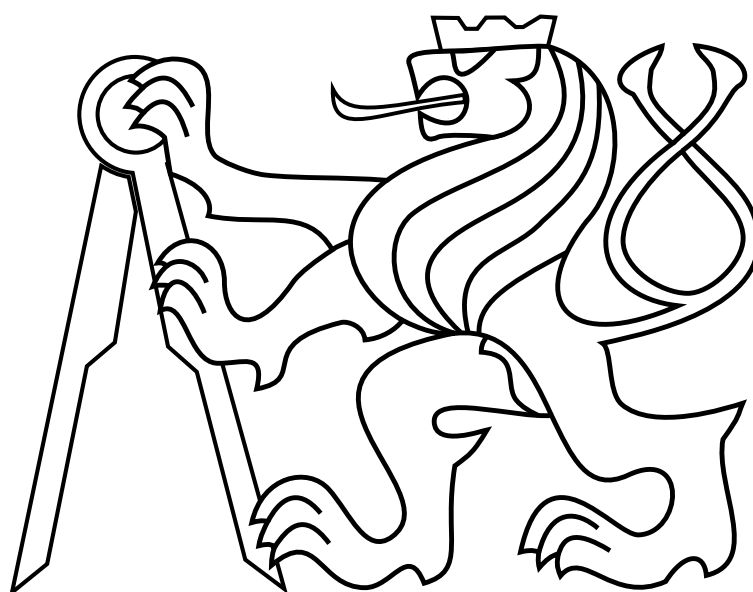


CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Bachelor thesis



Petr Čížek

Embedded module for image processing

Department of Cybernetics

Thesis supervisor: **Ing. Tomáš Krajník, Ph.D.**

BACHELOR PROJECT ASSIGNMENT

Student: **Petr Čížek**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Bachelor Project: **Embedded module for image processing**

Guidelines:

1. Learn about an embedded module for image processing used in the department of cybernetics.
2. Learn about 'teach and repeat' methods for mobile robot navigation.
3. Modify the FPGA module, that it can implement selected method.

Bibliography/Sources:

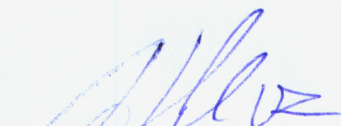
- [1] Krajník, T. - Faigl, J. - Vonásek, V. - Košnar, K. - Kulich, M. - et al.: Simple, Yet Stable Bearing-Only Navigation. Journal of Field Robotics. 2010, vol. 27, no. 5, p. 511-533. ISSN 1556-4959.
- [2] Šváb, J. - Krajník, T. - Faigl, J. - Přeučil, L.: FPGA-based Speeded Up Robust Features. In 2009 IEEE International Conference on Technologies for Practical Robot Applications. Boston: IEEE, 2009, p. 35-41. ISBN 978-1-4244-4992-7.
- [3] Pedre, S. - Krajník, T. - Todorovich, E. - Borensztein, P.: A Co-Design Methodology for Processor-Centric Embedded Systems with Hardware Acceleration Using FPGA. In Proceedings of the 3th Southern Programmable Logic Conference. Piscataway: IEEE, 2012, p. 7-14. ISBN 978-1-4673-0185-5.

Bachelor Project Supervisor: Ing. Tomáš Krajník, Ph.D.

Valid until the winter semester 2013/2014



prof. Ing. Michael Šebek, DrSc.
Head of Department



prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 18, 2012

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne.....23.5.2013.....

..........

Abstrakt

Tato práce předkládá návrh systému pro zpracování obrazu a následnou navigaci mobilního robotu specificky navržený pro architekturu programovatelných hradlových polí (FPGA). Základem práce je popis návrhu řešení, které vychází ze specifických vlastností FPGA architektury a je jím přizpůsobeno. Část zpracovávající obrazová data je rozdělena na dvě části: detektor významných bodů v obraze vycházející z algoritmu Speeded Up Robust Features (SURF) a deskriptor významných bodů založený na algoritmu Binary Robust Independent Elementary Features (BRIEF). Obě části jsou kompletně implementovány v logice hradlového pole. Navigační algoritmus je určen pro zpracování vestavným procesorem a vychází z algoritmu SURFnav vyvinutého na Fakultě elektrotechnické ČVUT. Začátek práce je věnován teoretickému úvodu do problematiky navigace mobilních robotů a strojového vidění. V závěru práce je vyhodnocen výkon detekční části algoritmu.

Abstract

This thesis propose a system design for image processing and mobile robot navigation specifically suitable for the architecture of the Field-programmable Gate Arrays (FPGAs). The image processing part of the design consists of the image feature detector based on the Speeded Up Robust Features (SURF) algorithm and the image feature descriptor based on the Binary Robust Independent Elementary Features (BRIEF) algorithm. The image processing part is completely implemented in the FPGA fabric. The navigation algorithm is designed as a software for the embedded processor of the module. It is based on the SURFnav navigation algorithm developed on the Faculty of Electrical Engineering of the CTU. This thesis also provides the reader with the background of the mobile robot navigation and image feature extraction methods. The image feature detector part of the design is evaluated in the end of this thesis.

Acknowledgment

I would like to express my thanks to my supervisor Tomáš Krajník. He taught me a lot and provided me with conditions and support for development of such a challenging bachelor project. I would like to thank my family for a full support throughout my studies at the Czech Technical University.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 2 | Vision based navigation | 3 |
| 2.1 | Localization | 3 |
| 2.1.1 | Incremental localization | 3 |
| 2.1.2 | Absolute localization | 4 |
| 2.2 | Mapping | 4 |
| 2.2.1 | Map based navigation | 6 |
| 2.2.2 | Map-building based navigation | 7 |
| 2.2.3 | Map-less navigation | 8 |
| 2.3 | Motion planning | 8 |
| 3 | Image feature extraction algorithms | 10 |
| 3.1 | Interest points detection | 10 |
| 3.1.1 | Speeded Up Robust Features | 11 |
| 3.2 | Interest point description | 13 |
| 3.2.1 | BRIEF descriptor | 13 |
| 3.3 | FPGA based image feature extractors | 14 |
| 4 | Method design | 16 |
| 4.1 | Field-programmable gate array (FPGA) | 16 |
| 4.2 | Image feature detector | 18 |
| 4.2.1 | Integral image generation | 19 |
| 4.2.2 | Hessian detector | 19 |
| 4.2.3 | Scale space construction | 20 |
| 4.2.4 | Interest point localization | 21 |
| 4.3 | Image feature descriptor | 22 |
| 4.3.1 | Descriptor generation | 22 |
| 4.3.2 | Orientation assignment | 22 |
| 4.4 | Navigation algorithm | 23 |
| 4.4.1 | Teach phase | 23 |
| 4.4.2 | Repeat phase | 23 |

| | | |
|----------|---|-----------|
| 5 | Implementation | 26 |
| 5.1 | Reference boards | 27 |
| 5.2 | FPGA configuration | 27 |
| 5.2.1 | Pixel I/O | 29 |
| 5.2.2 | Feature detector | 30 |
| 5.2.3 | Feature descriptor | 35 |
| 5.2.4 | Embedded processor and its peripherals | 39 |
| 5.2.5 | Implementation costs | 40 |
| 5.3 | Software description | 40 |
| 6 | Experiments | 42 |
| 6.1 | Stability test | 42 |
| 6.2 | Performance in the long term navigation scenario test | 43 |
| 7 | Conclusion | 45 |

List of Figures

| | | |
|----|--|----|
| 1 | Navigation tasks relations. Courtesy of [1]. | 3 |
| 2 | Map examples 1. Courtesy of [2]. | 5 |
| 3 | Map examples 2. | 6 |
| 4 | Path planning using RRT algorithm. Courtesy of [3]. | 9 |
| 5 | Integral image usage illustration. Courtesy of [4]. | 11 |
| 6 | Discretized Gaussian kernels and their approximations. Courtesy of [4]. . . | 12 |
| 7 | BRIEF descriptor binary tests examples. Courtesy of [5]. | 14 |
| 8 | FPGA architecture. | 17 |
| 9 | FPGA structure. | 18 |
| 10 | Comparison of convolution kernels. | 21 |
| 11 | Detected landmarks in the picture and the corresponding navigation his- togram. | 24 |
| 12 | Development platforms. | 26 |
| 13 | Digital video timing diagram. | 28 |
| 14 | Block diagram of the high-level FPGA architecture. | 29 |
| 15 | Feature detection chain architecture. | 30 |
| 16 | Simplified block diagram of the <code>PIXEL_BUFFER</code> core. | 32 |
| 17 | <code>PRELOC_MAX_FINDER</code> result value composition. | 33 |
| 18 | Zero Bus Turnaround SRAM memory timing diagram. | 36 |
| 19 | Simplified block diagram of the <code>FEATURE_BUFFER</code> core. | 37 |
| 20 | Timing diagram of SRAM memory bus utilization during display area. . . | 38 |
| 21 | The dataset capturing the seasonal changes. Location II. Courtesy of [6]. . | 43 |
| 22 | The dataset locations. Courtesy of [6]. | 44 |
| 23 | Heading estimation success rate.. . . . | 44 |

List of Tables

| | | |
|---|--|----|
| 1 | Description performance comparison. | 23 |
| 2 | Segment landmark map composition. | 24 |
| 3 | Reference boards features. | 27 |
| 4 | MASTER_CNTRL core user accessible registers. | 39 |
| 5 | Stability success rates. | 43 |
| 6 | CD content | 50 |

1 Introduction

For any autonomous vehicle the ability to navigate in its environment is essential. If we take evolution and nature as examples of perfect mastery of navigation we will realize that many of the living organisms navigate themselves by sight. This is also the main motivation for building systems that can navigate themselves by means of machine vision. With the growing computational performance of nowadays computers we have seen in recent years successes like first vehicles capable of autonomous navigation in an urban area [7] or a squad of flying quad-copters capable of synchronized movement in a confined space [8]. All these systems use powerful on-board or off-board computers for navigation. However sometimes the application imposes strict restrictions on hardware dimensions or power consumption of navigation system. For example the unmanned aerial vehicles (UAVs) are a typical platform which is restricted by its lifting capacity and a power consumption. These two demands make any use of conventional powerful computer platforms or hardware acceleration on graphics cards (GPUs) unusable because they are either too big to be carried by the small mobile robot or consume too much power which make them unsuitable for the long term operation on batteries. However if we take a closer look on the vision based navigation methods the major computationally demanding part usually consist in tasks related with the image processing. In the field of mobile robotics the local image feature extraction is especially utilize. The image feature extraction algorithms are computationally intensive however processes performed by them could be easily parallelized because they usually perform same computational operations on the different sets of data. This make the image feature extraction algorithms ideal for implementation on the Field-Programmable Gate Arrays (FPGAs). The FPGA architecture allows the programmer to built custom designs specifically suitable for given tasks.

In this thesis I would like to present a novel design for the FPGA architecture which is capable of real-time visual navigation by following natural landmarks. The design consists of the image feature detector and the image feature descriptor implemented in the FPGA fabric and the navigation algorithm implemented in software of embedded processor of the FPGA chip. The motivation behind the work on this thesis is to develop small, fast and reliable architecture capable of continuous video stream processing on low-end FPGA chips.

In early stage of my work at the Intelligent and Mobile Robotics Group I originally had to follow up on the work done by Jan Šváb in his master thesis [4]. He built a custom FPGA-based camera module specifically suitable for applications in mobile robotics and implemented a complete Speeded-Up Robust Features (SURF) algorithm [9] on the device. However it has shown that his solution is very consistent and provides only minimum space for any improvements. Few enhancements were introduced to the original design in order to allow the module to be used in navigation tasks. The results of this work is in review in the article [10].

During the work on the original design I suggested a new method of image feature detection specifically suitable for the FPGA architecture and tasks related with the mobile robot navigation. It was decided to implement and try the new design. This thesis summarizes the work done on this project which leads to the new design so far capable of real-time image processing.

This thesis is structured in four chapters. The first two chapters provide the reader with the background of the mobile robot navigation and image feature extraction methods. More attention is paid to the SURF algorithm and the SURFnav navigation method [11], [3], [1]. Chapter “Method design” describes the suggested new method of the image feature detection, the description and the navigation. Chapter “Implementation” describes the implementation of the image feature detector and the image feature descriptor in the FPGA fabric. Finally, the chapter Experiments contains evaluation of the image feature detector part of the design.

2 Vision based navigation

For any autonomous vehicle the ability to navigate in its environment is essential. The process of navigation could be roughly described as a process of determining suitable and safe path between the starting and the goal point. To achieve this objective the mobile robot has to solve tasks of localization, mapping and motion planning [12]. Although localization, mapping and motion planning are three different areas of interest and could be studied separately they are closely related and in the context of the intelligent navigation are often studied together. Figure 1 shows the simplified relations between these three tasks.

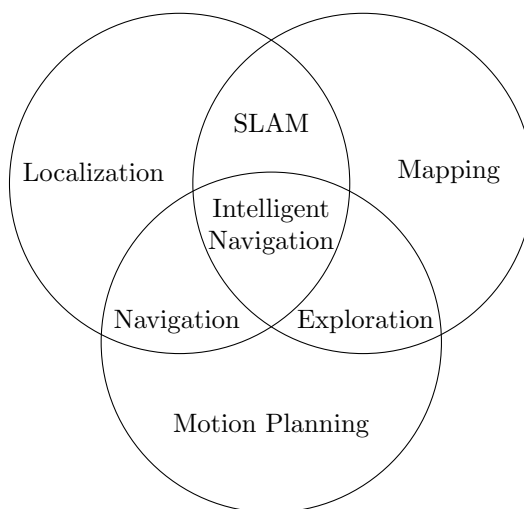


Figure 1: Navigation tasks relations. Courtesy of [1].

2.1 Localization

Localization is a process when the robot determines its position. Localization could be either incremental or absolute [13].

2.1.1 Incremental localization

Incremental localization, also called dead-reckoning, assumes that the starting position of mobile robot is known. During the navigation the robot estimates its current position from the previously known position and an increment of travelled distance. The biggest disadvantage of this method is that it accumulates position error over time. Most well known examples of dead-reckoning techniques are odometry or inertial navigation (i.e. navigation using accelerometers and gyroscopes). Due to accumulation errors these techniques

are not suitable for long term localization, however they are usually cheap to implement and easy to use so they are fairly popular. Examples of purely vision based incremental localization methods would be visual odometry or systems based on optical flow. Visual odometry (e.g. [14]) is a process of determining robot's position using sequential camera images to estimate the distance travelled. Optical flow is a principle used by optical mouse for estimating speed and heading. On the same principle works for example the PX4FLOW Smart Camera developed by 3D Robotics which uses sum of absolute differences (SAD) method for optical flow determination between two consecutive images [15]. The advantage of visual odometry and optical flow systems is that they provide the enhanced navigation accuracy for robots using any type of locomotion on almost any surface.

2.1.2 Absolute localization

Absolute positioning means that the mobile robot has no information about its starting position and has to estimate its position in a global coordinate system. This method usually relies on the map of the environment in which the navigation system must construct a match between the observations and the expectations. Typically the mobile robot estimates its position relatively to the landmarks in the area. These landmarks could be either natural or artificial [13]. Natural landmarks are usually objects which are naturally present in the environment (like walls, corners, etc.). The position of artificial landmarks, which are placed manually in the environment in order to improve navigation accuracy, is usually known in advance. By vision based localization it could be various blobs or patterns whose detection in the image is more easier than the detection of natural landmarks. Another examples of artificial landmarks are active beacon systems. Beacon systems always use triangulation or trilateration for localization relatively to three or more transmitters which actively transmits signals about its position. Beacon systems consist of beacons mounted at known positions in the environment and a receiver on board of the mobile robot (or vice versa). Examples of absolute positioning beacon systems could be global positioning system (GPS) or Ubisense system [16].

2.2 Mapping

The problem of localization is very closely related to mapping. Systems that use vision for navigation could be roughly divided into three main groups depending on whether they use map of the environment for navigation or not [17]. These three groups are

- map based navigation,
- map-building based navigation,
- map-less navigation.

In the first two cases the mobile robot localization is based on the map of its environment. This model (map) represents knowledge about the environment. Maps could be generally divided into four main groups [1], [18]: sensory, geometric, topological and landmark maps.

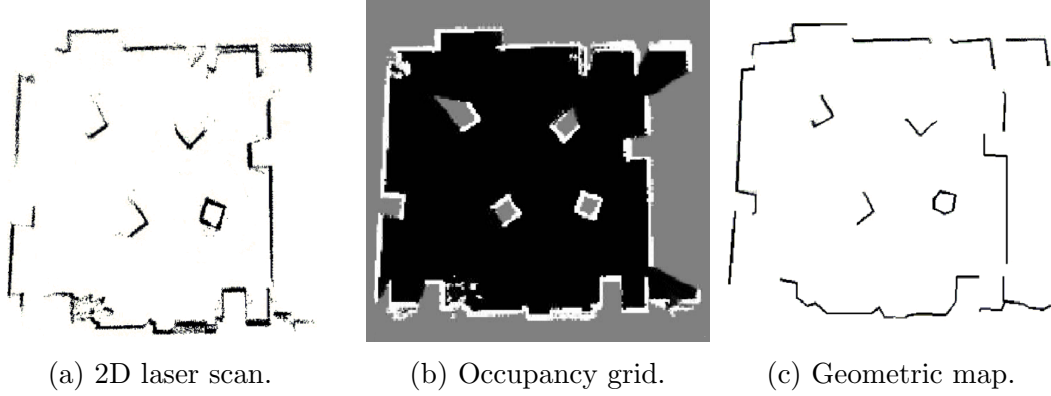
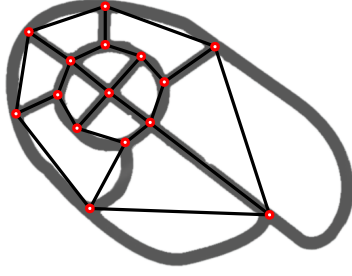


Figure 2: Map examples 1. Courtesy of [2].

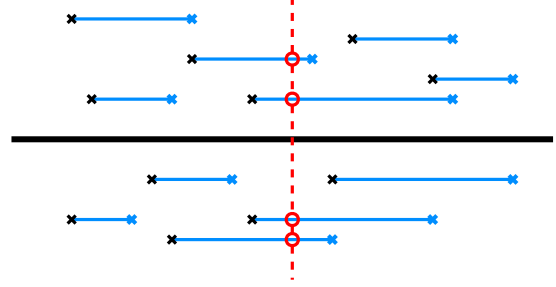
Sensory maps are only appropriately represented and saved sensory data. Sensory maps are quite simple to construct but they are always memory demanding particularly when the long term navigation is performed. The best example of sensory map is a point cloud produced by laser scanner. An example of the 2D point cloud is in Figure 2a. Point cloud data could be further simplified in a form of occupancy grid. Occupancy grids represent the environment by a grid consisting of cells where each cell contains information about whether the accordant place is occupied or not. Figure 2b shows the occupancy map of a room with three obstacles the black color corresponds to traversable area, white color represents detected obstacles. In the figure it can be seen that occupancy grid maps also rely on the position of the mobile robot. Areas which are out of sight or has not been mapped yet are considered occupied (depicted by gray color).

Geometric maps are usually built from geometrical primitives such as lines or simple bodies. These maps bring more abstraction than sensory maps and therefore occupy less memory. They are usable for example in indoor or urban environments where a lot of flat surfaces are present. Figure 2c shows the 2D geometric map of a room where obstacles and walls are approximated by line segments. An example of geometric mapping of the urban area is in the paper [19] where the authors used laser scan data of a large scale urban area and built the map from plane primitives.

Topological maps are basically graphs. Nodes represent the most characteristic places of the environment and edges represent routes. Information about how to navigate from one node to the another could be bound to edges. An example of topological



(a) Topological map. Courtesy of [3].



(b) Landmark map (SURFnav algorithm).

Figure 3: Map examples 2.

map would be a map of public transport system. Another example is in Figure 3a where a topological map of sidewalks in the park is depicted. An example of visual topological mapping is in the paper [20].

Landmark maps are used frequently in the field of mobile robot visual navigation. Landmark map contains information about landmarks which could be observed from given location. Landmarks should be salient points or areas in the image which are easily detectable, recognizable and traceable. Localization and navigation is done by detecting landmarks and matching them with those from the map. In Figure 3b there is a possible representation of the topological map. Black line represents the mobile robot path. Black points represent individual detected features and blue lines correspond to feature tracking during the robot's movement. The dashed red line corresponds to the current position of the mobile robot. This implies that the mobile robot should observe image features which are marked by red circles from its current position.

2.2.1 Map based navigation

These are systems that use map for navigation. This map is one of aforementioned types. There are two types of map based navigation systems [17]. The first one are systems that need a complete map of an environment before the navigation starts. The second type of map-based navigation techniques are those called “teach and repeat” techniques. The map is not provided to the robot in advance but it is built by the robot during the first guided pass through the environment. In the field of visual navigation it is mainly the sensory or the landmark map.

Paul Furgale and Timothy D. Barfoot published several papers on the long term teach and repeat methods. In the article [21] they presented solely visual teach and repeat method

based on the stereovision and tracking of image features. During the teaching phase the mobile robot is logging stereo images which are subsequently postprocessed into a series of overlapping submaps. During the repeat pass the mobile robot uses the database of submaps to repeat the route. The system can start at any place along the pre-learned path. The method combine topologically connected key frames (submaps) with a controller that attempts to drive the robot to the same viewpoints along the pre-trained path (visual odometry). It means that during the repeat phase the algorithm interleaves localization and visual odometry for path following. In the paper [22] they extended previously described solely visual method with laser scanners for motion estimation.

Article [23] presents a simple form of teach and repeat navigation. It utilizes a map consisting of salient image features remembered during a teaching phase and a simple left-right turn algorithm based on tracking image features for mobile robot steering during the repeat phase.

Another teach and repeat method called SURFnav [11] was developed on the Czech Technical University. The SURFnav navigation is based on the monocular vision supported by the dead-reckoning. The dead-reckoning techniques are separately not suitable for the long term localization because they accumulates position error. Therefore the mobile robot uses the vision for estimation of its position and heading. In the article [11] is mathematically proved that the heading correction itself can suppress odometric error keeping the position error bound and is sufficient for the long term robot localization. Moreover the algorithm's computational complexity doesn't grow together with the map growth which makes the SURFnav algorithm well suitable for long term navigation and embedded platforms with limited computational power.

During the teach phase the mobile robot is guided along straight line segments through the environment, measuring distance travelled by dead-reckoning and composing a landmark map of the environment. During the repeat phase the mobile robot drives the individual segments as follows. At each segment start the robot is turned to the desired direction according to compass value. During the traversal of the segment the heading of the robot is corrected based on matched visual features from the map and from the current observation. This method is called visual compass. The end of the segment is recognized according to traveled distance, which is measured by odometry.

Other examples of visual teach and repeat algorithms could be found in literature. In the paper [17] there is a brief overview of these methods.

2.2.2 Map-building based navigation

These are systems that build a map of the environment as they traverse through it. It means that they are self-localizing the robot in the environment simultaneously during

the map construction. This is called Simultaneous Localization and Mapping (SLAM) [24]. However this procedure leads to problems that are directly connected with the nature of such algorithms. On the one hand the map building of the environment depends on accurate localization of the robot but on the other hand the robot localizes itself within that map. This leads to the observation that localization and mapping errors accumulate over time and motion. SLAM is therefore defined as a problem of building a model leading to a new map, or repetitively improving an existing map. A lot of examples of SLAM algorithms could be found in literature (e.g. comprehensive overview of visual SLAM methods is in the paper [17]).

2.2.3 Map-less navigation

These are systems that do not use any map for navigation. Therefore they are also called reactive navigation systems. Navigation is only a reaction on current observation of the environment. Map-less navigation systems mostly include reactive techniques that use visual clues derived from the segmentation of an image, optical flow, or visual odometry for the navigation. Typical representative of these methods is GeNAV system [25] developed by Intelligent and Mobile Robotics Group. GeNAV is a system for unstructured roads following and crossroads recognition based on the texture segmentation of an image. The algorithm seeks the edges of the unstructured road and navigates the mobile robot in its center. If it detects the crossroad it informs the control computer.

2.3 Motion planning

Motion planning is a process of finding suitable and safe path between the start and the goal positions. Motion planning is usually based on results of the localization and the mapping.

By the map-based and the map-building based navigation it depends on the map. By the sensory or the geometric map the navigation can be performed for example by means of the Virtual Force Fields [26], where the goal point produces virtual attractive force and every obstacle produces virtual repulsive force, or by means of the Rapidly-exploring Random Tree (RRT) [27]. The RRT algorithm is capable of path planning for mobile robots respecting their kinodynamic constraints. In Figure 4 there is an example of path planning using RRT algorithm.

Navigation on topological maps is quite simple because it can use any graph traversal algorithm to reach the goal (e.g. A^* algorithm or the greedy algorithm)

By the Teach and Repeat methods the primary motion planning is provided by the human during the first guided pass through the environment.

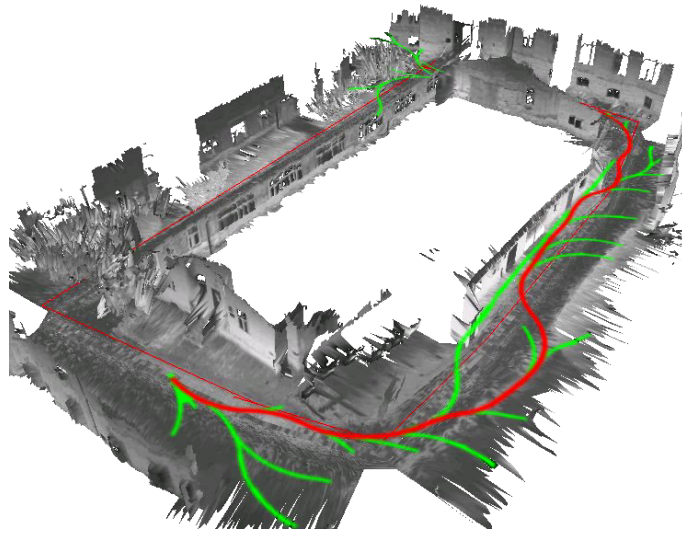


Figure 4: Path planning using RRT algorithm. Courtesy of [3].

Specific kind of motion planning is exploration, when the mobile robot plans its motions in order to explore specific area. By exploration the goal is to get the mobile robot in place where it can obtain new information of the environment.

Another field of motion planning is related with formation control [28], swarm robotics [8] and agent technologies [29] where the motion planning algorithms need to plan the suitable path for more mobile robots.

There are also algorithms which do not implement any higher intelligence for motion planning. Typical example are various household cleaning robots which are not aware of the shape of the area and perform only simple random reactive navigation inspired by insect behaviour.

In conclusion the efficiency of motion planning algorithms is generally influenced by their ability to retrieve the knowledge about their environment and the ability to process these knowledge.

3 Image feature extraction algorithms

Image feature extraction is considered to be a high-level image processing technique, which performs operations over an image in order to extract local features, which could be either points, regions or edge segments. Local features are areas of the image, which are expected to be reliably and repeatably detectable, preferably from a wider range of camera positions. They are used in many machine vision applications such as object recognition, robotic mapping and navigation, image stitching, 3D reconstruction or video tracking.

Image feature extraction always consists of two phases: the detection and the description. The interest points detection is a process which purpose is to identify the salient points in the image. Its input are the image data and output are locations of interest points in the picture optionally together with some more information about the located point (e.g. scale, contrast, etc.) The interest point description is a process when the surrounding of the detected interest point is described in order to allow the feature matching. The feature matching is a process when the description of two points is compared in order to determine whether these two points corresponds to the same feature of the environment.

This chapter provides the reader with basic overview of commonly used feature extractor algorithms with focus on interest point detectors and related descriptors which are used in robotic navigation. More attention is paid to the Speeded Up Robust Features (SURF) algorithm, which is a part of the original implementation [4]. Separate part discusses known implementations of image feature extractors on the FPGA architecture.

3.1 Interest points detection

Interest points detection is a process of feature extraction, which purpose is to identify interest points in the image. A lot of algorithms have been proposed on this topic. A comprehensive overview of image feature detector algorithms is in article [30].

One of the first interest point detectors was the Moravec operator introduced in the year 1980. After that the feature detection has been a growing field in the computer vision. Moravec operator works mainly as a corner detector defining corner as a point with low self-similarity. It measures the differences between a sliding image window and windows shifted in several directions. This method is called auto correlation.

The Harris corner detector [31] is based on the Moravec operator but it is more robust to noise, intensity and rotation changes. It uses a Gaussian to weight the derivatives inside the sliding window. Several Harris-based algorithms was proposed mainly to deal with the low scale invariance of the original detector. Their overview can be found in the article [30] One of the most robust image feature detectors is the Scale Invariant Feature Transform

(SIFT) detector [32] that apart from detecting features proposes a descriptor that is invariant to scale, rotation and illumination changes. It uses the difference of Gaussians for image feature detection. However fully implemented SIFT is highly computational demanding.

3.1.1 Speeded Up Robust Features

The original Speeded Up Robust Features (SURF) algorithm as proposed by Bay et al [9] relies on estimation of Gaussian and Haar wavelet filter responses by box filters. It consists of four stages: integral image generation, interest point detection, orientation assignment and descriptor generation.

Integral image generation This is a key component of SURF algorithm because it significantly reduces memory and computational demands. When using integral image it is necessary to perform only four read operations in order to calculate rectangular area integral of any size. The integral image is calculated using the equation 1.

$$I_{\Sigma}(x, y) = \sum_{i=0}^x \sum_{j=0}^y I(i, j) \quad (1)$$

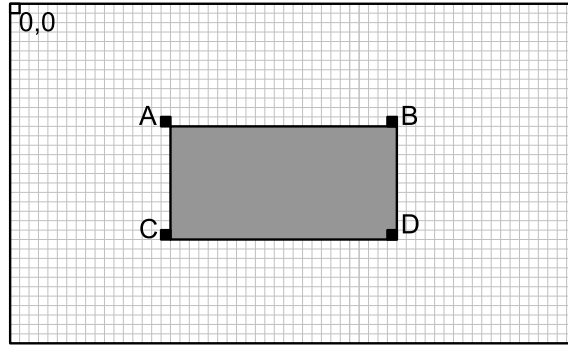


Figure 5: Integral image usage illustration. Courtesy of [4].

The usage of integral image is shown in figure 5, where the integral over the gray area is calculated using the equation 2.

$$I_{\Sigma}(ABCD) = I_{\Sigma}(A) + I_{\Sigma}(D) - I_{\Sigma}(B) - I_{\Sigma}(C) \quad (2)$$

Interest point detection The second stage identifies interest points by finding local maxima among determinants of Hessian matrices. Equation 3 shows definition of this determinant for a general two dimensional function f .

$$\mathcal{H}(x, y) = \begin{vmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{vmatrix} = \frac{\partial^2 f}{\partial x^2} \frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y} \right)^2 \quad (3)$$

In the case of machine vision the two dimensional function actually represents the image luminance. In the original SURF the second order partial derivatives are replaced by convolution of the image with derivatives of appropriate Gaussian kernels. Thus, the equation (3) becomes:

$$\mathcal{H}(x, y, \sigma) = \begin{vmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{vmatrix} \quad (4)$$

Where $L_{xx}(x, y, \sigma)$ represents a convolution with a second order derivative of a discretized twodimensional Gaussian of variance σ . The SURF detector approximates the Gaussian kernel responses L with box filter responses D . Figure 6 shows the comparison of the original Gaussian kernels L_{xx}, L_{yy}, L_{xy} with their box filters approximations D_{xx}, D_{yy}, D_{xy} . Convolution of the image with these kernels is calculated much faster using the integral image. The box filter approximation allows to rewrite equation (4) as $\mathcal{H} \sim D_{xx}D_{yy} - (0.9D_{xy})^2$. By the SURF algorithm the coefficient of 0.9 represents the weighting mechanism which compensates the box filters approximation.

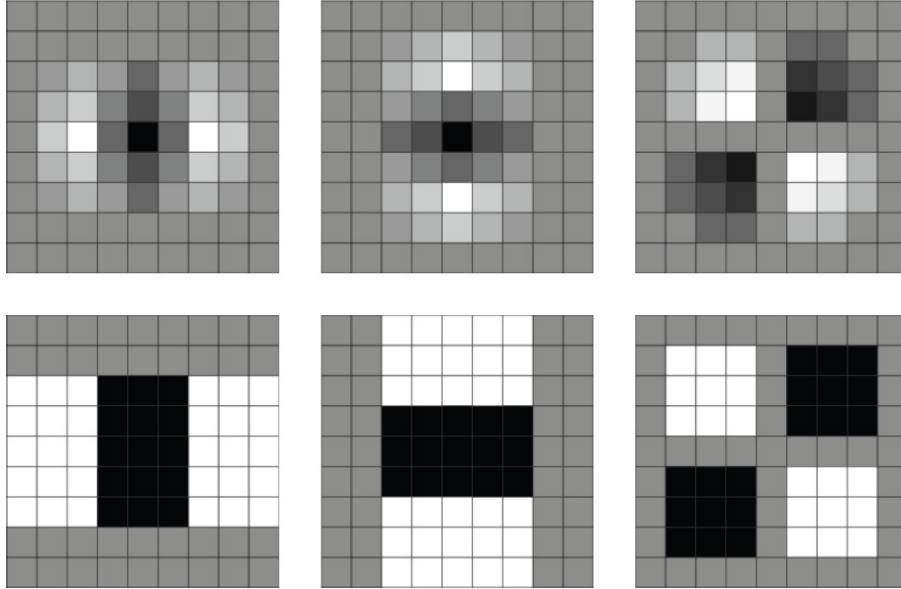


Figure 6: Discretized Gaussian kernels and their approximations. Courtesy of [4].

To achieve scale invariance the algorithm uses filters with multiple sizes. This creates a three dimensional space of determinant results called the scale space. After estimation of the Hessian determinants on all the scales for the whole image their local maxima are extracted. Located local maxima are then compared with predefined threshold. The point is declared an image feature if it passes both conditions.

Orientation assignment This phase is connected with the descriptor generation. To achieve the orientation invariance, which is important in many machine vision applications, the interest point is assigned a “dominant direction”. This direction is calculated from the responses of Haar wavelet filters centered around the interest point.

Descriptor generation The original SURF descriptor is a 64 element wide vector of floating point numbers calculated from a square area neighbouring the interest point rotated in the direction of the dominant direction. This neighborhood is divided in 16 equal sub-squares, which are regularly sampled by Haar wavelet filters. Horizontal d_x and vertical d_y Haar wavelet responses are weighted by a 2D Gaussian function centered at the interest point. The resulting values within each sub-square are summed to form a vector $\sum dx; \sum dy; \sum |dx|; \sum |dy|$. The vectors of all sub-squares are chained to form a vector and then normalized in order to obtain final descriptor. The SURF descriptor contains also the sign of the interest point’s Hessian value in order to simplify feature matching. Since the SURF detector usually finds interest points on blob-like structures, this sign distinguishes light blobs on dark background from the opposite case.

3.2 Interest point description

The interest point description is a process when the surrounding of the detected interest point is described in order to allow the feature matching. The resulting descriptors can be mutually matched using their euclidean distances (or other measures). Many algorithms features their own descriptor (e.g. SIFT, SURF) however there are also descriptors which can be used with almost any feature detector. The original SURF descriptor was presented earlier in this chapter. However it takes a lot of computation operations in floating point numbers to assemble it. This is not very good for embedded applications because specialized hardware funds need to be utilized by microprocessors or microcontrollers in order to be able to process floating point numbers efficiently. Therefore some other descriptor which is easy to construct was searched for.

3.2.1 BRIEF descriptor

The Binary Robust Independent Elementary Features (BRIEF) descriptor was proposed in paper [5]. It is a fast to implement easy to construct image feature descriptor which uses pairwise intensity comparison of pixels inside an image patch which surrounds located feature. These comparisons form a set of unique binary tests which are subsequently stored into an n_d -dimensional bit vector. In the original paper the n_d constant takes values of 128, 256 and 512. Several different approaches to choosing the test locations are proposed in the paper. However it has shown that completely random set of binary tests performs better than any artificially created set of tests. In Figure 7 there is an example of two random

sampling BRIEF and the artificially created BRIEF descriptor. The descriptor similarity can be evaluated using the Hamming distance, which is very efficient to compute.

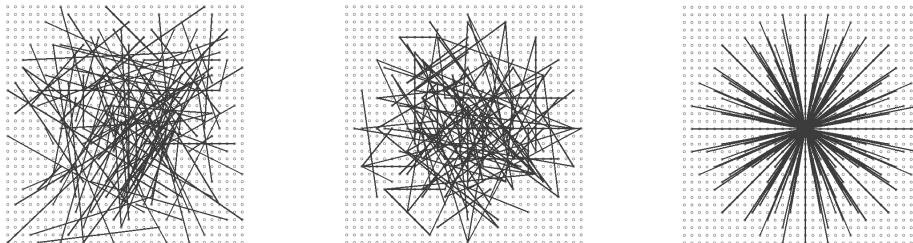


Figure 7: BRIEF descriptor binary tests examples. Courtesy of [5].

Several improvements were proposed to the BRIEF descriptor since the original paper was published. Mainly improving the descriptor's rotation invariance. Binary Robust Invariant Scalable Keypoints [33] (BRISK) is a scale and rotation invariant version of BRIEF which uses a deterministic comparison pattern. Oriented FAST and Rotated BRIEF [34](ORB) is another attempt to achieve a scale and rotation invariant BRIEF. It uses FAST (Features from Accelerated Segment Test) as a keypoint detector. By the same authors of [5] there is an improvement of original method D-BRIEF [35] which uses discriminative projections for descriptor construction.

3.3 FPGA based image feature extractors

In the recent years the popularity of the FPGA based image processing has grown rapidly. A lot of articles on image processing by FPGA was published. This section lists the known image feature extractors implemented in the field programmable gate arrays.

From the FPGA SIFT implementations there are implementations by Bonato et al [36] and Feng-Cheng Huang et al [37]. Bonato et al presented a complete SIFT implementation for the Altera Stratix II FPGA capable of feature detection on images with the resolution of 320×240 pixels in 33 ms. However they implemented the descriptor calculation in the soft-core processor with throughput of one descriptor per 11.7 ms which significantly restricts the number of detected features by the real time applications. Feng-Cheng Huang et al proposed an architecture for SIFT feature extraction which is capable of feature detection on the images with the resolution of 640×480 pixels in 3.4 ms and of the feature description in 0.03 ms per feature. However the article doesn't specify the utilized FPGA platform and it can be only roughly concluded that the design occupies 1.3 million logic gates.

The first published FPGA acceleration of SURF is [38]. Another implementation was proposed by Bouris et al [39]. They presented an implementation capable of processing the

images with the resolution of 640×480 pixels at 56 frames per second. However they perform only the feature detection and orientation assignment and not the feature description. From their paper the number of processed features is also unclear. Another implementation was proposed by Battezzati et al in the article [40]. Their approach uses massive parallelization of the whole algorithm in two Xilinx Virtex-6 FPGAs achieving feature detection on the images with the resolution of 640×480 pixels in 3.0 ms and the feature description in 0.01 ms per feature. However these results are achieved at the cost of extreme utilization of the FPGA fabric. Last known implementation of the FPGA SURF was proposed by Schaeferling et al [41]. Their Flex-SURF implementation is flexible in image size processing and features a specialized memory access management. The descriptors are computed in the embedded hardcore processor.

4 Method design

This chapter intends to provide the reader with the description of developed navigation algorithm. I originally had to follow up on the work done by Jan Šváb in his master thesis [4]. During his studies Jan Šváb developed an implementation of original SURF algorithm on FPGA and built a specialized hardware module suitable for the machine vision applications. However it has shown that his implementation was very consistent and in the hardware (FPGA configuration) part there is not much to improve. Partly because the FPGA fabric was almost all used by the original implementation. Partly because the hardware description language in which the implementation is written is almost impossible for reverse engineering. This is caused by the fact that this language is not interpreted by processor line by line but only specifies the dataflows through programmable logic which means all the actions are done in parallel. Room for improvement has shown to be in the software and in the minor improvements which make the minimodule more suitable for use as a self-contained navigation platform [10]. These improvements consisted in connecting auxiliary cores to the original design which perform control of the mobile robot, odometry reading and communication with the secure digital (SD) memory card.

Therefore I have developed a new algorithm for the image feature extraction specifically suitable for FPGA chips and realtime mobile robot navigation. In this chapter I would like to describe the algorithm and explain why I implemented the individual parts of the algorithm in a specific way. The algorithm consists of the image feature detection, the image feature description and own visual navigation. The image feature detection part is custom designed and it is highly optimized for use on the FPGA platform. In principle it is based on the original implementation but it is simplified so it can achieve the real time performance. The image feature description part uses BRIEF descriptor for description of interest points. The navigation algorithm is based on the SURFnav algorithm.

The first part of the chapter describes the architecture of FPGA chips which is necessary for definition of FPGA strengths and weaknesses. The second part describes the developed image feature detector. The third part is about used image feature descriptor and the last part is about the navigation algorithm.

4.1 Field-programmable gate array (FPGA)

The field-programmable gate array (FPGA) belongs to the family of semiconductor devices called Programmable logic devices (PLD). PLD is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of the manufacture. Before the PLD can be used it must be programmed.

FPGA contains the programmable logic components called Configurable logic blocks (CLBs) arranged in an array and a hierarchy of reconfigurable switches that can rearrange the interconnections between the logic blocks. Typical structure of this architecture is in

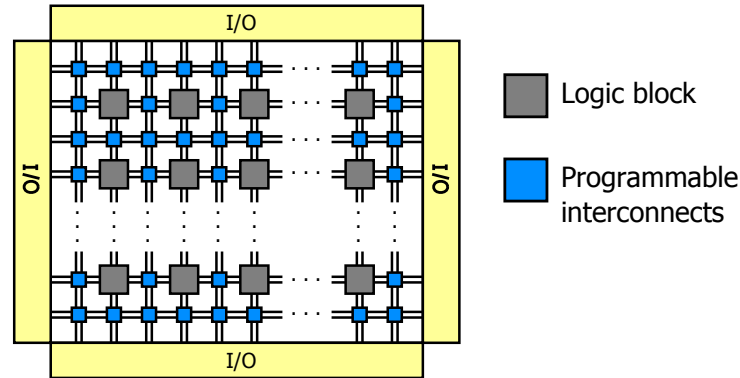


Figure 8: FPGA architecture.

Figure 8. Single CLB usually consists of a few logic cells. A typical cell consists of a configurable switch matrix with 4 or 6 inputs called Look-up table (LUT) which implements combinatorial logic, D-type flip-flop gate which implements sequential logic, full adder and some circuitry which determines global behaviour of the logic cell. Today's FPGAs can have up to several millions of logic cells.

Another important part of nowadays FPGA chips are clock management and generation circuitry (PLLs, DCMs), embedded memory blocks (block RAMs) and embedded multipliers. These components allow synthesizing designs with higher speed, clocking or memory demands. A recent trend is to incorporate traditional FPGAs with the embedded microprocessors and related peripherals to form a complete “system on a programmable chip”. Therefore we can meet devices with integrated hardcore processors and various endpoints for high-speed communication buses (e.g. Ethernet, SATA, PCI Express). An example of such architecture is depicted in Figure 9 which shows the structure of Xilinx Virtex-5 family chip.

The FPGA configuration is generally specified using the hardware description language (HDL) or the schematic design. Programming the FPGA is very different from the classical programming. Classical program runs on a processor and it is interpreted line-by-line. In contrary to the FPGA programming where the code only specifies the dataflows through the FPGA fabric. But this actually gives the programmer the ability to develop a specialized and highly optimized design which can outperform almost any conventional design. Moreover this optimization usually leads to the possibility of using slower clock and therefore reduce the device's power consumption.

When the design is ready to be implemented on FPGA chip, using an electronic design automation tool, a technology-mapped netlist is generated. Netlist can then be fitted to

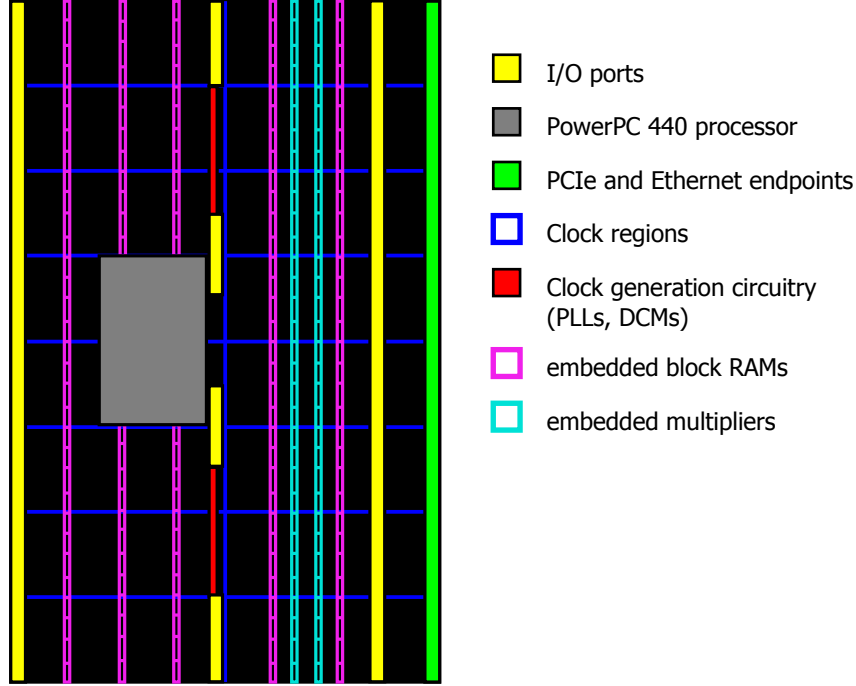


Figure 9: FPGA structure.

the actual FPGA architecture using the place-and-route process. It is also a good practice to write the simulation files for every design entity. It helps with the verification of design functionality and with the testing of the design for any unexpected situations.

The main drawbacks of the FPGAs are their cost and the time needed for development of an application. Compared with microprocessor designs FPGAs can offer true parallelism and the opportunity of designing fully custom and optimal designs. However learning to use or design complex FPGA systems is more challenging and also the debugging of the design is more difficult than by software approaches. Another issue is connected with the HDL programming because every developer uses his own coding style, making it almost impossible for reverse engineer of the code. This drawback can be partially suppressed by using and respecting the programming methodologies such as [42], [43].

4.2 Image feature detector

The image feature detector developed for purposes of this thesis is derived from original SURF detector (see chapter 3 for details) but it is highly optimized for the use on FPGAs. First of all I would like to present a closer look on SURF algorithm and comment the

individual parts from the view of FPGA programmer. I would like to provide the reader with the conclusions which lead me to implement the design the way I did.

4.2.1 Integral image generation

The whole algorithm starts with the generation of an integral image (see chapter 3 for details). The calculation of integral image is very simple and it is performed similarly on FPGA and in software. It consists of an array where the values of the integral image from previous image line are stored and of a sum register. The calculation is performed so that with every pixel the integral image value is calculated as a sum of value of sum register which sums luminance of pixels from the beginning of the image line, and a value of integral image of pixel right above the current pixel. The generation of integral image has only one issue. For example if we want to calculate an integral image of a picture with the resolution of 1024×768 pixels in grayscale (8 bit per pixel) the integral image value binary representation can occupy 28 bits so the whole image occupies 3.5 times more memory than the original image. So if we don't want to lose any data, we need to dimension all the FPGA logic the way it can handle such numbers. This consequently leads to higher utilization of the FPGA fabric. Another problem with integral image is its storage. Despite the modern FPGAs incorporate embedded block RAMs the size of an integral image on standard resolution frame can't be stored in the device. To my best knowledge there are two implementations of SURF algorithm on FPGA which rely on storing an image in embedded block RAMs. Bouris et al [39] presented an implementation which can process images with resolution only up to 640×480 pixels where the integral image storage occupies 72% of block RAM resources in the Xilinx Virtex-5 device. Battezzati et al [40] presented a short paper on the SURF architecture which also relies on storing the integral image data inside the FPGA. Their solution occupies 90% of two high-end Xilinx Virtex-6 FPGAs. Based on the above it is obvious that for the price of higher speed extreme utilization of FPGA fabric needs to be paid.

4.2.2 Hessian detector

In software the contribution of integral image is undeniable. As it was mentioned in chapter 3 it is necessary to perform only four read operations in order to calculate the rectangular area integral of any size. This feature is mostly utilized by Hessian point detector. It significantly helps to speed up the convolution of an image with the appropriate Gaussian kernels. Considering the size and number of these kernels it would take much more time to make the convolution pixel by pixel. This is partly due the processors sequential nature. On the other side the FPGA can utilize parallel logic so it should be possible to do the convolution "on-line". This would mean buffering only enough data, to be able to calculate the appropriate responses of Gaussian kernels. But if we take a closer look it would be impossible to actually calculate the responses. The smallest Gaussian kernel in original implementation is of 9×9 pixels size and features 126 weighted pixels (for

all kernels D_{xx}, D_{yy}, D_{xy}). It would eat up an enormous portion of FPGA fabric to perform the sum of these weighted elements reliably and fast. With such an utilization the scale space construction wouldn't be even possible.

This leads to an idea of simplifying the Gaussian kernels. In my algorithm the Gaussian kernels are replaced with three point central numerical second order differences [44]. Equations 5, 6 and 7 show numerically calculated second order partial differences of two dimensional function at the point x, y where h is a stepping distance.

$$\frac{\partial^2 f(x, y)}{\partial x^2} = \frac{f(x + h, y) - 2f(x, y) + f(x - h, y)}{h^2} \quad (5)$$

$$\frac{\partial^2 f(x, y)}{\partial y^2} = \frac{f(x, y + h) - 2f(x, y) + f(x, y - h)}{h^2} \quad (6)$$

$$\frac{\partial^2 f(x, y)}{\partial x \partial y} = \frac{f(x + h, y + h) + f(x - h, y - h) - f(x + h, y - h) - f(x - h, y + h)}{4h^2} \quad (7)$$

After convolution with the simplified kernels the data are processed by Hessian detector. As an original SURF algorithm, my implementation uses the determinants of Hessian matrices to locate the image's significant points. The determinants are calculated exactly by definition described by the equation 4 in chapter 3.

4.2.3 Scale space construction

The size of convolution kernels is closely related with the scale space construction. The scale space can be constructed in two different ways. The first one is to scale dimensions of convolution kernels and perform convolution on fixed image. This way is used by the original SURF algorithm because due to the use of an integral image it takes same time to compute the response of convolution kernel of any size. The second one is to iteratively smooth and subsample the original image with the fixed size of convolution kernels. My algorithm uses the second way of constructing the scale space. The reason is in the implementation of the algorithm in FPGA fabric.

As it has been shown thanks to simplifying the convolution kernels and lack of usability of an integral image in further stages of the navigation algorithm the integral image generation has been omitted. The image feature detector was developed as on-line from the beginning. This means that it processes the data stream of any source directly. With convolution kernel of 3×3 pixels size on base scale it is only necessary to buffer 3 consecutive image lines. After that, with every incoming pixel the kernels produce the result of convolution with underlying 9 pixels for further processing by Hessian calculator (the whole architecture is described and depicted in chapter 5). The architecture of image feature detector is divided into the separate processing chains, each detecting features on separate scale. Each chain takes pixel data on the input and in the first stage buffers 3 lines of them and produces the rescaled pixel data and the convolution data for further processing.

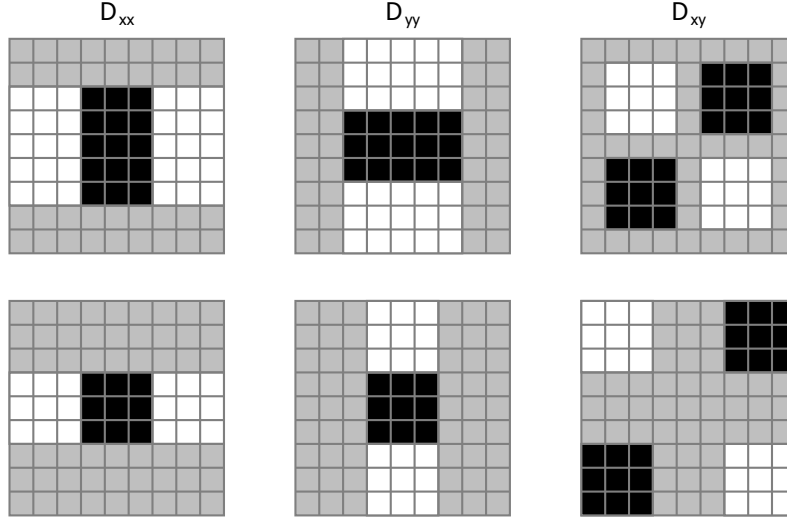


Figure 10: Comparison of convolution kernels.

These rescaled data are connected on the input of another processing chain. It can be said that the scale space construction is only a residual product of buffering three lines of pixel data which is necessary. So far the four scales are implemented in the hardware with scale factor of two. This corresponds to convolution with kernels of sizes 3×3 , 6×6 , 12×12 and 24×24 pixels. The reason is in the implementation because it is more economical to use the division by powers of two (in this case the sum of four pixels is divided by four) than to divide by any other number. However it is possible to divide by any number but it would take more FPGA fabric to do so. In Figure 10 there is a comparison of convolution kernels of the same size from the original implementation and my implementation. The original implementation kernels are in the first row, the simplified kernels are in the second row. By D_{xx} and D_{yy} kernels black squares represent the weighting of each image pixel by coefficient -2. By D_{xy} black squares represent the weighting of each image pixel by coefficient -1. White squares represent weighting coefficient 1 and gray squares 0. It can be seen that besides the D_{xy} kernel there is not a significant difference.

4.2.4 Interest point localization

The next step is to locate the local maxima amongst Hessian values. This is done by comparing Hessian value with its eight closest neighbours. In order to achieve this three lines of Hessians need to be buffered and after that processed simultaneously. After locating the local maxima these are compared with the pre-defined threshold to produce the final localization of interest points. In further stage of implementation these points could be simply compared throughout the scale space in order to obtain the highest local maxima amongst all located features but this functionality has not yet been implemented.

4.3 Image feature descriptor

I chose the BRIEF descriptor (see chapter 3 for details) for my work because it had shown that on a long term navigation scenario dataset it outperformed other feature descriptors in speed and repeatability [6]. The BRIEF descriptor uses intensity comparison of pixels inside an image patch which surrounds located feature. These comparisons form a set of unique binary tests which are subsequently stored into an n_d bit vector. In original paper the n_d constant takes values of 128, 256 and 512. For my work I decided to implement 128 bit long BRIEF descriptor on the patch of size 50×50 pixels. The composition of unique binary tests has been generated randomly.

4.3.1 Descriptor generation

The generation of BRIEF is completely done in FPGA fabric. There is an implementation of BRIEF descriptor in the OpenCV library which uses an integral image to compute the descriptor. This is logical since the features are normally detected on several scales and therefore the patch and consecutively the pixel sizes need to be properly enlarged. But my implementation doesn't feature integral image generation and storage therefore the descriptor needs to be calculated in another way. Descriptor calculation is done from the original image which is stored in an external memory. If I save only the original image, it would take a lot of memory read operations to calculate descriptors on higher scales. Therefore the algorithm saves the original image together with its smoothed and subsampled copies. Table 1 shows the performance comparison of description based on the integral image, the original image and the original image with all the scales saved in the memory. Values in the table are derived from computing necessary data for image with the resolution of 1024×768 pixels. The overall size in the memory is calculated as a number of pixels times bit width. This bit width is 8 bit per pixel for the original and the rescaled images and 28 bit per pixel for the integral image. The entry of number of write operations assumes a fully utilized 32 bit wide memory bus to an external memory. Number of read operations expresses an amount of memory read operations which are necessary for calculation of one 128 bit wide BRIEF descriptor on an appropriate scale. For the integral image and the original image with all scales this amount is constant. For the original image saved in the memory it is necessary to perform four times more read operations to calculate the descriptor with the every following scale. Table 1 shows that when operating only on four scales it is most efficient to buffer the image data for all of the scales. The precise implementation is described further in the chapter 5.

4.3.2 Orientation assignment

By the computer vision algorithms it is usual to perform orientation assignment of the descriptor so the interest point description is immune to rotation changes. This is necessary for computer vision applications like image stitching or object recognition where there is

| | Integral image | Original image | Original image with scales |
|----------------------------|----------------|---------------------|-------------------------------|
| Size in the memory | 22 020 096b | 6 291 456b | 8 454 144b |
| Number of write operations | 688 128 | 196 608 | 264 192 |
| Number of read operations | 1024 | 256/1024/4096/16384 | 256 |

Table 1: Description performance comparison.

high possibility of a bigger viewpoint change. However for purposes of the mobile robotics the orientation assignment of the descriptor is not necessary. When using ground vehicles the camera is usually mounted on top of the mobile robot and aims forward. When not operating in extremely rough terrains the vertical axis of the camera remains still.

4.4 Navigation algorithm

The navigation algorithm is based on the SURFnav algorithm [11], [3], [1] briefly described in the chapter 2. Now the algorithm will be described in more detail together with proposed modifications. The algorithm consists of two main phases the teach phase and the repeat phase.

4.4.1 Teach phase

During the teach phase the mobile robot is guided through the environment along the polyline path. The polyline path is split into individual line segments creating the landmark map of the environment. The algorithm uses the fact that when traversing the line segment with camera heading forward the detected image features are migrating through the image along the imaginary lines from the center of the image to the edges. Making the path a virtual tunnel of image features. The local segment landmark map contains information about in which part of the segment the image features were visible and what were their image coordinates. This information is sufficient to compute image coordinates of the landmarks for particular robot position. The landmark map is composed from entries which structure can be seen in the table 2. In my implementation the image feature positions are integer values and they are described by the BRIEF descriptor.

4.4.2 Repeat phase

The repeat phase of the algorithm navigate the mobile robot along the individual segments of the composed map. At the beginning of each segment the mobile robot turns to the predefined direction by means of odometry and starts traversing the segment. During

| Record | Value |
|----------------------------------|----------------------------------|
| Initial azimuth and length: | 2.13, 7.03 |
| Landmark 0: | |
| First position: | 760, 163 |
| Last position: | 894, 54 |
| First and last visible distance: | 0.00, 4.25 |
| Descriptor: | 1, 11010110001010001011101001... |
| Landmark 1: | |
| First position: | 593, 381 |
| Last position: | 689, 377 |
| First and last visible distance: | 0.72, 6.73 |
| Descriptor: | -1, 0100100011100110111001110... |

Table 2: Segment landmark map composition.

the traversal the system detects image features and match them with the learned ones. Figure 3b in chapter 2 shows the situation from the mobile robot's point of view. The red circles in the figure highlight image features which are searched in a current view. The position of the red dashed line is determined by the means of odometry. If the match between keypoints is detected the difference between observed and expected position of image feature in the image is calculated. The differences are used to create navigation histogram with fixed number of bins. The maximum peak in the histogram determines the correction of heading of the mobile robot.

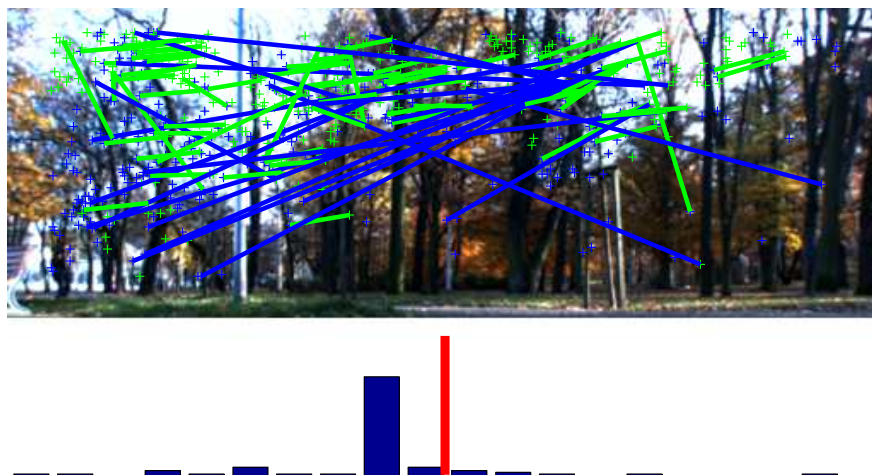


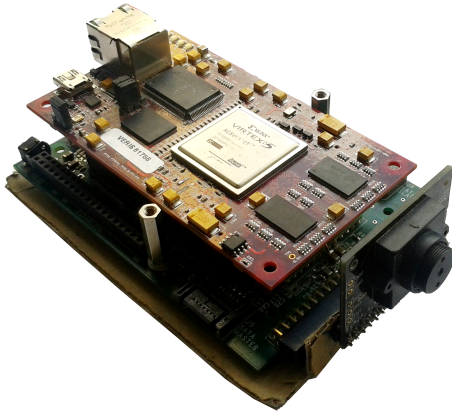
Figure 11: Detected landmarks in the picture and the corresponding navigation histogram.

Figure 11 shows the typical result of visual compass. Blue crosses mark currently visible image features and green crosses corresponds to landmarks stored in the map. Matched landmarks are connected with the lines. Correspondences near the center value are connected with green lines, others are connected with the blue ones. The navigation histogram is depicted under the picture showing that the mobile robot is slightly left from the pre-learned trajectory.

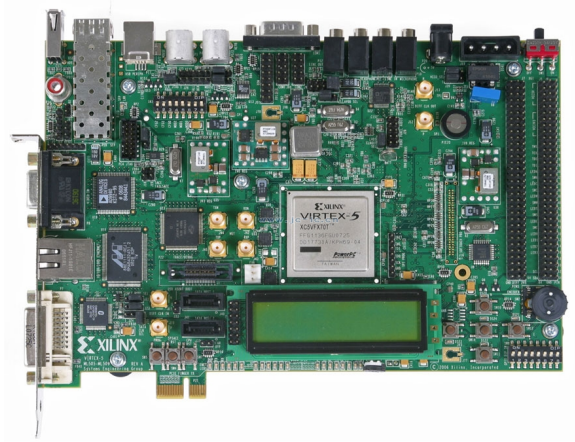
5 Implementation

In the previous chapter the theoretical basis for implementation is described. This chapter describes the actual implementation of the algorithm in hardware (FPGA configuration) and software. The first part of this chapter provides the reader with an overview of reference boards on which the implementation has been tested. The second part discusses the proposed FPGA architecture and the last part is about the implementation of SURFnav navigation algorithm in the software of the module.

Due to the necessity of previous theoretical testing of the detection and description algorithm an implementation in MathWorks Matlab software was also developed. The actual implementation has been developed in Xilinx ISE Design Suite System Edition [45]. The ISE Design Suite System Edition includes ISE Design Software(XISE), Xilinx Platform Studio(XPS), Software Development Kit(SDK) and System Generator for DSP. The ISE Design software is the main application of the suite. It is the project manager, text and schematic editor. It allows the user to build custom FPGA designs by means of HDL programming, schematic drawing or by adding macro components from various libraries to the design. Xilinx Platform Studio is used for creating the whole System On a Programmable Chip (SOPC) designs by adding and connecting components from the vast database of embedded Intellectual Property(IP) cores. It also allows the user to build the new IP cores by providing the assisted design flow. Both XPS and XISE are intended to build the FPGA configuration and provide the user with the resulting bitstream. Xilinx Software Development Kit provides programming environment for designs which features embedded processors. It allows to build and compile software projects which can be then run on FPGA specific processor architecture (softcore or hardcore).



(a) Schwab minimodule



(b) Xilinx ML507 board

Figure 12: Development platforms.

5.1 Reference boards

For the purposes of algorithm evaluation two development boards featuring same FPGA chip have been chosen. The first of them is a Schvab minimodule, built by Jan Šváb [4], [10]. The minimodule composes of a standard off-the shelf Avnet AES-MMP-V5FXT70-G Mini-Module Plus board and a custom baseboard specifically designed for computer vision applications. Figure 12a shows the minimodule with the connected heathsink over the FPGA chip.

The second one is a Xilinx ML507 evaluation platform board. This is a high-end development board which features a lot of auxiliary circuitry. Figure 12b shows the layout of the development board. In the table 3 the main features of both boards are listed. By the ML507 development board only circuitry which was utilized during the development of this thesis is listed.

| | Schvab minimodule | ML507 development board |
|--------------------------|--|---|
| FPGA | Xilinx Virtex-5 XC5VFX70T-2-FF665 | Xilinx Virtex-5 XC5VFX70T-1-FFG1136 |
| Memory | 64MB DDR2 SDRAM 32MB flash memory 4MB SSRAM SD Card slot | 256MB DDR2 SDRAM 32MB flash memory 1.125MB ZBT synchronous SRAM SD Card slot |
| I/O and Communication | serial port USB 2.0 PHY 10/100/1000 Ethernet PHY SATA master/slave subsystem Expansion connector (23 pins) Camera connector | RS232 serial port Video input (VGA) Video output (DVI connector) LEDs, DIP switches, Push buttons Expansion connector (96 pins) |

Table 3: Reference boards features.

5.2 FPGA configuration

From the beginning of my bachelor project I set up a goal to develop an implementation which can operate in real time. This means that all image features with corresponding descriptors are calculated within the duration of one image frame. One image frame is defined as a duration of full frame with digital video signal timing specifications. Digital video signal consists of two main areas. Display area, where image pixel data are provided, and blanking area, where there are no pixel data provided (corresponds to black color). This is due to the historical development because the cathode ray tube (CRT) monitors

which compose picture on the screen by beam deflection need some time to move the beam from the end of one line to the beginning of the next line. For precise synchronization horizontal and vertical synchronization pulses are provided. For standardized 4:3 resolutions the display area forms about 70% of the whole frame. In Figure 13 there is a digital video timing diagram showing both display and blanking area of the image and corresponding synchronization signals.

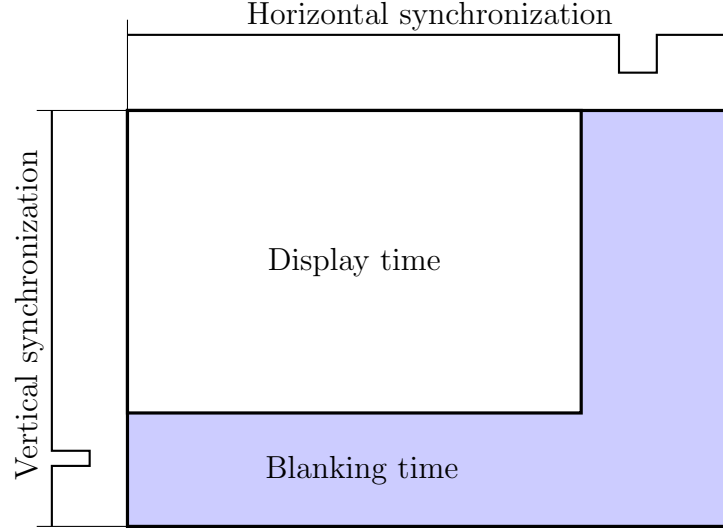


Figure 13: Digital video timing diagram.

In Figure 14 there is an overview of the proposed FPGA architecture. The design is divided into four main parts: pixel I/O, feature detector, feature descriptor and embedded processor with its peripherals. Pixel I/O part comprises of several cores, which purpose is to provide further logic with pixel data and corresponding data-aligned clock signals together with the signals determining position in the image. Feature detector takes the stream of pixel data on the input and provides locations of image features on the output. Feature descriptor part of the design takes positions of located features and calculates corresponding descriptors. The Feature descriptor part utilizes the memory interface with the on-board SRAM memory which has shown to be the key component in feature description process. Both the Feature detector and descriptor parts are custom designed and completely standalone. They form a complete feature extractor which takes the image data on the input and provide locations of key features together with their description on the output. The last part in the design features embedded processor core with peripherals. These peripherals are except one communication core all Xilinx's IP cores. The Embedded processor part is used for the actual visual navigation algorithm. Now every part of the design will be presented in detail.

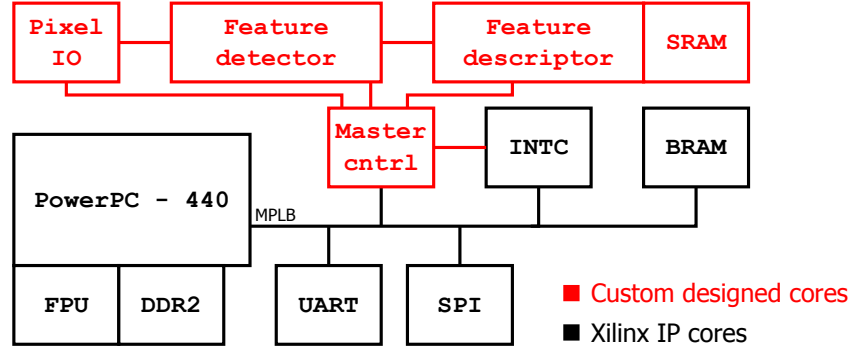


Figure 14: Block diagram of the high-level FPGA architecture.

5.2.1 Pixel I/O

The reference board Xilinx ML507 features VGA in and DVI out circuitry making it possible to connect the board with VGA output of the computer on one side and with the display on the other side. This is a very useful feature for development of computer vision algorithms. Thanks to this feature the programmer can instantly verify functionality of the solution only by sight. Thorough testing can be performed later but for basic check it provides an excellent help. Pixel I/O comprises of three individual cores: **I2C_CNTRL**, **VGA_COUNTER** and **VGA_CNTRL**.

I2C_CNTRL core is a simple core which performs initialization of on board chips connected to the VGA input and DVI output by means of i²c serial bus in order to assure their correct functionality.

By VGA input chip (Analog Devices AD9980 High Performance 8-Bit Display Interface chip) the initialization consists of setting correctly registry information about incoming VGA signal. This helps the device to correctly synchronize with the incoming analog data signal and provide clear and stable image and pixel clock for further processing in FPGA fabric. It also provides the horizontal and vertical synchronization signals, which all together form an external input of the whole implementation. By DVI output chip (Chrontel CH7301C) the initialization consists of master enable of the chip functionality and provision of information about incoming data signal (from FPGA).

The protocol for communication with both devices is a standard i²c protocol without acknowledging. As has been said this core is used only for initialization of auxiliary circuitry after start (or restart) therefore there is no need for implementing the whole i²c communication interface. This core is suitable to perform initialization of almost any i²c or i²c-like device, such as camera module (OmniVision OV9653 CMOS camera) attached to the Schvab minimodule board.

VGA_COUNTER core is a second key component on the input of the design. It's purpose is to synchronize with the incoming horizontal and vertical pulses and provide the logic with x,y coordinates of the right incoming pixel data. Together with x,y coordinates there is one more output from the core which marks the display and blanking area of the image.

VGA_CNTRL core is an auxiliary output core which helps graphically represent any debugging data from the FPGA logic. Moreover it can read the content of SRAM memory which is used for storing the image data. It forms something like a presentation layer to the DVI output chip, rearranging the data to match the correct timing and format for displaying.

5.2.2 Feature detector

Feature detector is a system of heavily pipelined cores which are divided into four separate feature detection processing chains, each processing the image on a different scale. Feature detection processing chain takes 8 bit wide pixel data stream on the input, calculates determinants of Hessian matrix and locates the local maxima amongst them. There are seven cores in the design of the feature detection processing chain, each of them represented by an VHDL entity. The interconnection of individual entities is shown in Figure 15 (for better readability clock and reset signals are not depicted because they are connected to all of the cores). The processing chain works as follows.

First of all three lines of pixel data are buffered in the **PIXEL_BUFFER** core. This core stores incoming image data in local block RAM resources and release them in appropriate

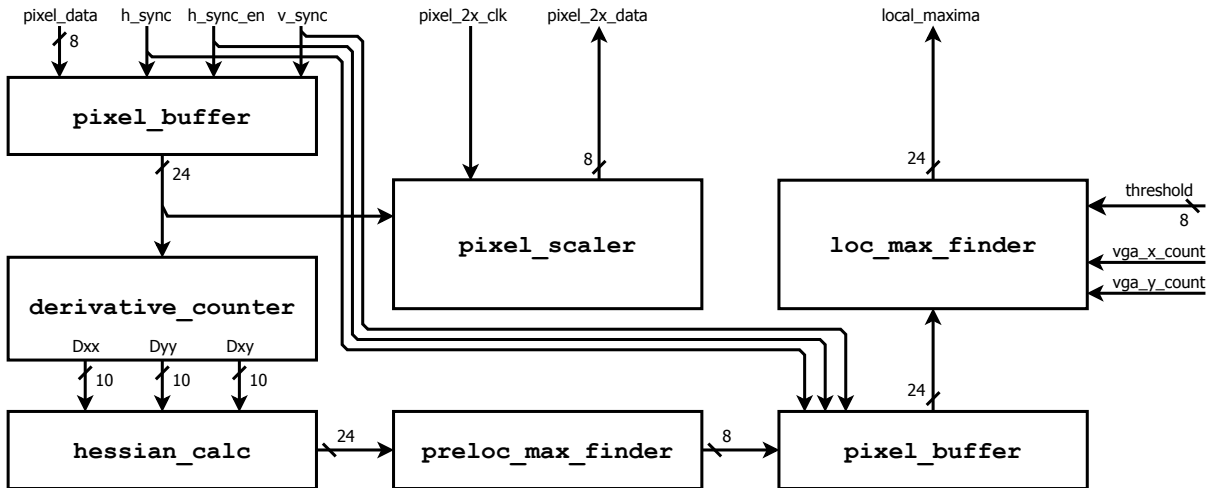


Figure 15: Feature detection chain architecture.

moment. It means that with every incoming pixel the core provides the luminance data for three pixels located in the image above each other.

These pixel triplets are processed by two cores. The first of them is an auxiliary core `PIXEL_SCALER`, which performs the operations needed for subsampling incoming data. At current state the `PIXEL_SCALER` core utilizes logic of moving sum and provides the average value of luminance of four pixels in the shape of square with an edge length of 2 pixels. The second core to utilize pixel triplets is the `DERIVATIVE_COUNTER` core. This core computes the convolution of an image with simplified 3×3 kernels (see details in chapter 4).

The resulting data from the `DERIVATIVE_COUNTER` core are processed by the `HESSIAN_CALC` core. This core computes the resulting determinants of Hessian matrix. These determinant values are in fact signed numbers which can take values from -524 288 ($[D_{xx}, D_{yy}, D_{xy}] = [-512, 512, 512]$) up to 262 544 ($[D_{xx}, D_{yy}, D_{xy}] = [512, 512, 0]$). In binary representation it takes 20 bits to represent the determinant values.

To reduce the FPGA fabric utilization I decided to implement something like a floating point representation of Hessian values by numbers which occupies only 8 bits. This looks like a significant loss of precision but this operation is legit because the algorithm is not interested in global maxima but local maxima. Therefore every Hessian value is compared only with its 8 surrounding neighbours. This leads to an idea of comparing only the most significant bits of the Hessian values behind the first logical 1 in the binary representation of the value. Moreover this simplification partially suppresses jitter of the detector. This is due to the fact that when the detector is fed by synthetic, fully digital, signal (e.g. image buffered in the memory of the device) the luminance values of individual pixels are same all the time (when processing the image over and over). When using any type of analog to digital signal conversion (e.g. camera or VGA input) the luminance of pixels jitters a little. This small jitter is then multiplied by `DERIVATIVE_COUNTER` and `HESSIAN_CALC` cores and consequently produces big jitter of the whole detector. So the FPGA fabric utilization and the jitter reduction are two main reasons to implement only comparison of the most significant bits of Hessian values. Hessian values shrinking is implemented in a logic of `PRELOC_MAX_FINDER` core.

Since the bit width of Hessian values is reduced to 8 bits it is possible to use the second instance of `PIXEL_BUFFER` core to buffer three lines of Hessian values.

The last core in the design of feature processing detection chain is the `LOC_MAX_FINDER` core. This core compares the incoming Hessian values with its 8 closest surroundings and produce a record with position specification of key feature if it locates a local maxima which satisfies the threshold condition.

Now every core will be described in more detail.

PIXEL_BUFFER

It is a multi-purpose core for data buffering. It stores incoming data in three instances of first-in first-out (FIFO) memory connected in chain and release them in appropriate moment. The data buffering is completely controlled by the horizontal synchronization pulses. In Figure 16 there is a simplified schematic of the **PIXEL_BUFFER** core. For better readability there are not clock and reset signals in the schematic. Clock signals are only connected to the input and to the output ports of the individual FIFOs and the reset signal is connected with the vertical synchronization signal to form the master reset for the whole core.

With every incoming pixel the core provides data for three pixels located in the image above each other. The number of stored pixels in each line (corresponds to each FIFO) is controlled only by the first three horizontal synchronization pulses after the vertical synchronization pulse which resets all the FIFOs. This allows changing the width of the processed image without the need to reconfigure the core (this feature could be utilize for example for multiplexing of the camera images from more sources through one detector). There is also a horizontal synchronization enable signal which is related with the design clocking and scale space construction which is described further in this chapter.

The **PIXEL_BUFFER** core utilizes three Xilinx FIFO18 primitives.

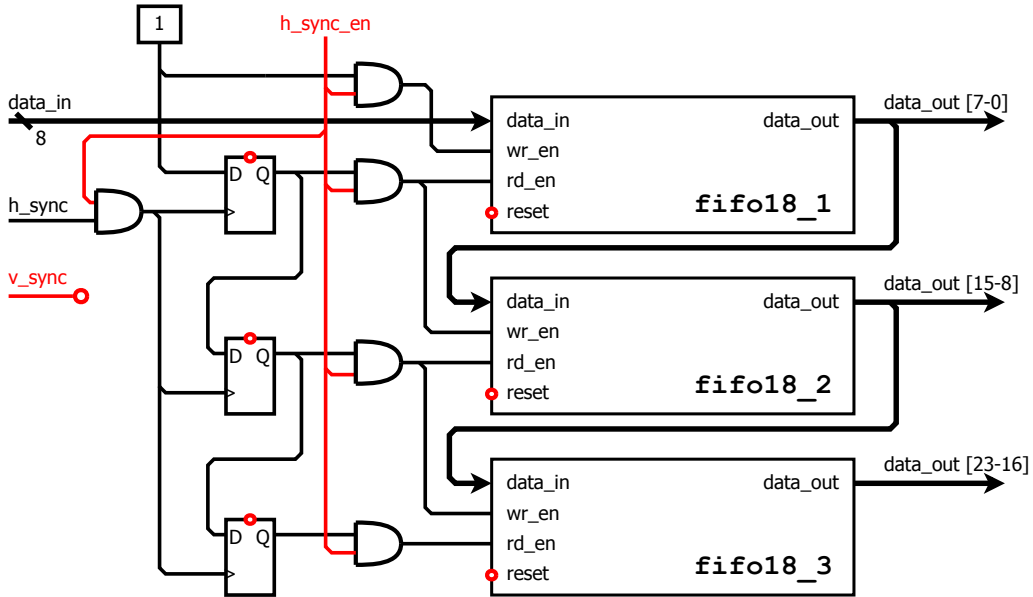


Figure 16: Simplified block diagram of the **PIXEL_BUFFER** core.

DERIVATIVE_COUNTER

This core calculates the responses of derivative filters as stated in chapter 4. It consists of three element deep 24bit wide shift registr which buffers incoming pixel data from

PIXEL_BUFFER. These data form 3×3 matrix from which the signed values of Dxx , Dyy and Dxy filter responses are calculated. With every falling edge of the clock incoming data and content of the shift register are shifted. With every rising edge of the clock the core produces values of Dxx , Dyy and Dxy filter responses.

HESSIAN_CALC

This core calculates the final value of the determinant of Hessian matrix according to the equation 4 in chapter 3. It produces 24 bit wide signed number. The **HESSIAN_CALC** core utilizes two embedded multipliers for the calculation of the descriptor.

PRELOC_MAX_FINDER

Simple core which shrinks 24 bit wide Hessian values to 8 bit. The resulting 8 bit value is formed from the sign bit, the 2 bit wide exponent and the 5 bit wide mantissa. The resulting 8 bit value is shown in Figure 17. As mentioned above the Hessian values can occupy maximum of 20 bits. Therefore the exponent part of the representation indicates the position of the most significant ‘one’ in the Hessian value representation, tailing the Hessian value into four quintuplets. Mantissa is then the content of that quintuplet with the highest ‘one’ in binary representation.

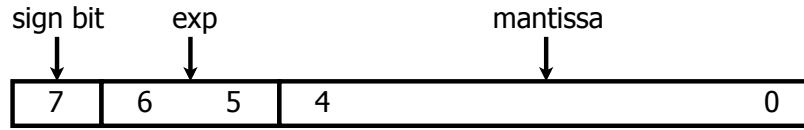


Figure 17: **PRELOC_MAX_FINDER** result value composition.

LOC_MAX_FINDER

This core is intended to locate the local maxima amongst Hessian values which satisfies the threshold condition and produce final entries with x,y coordinates of located feature. It consists of three element deep 24bit wide shift register (same as by **DERIVATIVE_COUNTER** core), which buffers incoming Hessian values from the second instance of the **PIXEL_BUFFER** in the feature detection chain. The content of the shift register forms 3×3 matrix of Hessian values in which the center value is compared with its 8 surroundings. If there is no higher Hessian value in the matrix and the value satisfies the external 7 bit wide threshold condition the point is declared the key feature. The **LOC_MAX_FINDER** core is the only core in the design of the Feature detector which takes on the input x,y coordinates provided by the **VGA_COUNTER** core. These coordinates are used for the generation of key feature entry. This entry is a 24 bit wide logic vector which consists of the sign bit of Hessian value and the x, y coordinates which are currently provided on the input of the core.

PIXEL_SCALER

It is an auxiliary core which does not have to be implemented. Its purpose is to

produce rescaled pixel data. As mentioned in chapter 4 the scale space construction is done by resampling the original image data by the scale of two. The core takes the data from the `PIXEL_BUFFER` core on the input and performs moving sum over four pixels in a shape of square with the edge length of two pixels. The resulting average, which is performed by a simple bit shift, is sampled with provided clock signal from higher scale. This is related with the clock generation mechanism which is described further in this chapter. This approach also ensures best possible alignment of the rescaled pixel data with the higher scale pixel clock.

So far there are four feature detection processing chains in Feature detector part of the implementation. The first of them is connected directly with the Pixel I/O, processing the image data on basic scale. The rest of chains are connected in cascade on the rescaled data output of previous chain, together forming a four scale image feature detector.

The biggest task in development of Feature detector was to provide the logic with correct and precise timing. Clock signals are derived from signals provided by `VGA_COUNTER` core, namely the `vga_x_count` and `vga_y_count`. The `vga_x_count` signal is incremented with every incoming pixel clock positive edge. When the horizontal synchronization pulse is detected the `vga_y_count` signal is incremented. To correctly implement the scale space construction it is necessary to adjust the base clock signal. This consists of dividing the base clock by two and skipping every odd line of the image (means the clock signal is holding steady during the whole line). This procedure is repeated for every consecutive scale. If we take a closer look it can be shown that the clocking signals can be derived from the `vga_x_count` and `vga_y_count` signals. The division of the base clock on x axis is quite obvious. If we look at the `vga_x_count` signal as the counter or register, which value is incremented every clock cycle, it can be easily concluded that every bit of such a counter (taken from the least significant bit) corresponds to division of base clock by two. Skipping of odd lines could be derived from `vga_y_count` signal. For the base scale there is no skipping. For the first scale every odd line needs to be skipped, this corresponds to the least significant bit of `vga_y_count` register. For the second scale every odd line from the first scale needs to be skipped, this corresponds to state where the least significant bit and the second bit of the `vga_y_count` register are both in logic one. Etc.

Clock signals for every scale could be synthesize by only adding logical AND between the respective signals, but it would mean gated clock signals which is bad design practice. Using signals of clock and clock enable seems to be better idea but it would mean connecting these signals in all the cores of each chain, which utilizes more FPGA fabric than the third approach. The third approach is to provide the logic with continuous clock signal, appropriately divided according to scale and the line skipping perform in the `PIXEL_BUFFER` core. Line skipping then consists of enabling inputs and outputs of the FIFOs in the `PIXEL_BUFFER` core which in fact is equal to enabling the horizontal synchronization pulse. In conclusion the precise timing is provided by the clock signals which are derived from the `vga_x_count` signal (synthesize tools can recognize dividing the clock by two which is

not gated operation in FPGA fabric) and horizontal sync enable signals which are derived from `vga_y_count`. Both the clock and the horizontal sync enable signals determine the correct behaviour of the Feature detector.

Considering the performance of an individual chain it can process the continuous stream of data, with only appropriate clock, horizontal synchronization and vertical synchronization signals provided. The information about whether the point is a local feature or not is delayed only by 4 image lines and 4 pixels behind the pixel data are provided on the input of the chain.

5.2.3 Feature descriptor

This is a part of implementation which takes care of computing BRIEF descriptors. During the development work on this thesis there was an idea to compute the descriptor in the software on the embedded processor core (this strategy is utilized for example by [4], [40], [39]). However it would mean a significant performance bottleneck for both the memory bus utilization and processor cycles which will be spent by transfusing the image data for and there. Therefore I decided to implement the description cores together with the corresponding memory interface in FPGA fabric. The last question remains: “Which memory should be utilized?”

If we take a closer look on method design (see chapter 4 for details) first of all the whole image together with its subscaled copies needs to be stored in the memory. As it was mentioned in the same chapter the inner block RAM resources can’t store such amount of data. Therefore the implementation uses the SRAM memory on the development board. Moreover the SRAM memory on the development board is a Zero Bus Turnaround (ZBT) synchronous SRAM memory. ZBT SRAM memory eliminates dead bus cycles when turning the bus around between reads and writes, or writes and reads. It means that it has zero latency, the data are only delayed by two clock cycles. Figure 18 shows the simplified timing diagram of ZBT SRAM memory. It demonstrates the ability of the memory to perform read and write cycles consecutively without any waste cycles. SRAM memories are also better in accessing stored data in a random order than other memories which are designed for sequential access.

These reasons have been recognized as a key feature of interest point description implementation. When implemented in FPGA fabric, the implementation can fully utilize the memory bus bandwidth and thus achieve the highest possible performance.

The Feature descriptor part of the design comprises of `MEMORY_MANAGER`, `FEATURE_BUFFER` and `DESCRIPTION_CNTRL` cores.

The `MEMORY_MANAGER` core handles appropriate communication with the ZBT SRAM memory. It is the point where five different clock signals meets together. First of them is a master clock signal for synchronizing the communication with the SRAM memory. At the current point this clock corresponds to the base pixel clock. Several attempts had been made to speed up the communication with SRAM memory in order to achieve higher performance but with little or no success. The rest of the incoming clock signals are the

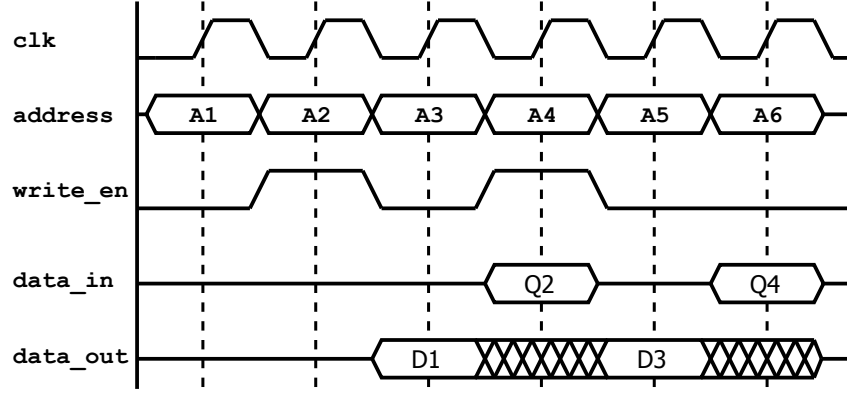


Figure 18: Zero Bus Turnaround SRAM memory timing diagram.

pixel clocks on all the scales. Together with the pixel clock signals pixel data are provided. The **MEMORY_MANAGER** core saves the incoming pixel data to the SRAM memory and when there is no need for writing transaction it performs read operations on the address defined by input **rd_address**. This leads to maximum possible utilization of data memory bus bandwidth. In the whole frame with the display resolution of 1024×768 pixels there are 1 083 264 clock cycles. 264 192 clock cycles are used for write operations (according to the Table 1 in the chapter 4). This allows 819 072 possible read cycles.

The second core in the design is the **FEATURE_BUFFER** core. This core is connected to the processing chains and performs the final selection of image features for description, automatic thresholding and image feature storage. Feature selection filters out points so close to the image border, that their descriptor could not be calculated. The automatic thresholding is a process when the implementation modifies the threshold for **LOC_MAX_FINDER** core in order to maintain an approximately constant number of located image features per image. This number is defined for every scale by two generics. The first of them specifies the maximum allowable number of image features in the image, the second one the minimum number of image features.

The last core in the Feature descriptor is the **DESCRIPTION_CNTRL** core. This core reads the stored image features from **FEATURE_BUFFER** core and performs composition of the BRIEF descriptor. This consists of providing the **MEMORY_MANAGER** core with appropriate addresses to read pixel data from. As mentioned in the Table 1 it is necessary to perform exactly 256 read operations to construct the BRIEF descriptor. Now every core will be described in more detail.

FEATURE_BUFFER

This core is connected to all four processing chains and performs final selection of image features for description, automatic thresholding and image feature storage.

In Figure 19 there is a simplified block diagram of the **FEATURE_BUFFER** core. The

core takes feature data from `LOC_MAX_FINDER` core on the input. Then it compares the x,y coordinates with the predefined set of constants x_{min} , x_{max} , y_{min} and y_{max} which filter out all the key features that could not be described (they are too close to the frame border). When the key feature succeed conditions the write enable signal for FIFO is generated and the feature is stored in the FIFO memory for further processing by the `DESCRIPTION_CNTRL` core. All the write pulses are summed during the duration of the one frame and this sum is compared with predefined constants t_{min} and t_{max} . If this sum exceeds the t_{max} value the threshold is incremented by one. When the sum underflows the t_{min} value the threshold is decremented by one. This is the principle of the automatic thresholding which maintain the number of located features between t_{min} and t_{max} values. The `FEATURE_BUFFER` core utilizes four instances of Xilinx FIFO36 primitive for the key feature storage.

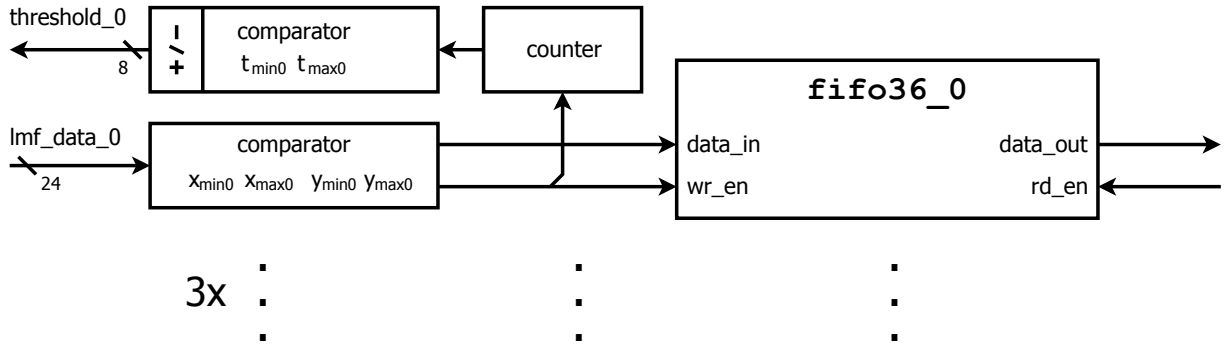


Figure 19: Simplified block diagram of the `FEATURE_BUFFER` core.

MEMORY_MANAGER

The `MEMORY_MANAGER` core has shown to be the toughest core for development. Its complexity lies in a fact that there are five different and discontinuous (odd line skipping) clock signals which meet together in this core. The purpose of the core is to save the image data that came from `PIXEL_SCALER` cores of the detector chains and in the spare time provides data to the `DESCRIPTION_CNTRL` core for descriptor construction. The core consists of four 32 bit wide buffers for incoming pixel data and a state machine which controls the communication with the SRAM memory. When the 32 bit wide register for any chain is full its content needs to be written to the SRAM memory before next pixel data arrive. Experiments with “write on demand” system didn’t work well so the precise deterministic write system has been utilized. The SRAM memory clock is the base pixel clock. It means that every four clock cycles the data for the base scale chain needs to be stored. Then every eight clock cycles the data for the first scale chain needs to be stored and so on. Figure 20 shows

the timing diagram of two write cycles. The diagram apply for the display area of the image frame. During the blanking area only read operations are performed.

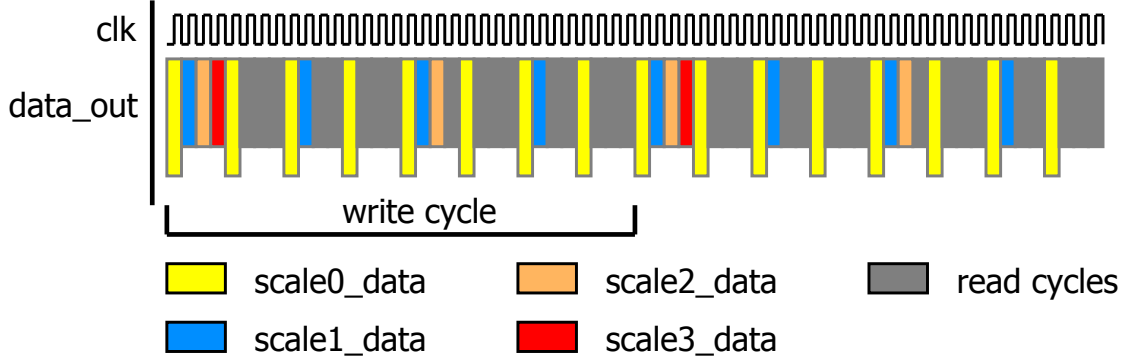


Figure 20: Timing diagram of SRAM memory bus utilization during display area.

Every chain has also its own address range for the data storage. Images are stored in the memory as if they were parts of a frame with the resolution of 1024×768 pixels which is the base scale resolution. It means that when the memory is read out as a static image with resolution of 1024×768 pixels, subscaled images are not distorted. This allows using same address offsets for computation of descriptors on any scale. There is only one issue with the Xilinx ML507 development board because it features only 9 Megabits of SRAM memory. This is unfortunately not enough to save the whole image on all the scales in desired format. Therefore only half of the base scale image is present at the memory at any moment. When the base scale image reading reaches the half of the image, it moves the writing address back to the first pixel and starts rewriting the image from the beginning. It gives the `DESCRIPTION_CNTRL` core only 512 image lines to perform descriptor composition for points located in the first half of the image. However the Schvab minimodule board features 4 Megabytes of the SRAM memory which is enough for storing whole images.

DESCRIPTION_CNTRL

This core's functionality is to assemble the descriptor for located features. It reads the data from `FEATURE_BUFFER` FIFOs and test them whether it is possible to perform description of located points. If it is possible it generates sequence of addresses to be read out from the SRAM memory in order to assemble the descriptor. The testing lies in comparing current value of y coordinate in the image (`vga_y_count`) with y coordinate of located feature. The BRIEF descriptor requires the y coordinate of located feature to be at least 25 lines above the current y coordinate. If it was not, the descriptor would be assembled from two camera images instead of one (the current one). The address offsets are stored in one instance of Xilinx RAMB18

primitive. During the composition of the descriptor they are read out and added or subtracted to the x and y coordinates of located feature. These new coordinates are then serialized to an address and send to the **MEMORY_MANAGER** core. Incoming data from the **MEMORY_MANAGER** core are then compared in pairs and form the resulting descriptor. When the generation is finished 'done' signal is generated and the data are send to the Embedded processor part of the design. The number of descriptors is limited by clock cycles during the blanking time and spared cycles during the display time. Theoretically it is possible to compute 3199 descriptors during the duration of the one frame, however the random distribution of features together with the need of waiting for enough data to be buffered decreases the real amount of computed descriptors somewhere about 400-500 descriptors per image.

5.2.4 Embedded processor and its peripherals

The Xilinx Virtex 5 FPGA contains hardcore PowerPC 440 processor (depicted in Figure 9 in the chapter 4). It is a main part of the embedded processor subsystem block. Implementation of this part is slightly different than of the other parts of the design because it is almost completely built from Xilinx's Intellectual Property (IP) cores. Building such a SOPC design from IP components is quite simple because it is only necessary to correctly connect the predefined IP cores and assign them correct properties (e.g. address range, baud rates, etc.). The Embedded processor subsystem consists of several Xilinx IP cores and one custom designed IP core. Xilinx cores provides functionality of the DDR2 memory controller for communication with auxiliary DDR2 memory, the BRAM controller for storing smaller software designs, the UART communication interface, the floating point unit attached by an auxiliary processing unit(APU) bridge directly to the PowerPC and finally the interrupt controller.

| Address | R/W | Bits | Description |
|---------|-----|-------|--|
| 0x00 | R | 0 | frame sync - bit distinguishing consecutive frames |
| | | 21:31 | x coordinate position of the feature |
| 0x04 | R | 0:1 | feature scale: 00 - base scale 01 - first scale 10 - second scale 11 - third scale |
| | | 20:31 | y coordinate position of the feature |
| 0x08 | R | 0:31 | descriptor first word (0:31) |
| 0x0C | R | 0:31 | descriptor second word (32:63) |
| 0x10 | R | 0:31 | descriptor third word (64:95) |
| 0x14 | R | 0:31 | descriptor fourth word (96:127) |

Table 4: **MASTER_CNTRL** core user accessible registers.

The custom designed **MASTER_CNTRL** core provides communication bridge between the image feature extraction part of the implementation and the Embedded processor subsystem. So far it has six accessible registers where the information about the incoming image feature is stored. This core is directly connected with the **DESCRIPTION_CNTRL** core on the one side, and with the PLB bus and the interrupt controller on the other side. When the **DESCRIPTION_CNTRL** core generates the ‘done’ signal the key feature x,y coordinates together with corresponding descriptor are read out and stored in user accessible registers of the core. After that the **MASTER_CNTRL** core generates the interrupt signal which tells the software running on the PowerPC processor that the new data are ready to be read out. The list of user accessible registers is in the table 4.

5.2.5 Implementation costs

At the current state the Feature detector and the Feature descriptor is capable of processing any image with the full frame width of 2048 pixels (including display and blanking time) and a pixel clock of 200 MHz in real time. This corresponds for example to the VESA Signal 1280×1024 @ 100 Hz timing with 1760 pixels full frame width and the pixel clock of 190.96 MHz. Higher resolutions can be achieved by utilization of more internal Block RAM resources for the **PIXEL_BUFFER** core or by data buffering exclusion during the blanking area of the image line. Higher pixel clock rate is achievable only in the Feature detector part of the design. The currently used SRAM memory is limited by 200 MHz clock.

The custom designed part of the system occupies only 5% of Slices and 19% of inner Block RAM resources when placed and routed on the Xilinx Virtex-5 FX70T FPGA. Full design together with embedded processor and its peripherals occupies 36% of Slices and 25% of inner Block RAM resources. The largest part of Slices is occupied by auxiliary floating point unit and DDR2 controller of the processor.

5.3 Software description

This section describes the software written in C language which runs on the embedded PowerPC 440 processor. As stated until now the implementation of the feature detector and the feature descriptor leaves the CPU performance intact. So the whole CPU performance is dedicated to the tasks related with visual navigation. However due to the complexity of the development of this thesis the actual navigation algorithm hasn’t been implemented yet and it is the subject of the future work. So far the software of the device is capable of initialization of every peripheral including the interrupt controller. After initialization it enable interrupts for the PowerPC processor and defines an interrupt handler method. When the interrupt is generated by the **MASTER_CNTRL** core the interrupt handler method reads out the **MASTER_CNTRL** core user accessible registers and stores the incoming key feature data to the main system memory. These data are then sent to the computer via the

UART interface. Due to the fact that the implementation processes 60 frames per second with about 500 descriptors per frame the interrupt handling needs to be temporarily suspend during the slow UART communication. It means that the implementation buffers data for few consecutive frames then suspends interrupt handling and sends the data via the UART interface to the computer.

6 Experiments

This thesis introduced a new feature detection method which had to be evaluated. Because it brings a significant simplification into the process of feature detection the main question was if it can achieve comparable results in the mobile robot navigation. This chapter describes the experiments which were done in order to evaluate the behaviour of the feature detector.

The experiments were performed on the Xilinx ML507 development board which was connected to the video output of the computer. The processed video signal was a standardized VESA 1024×768 @ 60Hz video signal. This layout of the experiment also helped verify the ability of the detector to operate in real conditions because it had to process basically analog signal (the device was connected via the VGA interface). Two experiments were performed in order to evaluate the stability of the feature detector and the performance in a long term navigation task.

6.1 Stability test

The first test used the fact that the device was connected via the VGA interface to evaluate the stability of the feature detector. By stability it is understood the detector's ability to repeatably detect the same image features in the image. In the earlier phase of the development of this thesis the input image data were stored on the MMC memory card connected to the development board. On such a stable data the detector performed flawlessly without any jitter. However when connected to the VGA interface the detector starts to jitter. The first test intended to test the detector's ability to deal with this kind of unstable signal therefore one image was processed 20 times by the detector and positions of located image features were stored in the memory. The stability was measured as a number of points which were detected in the image on the exact same position between more images. The stability was calculated for 2, 10 and 20 consecutive frames. The image feature detector produced in average 372 image feature positions per image and during the whole test it marked 802 different feature positions in the image. Table 5 shows the measured stability success rates for individual scales and number of processed consecutive frames.

The results in table 5 show that approximately 50% of all the image feature data suffer from the detector jitter in a long term. In my opinion these results are caused by minor problems with the description part of the design which processes the data from the detection part. When evaluated only by sight on the screen which is connected between the detector and the descriptor part of the design it doesn't seem that the detector would produce such a big jitter. Moreover on the higher scales the detector jitter ceases which is in conflict with the results. Further testing is necessary for evaluation of these results.

| Scale | Stability[%] (20 frames) | Stability[%] (10 frames) | Stability[%] (2 frames) |
|-------|-----------------------------|-----------------------------|----------------------------|
| 0 | 48.7 | 69.8 | 79.5 |
| 1 | 53.9 | 71.2 | 84.3 |
| 2 | 52.3 | 72.3 | 87.0 |
| 3 | 49.5 | 73.1 | 92.6 |

Table 5: Stability success rates.

6.2 Performance in the long term navigation scenario test

The second test measured the performance of the feature detector in a scenario of outdoor vision-based longterm autonomous navigation. The evaluation method is based on the article [11] and it is described in the article [6]. It focuses on the ability of the feature extraction and matching algorithm to establish heading of the robot relatively to the intended path. The evaluation algorithm establishes the robot heading by finding a modus of horizontal displacements of the tentative correspondences. The modus is found by histogram voting (as it is depicted in the figure 11 in the chapter 4). The dataset used for evaluation contains 12 images from five locations covering seasonal changes of the Stromovka forest park in Prague throughout the year. Figure 21 shows the seasonal changes on one of the five locations. Figure 22 shows the rest of the locations. Individual images at given location vary in seasonal changes and slightly in the viewpoint.

The evaluation was done for the sample Matlab implementation and for the key feature position data obtained from the FPGA feature detector for the rates of 100, 200 and 400 detected features per image. The detection rates for the other feature detectors were

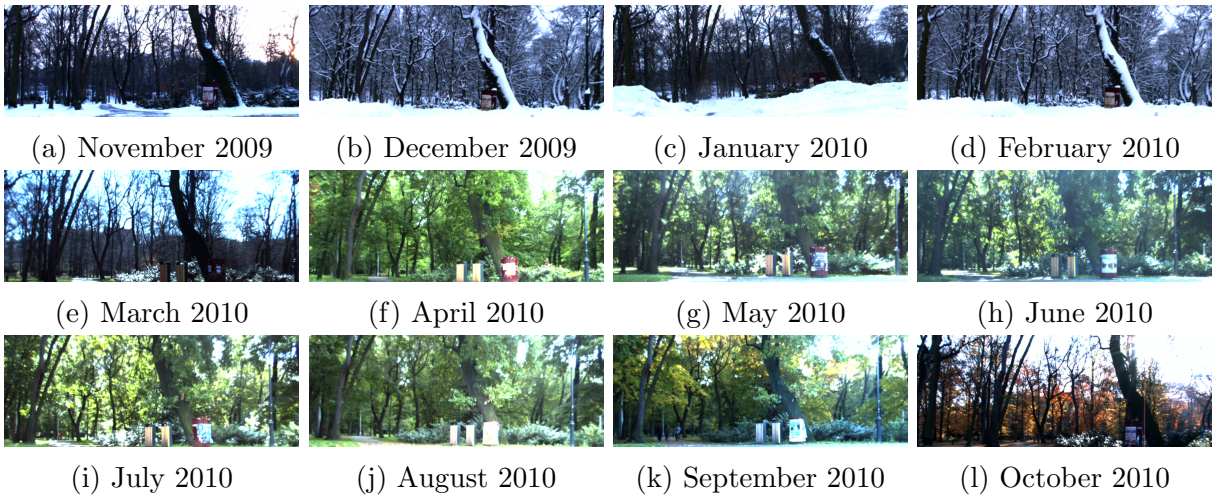


Figure 21: The dataset capturing the seasonal changes. Location II. Courtesy of [6].

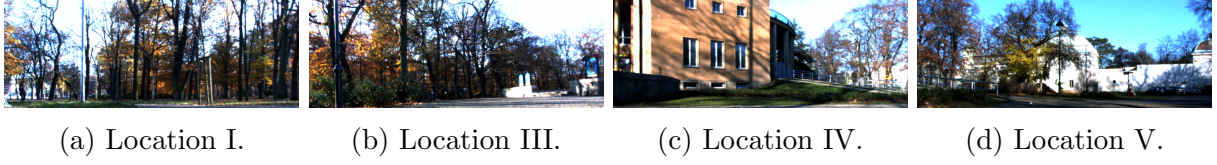


Figure 22: The dataset locations. Courtesy of [6].

provided by thesis supervisor (brief description of these algorithms is in the chapter 3). Due to some issues with the feature descriptor part of the design the descriptors used for the evaluation were computed in software from position data obtained from the FPGA. Two tests were performed. One with the descriptor generated by Matlab according to the implementation (means 128 bit long BRIEF descriptor) and the second with the OpenCV sample BRIEF descriptor(512 bit long). Figures 23a and 23b show the dependence of the success rate on the number of extracted features for the implemented BRIEF128 descriptor and OpenCV BRIEF512 descriptor respectively.

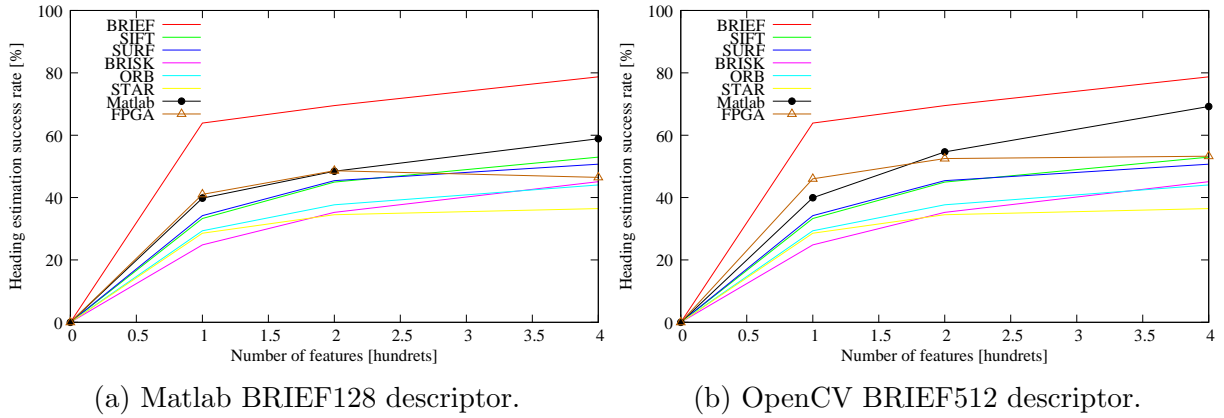


Figure 23: Heading estimation success rate..

The results show that both the FPGA and the Matlab implementations report approximately the same estimation success rate. With the OpenCV BRIEF512 descriptor the results are even better than for the implemented BRIEF128. This will be considered for future work. On this particular dataset both the PC and FPGA implementations even outperform robust image feature detectors like SIFT or SURF. In the evaluation only the original BRIEF descriptor which uses STAR as a detector performs better.

7 Conclusion

This thesis presented a novel FPGA design which is capable of real time image feature detection and description. The main goal of this thesis was to develop an FPGA architecture which is capable of the mobile robot navigation. Therefore I suggested a new method of image feature extraction specifically suitable for the FPGA architecture and tasks related with the mobile robot navigation.

The suitability of the design for FPGA platform is best expressed by the device utilization. The custom designed part of the architecture occupies when placed and routed on high-end Xilinx Virtex-5 FX70T FPGA chip only 5% of Slices and 19% of inner Block RAM resources. This is so few that I tried to fit the architecture into the low-end Xilinx Spartan-3E device and the design occupied only 32% of Slices. The design can also fit with minor modifications without problems into the Altera Cyclone-IV E FPGA on the DE0-nano development board [46] which is sold for 79\$.

The implementation in the current state is designed to be capable of processing in real-time any image with the full frame width up to 2048 pixels (including the display and the blanking time) and a pixel clock up to 200 MHz. This corresponds for example to the VESA Signal 1280×1024 @ 100 Hz timing with 1760 pixels full frame width and the pixel clock of 190.96 MHz. The implementation is capable of feature extraction during the duration of one image frame with the theoretical amount of about 3000 described features per image (depends on the resolution).

All of these results are complemented with the feature detector evaluation results. Due to the introduced simplifications I didn't expect that the implementation can outperform robust feature extractors like SURF or SIFT in the long term scenario navigation task. Therefore I am convinced that the main goal of this thesis was achieved and that the implementation can provide features for the mobile robot navigation.

Considering the future work on this project first I would like to finish the navigation part of the design and test it with an actual mobile robot. The results of the detector evaluation convinced me to push the design further and already now I believe that the whole navigation algorithm can be implemented in the FPGA fabric.

The work on this project was very challenging and it definitely enriched me with a lot of new experience that I would like to use in a further development of this project.

References

- [1] Hana Szücssová. Computer Vision-based Mobile Robot Navigation. Master's thesis, CTU, Faculty of Electrical Engineering, 2011.
- [2] Miroslav Kulich. *Lokalizace a tvorba modelu prostředí v inteligentní robotice*. PhD thesis, CTU, Faculty of Electrical Engineering, 2003.
- [3] Tomáš Krajník. *Large-scale Mobile Robot Navigation and Map Building*. PhD thesis, CTU, Faculty of Electrical Engineering, 2011.
- [4] Jan Šváb. FPGA-based Computer Vision Embedded Module. Master's thesis, CTU, Faculty of Electrical Engineering, 2011.
- [5] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: binary robust independent elementary features. In *Computer Vision–ECCV 2010*, pages 778–792. Springer, 2010.
- [6] T. Krajník et al. Image Features for Long-Term Mobile Robot Autonomy. In *ICRA 2013 Workshop on Long-Term Autonomy*, Karlsruhe, 2013. IEEE. <https://sites.google.com/site/icra2013ltaworkshop/>, [Cit.: 2013-05–2].
- [7] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: Autonomous vehicles in city traffic*, volume 56. springer, 2009.
- [8] Nathan Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The GRASP Multiple Micro-UAV Testbed. *Robotics Automation Magazine, IEEE*, 17(3):56–65, 2010.
- [9] H. Bay, A. Ess, T. Tuytelaars, and L. van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.
- [10] T. Krajník, S. Šváb J., Pedre, P. Čížek, and Přeučil L. FPGA-based module for SURF extraction. *Machine Vision and Applications*, 2013. (in review).
- [11] T. Krajník, J. Faigl, M. Vonásek, V. Kulich, K. Košnar, and L. Přeučil. Simple yet Stable Bearing-only Navigation. *J. Field Robot.*, 2010.
- [12] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *Robotics and Automation, IEEE Transactions on*, 7(3):376–382, 1991.
- [13] Johann Borenstein, HR Everett, Liqiang Feng, and David Wehe. Mobile robot positioning-sensors and techniques. Technical report, DTIC Document, 1997.
- [14] Eric Royer, Maxime Lhuillier, Michel Dhome, and Jean-Marc Lavest. Monocular vision for mobile robot localization and autonomous navigation. *International Journal of Computer Vision*, 74(3):237–260, 2007.

REFERENCES

- [15] Petri Tanskanen Marc Pollefeys Dominik Honegger, Lorenz Meier. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. *ICRA 2013, proceedings of IEEE International Conference on Robotics and Automation*, page 49, 2013.
- [16] Various authors. Ubisense group plc.@ONLINE. <http://www.ubisense.net/en/>.
- [17] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263–296, 2008.
- [18] Chen Chen and Yinhang Cheng. Research on map building by mobile robots. In *Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on*, volume 2, pages 673–677, 2008.
- [19] D. Wolf, A. Howard, and G. Sukhatme. Towards geometric 3D mapping of outdoor environments using mobile robots. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1507–1512, 2005.
- [20] Karel Košnar, Tomáš Krajník, and Libor Přeučil. Visual Topological Mapping. In Herman Bruyninckx, Libor Přeučil, and Miroslav Kulich, editors, *European Robotics Symposium 2008*, volume 44 of *Springer Tracts in Advanced Robotics*, pages 333–342. Springer Berlin Heidelberg, 2008.
- [21] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.
- [22] Colin McManus, Paul Furgale, Braden Stenning, and Timothy D Barfoot. Visual teach and repeat using appearance-based lidar. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 389–396. IEEE, 2012.
- [23] Zhichao Chen and S.T. Birchfield. Qualitative vision-based mobile robot navigation. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2686–2692, 2006.
- [24] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.
- [25] P. De Cristóforis, M. Nitsche, and T. Krajník. Real-time image-based autonomous robot navigation method for unstructured outdoor roads. *Journal of Real Time Image Processing*, 2013. (to appear).
- [26] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environments. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 572–577 vol.1, 1990.
- [27] Steven M LaValle. Rapidly-Exploring Random Trees A New Tool for Path Planning. 1998.

REFERENCES

- [28] Martin Saska, Juan S. Mejía, Dušan M. Stipanović, Vojtěch Vonásek, Klaus Schilling, and Libor Přeučil. Control and navigation in manoeuvres of formations of unmanned mobile vehicles. *European Journal of Control*, 19(2):157 – 171, 2013.
- [29] Michal Čáp, Peter Novák, Jiří Vokřínek, and Michal Pěchouček. Asynchronous decentralized algorithm for space-time cooperative pathfinding. *arXiv preprint arXiv:1210.6855*, 2012.
- [30] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends® in Computer Graphics and Vision*, 3(3):177–280, 2008.
- [31] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.
- [32] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999.
- [33] S. Leutenegger, M. Chli, and R.Y. Siegwart. BRISK: Binary Robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555, 2011.
- [34] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571, 2011.
- [35] Tomasz Trzcinski and Vincent Lepetit. Efficient discriminative projections for compact binary descriptors. In *Computer Vision–ECCV 2012*, pages 228–242. Springer, 2012.
- [36] V. Bonato, E. Marques, and G.A. Constantinides. A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(12):1703–1712, 2008.
- [37] Feng-Cheng Huang, Shi-Yu Huang, Ji-Wei Ker, and Yung-Chang Chen. High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(3):340–351, 2012.
- [38] J Šváb, Tomáš Krajník, Jan Faigl, and L Přeučil. FPGA based speeded up robust features. In *Technologies for Practical Robot Applications, 2009. TePRA 2009. IEEE International Conference on*, pages 35–41. IEEE, 2009.
- [39] D. Bouris, A. Nikitakis, and J. Walters. Fast and Efficient FPGA-Based Feature Detection Employing the SURF Algorithm. In *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, pages 3–10, 2010.

REFERENCES

- [40] N. Battezzati, S. Colazzo, M. Maffione, and L. Senepa. SURF algorithm in FPGA: A novel architecture for high demanding industrial applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 161–162, 2012.
- [41] M. Schaeferling and G. Kiefer. Flex-SURF: A Flexible Architecture for FPGA-Based Robust Feature Extraction for Optical Tracking Systems. In *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, pages 458–463, 2010.
- [42] S. Pedre, T. Krajnik, E. Todorovich, and P. Borensztein. A co-design methodology for processor-centric embedded systems with hardware acceleration using FPGA. In *Programmable Logic (SPL), 2012 VIII Southern Conference on*, pages 1–8, 2012.
- [43] S. Pedre, T. Krajnik, E. Todorovich, and P. Borensztein. Accelerating embedded image processing for real time: a case study. *Journal of Real-Time Image Processing*, pages 1–26, 2013.
- [44] Pavel Holoborodko. Central differences @ONLINE. <http://www.holoborodko.com/pavel/numerical-methods/numerical-derivative/central-differences/>, 2009.
- [45] Various authors. Xilinx, inc.@ONLINE. <http://www.xilinx.com>.
- [46] Various authors. Altera, corp.@ONLINE. <http://www.altera.com>.

Appendix

CD Content

In table 6 are listed names of all root directories on CD

| Directory name | Description |
|----------------|--------------------------------|
| bp | bachelor thesis in pdf format. |
| sources | source codes |

Table 6: CD content