Czech
Technical
University
in Prague

Faculty of Electrical Engineering
Department of Cybernetics

Master's thesis

Embedded module for image processing

Petr Čížek

May 2015
Thesis supervisor: Ing. Tomáš Krajník, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	Bc. Petr Č í ž e k
Studijní program:	Kybernetika a robotika (magisterský)
Obor:	Robotika
Název tématu:	Vestavný modul pro zpracování obrazu

Pokyny pro vypracování:

Cílem práce je zprovoznit a otestovat funkčnost FPGA modulu AB-DE0-Cam pro zpracování obrazu.

1. Seznamte se s daným modulem a jeho rozhraními.
2. Seznamte se s metodami vizuální lokalizace a navigace mobilních robotů.
3. Implementujte vybraný algoritmus na daném FPGA modulu.
4. Algoritmus experimentálně ověřte jak na dodaných datových sadách, tak nasazením na reálném robotu.
5. Vytvořte technický popis modulu tak, aby byl snadno použitelný i na jiných platformách.

Seznam odborné literatury:

[1] Krajník et al.: Simple, Yet Stable Bearing-Only Navigation. Journal of Field Robotics, 2010
[2] Krajník, Šváb, Pedre, Čížek, Přeučil: FPGA-Based Module for SURF Extraction. Machine Vision and Applications. 2014
[3] Engel et al: LSD-SLAM: Large-Scale Direct Monocular SLAM, ECCV 2014
[4] Konolige et al: Large-scale visual odometry for rough terrain. In Robotics Research. Springer, 2011
[5] Calonder et al: BRIEF: Binary Robust Independent Elementary Features, ECCV 2011

Vedoucí diplomové práce: Ing. Tomáš Krajník, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 21. 1. 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student:	Bc. Petr Č í ž e k
Study programme:	Cybernetics and Robotics
Specialisation:	Robotics
Title of Diploma Thesis:	Embedded Module for Image Processing

Guidelines:

Scope of the thesis is to design, implement and experimentally verify a visual-based mobile-robot navigation method on an FPGA embedded module.

1. Learn about the embedded module and its peripherals.
2. Learn about mobile robot visual localisation and navigation methods.
3. Implement selected navigation algorithm on the module.
4. Verify the algorithm in the real navigation task and on the given datasets.
5. Provide the technical documentation of the module.

Bibliography/Sources:

[1] Krajník et al.: Simple, Yet Stable Bearing-Only Navigation. Journal of Field Robotics, 2010
[2] Krajník, Šváb, Pedre, Čížek, Přeučil: FPGA-Based Module for SURF Extraction. Machine Vision and Applications. 2014
[3] Engel et al: LSD-SLAM: Large-Scale Direct Monocular SLAM, ECCV 2014
[4] Konolige et al: Large-scale visual odometry for rough terrain. In Robotics Research. Springer, 2011
[5] Calonder et al: BRIEF: Binary Robust Independent Elementary Features, ECCV 2011

Diploma Thesis Supervisor: Ing. Tomáš Krajník, Ph.D.

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, January 21, 2015

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 11. 5. 2015

Acknowledgement

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

Abstrakt

Tato práce předkládá kompletní návrh vestavného FPGA modulu pro vizuální navigaci mobilního robotu. Jádrem práce je implementace metody pro extrakci lokálních rysů obrazu v logice FPGA. Metoda se skládá z detekce významných bodů v obraze, vytvoření popisu jejich okolí, porovnání jejich popisu se známou mapou a výpočtu řídících povelů pro mobilní robot. Práce navíc předkládá kompletní metodologii pro návrh a akceleraci algoritmů strojového vidění na FPGA jako výsledek tříleté zkušenosti s výzkumem v oblastech FPGA architektur, strojového vidění a mobilní robotiky. V závěru této práce jsou experimentálně vyhodnoceny výkon prezentovaného vestavného modulu a jeho porovnání s ostatními existujícími implementacemi podobného charakteru.

Klíčová slova: FPGA, Strojové vidění, Mobilní robotika, Vestavný systém

Abstract

This thesis presents a complete hardware and software design of an embedded computer vision module based on a low-end FPGA platform capable of real-time mobile robot navigation. The core of the presented work is a method for a real-time image feature extraction and matching, which is completely implemented in the FPGA fabric. The method implements all stages required for visual-based teach-and-repeat mobile robot navigation, i.e. detection of salient points in the image, description of their surroundings, establishing their correspondence with a known map and calculation of the robot's steering commands. Moreover three years of experience and continuous research in the fields of FPGA prototyping, computer vision and mobile robotics resulted in a generalization of complete methodology and framework for acceleration of computer vision applications on the FPGA platform which is also presented in the thesis. The proposed embedded module performance is evaluated in the end of this thesis and compared to the other embedded solutions used in the field of the mobile robotics.

Keywords: FPGA, Computer vision, Mobile robotics, Embedded systems

Contents

List of Figures

List of Tables

Part 1
Introduction

Visual based mobile robot navigation is a very comprehensive task which is in the scope of the mobile robotics and automation for nearly five decades. The main reason for studying how to guide robots by means of visual information is the natural way humans perceive their environment. With the growing power of computers and availability of cheap camera sensors it is possible to build more and more complex system capable of intelligent behaviour. It is possible to perform more complex processing and reasoning about the visual information and well known results like autonomously driving cars [11], [7], Mars exploration rovers [12] or squads of quadcopters capable of synchronous movement in an confined spaces [13] are only a tip of an iceberg of image processing applications in mobile robotics.

However sometimes the application imposes strict restrictions on hardware dimensions or power consumption of image processing systems. For example unmanned aerial vehicles (UAVs) or space probes are a typical platforms which are restricted by lifting capacity and a power consumption.

In conventional processor architectures the computation power goes hand in hand with power consumption so the resulting visual navigation algorithm needs to balance between speed, power consumption and complexity. In such case advantages of the Field-programmable gate arrays (FPGA) can be exploited. The FPGA architecture allows the programmer to built custom designs specifically suitable for given tasks by programming the actual logic architecture of the digital circuit. They are well suited to tasks that can be pipelined or run in parallel. Since the FPGA programming is effectively designing an electronic circuit, it is low-level and highly abstract. The optimisation of the architecture generally leads to the lower power consumption and higher processing speed compared to the classical processor approach. Lately the trend is to provide a fusion of classical processor architectures together with versatility of the FPGA fabric in order to benefit from strengths of both platforms. This System on a chip (SoC) approach allows developers to apply a combination of serial and parallel processing in order to achieve higher performance of their architectures. However the development may last for months or even years. Lately this problem was addressed by methodologies [14] and subsequently by official development tools [15] which simplifies the design flow and allows for rapid development of FPGA systems. Last but not least, FPGAs as a platform proved to be radiation tolerant [16] which makes it well suitable for space or hazardous radioactive environments applications.

This thesis builds on the results of a three year long research which focuses on building an embedded module capable of real-time mobile robot navigation. First attempts to build a module capable of mobile robot navigation were described in the thesis [17] and the following paper [18]. Although presented module was capable of advanced image processing, it was not able to perform the visual navigation algorithm and it was relatively expensive. The room for improvement was in making the whole system smaller, cheaper and faster.

This thesis presents an embedded module for image processing and mobile robot navigation based on a low-cost FPGA development board which was extended by a camera sensor. It proposes a complete design of a standalone embedded module capable of real-time mobile robot navigation based on extraction of natural visual landmarks from onboard camera. Besides the custom designed hardware, a complete system on a chip solution is presented which consists of the image feature detector, the image feature descriptor and navigation algorithm completely implemented in the FPGA fabric. Moreover the thesis generalizes a framework for building FPGA architectures specifically suitable for accelerating computer vision applications by introducing a set of basic building blocks and unifying their interfaces.

The thesis is structured in three chapters. First chapter provides the reader with the background of computer vision methods and mobile robot navigation. Division of mobile robot navigation methods based on the level of interpretation and processing of the input visual data is presented. Second chapter presents a complete design of the embedded module for mobile robot navigation. Overall system setup together with the architecture design details, hardware specification and software implementation are covered in this chapter. Last chapter contains evaluation of the presented module on real experiments and real datasets.

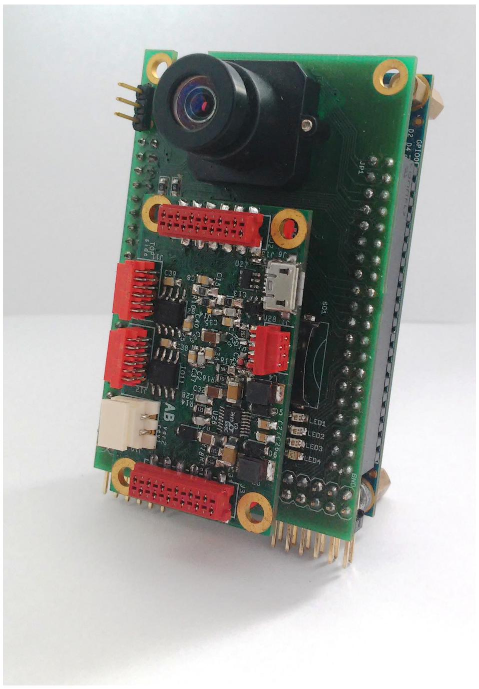


Figure 1: Developed embedded module.

Part 2

Computer vision for mobile robot navigation

This chapter provides the reader with the overview of computer vision techniques as well as associated navigation techniques used in mobile robotics. In surveys on computer vision in mobile robotics [19], [20] computer vision methods are usually described with respect to navigation methods. Namely based on the utilisation of model (map) of the environment into three groups: map based navigation, map-building based navigation and map-less navigation. Computer vision methods are in these surveys described only as a tool of selected navigation algorithms. Recently there was a paper published dealing with classification of vision systems for ground mobile robots [21]. It presents an overview of vision sensors and techniques, but the computer vision methods are again presented only as a tool of selected navigation methods.

This is somehow convenient because computer vision is a very extensive discipline and individual methods can be used in various ways in the field of mobile robotics. Moreover there is no recognised taxonomy of computer vision methods and therefore it is very hard to find a key division criteria. Despite these facts this chapter presents the division of visual mobile robot navigation methods with respect to computer vision methods.

Hence this chapter does not intend to give an exhaustive survey on robotic navigation nor computer vision techniques rather to explain key methods of computer vision used in mobile robotics and provide an overview of navigation techniques relying on them.

This chapter is structured in two parts. First part of this chapter provides brief overview of mobile robot navigation processes and tasks together with few examples. The second part describes individual computer vision methods together with navigation algorithm examples.

2.1 Mobile robot navigation

Navigation is in general a problem that encompasses localisation, mapping and motion planning [22]. While localization is a process of robots position determination in the environment by acquiring and processing sensory information, mapping is a process of building a spatial or spatio-temporal model (map) of the environment in which the mobile robot operates and motion planning is a process of finding a safe and suitable path through the environment. In a broader sense, navigation is the determination of position and direction to the target goal. Figure 2 shows simplified relations between these three tasks.

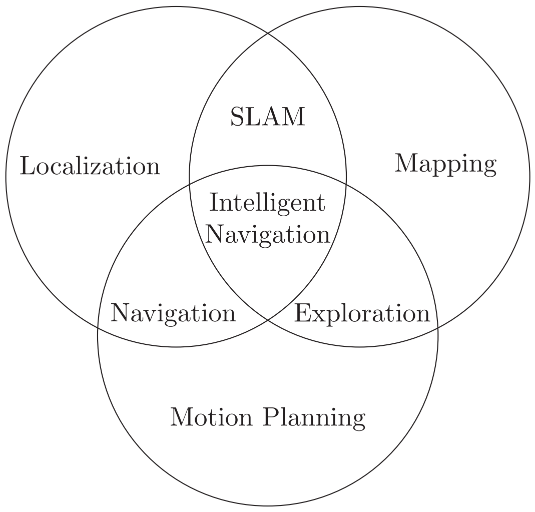


Figure 2: Navigation tasks relations. Courtesy of [1].

2.1.1 Mapping

Mapping is a process of building a spatial or spatio-temporal model (map) of the environment. Based on the utilisation of model (map) of the environment three main groups of navigation methods which uses computer vision are recognised in the literature [19], [20]. These three groups are

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

Main idea of the first two groups is to provide the robot with a set of landmarks expected to be found during the navigation. By map-based navigation the map is a priory given to the robot before the navigation starts or though in a teach-and-repeat like scenario during the first guided pass through the environment [19].

Map-building-based navigation can also be referred to as simultaneous localisation and mapping (SLAM) [23]. In the SLAM the robot maps the environment while using the map for localisation. This leads to a problem when the map building of the environment depends on accurate localization of the robotbut the quality of localization depends on the fidelity of the built map. This results in error accumulation over time. The accumulated error can be reduced through loop-closure techniques, where the robot would revisit known locations and use these as 'anchor points' for graph-relaxation methods, that would redistribute the accumulated error more evenly throughout the SLAM-created map.

There are generally four basic types of maps varying in the level of abstraction which is inserted into the process of navigation [1], [24]. These four groups are sensory, geometric, topological and landmark maps.

Sensory maps are only appropriately represented and saved sensory data. Sensory maps are quite simple to construct but they are always memory demanding because of a large number of observations accumulated over time particularly when the long-term navigation is performed. Typical example of sensory maps are 3D point clouds or their memory efficient version, occupancy grids [25], which represents basically the same data in the discrete raster.

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 well-defined corners and flat surfaces are present.

Article [26] presents a method for extracting geometric primitives from the laser scan point cloud data and investigates mutual relations between these primitives in order to be able to re-identify commonly appearing structures in the environment.

Another example of geometric mapping is in the paper [27] where the authors used laser scan data of a large scale urban area to built the map from simple plane primitives.

Topological maps are basically graphs. Nodes represent 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 map usage is in the paper [28] where authors present an outdoor topological exploration system based on visual recognition. In their experiment the mobile robot followed paths in the park environment mapping intersections and building a topological map which can be later used for navigation. Nodes represented intersections and edges individual pathways. The mobile robot navigated itself reactively along these pathways.

Landmark maps contains information about landmarks which could be observed from given location. Landmarks should be salient features of the environment which are easily detectable, recognizable and traceable. Localization and navigation is done by detecting landmarks and matching them with those from the map. Landmark maps are frequently utilised in visual based navigation systems. Either in a teach and repeat scenarios [29], [30], [31] where they represent an expected observation of the robot at a given position or in map-building based navigation systems like MonoSLAM [32] which uses landmark map for probabilistic feature tracking and real-time camera position estimation.

In contrast to aforementioned, map-less navigation systems do not use any map for navigation. Therefore they are also called reactive navigation systems because the navigation is only a reaction on current observation of the environment. These architectures were first introduced by Marvin Minsky [33] and are essential building blocks of the subsumption architectures introduces by Rodney Brooks [34]. Vision based map-less navigation algorithms usually uses visual clues derived from the segmentation of an image [35], optical flow [36] or visual odometry for obstacle avoidance or path-following [3].

2.1.2 Localisation

Localisation is a process of robots position estimation in the environment. It is very closely related to the mapping because in navigation methods which use maps for navigation it is essential to localise the robot within the map. Localisation can be either incremental or absolute [37].

Incremental localisation (also called dead-reckoning) is usually used in map-less navigation methods. It assumes starting position of the robot to be known. During the navigation the robot estimates its current position from the previously known position and an increment of travelled distance. However integration of the travelled distance leads inevitably to the unbounded accumulation of errors and therefore incremental localisation is solely not usable for long-term navigation. Well-known and widespread examples of incremental localisation are odometry and inertial navigation (i.e. navigation using accelerometers and gyroscopes). Incremental localisation techniques are widely utilised for their speed and simplicity.

Absolute localisation, on the other hand, has no apriory information about the starting position of the robot and it has to estimate its position in a global coordinate system. It usually relies on the map of the environment in which the navigation system must construct a match between the observations and the expectations. Special type of absolute localisation is the one using external beacons artificially placed in the environment like Ubisense systems, or global positioning systems like GPS or Galileo.

In a lot of systems both types of localisation are utilised. Usually the incremental localisation is supportive to the absolute localisation by providing frequent updates to the vehicle position estimation while the absolute localization is performed at lower rates. A common framework to fuse both types of localization is for example Kalman filtering [22].

2.1.3 Motion planning

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

By map-less navigation methods motion planning is based directly on current observation of the environment. Collision avoidance is the primary goal of the navigation. Methods like virtual force fields [38] or drag processes [35] can be used in the reactive control of the mobile robot.

By map based and map-building based navigation, motion planning seeks a path in a state space or configuration space of the robot. Used algorithms strongly depends on the utilised map type and the dimensionality of the problem. If the state space is too big randomised algorithms are usually utilised. Examples of randomised algorithms are Markov processes, Simulated Annealing or Rapidly Growing Random Trees. Otherwise the motion planning problem is transfer to the discrete raster and solved by any graph traversal algorithms like A* or Dijkstra.

2.2 Computer vision for mobile robotics

First of all it has to be noted that the computer vision is a very extensive discipline involving a lot of results and methods of mathematics, pattern recognition, artificial intelligence, computer science, electronics and other scientific disciplines. A lot of high-level knowledge, semantic information and context is explored in order to imitate the human natural way of understanding a visual information. The main purpose of all computer vision methods is to interpret the content of visual observation and reduce the information in the image to relevant information for the application domain [39], [40] which is in our case mobile robot navigation. This process of abstraction is one of main reasons why is the computer vision so difficult.

However interpretation in this context is more like a getting viable informations from the image or sequence of images rather than understanding the content of the scene in a way humans perceive the environment. Its upon the user and implementation to choose which informations are relevant for the application and which are not.

In the field of mobile robotic mapping and localisation all operations performed over the visual data lead to some informations about the robot movement or surrounding environment which are utilised in the navigation algorithm. Several key concepts from computer vision are used. These are optical flow, feature detection, segmentation, pattern and object recognition, neural networks and lately deep-learning. Besides these methods which may be solely monocular there is a whole field of stereo-vision and multi-view systems. Now every method will be discussed in more detail together with its applications in the mobile robot navigation.

2.2.1 Optical flow

Optical flow is one of the tools of the motion analysis. It represents the motion of objects in a visual scene typically by a vector field. Each vector approximates the apparent motion of a pixel or a group of pixels within an image. Optical flow is calculated between consecutive frames in a video sequence and it forms a projection of 3D velocity field on the image plane [41] as can be seen in the Figure 3.

While several different approaches to optical flow estimation were proposed the basic gradient-based optical flow computation can be expressed by the equation 1 where $\mathbf{I}(x, y, t)$ is the image brightness of point x, y at the time t and $\mathrm{d}x, \mathrm{d}y, \mathrm{d}t$ are small increments in position and time respectively.

$$\mathbf{I}(x, y, t) = \mathbf{I}(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t). \tag{1}$$

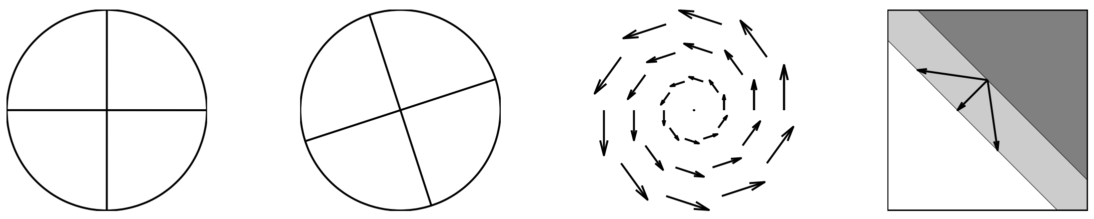

(a) Frame k. (b) Frame $k+1$. (c) Optical flow. (d) Aperture problem.

Figure 3: Optical flow principle.

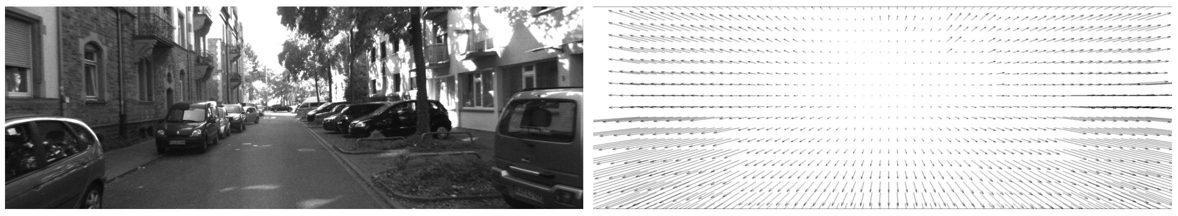

(a) Image from the video sequence. (b) Estimated optical flow.

Figure 4: Optical flow estimation example. Courtesy of [2].

The desired result is a motion direction and motion velocity at (possibly all) image points expressed as

$$\mathbf{c} = \left(\frac{\mathrm{d}x}{\mathrm{d}t}, \frac{\mathrm{d}y}{\mathrm{d}t}\right) = (u, v) \,. \tag{2}$$

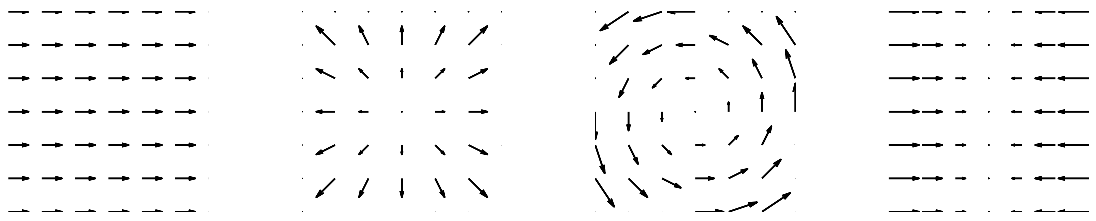
This applies under the following assumptions:

1. image brightness at given point is constant over time,
2. nearby points in the image move in a similar manner.

However these assumptions are often violated in practice which may result in significant differences between optical flow field and the actual motion field. This is mostly due to the aperture problem which means that regions of constant intensity and edges parallel to the direction of motion exhibit no apparent sign of motion. Only edges with a component normal to the direction of motion carry information about the motion. Figure 3d illustrates the aperture problem. When there is only a limited field of view the apparent motion can not be uniquely determined.

Probably the most comprehensive list of optical flow estimation algorithms is maintained by the Karlsruhe Institute of Technology as a part of their benchmarking suite [42] for optical flow algorithms. Figure 4 illustrates an optical flow estimation from the dataset sequence of camera images taken from a moving car.

In the field of mobile robot navigation optical flow is used to recover robot motion, detect obstacles, avoid collisions, recover scene depth and track moving objects.



(a) Translation at constant distance. (b) Translation in depth. (c) Rotation in constant distance. (d) Rotation in depth.

Figure 5: Motion recognition from optical flow.

Motion estimation

Optical flow is primarily used in motion estimation. By examining the resulting optical flow vector field, relationships can be concluded on extracting the camera motion. Usually considered motions appearing in the image are

1. translation at constant distance (Fig. 5a),
2. translation in depth (Fig. 5b),
3. rotation at constant distance (Fig. 5c),
4. rotation of a planar object perpendicular to the view axis (Fig. 5d).

From these primitives motion of the camera can be estimated. Very important element of the analysis is the focus of expansion (FOE) point. All of the velocity vectors in the stationary scene will meet at the FOE. If several independently moving objects are presented in the image, each motion has its own FOE.

The method itself, however, provides only velocity measurements which may be integrated over time in order to obtain position measurement. Therefore it is suitable for relative localisation only. In the long time horizon, accumulation of position error is unbounded. The principle was first used in computer optical mice however nowadays it is fairly popular because it provides an enhanced navigation accuracy for robots using any type of locomotion on almost any surface. Due to their affordability, the optical mice are actually used in some robotic systems [43], [44].

An overview of optical flow methods used in mobile robotics is presented in the survey [45] which focuses on a survey of both the optical flow related sensor hardware, the software and optical flow field estimation models used for UAV navigation.

Another example of motion estimation from optical flow is presented in the article [36]. It presents a reactive navigation architecture for corridor navigation by balancing an optical flow on the left and the right side of the robot. This approach is inspired by behaviour of insects and birds .

Concerning hardware for optical flow calculation targeting field of mobile robotics, article [8] presents an open source and open hardware design of the PX4Flow optical flow sensor for indoor and outdoor robotic navigation.

Scene depth estimation.

As long as the camera movement is along its optical axis, depth of each point with known velocity can be computed according to the equation:

$$\frac{D(t)}{V(t)} = \frac{z(t)}{w(t)}, \tag{3}$$

where $D(t)$ is the distance of a point from the FOE, $V(t)$ is its computed velocity and the ratio $\frac{z}{w}$ specifies the time at which the point moving at constant velocity w crosses the image plane. At least one actual distance z has to be known in order to evaluate the distance exactly. However when there is no movement of the camera the depth can not be recovered.

Obstacle detection and collision avoidance

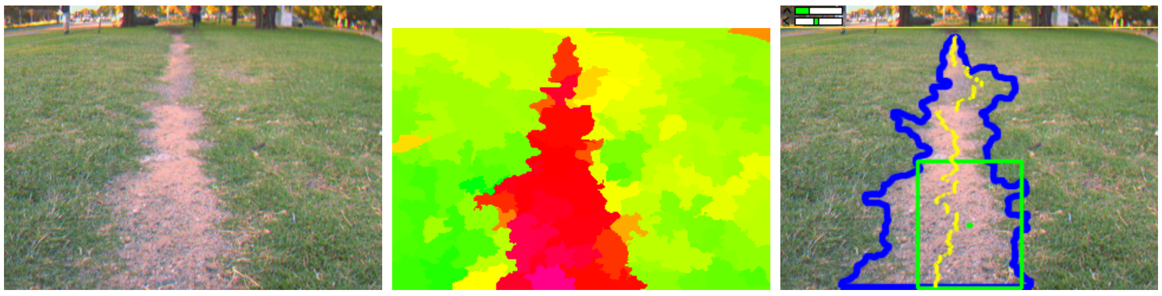
The obstacle detection and collision avoidance is based on segmentation of the optical flow, establishing the FOE of each object and then estimating its depth. In most cases it is sufficient to calculate the time to contact (TTC) without the exact metric depth information like it is presented in the paper [46] where the reactive navigation algorithm is presented to guide the robot through the corridor while avoiding obstacles.

Article [47] presents a method for obstacle detection and collision avoidance by combining optical flow with a stereo-based depth estimation. In this case the depth estimation provides the metric measurement and helps recognize the boundaries of obstacles.

The main advantage of optical flow utilisation in mobile robotics is its simplicity and speed. It can provide sufficiently precise informations for relative localisation and obstacle avoidance when the apparent motion in the scene is small enough. The main disadvantage is then connected with no interpretation of the scene. Hence the optical flow is not usable for absolute localisation. There are no direct means for recognizing previously visited places using optical flow.

2.2.2 Segmentation

Another image processing method is segmentation. Its purpose is to assign each pixel in the image with a label such as pixels with same label share some characteristic or computed property like color, intensity, or texture [40]. It could be either binary or multilevel. Binary segmentation assigns each pixel only one of two values (has or has not a property). Multilevel segmentation assign pixels in more classes based on some heuristic. This heuristic may be purely image based or utilise informations from other



(a) Original image. (b) Segmentation. (c) Contour and control law.

Figure 6: Segmentation of unstructured road example. Courtesy of [3].

(a) Original image. (b) Depth map. (c) Segmentation.

Figure 7: Segmentation example. Courtesy of [4].

methods like depth estimation or optical flow. Typical methods of segmentation are region growing and thresholding.

In the field of mobile robotics segmentation is usually used for locating objects and boundaries in order to determine where the free navigable space is. There are a lot of papers published on this topic regarding mainly autonomous vehicles and drive assistant systems where segmentation is used to demark the road and obstacles. It can also help with building of geometrical and topological maps of the environment.

An image based heuristic presented in paper [3] uses mean value of color and hue to connect individual segments. The mobile robot then follows the path demarked by the center of traversable area contour as it is depicted in the Figure 6. Moreover this method presents a way to incorporate the results in topological mapping [28], [48], where the robot traverses paths in the semi-urban environment detecting crossroads and navigating along individual paths in a reactive way.

Another method proposed in [35] uses multilevel segmentation to define the region of interest (navigable area) and then seeks an ideal threshold for segmenting this area by using drag forces to classify the optimal way in terms of fluidity and navigability. The method was tested in the obstacle detection scenario and a reactive navigation system which corrects the robots steering angle to the center of the segmented navigable area.

Method [4] segments the image based on the depth information from a camera stereo pair. Segments are connected based on persistence analysis thus according to the sensitivity to the slight variations in illumination. Figure 7 shows the result of segmentation of a traffic scene. The method was benchmarked on the KITTY dataset [42].

These are just few examples of segmentation usage in the field of the mobile robot navigation. Although segmentation can be used solely for the reactive navigation, it can be also incorporated in high-level image processing techniques like object recognition and several feature extraction methods rely directly on the segmentation of the image.

2.2.3 Image features

A local feature is an image pattern which differs from its local neighbourhood and is expected to be reliably and repeatably detectable, preferably invariantly to camera viewpoint changes. Features can be either artificial, like blobs or patterns, or naturally occurring in the environment. Methods for their detection can be then engineered artificially, based on some property of the image, or adaptive and self thought. This gives us a natural way of division of methods relying on image features into the following three classes.

1. Artificial features, artificial algorithm.
2. Natural features, artificial algorithm.
3. Natural features, adaptive algorithm.

The first class is represented by pattern detection methods where the features are not commonly occurring in the environment and they are built with the intention of simple detection and tracking [49], [50], [51], [52]. The second class is represented by feature extraction which uses natural features of the environment which are, however, defined by some engineered property like large gradient [53], [54] or uniformity of some image area [55]. The third class includes neural networks and deep learning which adapts their abilities directly to the environment [56], [57]. These algorithms are, however, considered black box and they can suffer by an unpredictable behaviour. Now each class will be discussed in more detail

2.2.3.1 Pattern detection

Pattern detection falls into the class of artificial features and artificial algorithms. It is a process of detection of artificial landmarks which are usually well distinguishable in the environment and easily traceable using template matching and tracking. Typical artificial features are defined by high contrast or unusual colors. There are various types of landmarks from circles with a unique bar-code for each landmark to reflecting tapes or colourful sheets of paper set in the environment or on the robots. Pattern detection frequently utilises other methods of computer vision like segmentation with binary thresholding or edge detection.

Paper [58] presents an overview of binary markers like those depicted in the Figure 8 which shows four different black-white landmarks examples with an inner code pattern for identification of single landmark. However this paper is not primarily targeting field of mobile robot navigation.

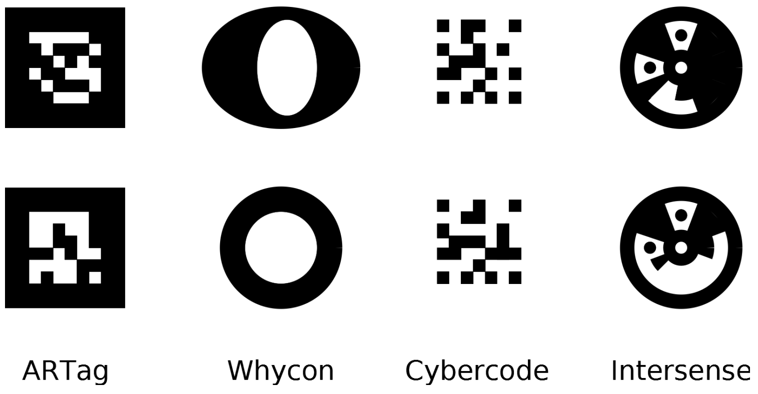


Figure 8: Artificial landmarks examples.(ARTag [49], Whycon [50], Cybercode [51], Intersense [52])

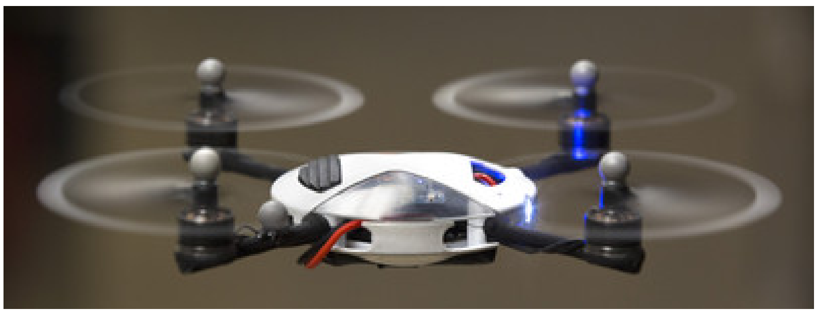


Figure 9: Vicon markers (grey balls at the top of the rotors), Courtesy of [5]

A practical multirobot localization system named Whycon presented in article [50] uses elliptical patterns as depicted in the Figure 8. It uses the fact that circles transforms to ellipses under projective transformations. By measuring the length of principal axes of detected ellipses their spatial position can be determined.

Another example of well known landmark system used in robotics is a Vicon motion capture system [5]. It consists of a set of cameras placed in the environment and markers which are set on the robotic platform. Figure 9 shows an quadrotor with attached Vicon markers on it (silver balls). The system provides an external localisation to the robot.

Pattern detection is mostly utilised in the environments which can be set up in advance. Such conditions are usually possible in the industry. Another usage of pattern detection is for ground truth provision for testing of other methods. Pattern detection is usually very fast and very precise in estimation of robot position.

2.2.3.2 Feature extraction

A local feature is an image pattern which differs from its local neighbourhood and is expected to be reliably and repeatably detectable, preferably invariantly to camera viewpoint changes and seasonal changes. Features can be either points, edgels or image patches which posses some characteristic property or properties. Commonly considered properties are intensity, color and texture [6].

In order to be able to distinguish between individual features, extraction 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 additional information about the located feature (e.g. scale, contrast, etc.). The interest point description is a process when the surrounding of the detected feature is described in order to allow distinguishing individual features. The process of establishing feature correspondences is called feature matching where the description of two points is compared in order to determine whether these two points correspond to the same feature of the environment.

Feature extractor performance is usually evaluated by the following properties.

Repeatability characterizes how well the detector handles variations in lighting conditions and camera viewpoint changes.

Distinctiveness provides uniqueness of the feature. In broader sense, how well will the feature matching perform.

Computation efficiency tells how fast can be features extracted from the image.

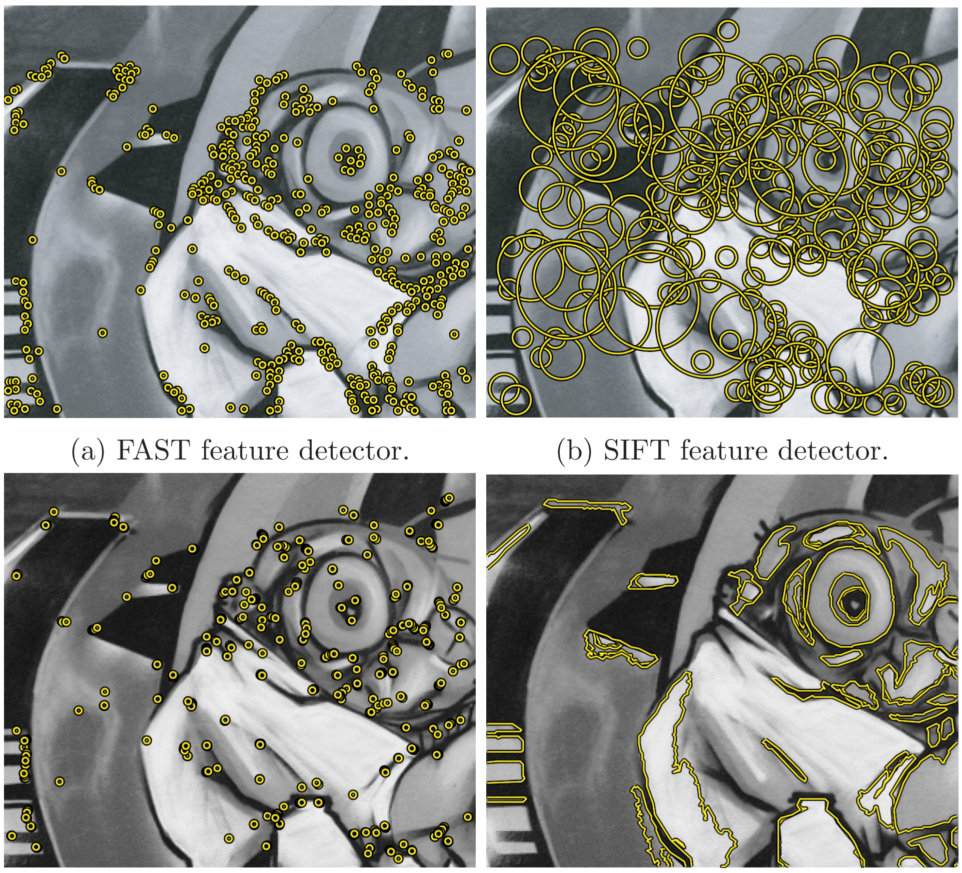
Quantity tells how many features are detected in the image. It should be adjustable while features should provide complex image representation and reflect image information content.

There is a standardised methodology described by Mikolajczyk [6] for evaluation of extractor invariance to image scale, rotation, exposure and camera viewpoint changes. A new, comprehensive evaluation of state-of-the-art image feature extractors was presented in the article [59].

In the scope of mobile robotics is nowadays also a long-term mobile robot autonomy. The article of T. Krajník et al. [10] presents a survey on image feature extractors and their ability to cope with the seasonal changes.

A lot of different feature extractors varying in complexity are described in the literature and it would be impossible to cover an sufficiently detailed survey on individual methods in this chapter. It also has to be distinguished between feature extraction, feature detection and feature description because some of the methods presents only detectors of visual landmarks and some only description methods. It is usual to combine various detectors and descriptors in order to achieve optimal performance for the given application domain.

Two basic and mostly utilised feature extractors, two feature detectors and one feature descriptor are described in the following listing. All of them are part of the Open Source Computer Vision (OpenCV) software library.


(a) FAST feature detector. (b) SIFT feature detector.

(c) Harris corner detector. (d) MSER region detector.

Figure 10: Feature detection examples. Courtesy of [6].

Scale Invariant Feature Transform (SIFT) extractor

SIFT feature extractor is well established detector and descriptor presented by D. Lowe [53] in 1999. It uses a set of Difference of Gaussian filters in a scale space Gaussian pyramid to detect corners in the image. It features high precision and good robustness, however it is computationally demanding. An example of detected SIFT features is in the Figure 10b. The descriptor is based on local gradient orientation histograms in the 16×16 neighbourhood of the feature while taking into account the orientation of the feature.

Speeded Up Robust Features (SURF) extractor

SURF feature extractor [54] is based on the SIFT extractor but uses several approximations which allow acceleration of the computation. The main difference is utilisation of so-called integral image for estimation of Gaussian filter responses approximated by box filters. This allows computation of filter responses for any scale in constant time. The descriptor is formed in a similar way as by SIFT using gradient orientation histograms where the gradient is calculated using the box filter responses.

Features From Accelerated Segment Test (FAST) detector
FAST feature detector [9] belongs to the family of so called appearance based detectors because it searches the image for “corner-like” structures. It is optimised with respect to the computation speed. It uses a carefully chosen set of comparisons in the circular neighbourhood of the expected feature point in order to determine whether it is or it is not a corner. An example of detected FAST features is in the Figure 10a. The FAST feature detector was selected for implementation in this thesis therefore a more thorough description is presented in the section 3.1.1.

Maximally Stable Extremal Regions (MSER) detector
The MSER detector [55] is a region detector based on the region growing in a binary thresholded image. While increasing the threshold, new regions appear in the image and are merged together. During the region growing, the region statistics (area and the border length) are monitored. The detector finds intensity ranges where the ratio of the area and border length exhibit smallest changes. An example of MSER features is in the Figure 10d.

Binary Robust Independent Elementary Features (BRIEF) descriptor
BRIEF is a binary feature descriptor presented in the article [60]. It uses pairwise intensity comparisons of pixels inside an image patch surrounding the located feature. These comparisons form a set of unique binary tests which are subsequently stored into an n-dimensional bit vector. The pairwise comparison locations may be artificial or randomised as it is presented in the original paper or genetically optimised for the given environment [56]. It is well suitable for the mobile robotics due to its simple computation and in certain scenarios it showed to outperform other descriptors [10].

For mobile robotics it is important that features provide a limited set of anchor points in the environment. These points represent a description of the environment which can be related with the description from a different time or with some kind of a model (map). Process of establishing pairwise correspondences is called feature matching. Its input are two sets of points and the output are pairwise correspondences of points preferably in a 1:1 manner called tentative correspondences. However tentative correspondences always contain some outliers which are usually filtered away in post-processing steps. Anyway these correspondences are the key input to a lot of various navigation methods.

Image features are especially useful for wide-baseline matching, where the viewpoints differ in a significant way.

In the field of mobile robotics features are used to calculate sparse optical flow, recover robot motion, localise robot in the environment and recover scene depth. All these tasks are very closely related and in principle share the same set of methods of handling feature correspondences.

Sparse optical flow

Results of the feature matching can be directly used as a sparse optical flow. The correspondences are tracked in a frame-by-frame manner and their difference in coordinates is used for optical flow recovery. Everything mentioned in the section 2.2.1 then applies in the same way. Using features in the optical flow estimation has an advantage in a more robust estimation because only a well traceable set of points is chosen for flow calculation. Moreover the sparse points can be used for dense flow calculation and significantly improve its results because they provide a set of well defined anchor points. On the other hand feature extraction and tracking is a computationally more intensive task than the simple gradient methods of the optical flow.

The correspondence outlier problem can be addressed for example by efficient feature selection like it is described in the article [61] which presents a methodology for efficient feature selection for purposes of motion estimation from sparse optical flow.

Motion estimation and localisation

Both the motion estimation and localisation are trying to determine the position of the camera either with respect to the map of the environment or a previous state of the camera. So the main goal is to determine the extrinsic parameters of the camera which are basically the 6 degrees of freedom (3D rotation and movement). In many applications, however, the mobile robot movement is somehow constrained and therefore it is unnecessary to determine all 6 degrees of freedom [62]. The determination of extrinsic parameters of the camera from feature correspondences also helps with verifying these correspondences and filtering outliers.

The core source of knowledge describing geometry of computer vision is Richard Hartley's book Multiple view geometry in computer vision [63].

In the context of map-less navigation, features are tracked only for a short time. The camera position is estimated in a frame-by-frame manner usually by calculation of homography or projective transformation between individual frames in a RANSAC scheme [63]. Both monocular and stereo approaches are described in the literature. Well known example is an article presented by Kurt Konolige et al. [64] dealing with large scale visual odometry for outdoor use. Their approach uses camera stereo-pair and feature tracking for precise visual odometry estimation.

The main source of error in visual odometry are inaccurately determined positions of features in the image. While this is inevitable, a lot of methods use so called keyframes for camera motion estimation. Between individual frames the camera motion is usually not big enough and the calculation of camera motion would be erroneous due to numerical instabilities. Therefore the landmarks are tracked until their displacement with respect to the keyframe is not sufficiently large and then the camera motion is estimated and new keyframe is selected.

In map-building-based navigation features are repeatedly added to the map as the robot traverses the environment. The robot then localises itself in this map. These methods are called simultaneous localisation and mapping (SLAM) and represents a huge branch in the field of mobile robot navigation. Because the detection of image features is not infinitely precise there is always some uncertainty in the position of features which results in uncertainty in position of the robot. This problem is addressed both by bundle adjustment [65] and loop closing [66]. Through application of these two principles, SLAM performs better than visual odometry in terms of precision. The purpose of the SLAM is to built a map of the environment and provide better localisation than when using visual odometry only. An example of such a method is a semi-direct monocular visual odometry presented by Christian Forster, et al. [65] which uses tracking of FAST feature points and map building for visual odometry estimation.

Typical examples of the visual SLAM methods using mono and multi-view vision are methods by E.Royer et al. [67] and S. Se et al. [68] respectively.

Another example is a MonoSLAM [32] which uses monocular camera in a map-building based navigation systems which utilises landmark map for probabilistic feature tracking and real-time camera position estimation.

In the map-based navigation map of expected features is a priory given to the robot or though in a teach-and-repeat like scenario during the first guided pass through the environment. Features are matched to the ones from the map in order to localise the robot in the environment.

The teach-and-repeat methods are in some sense very similar to the SLAM with only one big difference. There is no need for metric precision maps and globally consistent environment representations. Therefore there is usually no loop closure checking or recovery from a kidnapped robot problem. During the first guided pass the robot builds so called a visual memory of the route and it is then capable of reproducing the trajectory. Several methods use bearing-only navigation which corrects only one degree of camera freedom, which corresponds to the heading of the mobile robot. There are several examples of these systems in the literature.

SURFNav algorithm [29] presents a method where the robot relies on the relative localisation provided by odometry, which would integrate unbounded position error over time, but uses visual landmarks to correct its heading and suppress the odometric error. Only a histogram of horizontal displacements of corresponding pairs of previously mapped and currently perceived visual features is used to correct the heading of the mobile robot. More thorough description of the method is in the section 3.1.3.

Paper [69] uses omni-directional vision and feature correspondences in a local control law that enables a robot to reach any position on the plane by exploiting the bearings of at least three landmarks of unknown position. Attractive and repulsive virtual forces derived from the displacement of feature correspondences are used to control the robot movement.

Another examples of teach-and-repeat methods are in articles [31],[30] and [70] utilising simple left-right turn algorithm and series of overlapping feature submaps respectively.

2.2.4 Neural networks and deep-learning

Although neural networks and deep-learning techniques were mentioned in the previous section under the class of natural features and adaptive algorithms, they were given their own section. This is because the neural networks can be used to extract different types of features depending on their training [71] and in several applications they behave like a complete black box [72], [7] and therefore it can not be positively answered on which characteristics of the environment the neural network reacts.

The systems using neural networks consists of individual nodes(neurons) connected in layers. These layers are an input layer then several hidden layers and an output layer. Every node has several inputs and it characterised by its activation function as it is depicted in the Figure 11. There are known procedures how to teach the neural network using both supervised and unsupervised learning methods.

In the field of mobile robotics neural networks are utilised in reactive based navigation systems, localisation methods and obstacle avoidance in dynamical environment.

The Carnegie Mellon University Navigation Laboratory presented several visual based navigation methods for assisted and autonomous driving based on neural networks. Well known example is the ALVINN experiment [72] which featured 5 layer neural network capable of road following after several minutes of learning from human driver. The neural network in ALVINN has an input layer consisting of 30×32 nodes, 5-node deep hidden layer and 30 nodes of output layer representing each of the possible steering angles that the vehicle must follow in order to keep travelling in the middle of the road.

Another example of mobile robot navigation using neural networks is NEURO-NAV [57] which uses topological map for indoor navigation. The NEURO-NAV consists

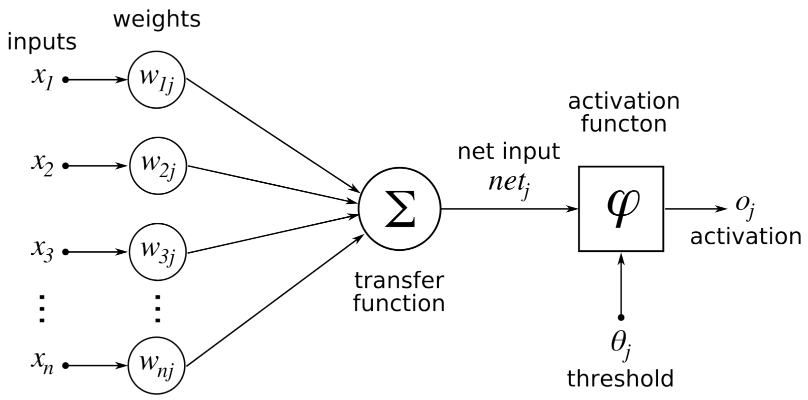


Figure 11: Neural network node.

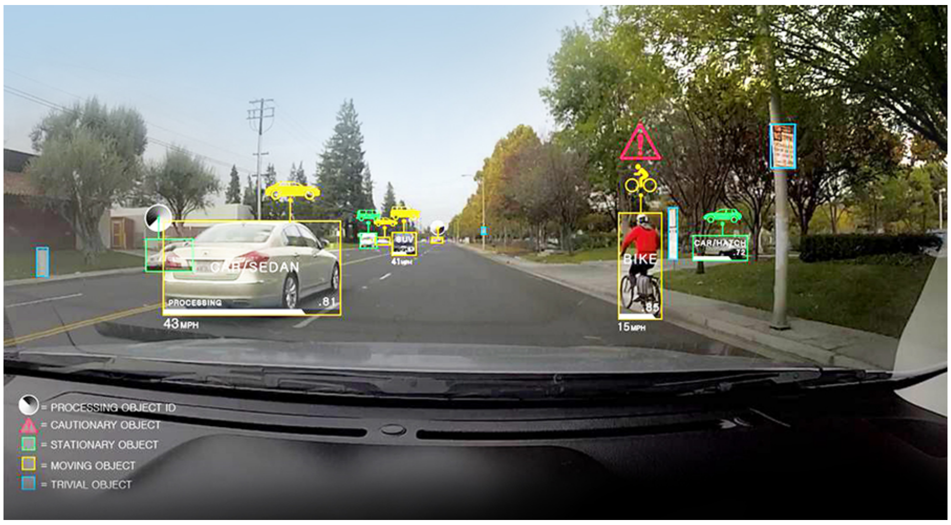


Figure 12: Deep learning based object classification. Courtesy of [7].

of two modules. One for corridor following based on the same principle as ALVINN and one for landmark detection and self-localisation in the map.

Lately deep learning methods with 15-node deep hidden layer were introduced and are incorporated in the mobile robot navigation methods. For example, automotive industry uses the neural networks for segmentation, obstacle detection and road-sign recognition [7]. An example of object classification in a traffic scene using deep neural networks is depicted in the Figure 12.

Part 3
Architecture

This thesis builds on the results of a three year long research in the field of the FPGAs, computer vision and mobile robotics. The basic idea of the project was to design and develop a custom designed module capable of real-time visual mobile robot navigation specifically suitable for small mobile robot platforms restricted in power consumption and lifting capacity. This project was very specific by porting an application on a hardware with limited resources. However it was not an intention to build a single purpose hardware rather to present and develop a hardware and software architectures with respect to further scalability and versatility. This is also the main reason why the FPGA platform was chosen for the implementation.

Although nowadays computers are powerful enough to perform complex processing of visual data together with the navigation of the mobile robot, their ability to operate in real-time goes hand in hand with power consumption. Moreover, in the field of computer vision, algorithms are often accelerated using the GPUs which are either too heavy to be carried by small robotic platforms or consume too much power for the long term operation on batteries. Therefore the FPGA architecture was chosen as a potentially advantageous platform for such task. The FPGA architecture allows the programmer to build a digital circuit architecture specifically suitable for the given task. It also offers true parallelism and it proved to be an efficient platform for computer vision applications. Hence the custom designed embedded module based on the FPGA platform was developed during the work on this thesis. Moreover this chapter generalizes a framework for building FPGA architectures specifically suitable for accelerating computer vision applications by introducing a set of basic building blocks and unifying their mutual interfaces.

This chapter provides the reader with a comprehensive description of the developed embedded module together with the hardware and the software implementations. The first part discusses requirements on the module followed by the description of the overall system setup together with justification of particular design choices. The second part describes briefly the hardware of the embedded module and the third part describes the most important part of the work, the architecture and implementation of the visual navigation system on the embedded module together with the proposed framework for building computer vision applications on FPGA platforms.

3.1 System setup

The basic idea of the project was to develop a lightweight embedded module capable of mobile robot navigation which could be used on a tightly constrained robotic platforms like μUAVs yet provide sufficient computation power for various navigation

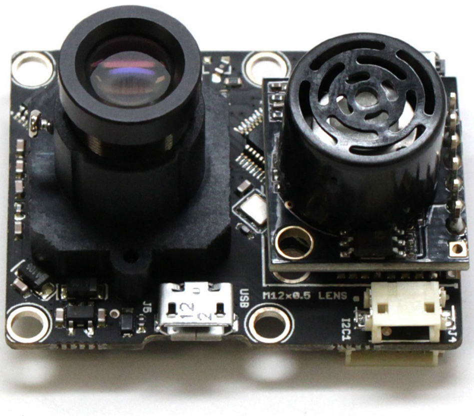


Figure 13: PX4 Flow sensor. Courtesy of [8].

tasks. Regarding such systems there are only a few implementations recognised in the literature which appeared recently. Well established example of a specialised hardware for mobile robotics is a PX4 Flow optical flow sensor [8]. It is a small microprocessor based camera sensor which is able to provide an optical-flow-based visual odometry. It is also equipped with ultrasonic range sensor which measures the distance to the scene and scales the optical flow values to metric values and a gyroscope for angular rate compensation. Figure 13 shows the sensor with mounted lens and ultrasonic range sensor.

Although this sensor computes the optical flow at 250 frames per second it has to be noted that only in a discrete raster of 64 points on a subsampled image with resolution of 64×64 pixels without any lens distortion correction. It also does not compute angular velocity but only translational velocities. PX4 Flow power consumption is only 0.6 Watts.

Another example of vision based navigation implemented on μUAV is a semi-direct monocular visual odometry presented by Christian Forster, et al. [65] which uses tracking of FAST feature points and map building for visual odometry estimation. They presented the evaluation of the method on the Odroid-U2 ARM-9 Quadcore microcomputer on which their method achieves frame rate of 55 frames per second on images with the resolution of 752×480 pixels tracking approximately 120 features at a time. However the power consumption of the board was over 10 Watts.

Recently introduced embedded module [73] based on FPGA and mobile CPU calculates dense disparity images with the resolution of 752×480 pixels at 60 frames per second. The disparity estimation rely on the semi global matching approach which is computationally a very intensive task. All of this while maintaining power consumption under 5 Watts.

Perhaps the most relevant implementation was presented in the thesis [17] by Jan Šváb and the following paper [18] which presents an embedded module capable of image processing suitable for mobile robotics. The module was capable of SURF feature

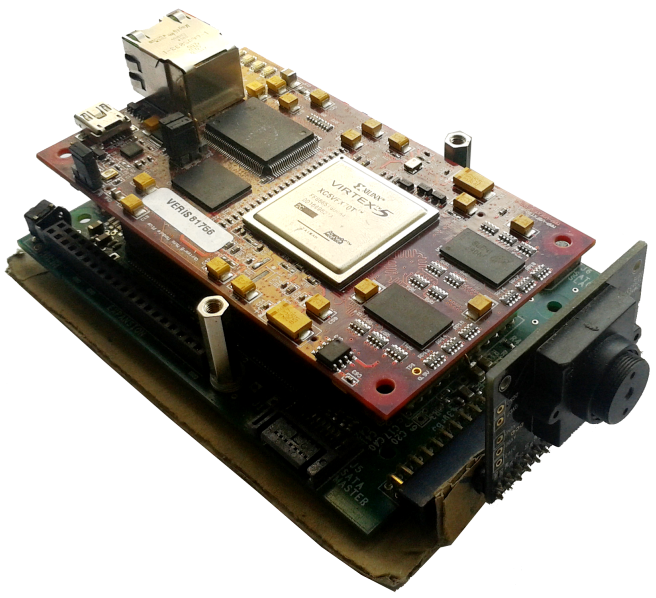


Figure 14: The SCHVAB minimodule.

extraction from images with the resolution of 1024 × 768 pixels at frame rates up to 20 frames per second. However it was not able to perform the whole navigation. The module is depicted in the Figure 14. The module power consumption was 6 Watts while utilising 95% of inner FPGA resources.

These examples show the weaknesses of CPU based implementations of computer vision methods on constrained hardware. The complexity of the algorithm together with the requirement on real-time computation goes hand in hand with power consumption. Therefore the FPGA architecture was chosen as a key component of the hardware implementation. However development of a printed circuit board utilizing as advanced technology as FPGA is a very complicated task by itself. When considering this in combination with the price of a single FPGA chip (when buying only one IC) it is better to utilise a complete development board and extend its functionality by custom designed hardware. Therefore the module is built on the DE0 Nano development board [74] by Terasic Technologies. This board was chosen because it is small, cheap and features circuitry usable in mobile robotics. The full specification of the module is described further in this chapter in section 3.2.

The FPGA platform has proved to be very helpful in accelerating computer vision methods or process simultaneously data from sensors, however it is not so well suitable for general programming. Moreover it is not beneficial to try to accelerate everything in the FPGA fabric. General computation is better performed by ordinary processor architecture. Regardless of the fact that the development for FPGAs is much more time consuming. Methodology proposed in the thesis [14] and the article [75] supports the decision to utilise both FPGA fabric and ordinary CPU in processor-centric embedded systems.

Although modern FPGAs are designed as a system on a chip with already integrated hardcore processors they are so far too expensive. Another possibility is to utilise a softcore processor which is implemented in the FPGA fabric. This solution is, however, not optimal because it occupies a lot of fabric and is constrained in maximal clock frequency due to the electrical characteristics of the FPGA chip. Therefore the module was extended by an external auxiliary microprocessor board connected directly to the FPGA fabric. The board features an STM ARM Cortex microprocessor clocked at 160MHz with a variety of interface options.

Last in the hardware design of the module was the camera part. Selection of a right camera sensor shown to be a difficult problem. In the end the camera sensor MT9V034 from Aptina Imaging was utilised. It is a CMOS camera sensor with full resolution of 752×480 pixels and frame rate of 60 frames per second. It is the same sensor which is utilised in the PX4 Flow module [8] and the dense stereo module [73] described earlier in this section.

Hardware of the module together with the main features of the individual boards is described further in this chapter in the section 3.2.

The problem of visual based mobile robot navigation was thoroughly discussed in the previous chapter. For the implementation on the module a feature based visual teach and repeat navigation algorithm was chosen. A modified SURFNav navigation algorithm [29] was chosen for the implementation due to its relative simplicity and the fact that its complexity doesn't grow with the size of the environment. Therefore it is suitable even for a large-scale environments. From the computer vision methods the SURFNav rely on the utilisation of image feature extraction. Thesis [1] and article [10] evaluated several feature extractors and discussed their influence on the quality of the SURFNav navigation. From the comparison the best performing feature extractors were those using BRIEF descriptor [60]. Therefore FAST feature detector and BRIEF descriptor were chosen with respect to the implementation in the FPGA architecture to be utilised.

The rest of this section describes thoroughly individual parts of the implementation together with the justification of their suitability for implementation in the module.

3.1.1 Features from accelerated segment test (FAST)

Features from accelerated segment test (FAST) is a feature detector proposed by Edward Rosten and Tom Drummond in the paper [9]. It is a corner detector optimised with respect to the computation speed. The main idea behind the method is to examine the image for small patches that resemble a corner. Since derivatives of the image are not computed and therefore no smoothing of the image is necessary the calculation is very fast. The algorithm uses a set of comparisons between the center pixel p and pixels defined on the 7×7 neighbourhood by a bresenham circle as it is shown in the Figure 15.

The feature is detected if there are n contiguous pixels in the circle which are all brighter than the intensity of the candidate pixel I_p plus a threshold t, or all darker than

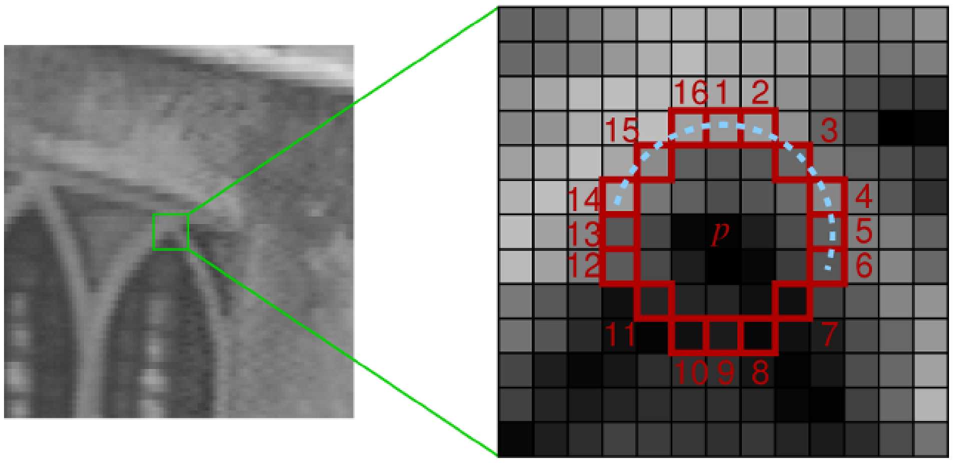


Figure 15: FAST feature detection. Courtesy of [9].

I_p minus t. The value of n is in the original implementation defined as $n = 12$ because in that case a rapid rejection method can be used which only checks comparisons in four compass directions. If p is a corner then at least three of these comparisons must meet the above condition. However the paper approved that the best repeatability exhibits the setting of $n = 9$.

The problem of non-maximal suppression is in the original paper addressed by three possible methods.

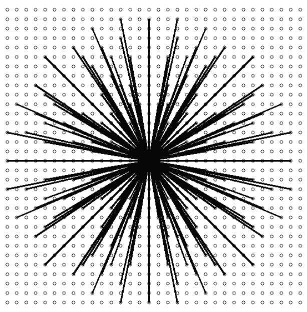
1. The maximum value of n for which p is still a corner.
2. The maximum value of t for which p is still a corner.
3. The sum of the absolute difference between the pixels in the contiguous arc and the central pixel.

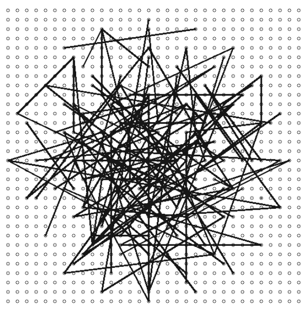
In the original algorithm a slightly modified version of the option 3 is used.

FAST feature detector is well suitable for the implementation on the FPGA. Unlike other feature extractors it does relies on pixel-wise comparisons rather than somewhat heavy calculations. The image patch needed for the detection is also small and therefore not so much resources are needed for the whole architecture. The specialised architecture for the detection of FAST features is described further in this chapter.

3.1.2 BRIEF feature descriptor

Binary robust independent elementary features (BRIEF) is a binary feature descriptor presented first by Michael Calonder and Vincent Lepetit in the article [60]. The BRIEF descriptor uses pairwise intensity comparisons of pixels inside an image patch surrounding the located feature. These comparisons form a set of unique binary tests which are subsequently stored into an n-dimensional bit vector. There are several


(a) Artificial BRIEF descriptor. Courtesy of [60].


(b) Randomised BRIEF descriptor. Courtesy of [60].

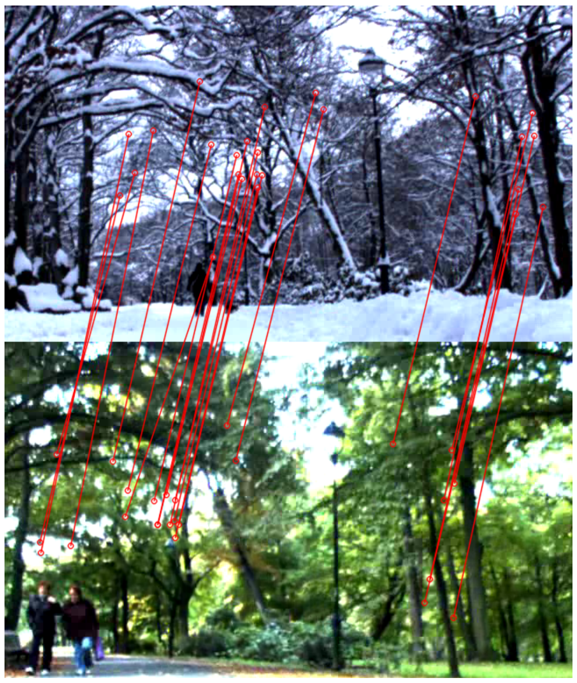

(c) GRIEF descriptor tentative matches. Courtesy of [56].

Figure 16: Descriptor examples.

methods for choosing the pairwise comparison locations. In the original paper artificial and randomised methods for distribution of comparisons are studied from which the randomised are recognised as better. Later contribution presents a reinforced learning method which evaluates individual comparisons and their contributions to the resulting distinctiveness of the descriptor on a given dataset and by means of genetic evolution builds a comparison sequence which is optimised for the given environment. The enhanced version of the descriptor is called GRIEF [56] (Genetic BRIEF). Moreover article [10] proved BRIEF features outperforms other more complex feature extractor in repeatability in a long-term navigation scenario. Figures 16a, 16b shows an example of artificial and random BRIEF descriptor comparisons and the Figure 16c shows tentative matches of the GRIEF feature across seasonal changes.

The descriptor similarity is evaluated using the Hamming distance which states in how many bits the two vectors differ. The Hamming distance computation is performed in two steps. In the first step the two descriptors are xor-ed which results in a binary vector with ones in positions in which the two descriptors differ. Second step is counting the ones in the resulting vector which can be very fast on machines utilising the `popcnt` instruction otherwise it is necessary to iterate the whole vector through.

For the purposes of this thesis GRIEF descriptor was chosen because of its low computation complexity and good performance in the long-term, large-scale navigation scenarios. The main implementation problem of feature descriptors in the FPGA fabric is that they are formed from a relatively big neighbourhood of the center point typically with a minimum size of 25×25 pixels which requires memory resources of the FPGA. Generally it is not possible to do the description on-line without massive utilisation of the FPGA fabric. Moreover it is not desirable because with utilisation grows the power consumption as well. Typically, there are also not so much features detected in proportion to the total amount of pixels in the image. Therefore an architecture was developed which buffers the incoming pixel data and when the feature is detected it assembles the descriptor.

3.1.3 SURFNav navigation algorithm

SURFNav [29], [76], [1] is a teach-and-repeat navigation algorithm which utilises visual features for bearing-only navigation. It uses odometry as a main source of localisation information and uses image features for heading correction only. Article [29] proved that the heading correction is sufficient for keeping the odometric error bounded even in long-term navigation scenarios. 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. The algorithm consists of two phases. The teach phase, when the mobile robot is guided along straight line segments through the environment manually, and the repeat phase when the robot repeats the pre-learned trajectory as follows.

At each segment start the robot is turned to the desired direction according to compass value or odometry. 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 similar to the visual compass principle. However, the features do not lie in infinity, which causes the 'visual compass' to point not just along the navigated trajectory, but to intersect the intended trajectory at an average distance of the visual features. This bias allows to compensate the lateral displacement of the robot and causes the method's stability. The horizontal displacement between observed and expected features are used to create a navigation histogram with fixed number of bins. The maximum peak in the histogram determines the correction of heading of the mobile robot as it is shown in the Figure 17. The end of the segment is recognized according to traveled distance, which is measured only by odometry.

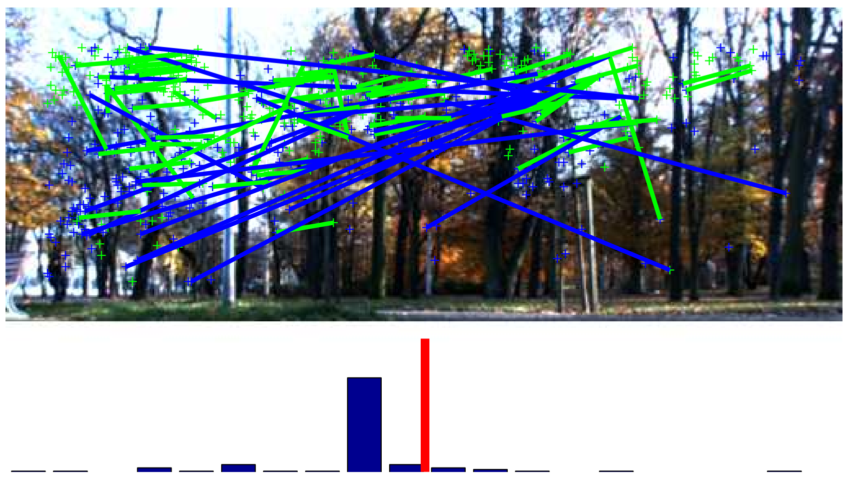


Figure 17: Matched features and the corresponding navigation histogram.

The SURFNav navigation algorithm was chosen because it is computationally very efficient and well suitable even for long-term navigation [77]. The algorithm requirements are made only on the size of memory for map storage. Therefore the module is equipped with a microSD card slot for large, cheap and easily accessible nonvolatile memory.

3.2 Hardware design

Custom designed embedded module specifically suitable for computer vision applications and robotics was developed during the work on this thesis. The module comprises of three independent boards which are stacked together. These boards are the DE0 Nano development board by Terasic Technologies featuring the FPGA chip, ARMBoard module featuring the STM ARM Cortex microprocessor and a camera module board which features camera sensor and acts as an interface board between the DE0 Nano and ARMBoard. A schematic diagram of the module key features is depicted in the Figure 18. Figure 19 shows the individual boards of the module.

Direct connectivity options of the module are USB PHY and CAN bus provided by the ARMBoard, UART and SPI interfaces and expansion connectors. Moreover the ARMBoard provides the means for connecting the Ethernet interface through the dedicated shield board. A specialised dedicated VGA output board which can be connected directly to the FPGA was also developed during the work on the project.

DE0 Nano board

The DE0 Nano board (further only DE0) is a low-end FPGA development board

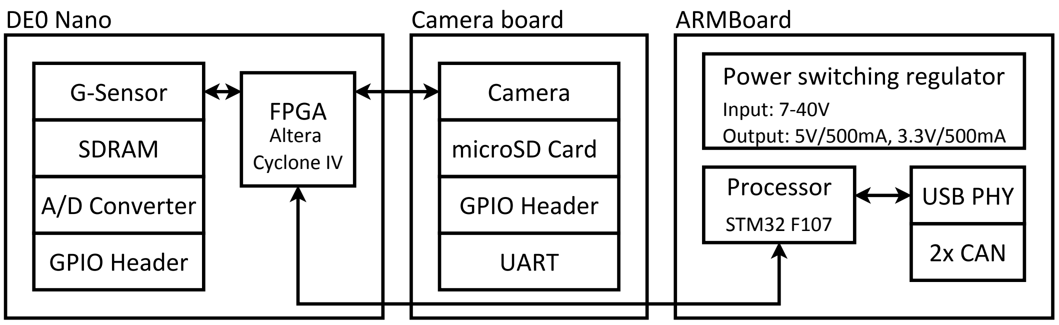
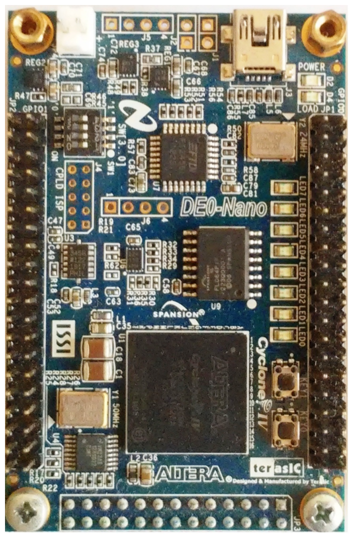
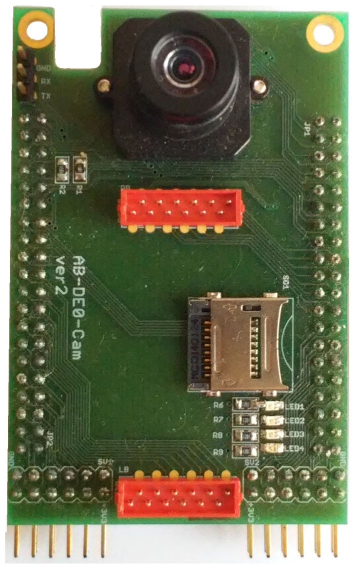


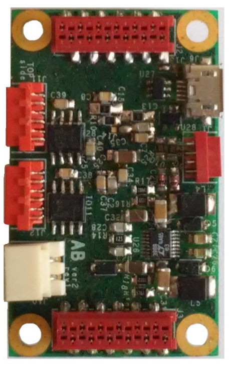
Figure 18: Overview of the module key components.



(a) DE0 Nano dev. board.



(b) Camera board.



(c) ARMBoard.

Figure 19: Individual boards of the embedded module.

with only few resources but very good connectivity options. It features Altera Cyclone IV FPGA with 22320 Logic Elements and 594 Kbits of embedded memory and 66 embedded multipliers. Utilised FPGA has the fastest speed grade in the device family allowing maximum clock of around 350 MHz. Table 1 lists all the features of the DE0 board.

FPGA chip	Altera Cyclone IV	EP4CE22F17C6N
Memory devices	SDRAM	32 MB
	I2C EEPROM	2Kb
GPIO	LED	8
	Pushbuttons	2
	DIP-switch	4
Expansion	2x40 pin header	72 user I/O
	1x26 pin header	13 user I/O
Other	3-axis accelerometer	ADI ADXL345
	A/D converter	NS ADC128S022, 8channel
Power	USB mini AB	5V
	2-pin header	3.6 - 5.7V

Table 1: Features of the DE0 Nano development board.

The advantages of the DE0 board include its size and weight as well as low power consumption. Good connectivity options together with presence of a 3-axis accelerometer on board are also beneficial for mobile robotics applications. Last but not least the programming of the module is performed through the on-board programmer therefore no additional programming modules are necessary for the reconfiguration.

ARMBoard

ARMBoard is an open-source modular electronic system for robotic applications. It is based on an STM32F107 ARM Cortex MCU clocked at 160 MHz with USB PHY, galvanically isolated CAN bus, on-board step-down converter and four extension connectors. It is also used as a power supply to the whole module. The on-board step-down converter operates on a wide range of supply voltages (7-40V) providing stable 5V and 3.3V power outputs. The ARMBoard provides means for further extending the module by Ethernet or direct motor control dedicated shield boards.

Camera board

The camera board features two 40 pin connectors for connection with the DE0, two 12 pin micromatch connectors for connection with the ARMBoard, two 12 pin expansion connectors, microSD card slot and the camera sensor with needed circuitry and lens mount. The camera sensor is a MT9V034 from Aptina Imaging. It is a 1/3 Inch CMOS camera sensor with the full resolution of 752×480 pixels, supply voltage 3.3 V and LVDS functionality. With the maximum master clock of 27 MHz it can output 60 frames per second in a full resolution. The frame rate can raise up to 240 fps when row and column binning are enabled. In that case the resolution drops down to 188 × 120 pixels. For the purposes of robotics it is very important that the camera features global shutter which is essential for computer vision applications dealing with dynamic scenes captured by rapidly moving cameras.

All the connections with the DE0 board through the 40 pin connectors are designed to reduce the complexity of the HDL design and reduce possible timing hazards in communications by ending all associated signals in the same Bank of the FPGA chip. These spatial relations are necessary for designing high speed communication interfaces.

The camera board can be used solely with the DE0 board. Both the camera sensor and microSD card slot are connected and powered from the DE0 board.

VGA debug board

An auxiliary module designed specifically to be used with the module extends the module debugging options by the VGA output. The possibility of visually verifying the results of the processing is essential for development of machine vision algorithms. Therefore a very simple debugging board was designed which allows connecting any VGA monitor directly to the device.

3.3 FPGA configuration

This project was very specific by porting a computationally intensive application on a hardware with limited resources. However it was not an intention to build a single-purpose hardware. Rather, it intends to present and develop an FPGA architecture with respect to scalability and versatility. Therefore this section presents a set of building blocks for developing a specialised but easily modifiable and portable architecture for computer vision applications. In addition, it discusses the implementation details of the proposed and developed architecture.

In order to provide a clear description of the proposed solution the path of the visual data through the architecture is followed in this section. The architecture is divided into several core building blocks which structure is depicted in the Figure 20. The framework defines the overall behaviour and interfaces of individual modules.

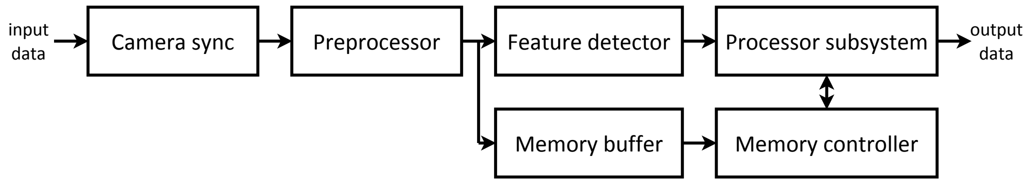


Figure 20: Overall FPGA architecture.

For the visual data the **Camera sync** core is the gate to the system. Its purpose is to provide the rest of the processing pipeline with the continuous stream of visual data together with several supporting signals. This core also provides a clock signal for the feature detection chain.

Next in the pipeline is the **Preprocessing core** which is used to perform operations like color conversion, histogram equalisation or rectification of the incoming data. It is not necessary in the architecture and in the proposed architecture of the module it is not implemented.

In the scheme of feature extraction there are two stages: feature detection and feature description. Feature detection is usually more suitable for the implementation in the FPGA fabric because it can benefit from the true parallelism and it is usually calculated from a smaller area of the image. On the contrary, the descriptor calculation is usually based on a larger area around the detected interest point. It also has to be noted that all of the points in the image need to be checked for presence of the feature however only a handful of them are recognised as features and are selected for feature description. Therefore it is better to perform feature description after the detection of the feature point, not concurrently and on-line. Moreover the on-line calculation of the descriptor from a vast area around the feature point is extremely wasteful in terms of FPGA fabric resources. Therefore the framework presents a way how to deal with the feature description by using a dedicated **Processor subsystem**.

After the preprocessing of the image, the path of the visual data splits between feature detection and preparation for feature description. The **Feature detector** takes the stream of the visual data on its input and provides coordinates of individual feature points on its output optionally with some additional information about the features. The coordinates of individual features are then provided to the **Processor subsystem** for the descriptor calculation and further processing. Three highly optimised pipelined architectures for feature detection were introduced during the work on this project. However this thesis describes only the FAST feature detector in detail but generalises a framework for efficient implementation of other feature detectors on FPGA architecture.

The second path of the visual data goes through the **Memory buffer** to the available memory storage. This storage needs to be large enough to hold at least a part of the image. On small FPGAs such storage is not available in the FPGA fabric. Therefore it is advantageous to use an external memory like SRAM or SDRAM which is usually present on FPGA development boards. This memory is then shared between the **Memory buffer** and the **Processor subsystem**. In conclusion the **Processor subsystem** has information about the features as well as an access to all the image data which is then required for the feature description. Moreover the **Processor subsystem** may be used for other tasks like navigation where certain operations may benefit from additional acceleration in the FPGA fabric. In certain cases the **Processor subsystem** can be substituted for a deterministic state machine which performs only reads and writes on predetermined locations. Last but not least, the **Processor subsystem** may run at its maximum clock frequency achieving the best complete system performance possible.

The implementation on the module uses simple **Camera sync** core for the camera data acquisition, FAST feature detector and NIOS II softcore **Processor subsystem**. 32MB SDRAM memory on the DE0 board is utilised as a main external memory for buffering of image data.

Now every part of the architecture will be discussed in more detail. For every core its usage in framework is described followed by the implementation details for the proposed embedded module.

3.3.1 Camera sync

The **Camera sync** core is the first core in the feature detection and description pipeline. It forms an input gateway for the visual data. Its input may be any video signal source or memory device. Its output is a stream of pixel data together with the pixel clock and informations about the coordinates of the individual pixels. Mandatory ports for the **Camera sync** core are listed in the Table 2. The framework defines only output ports of the **Camera sync** core because it is upon the user and implementation to provide the video signal source. This source can be either input from the camera or other digital video source, data stream from memory. Additionally, the core can provide artificial images that serve as test patterns for the processing pipeline.

Name	**Type**	**Size**	**Function**
PIXEL_CLK	Output	bit	Master clock signal. All signals are valid on the rising edge of this clock.
PIXEL_DATA	Output	vector	Parallel pixel data output.
FRAME_X_CNT	Output	vector	Parallel signal. Contains x coordinate of the right outputted pixel in the current frame.
FRAME_Y_CNT	Output	vector	Parallel signal. Contains y coordinate of the right outputted pixel in the current frame.
FRAME_BLANK	Output	bit	Binary signal. When high indicates blanking area of the image.
FRAME_SYNC	Output	bit	Binary signal. On rising edge indicates end of the current frame.

Table 2: **Camera sync** core entity ports.

The output data stream is defined to be a standard digital video signal with continuous clock and horizontal and vertical blanking areas. It is also essential that each image line has a precisely defined number of pixels. These requirements greatly reduce the complexity of the rest of the architecture, especially the feature detection. Timing diagram of the **Camera sync** core is depicted in the Figure 21, showing the readout of one image.

Implementation on the module uses **Camera sync** core to synchronise with the incoming data from the MT9V034 CMOS camera sensor on the Camera board. The sensor requires only power supply and 26.66 MHz master clock generated by the FPGA

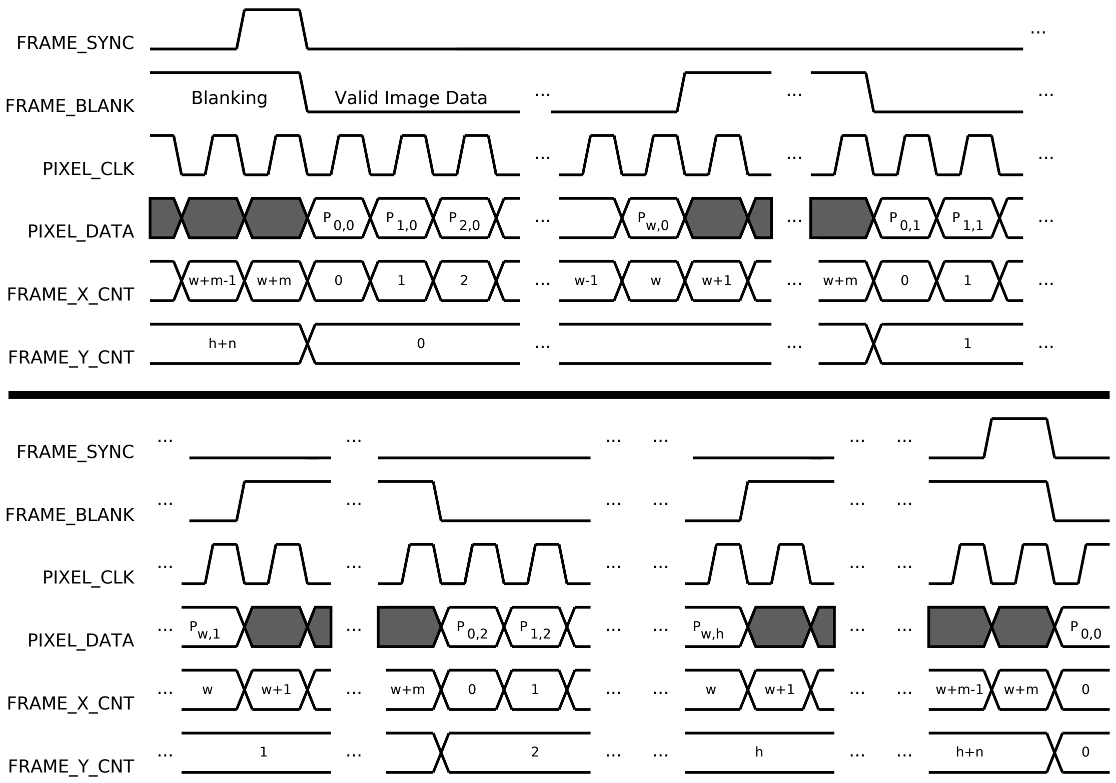


Figure 21: **Camera sync** core output timing diagram - one image readout. [w,h] are numbers of active image column and row pixels respectively, [m,n] are numbers of horizontal and vertical blanking pixels.

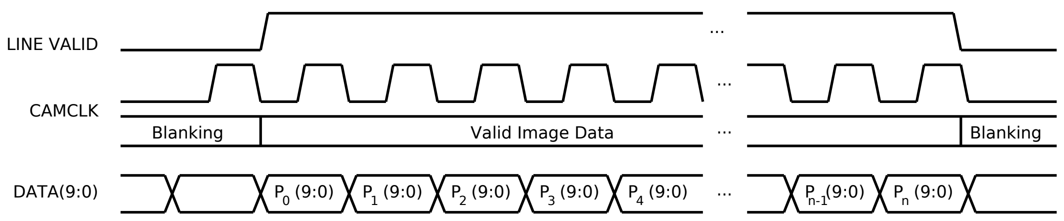

LINE VALID
CAMCLK
Blanking
Valid Image Data
Blanking
DATA(9:0)
P0 (9:0) P1 (9:0) P2 (9:0) P3 (9:0) P4 (9:0) Pn-1(9:0) Pn (9:0)

Figure 22: Camera sensor data timing. Readout of one image line.

in order to operate in a native mode. In the native mode the sensor outputs 752×480 WVGA grayscale images at the frame rate of 60fps. To change the behaviour of the sensor, internal registers need to be written with appropriate configuration through the I^{2}C-like serial interface. Therefore the **Camera sync** core utilises a simple I2C_INIT core which initialises the camera registers with appropriate setup values on startup.

The camera sensor outputs the data according to the timing depicted in the Figure 22 which shows signals asserted during the readout of one line of the image. The pixel data are synchronized with the rising edge of CAMCLK output. The LINE_VALID and FRAME_VALID(not depicted in the timing diagram) signals are high during the display time and low during the horizontal and vertical blanking time respectively. The **Camera sync** core samples the pixel data input and based on CAMCLK, FRAME_VALID and LINE_VALID signals provide the rest of the architecture with signals listed in the Table 2.

3.3.2 Preprocessor

The **Preprocessor** core is a functional block defined by the framework which is used to pre-process the raw image data. It can be used to perform various operations like color conversion, histogram equalisation, lens distortion correction, rectification or integral image calculation but it does not have to be implemented.

The **Preprocessor** core may change almost anything about the input image data which is for example the case of rectification(see Hartley an Zisserman for details [63]) however the requirement on precise horizontal timing of output signal has to be met. The outputted signal has to cope with the timing depicted in the Figure 21.

Table 3 lists the required input and output ports of the **Preprocessor** core.

In the architecture of the module the preprocessing core is not utilised because it is not necessary. However the preprocessing step is a usual component of computer vision applications and therefore the core is a part of the framework. For example an architecture of the SCHVAB minimodule [18], which is closely related with this thesis, uses integral image calculation and image sub-sampling as the preprocessing steps in the feature detection pipeline.

3.3.3 Feature detector

The **Feature detector** core takes the stream of the visual data on its input and provides coordinates of individual feature points on its output. The main idea behind feature detection in FPGA is to use the true parallelism of FPGA fabric together with pipelining in order to design an architecture which is able to process the stream of visual data on-line. Table 4 lists all the necessary ports of the **Feature detector** core defined by the developed framework.

Feature detection in the FPGA can be done very efficiently by using the true parallelism and pipelining. A typical feature detector consists of one or more convolution

Name	Type	Size	Function
PIXEL_CLK	Input	bit	Master clock signal from the **Camera sync** core. All signals are valid on the rising edge of this clock.
PIXEL_DATA_IN	Input	vector	Parallel pixel data input from **Camera sync** core.
FRAME_X_CNT_IN	Input	vector	Parallel signal. Contains x coordinate of the right outputted pixel in the current frame.
FRAME_Y_CNT_IN	Input	vector	Parallel signal. Contains y coordinate of the right outputted pixel in the current frame.
FRAME_BLANK_IN	Input	bit	Binary signal. When high indicates blanking area of the image.
FRAME_SYNC_IN	Input	bit	Binary signal. On rising edge indicates end of the current frame.
PIXEL_DATA_OUT	Output	vector	Parallel pixel data output.
FRAME_X_CNT_OUT	Output	vector	Parallel signal. Contains x coordinate of the right outputted pixel.
FRAME_Y_CNT_OUT	Output	vector	Parallel signal. Contains y coordinate of the right outputted pixel.
FRAME_BLANK_OUT	Output	bit	Binary signal. When high indicates blanking area of the image.
FRAME_SYNC_OUT	Output	bit	Binary signal. On rising edge indicates end of the current frame.

Table 3: **Preprocessor** core entity ports.

Name	Type	Size	Function
PIXEL_CLK	Input	bit	Master clock signal from the **Camera sync** core. All signals are valid on the rising edge of this clock.
PIXEL_DATA_IN	Input	vector	Parallel pixel data input.
FRAME_X_CNT_IN	Input	vector	Parallel signal. Contains x coordinate of the right outputted pixel in the current frame.
FRAME_Y_CNT_IN	Input	vector	Parallel signal. Contains y coordinate of the right outputted pixel in the current frame.
FRAME_BLANK_IN	Input	bit	Binary signal. When high indicates blanking area of the image.
FRAME_SYNC_IN	Input	bit	Binary signal. On rising edge indicates end of the current frame.
PROCESSOR_BUS	I/O	multiple	A **Processor subsystem** memory mapped slave bus.

Table 4: **Feature detector** core entity ports.

kernels which are used to calculate the response function in every pixel. Typical examples are Difference of Gaussians kernels of SIFT feature detector or their approximated

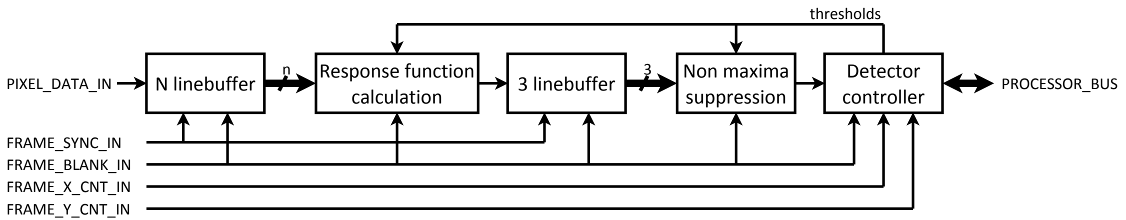


Figure 23: Architecture of the **Feature detector**.

counterparts in SURF feature detector. A smaller 3×3 convolution kernel is used for example in Sobel edge detection filter. FAST feature detector does not use directly the operation of convolution but it has also a kernel-like structure for image feature detection.

In order for the detector to operate in real time on the streamed video data it is necessary and desirable that with each rising edge of PIXEL_CLK a feature response function is calculated for one pixel position. Hence with each PIXEL_CLK the output of the core indicates whether there is or there is not a feature at the given x and y image coordinates. Figure 23 proposes an internal architecture of the feature detection processing chain. The chain consists of five cores which are pipelined in order to allow the maximum data throughput. All the cores take stream of the data on their inputs and provide a stream of the data on their outputs.

N linebuffer

The **Linebuffer** core is the first in the feature processing pipeline. It is used to provide a simultaneous access to n vertically aligned pixels with every rising edge of the PIXEL_CLK by using the internal block RAM resources of the FPGA for temporary storage of the visual data.

The architecture of the **Linebuffer** core is depicted in the Figure 24. It uses first in first out (FIFO) memory resources of the FPGA fabric together with a simple logic defining memory writes and reads. Simple state machine mechanism triggered by the FRAME_BLANK_IN signal defines a number of stored pixels in each FIFO memory which shall be exactly one line of image data. For this part it is extremely important the preciseness of the blanking signal timing provided by the **Camera sync** core. The FRAME_SYNC_IN signal is used as an internal reset for the FIFOs and the buffering logic. It also has to be noted that for a simultaneous access to n vertically aligned pixels it is necessary to use only $n-1$ FIFOs.

Concerning the portability between individual FPGAs it is sufficient to swap the FIFO component of the architecture with the one appropriate for the given FPGA vendor.

The FAST feature detector implementation on the module uses the core to buffer 7 vertically-aligned pixels which are necessary for the FAST response function calculation.

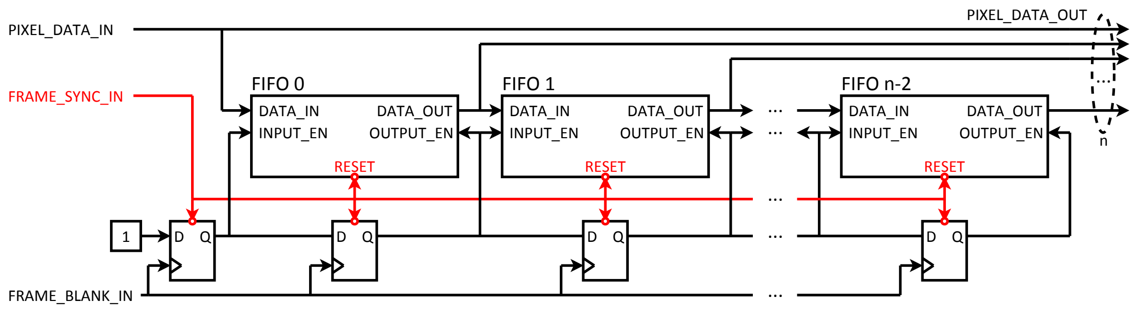


Figure 24: Architecture of the **Linebuffer** core.

Response function calculation

As has been said the response function usually consists of convolutions and some additional operations. The **Response function calculation** core takes a stream of n vertically aligned pixels on its input and provides a value of response at the given point on its output. This point is usually a center of the convolution window which is continuously sliding over the input image as the pixels are streamed from the **Linebuffer** core.

The development of the **Response function calculation** core is the most challenging part of the whole design because two considerations need to be taken into account. First consideration regards the FPGA fabric utilisation and the second one is the metastability of the architecture.

In order to explain the utilisation problem it is necessary to look on the convolution from the implementation perspective. The main problem lies in the size of the convolution window. In conventional processor architectures the data are stored in the memory and the processor pulls the data on demand. Generally it can not lose any data during the process. However, the proposed FPGA implementation intends to process the data on-the-fly as they arrive from the camera. It is like in a hard real-time systems. When the data are not processed in a certain amount of time they are lost. Therefore it is necessary to develop such architecture which would utilise all the data necessary for the **Response function calculation** as they are coming through the architecture because the data are present and valid for just a limited time. The natural way of solving this problem is to use the true parallelism of the FPGA and implement the whole convolution kernel in the FPGA fabric. This method is illustrated in the Figure 25 which shows the convolution of image with the Sobel edge filter. The filter is placed over the image and the resulting response function is calculated according to the equation prescribed on the right side of the Figure. This equation can be directly wired into the FPGA fabric thanks to the true parallelism. Concerning a really simple implementation for 3×3 convolution window with any integer number coefficients it is sufficient to utilise only 9 registers, 9 multipliers and 8 adders for the convolution and resources for the **Linebuffer** core. However with growing size of the convolution window the number of necessary resources increases faster than quadratically.

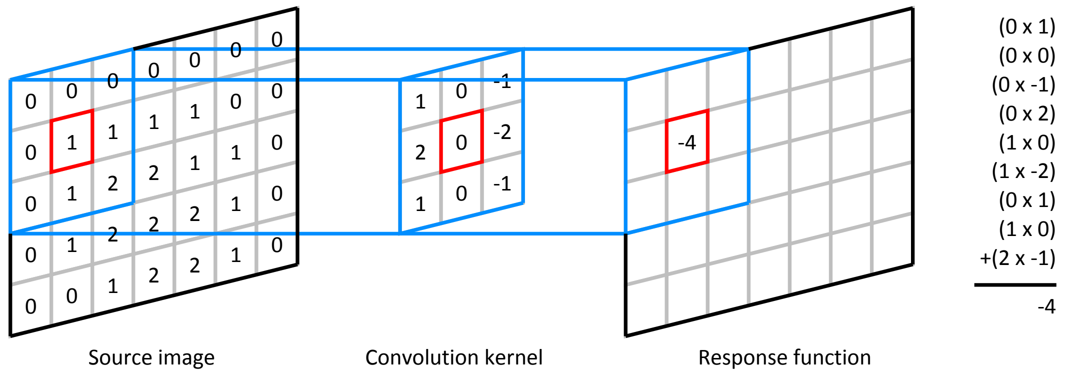


Figure 25: Convolution architecture example.

This problem can be addressed by either separable convolution or subsampling. Both can save a lot of internal FPGA resources. Separable convolution in FPGA architectures uses pipelining for its operation. It is performed by first doing the vertical convolution resulting in one value which is then fed on the input of an m element deep shift register where m is the width of the separable kernel. The data are shifted with the PIXEL_CLK resulting in m horizontally aligned values of vertical convolution stored in the shift register. The shift register then acts directly as an input for horizontal convolution. It is important to note that in any time, m response function calculations are concurrently carried out but are in a different processing state resulting in calculation of only one response function with each PIXEL_CLK. The separable convolution can be also performed conversely with first doing horizontal convolution and then buffering m lines of data, however it is preferable to perform first the vertical convolution directly on image data because the result of convolution has usually higher bit width then the visual data which results in utilisation of more memory resources.

Addressing the metastability of the architecture usually means a higher FPGA utilisation but results in a fast and scalable architecture. Metastability is an inevitable process appearing in the digital circuits whenever a result of particular operation is sampled before it stabilises. As a result, the circuit can act in unpredictable ways. This situation is more probable to happen when particular operation composes of a large amount of elementary steps and the sampling frequency is high. The large amount of operations is reflected by a large number of subsequent gates which a signal has to travel through which means that the signal requires a longer time to stabilise on the correct value. Recalling the previously mentioned example of 3×3 convolution window with any integer number coefficients it is necessary for the calculation of response function to perform 9 multiplications and 8 additions. The whole calculation can be directly wired in the FPGA fabric but it would take a lot of time before the result would settle on a correct value. Solution to this problem are either parallelism or pipelining.

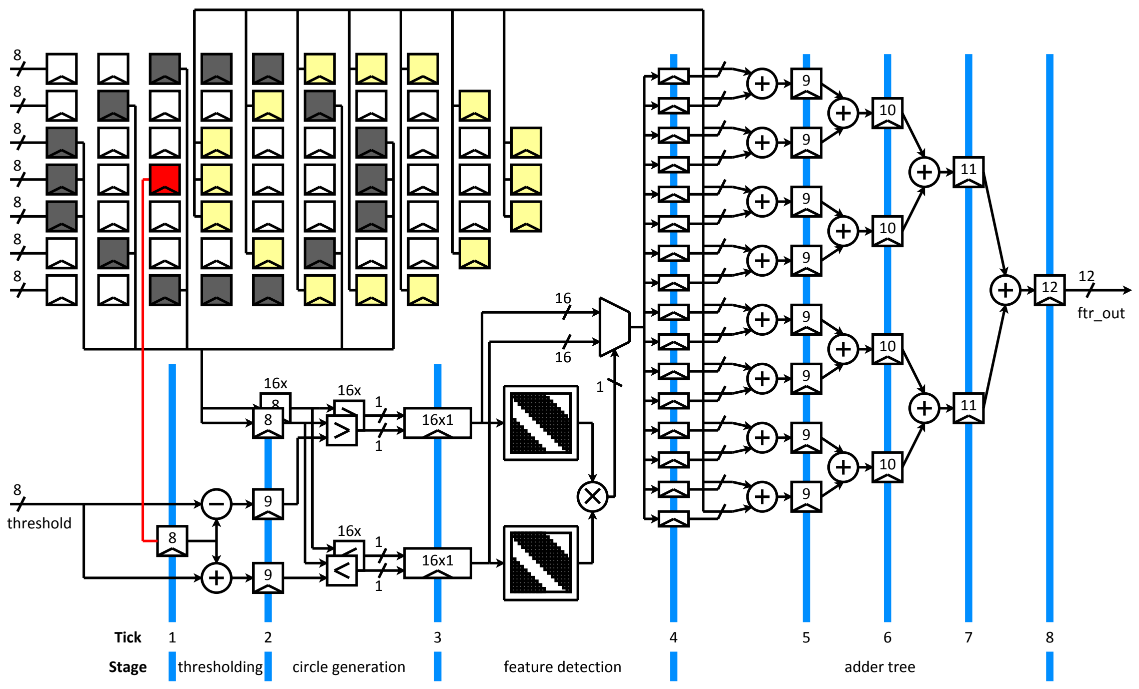


Figure 26: FPGA architecture of the FAST corner detector.

Parallelism means utilising more similar processing units in order to provide each unit with the necessary time for the asynchronous calculation of the result which is then sampled. Multiplexing between individual processing units is necessary and ensures the correct results.

Pipelining means dividing the calculation to a sequence of elementary operations from which each calculation can be performed in an incremental amount of time. Between the individual calculation steps, the results of previous ones are sampled with the PIXEL_CLK. The pipelining thus increases throughput of the architecture at the cost of latency. With increasing frequency of PIXEL_CLK the incremental calculations steps need to be more and more simple ensuring the protection from metastability. It is also important to note that in any time, multiple response functions are concurrently calculated but are in a different processing state resulting in calculation of only one response function with each PIXEL_CLK.

The proposed FAST feature detector architecture is depicted in the Figure 26. It uses 56 bit-wide 10 element-deep shift register for accessing all the data necessary for response function calculation. The actual response function calculation architecture is pipelined into 8 stages. The pipeline is optimised with respect to the processing speed and resource utilisation. The response function calculation is split between elementary operations which ensures metastability protection even when using maximum possible clock for a given FPGA architecture.

The architecture copes with the description of the FAST feature detector provided in section 3.1.1. The response function calculation consists of following steps.

1. Calculation of central pixel comparison values $I_p + t$ and $I_p - t$, Where I_p is the intensity value of the central pixel p and t is the threshold value.
2. Comparison of the pixels lying on the Bresenham circle around the central pixel according to the equation

$$P_{p\to x} = \begin{cases} 1 & I_{p\to x} > I_p + t \\ 0 & I_{p\to x} < I_p + t \end{cases}, \tag{4}$$

$$N_{p\to x} = \begin{cases} 1 & I_{p\to x} < I_p - t \\ 0 & I_{p\to x} > I_p - t \end{cases}, \tag{5}$$

   resulting in two 16 bit vectors describing the Bresenham circle with respect to the center pixel.
3. If there are n contiguous '1' in either P or N the feature is detected, the corresponding vector is used for selecting pixels for response function calculation.
4. Response function calculation as a sum of pixel values selected by '1' in a corresponding vector.

These steps are further decomposed between elementary operations resulting in a final 8 stage pipeline depicted in the Figure 26. Each stage of the pipeline is executed during the duration of one PIXEL_CLK.

1. Sampling of the central pixel value I_p. The value is sampled one step ahead of the rest of the Bresenham circle pixels in order to calculate the thresholded values in advance.
2. Sampling of the pixels on the Bresenham circle concurrently calculating I_p+t and $I_p - t$ values.
3. Comparison of $I_p + t$ and $I_p - t$ values with the pixels on Bresenham circle resulting in two 16 bit vectors describing the Bresenham circle with respect to the center pixel.
4. If there are n contiguous '1' in one of the vectors the feature is detected, the Bresenham circle vector is used for selecting pixels for response function calculation. Because the pixel values are continuously shifting in the register with each PIXEL_CLK, it is necessary to sample the values for response function calculation from 3 pixels horizontally displaced locations.
5. 6. 7. 8. Adder tree for computation of the response function.

3 linebuffer

The **3 linebuffer** core is used to provide a simultaneous access to the 3 vertically aligned feature response values. It functions exactly the same way as the **N linebuffer** core described earlier in this section. The only difference may be in the used Block RAM resources of the FPGA. The feature response values usually have a wider range compared to the visual data provided on the input of the **Feature detector**. Therefore it is usually necessary to utilise more Block RAM resources because of the stored data size.

The implementation on the module uses 6 M9K memories. For each line two memories are necessary because the width of feature response values is 12 bits.

Non-maxima suppression

Non-maxima suppression core is used for locating the local extrema amongst feature response values. The core consists of three element deep 24bit wide shift registr, which buffers incoming feature response values provided by the **3 linebuffer** core. The content of the shift register forms 3×3 matrix of feature response values in which the center value is compared with its 8 surroundings and an optional threshold value provided by the **Detector controller** core. If the central value is the local maxima the core output is asserted high otherwise it is asserted low. Optionally the **Non-maxima suppression** core can output additional information about the feature like feature response value.

Detector controller

The purpose of Detector controller core is to provide the **Processor subsystem** with coordinates of detected features as well as configure the behaviour of the **Feature detector**. It takes the informations about located features from **Non-maxima suppression** core on the input and merge it with the current FRAME_X_CNT_IN and FRAME_Y_CNT_IN coordinates in order to construct the feature entry. Due to the buffering and pipelining of the data the actual coordinates of the feature differ from the coordinates that are provided by FRAME_X_CNT_IN and FRAME_Y_CNT_IN signals at the moment when the **Non-maxima suppression** core signalises detected feature. However the delay induced by the processing is deterministic, therefore the core updates the x and y coordinates of the detected feature and provides them to the **Processor subsystem**. It can also directly filter out features which can not be described due to their position too close to the border of the image.

The core also functions as a controller of the detector architecture. It sets various thresholds that are either provided by the **Processor subsystem** or calculated from the number of detected features. Typically the core outputs a threshold value to **Non-maxima suppression** core in order to limit the overall number of detected features.

The interface to the **Processor subsystem** is implementation dependant. A well-tested recommendation is to incorporate the core partially to the **Processor subsystem** as a memory mapped slave device. When the feature is detected the core interrupts the processor and the processor pulls the feature data from the core for further processing. On the other side the **Processor subsystem** can write internal registers of the **Detector controller** core in order to change the behaviour of the **Feature detector**.

The implementation on the module uses the aforementioned system of **Processor subsystem** interfacing. The core also features a set of user accessible registers which are listed in the Table 5. The **Processor subsystem** may write these registers and thus change the behaviour of the feature detector.

Address	Bit	Default in Hex(Dec)	R/W	Description
0x00	0	0	W	Setting this bit to 1 enables interrupt generation by this core.
0x01	9:0	0x0000(0)	R	X coordinate of the feature.
0x02	9:0	0x0000(0)	R	Y coordinate of the feature.
0x03	7:0	0x10(16)	W	Minimum number of detected features per image.
0x04	7:0	0x40(64)	W	Maximum number of detected features per image.
0x05	9:0	0x19(25)	W	Left border for feature detection.
0x06	9:0	0x2D7(727)	W	Right border for feature detection.
0x07	9:0	0x19(25)	W	Top border for feature detection.
0x08	9:0	0x1C7(455)	W	Bottom border for feature detection.

Table 5: **Detector controller** core user accessible registers.

3.3.4 Memory buffer

The **Memory buffer** core is used in the description chain. Its purpose is to store image data to a predefined location in the external memory. It also acts as a clock crossing bridge for visual data between slower feature detection chain and faster **Processor subsystem**. The **Memory buffer** core takes the video stream on its input and provides the data in a format suitable for storage on its output. The interfacing ports are defined only for the visual data input as they are listed in the Table 6. The memory output is implementation dependant and it is greatly influenced by the memory controller core and **Processor subsystem**.

The core samples the visual data and store them to the memory shared with the **Processor subsystem**. It is recommended to utilise a small memory buffer for temporary visual data storage before the data are stored in the big external memory. Direct memory access or processor interrupts may be used to avoid collisions between processor and the **Memory buffer** core.

The implementation of the **Memory buffer** core in the module uses direct memory access through the Altera avalon bridge to store the visual data directly to the SDRAM memory on the DE0 board. The incoming data are first sampled on rising edge of the faster processor clock and then stored into the buffer FIFO memory. One M9K embedded block RAM is utilised as a buffer for incoming visual data. When the memory is almost full, the core interrupts the embedded NIOS processor and then starts directly

Name	Type	Size	Function
PIXEL_CLK	Input	bit	Master clock signal from the **Camera sync** core. All signals are valid on the rising edge of this clock.
PIXEL_DATA_IN	Input	vector	Parallel pixel data input from **Camera sync** core.
FRAME_X_CNT_IN	Input	vector	Parallel signal. Optional. Contains x coordinate of the right outputted pixel in the current frame.
FRAME_Y_CNT_IN	Input	vector	Parallel signal. Optional. Contains y coordinate of the right outputted pixel in the current frame.
FRAME_BLANK_IN	Input	bit	Binary signal. When high indicates blanking area of the image.
FRAME_SYNC_IN	Input	bit	Binary signal. On rising edge indicates end of the current frame.

Table 6: **Memory buffer** core entity ports.

writing to the SDRAM memory. Writes are performed on predetermined locations derived from the width and height of the image. Due to the internal buffering it is not possible to use FRAME_X_CNT_IN and FRAME_Y_CNT_IN to address calculation but in different FPGAs it might be possible to directly store the data in a dual port RAM. In the proposed architecture with every successful write operation an address counter is incremented. Both the address counter and FIFO buffer resets on the rising edge of the FRAME_SYNC_IN signal.

3.3.5 Processor subsystem

The **Processor subsystem** is primarily used for the feature description. It takes x and y coordinates of the features on its input and for each feature constructs a descriptor. Implementation details of the **Processor subsystem** are platform specific and therefore only guidelines for implementation are provided in this section.

First of all the **Processor subsystem** utilises a memory controller which is shared with the **Memory buffer** core. Therefore it is necessary to somehow avoid collisions during the memory access although the **Memory buffer** performs only write operations and **Processor subsystem** performs only read operations within the part of the memory where the image data are stored. There are two possible solutions to the collision avoidance problem. First one is to utilise a direct memory access and the second one is to interrupt the processor during the writing to the memory.

Second consideration concerns the overall performance of the **Processor subsystem**. The used processor can be either softcore or hardcore depending on the FPGA type and vendor. Hardcore processors are usually clocked at much higher frequencies then the FPGA fabric and therefore their processing time is not so expensive. On the contrary, the softcore processors which are implemented in the FPGA fabric can not achieve such high performance and therefore every performed instruction counts.

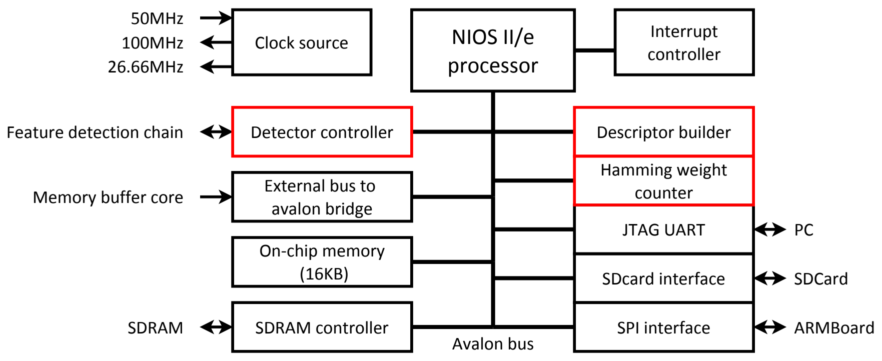


Figure 27: **Processor subsystem** overview. Red cores are custom designed.

By softcore processors it is therefore beneficial to use the processor only to move the data inside the architecture. In other words it is beneficial to use as much direct reads and direct writes as possible. The hard calculations shall be performed in the FPGA dedicated cores.

The implementation of the **Processor subsystem** on the module uses NIOS II/e embedded processor. Figure 27 shows embedded peripherals of the **Processor subsystem** and their connections. Custom designed cores are depicted with red border. Now every utilised core will be briefly described.

NIOS II/e processor
: It is a small lightweight softcore processor provided by the Altera. It features 32bit RISC architecture and it occupy only 600 LEs. In the proposed architecture it runs on 100MHz clock.

Clock source
: Clock source component is used for generating clock signals for embedded processor, SDRAM memory and input clock for the Camera sensor. Its input is 50MHz clock from the onboard crystal and its outputs are 100MHz clock for the **Processor subsystem**, 100MHz skewed clock for the SDRAM memory and 26.66MHz output for Camera sensor.

Detector controller
: **Detector controller** was already described previously in this chapter. It forms an interface between the Feature detector and the **Processor subsystem**.

External bus to avalon bridge
: Simple core which allows direct memory access from outside the **Processor subsystem**. It is connected directly to the **Memory buffer** core.

On-chip memory
: An on-chip 16KB memory was utilised in block RAM resources of the module. It contains the whole software for the **Processor subsystem**.

SDRAM controller

SDRAM controller is used for interfacing the 32MB SDRAM memory on the DE0 board. It is the only core which is platform dependant (concerning Altera devices).

Descriptor builder

Custom designed core which is used for 64bit BRIEF/GRIEF descriptor construction. It takes pixel values on its input and provides a descriptor on its output performing all the necessary comparisons. The Descriptor builder core acts as a memory mapped slave with 128 different addresses. The individual addresses directly indicate the positions of the comparisons that constitute the feature descriptor. The data are provided to the core in tuples in order to save the internal resources of the FPGA. The first bit of the address is used for distinguish between the first and the second value in the tuple. Bits 7:2 of the address indicates the position of the comparison in the BRIEF descriptor vector. For each tuple the first pixel value is stored in the buffer and then compared with the second pixel value.

Experimental results as well as code disassembly shows that this core provides 40% speed-up in assembling of the BRIEF descriptor compared to the implementation in C code which assigns each bit of the vector with the result of a comparison. When disassembling the code the solution utilising the CPU translates to 10 individual machine instructions compared to the solution utilising the Descriptor builder core which translates to 6 machine instructions.

Hamming weight counter

Custom designed core for Hamming weight computation. The core takes a 32 bit long vector on its input, count the number of '1' appearing in the vector and provides that number on the output. The latency of the calculation is only one clock cycle. The Hamming weight counter acts as a memory mapped slave with two accessible registers listed in the Table 7. The core is used for feature matching

Address	Bit	Default in Hex(Dec)	R/W	Description
0x00	31:0	0x0000(0)	W	Input register for the 32bit vector.
0x01	7:0	0x00(0)	R	Output register. Contains the number of ones in the register 0x00.

Table 7: Hamming weight counter core user accessible registers.

of BRIEF descriptors. First of all two BRIEF descriptors are xored and then the hamming weight of the result is calculated using this core.

On certain processor architectures the hamming weight count operation is known as a popcount (population count) instruction. However only a few embedded grade microprocessors features this instruction. Experimental results as well as code disassembly shows that this core provides 98% speed up in the feature matching of two BRIEF descriptors.

JTAG UART
Altera's programming and debugging interface for NIOS II processor architectures. The core acts as a stdio interface.

SDcard interface
Core for interfacing the MicroSD cards on the Camera board. The protocol of communication is an SPI protocol capable of communication at the speed of 25Mbits/s. The core acts as a memory mapped slave with several user accessible registers. The core is a part of the Altera University program.

The SD card is meant to store a topological-landmark map that the robot would use for feature-based navigation.

SPI interface
Simple SPI communication core used for data transmission between DE0 board and ARMBoard. It is capable of communication at the speed of 25Mbits/s.

3.3.5.1 Software description

During the work on this thesis it was supposed that the implementation on the FPGA will work only as an image feature extractor and that the ARMBoard will carry out the tasks of the mobile robot navigation. However after finishing the implementation of the feature extraction on the FPGA it has shown that the NIOS embedded processor has still enough performance reserves and therefore it was decided to try to implement the whole navigation algorithm in the FPGA part of the module. Therefore the ARMBoard was not utilised during the work on this thesis and still offers opportunities for further enhancing the whole architecture. Thus, the core of the SURFNav navigation algorithm, the navigation histogram calculation, was successfully implemented in the FPGA fabric.

The main purpose of the software running on the NIOS II processor is to fetch the coordinates of the individual features from the feature detector chain and for each feature construct the BRIEF descriptor from image data stored in the memory. After that the feature is matched to the set of features stored in the memory. The matching uses a 'ratio test' with a ratio threshold 0.875. If there is a positive match the horizontal displacement of the features is added to the histogram. The features are fetched from the **Detector controller** core to the buffer memory on interrupt. The main loop of the program consistently checks if the buffer is empty. If there are features in the buffer they are immediately processed. The set of the features which is used for comparison is so far implemented using a keyframe method. Every 20 seconds the module makes a snapshot and stores all the features as the default set. All the features from the consecutive frames are matched to this set of points. This simulates the computationally most demanding part of the SURFNav algorithm which is the construction of the heading correction histogram from feature correspondences. In each loop the maximum value of the histogram is found and its position in the histogram is outputted. This position would be used for a direct steering of the mobile robot.

The software is also capable of accessing the data on SDcard which is so far used for debugging purposes and storing of images from the module. However it is intended to be used as a storage for the SURFNav landmark map.

Concerning the implementation details almost all the operations are performed only by direct memory reads and writes. There are three interrupt sources in the design. First interrupt source is a FRAME_SYNC_IN signal which announces the end of the current frame. The second interrupt source is the **Memory buffer** core which interrupts the processor whenever it wants to write the visual data to the shared memory. During the memory write, the processor waits in a busy loop. The last source of interrupts is the **Detector controller** core which announces that there are new keypoints for processing. The descriptor calculation is performed by fetching x and y coordinates of the feature translating them to the shared memory address offset. This address is then added with the offsets of individual GRIEF comparisons and using the direct reads from the memory and the direct writes to the Descriptor builder core the GRIEF descriptor is constructed. The descriptor is then immediately matched to the set of the stored features from the keyframe utilising the Hamming weight counter core.

3.4 Implementation costs

The module uses Altera Cyclone IV FPGA chip which belongs to the low-end category of FPGAs. Nevertheless the whole implementation occupy when placed and routed only 28% of total logic elements and 36% of total embedded memory resources providing a plenty of area for further development of the architecture. Table 8 lists the utilisation by individual entities. The utilisation is before optimisation and fitting into the FPGA fabric so the resulting architecture occupy even less resources.

The price of the module comprises from the price of the DE0 Nano development board and manufacturing price of the Camera board and ARMBoard. The DE0 Nano board costs 79$ when purchased directly from Terasic Technologies, but they have also an accademic price of 61$. The price of the Camera board comprises mostly from the MT9V034 camera sensor which costs 29$. The rest of the electrical components together with manufacturing of one PCB costs approximately 15$. The construction costs of the ARMBoard are around 57$ but the ARMBoard is not necessary for the functioning of the module. So the total price for the module is 123$(105$ with academic licensed DE0 board) without the ARMBoard and 180$(162$) with ARMBoard.

Entity	**LC Combinationals**	**LC Registers**	**Memory Bits**
Camera sink	105	81	0
Feature detector	1111	974	65664
7 linebuffer	204	132	49152
Corner detector	608	517	128
3 linebuffer	65	46	16384
detector controller	132	160	0
Memory buffer	90	98	8192
Processor subsystem	4222	2704	146688
NIOS II cpu	940	501	10240
Descriptor builde	139	95	0
Popcount	24	32	0
SDcard interface	1148	893	4096
SDRAM controller	279	247	0
On-chip RAM	1	0	131072
Other	1691	936	1280

LC = Logic Cell

Table 8: Utilisation of the FPGA by individual entities.

Part 4
Results and experiments

The purpose of the performed experiments is to evaluate the overall module performance and test its correctness and applicability in a mobile robot navigation scenario. The module parameters are evaluated including spatial demands, power consumption, repeatability and distinctiveness of implemented feature extraction algorithm as well as the overall usability for long-term large-scale visual navigation of mobile robots.

First part of this chapter evaluates the performance of the feature extraction part of the design in terms of its speed and repeatability. The second part evaluates the most demanding part of the SURFNav algorithm which is the construction of the heading correction histogram from feature correspondences and subsequent histogram voting for generation of the robot steering command. The third part evaluates the performance of the module in a long term navigation task and the third part compares the proposed solution with other embedded modules used for the mobile robot navigation.

4.1 Feature extractor evaluation

Evaluation of the feature extractor examined the performance and correctness of the proposed feature detection and description implementation. The main purpose of the evaluation is to compare the performance of proposed FPGA implementations of FAST feature detection and GRIEF description algorithms to their original, CPU-based counterparts in terms of processing speed and the ability to produce same results. In order to assure correctness of the results, the tests were performed both on the real-world images captured by a mobile robot as well as on the artificially generated data.

4.1.1 FAST feature detector evaluation

The proposed architecture of the FAST feature detector was tested on the processing speed and the correctness of the implementation both on artificially generated data and camera images.

The artificially generated data consisted of 8 static and dynamic test patterns generated by the **Camera sync** core which were used to test the implemented FAST detector against its CPU counterpart. An example of two static test patterns used in the evaluation is in the Figure 28.

The outputted coordinates of FAST features were compared to those provided by the CPU implementation. First of all the patterns were processed on the CPU and coordinates of detected FAST feature points together with corresponding thresholds

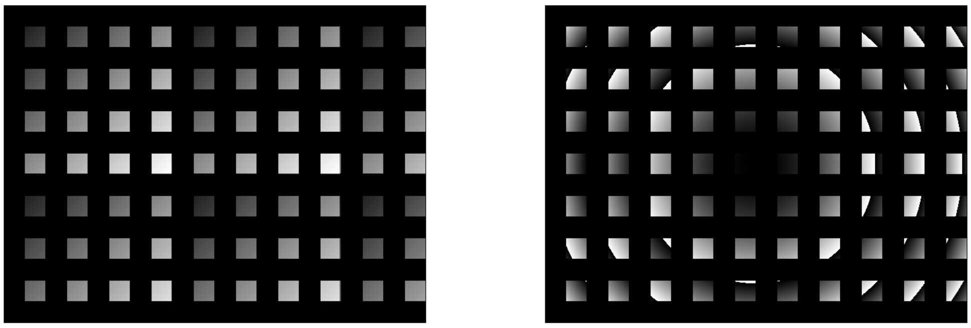


Figure 28: Example of artificiall test patterns.

were stored. These points were then compared directly by the embedded processor on the module with those extracted by the implemented FAST feature detector. The results matched in 99.6% of all cases. The rest 0.4% were points with outlying (out-of-image) coordinates. These points emerge during the data exchange between the **Processor subsystem** and the **Detector controller** core which did not implement a buffer for features in the time of testing and the features did not pass the clock crossing bridge correctly. However these outlying features can be easily removed from the calculation.

The artificially generated data were also used in the test of processing speed. Since the method allows to process the images on-the-fly, it does not make much sense to compare its speed to the CPU-based implementations in traditional ways that evaluate the algorithms speed in terms of the time needed to process one image. Thus, to compare the feature detector to standard, CPU-based implementation it is better to use the notion of latency. While the latency is crucial for fast and accurate vision based mobile robot navigation. The overall latency of the implemented FAST feature detector is deterministic and it is exactly 4 image rows and 15 pixel clocks. Concerning the implementation on the module it is only 128μs.

The processing speed test focused on the verification of metastability tolerance of the presented pipelined architecture. A specialised self-evaluating test pattern was generated because the utilised embedded processor would not be able to authenticate the results at such high frequencies. The architecture was able to correctly operate at the PIXEL_CLK frequency of 200MHz which is enough for use with almost any camera sensor. For example a Full HD 1920×1080 signal with frame rate of 60 Hz has a pixel clock of 182MHz.

4.1.2 GRIEF feature descriptor evaluation

The GRIEF feature description evaluation intended to test the speed of description process. The description process was tested on real camera images with variable threshold for detected number of features. During the test, the software of the module performed only the 64bit GRIEF description. This scenario simulated the usage of the FPGA part of the module only as a smart camera processing the image and providing only the extracted visual features on its output.

During the test the thresholds for the FAST feature detector were gradually increased up to the limit of 255 features per image. While utilising the **Feature description** core, the module was able to provide the description of the 255 detected features during the duration of the currently processed frame.

The individual descriptors generated by the FPGA implementation were compared to the descriptors calculated by the CPU version of the GRIEF. In all cases, the descriptors of the FPGA and CPU implementations were identical.

4.2 Navigation algorithm evaluation

In order to extensively test the suitability of the module for mobile robot navigation, the most demanding part of the SURFNav algorithm which is the construction of the heading correction histogram from feature correspondences and subsequent histogram voting for steering command generation was evaluated. The whole navigation stack required for visual-based teach-and-repeat mobile robot navigation was utilised in this test. The stack consists of detection of FAST feature points in the image, description of their surroundings using GRIEF descriptor, establishing their correspondence with a known map and finally calculation of the robot's steering commands based on the histogram voting.

At the beginning of the test a snapshot of the local features in the environment was created and stored in the memory of the module. The module was then displaced from its original position and based on the steering command provided by the histogram voting was navigated back to its initial position.

Two testing scenarios were examined. First scenario simulated a typical SURFNav navigation problem when the mobile robot deviated from the original path due to the erroneous odometry. In the scenario the camera was pointing in the direction of the mobile robot movement and the induced lateral error was compensated based on the steering command provided by the histogram voting. The algorithm was able to provide the correct heading estimation in cases when the induced displacement allows for tracking off well distinguishable features.

Second scenario simulated the usage of the module for the position stabilisation of a μUAV robotic platform. In this case, the module's camera faced downwards and the histograms were used to calculate and correct quad-rotor displacement in horizontal plane. This scenario however did not simulated the angular motions of the platform, because there is no angular motion compensation mechanism implemented in the module and the generated descriptor is not rotation invariant. In this scenario the module was connected directly to the computer screen which displayed the position informations based on the current displacement of the module with respect to the starting position. In this scenario rapid movements were tested in order to verify the module overall latency.

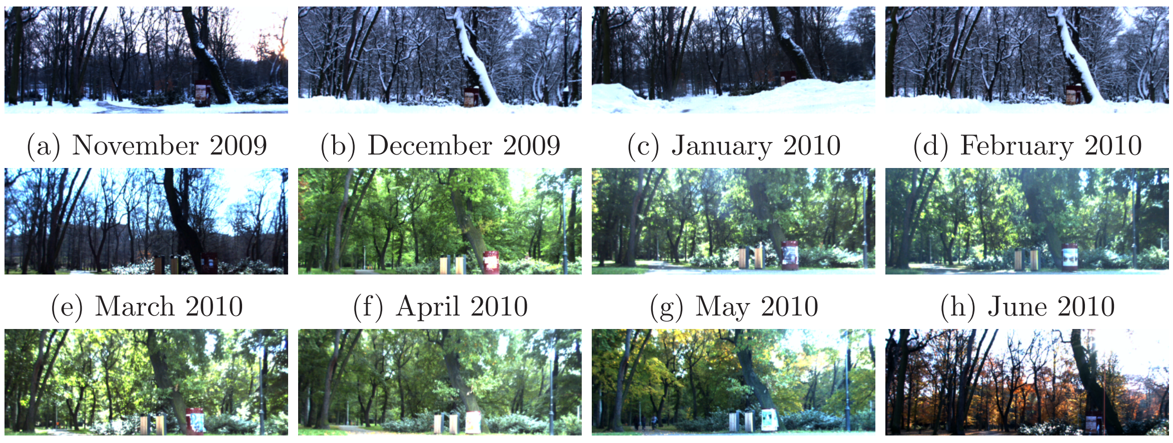

(a) November 2009 (b) December 2009 (c) January 2010 (d) February 2010
(e) March 2010 (f) April 2010 (g) May 2010 (h) June 2010
(i) July 2010 (j) August 2010 (k) September 2010 (l) October 2010

Figure 29: The dataset capturing the seasonal changes. Location II. Courtesy of [10].

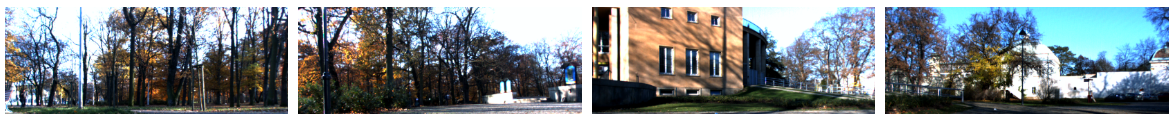

(a) Location I. (b) Location III. (c) Location IV. (d) Location V.

Figure 30: The dataset locations. Courtesy of [10].

4.3 Long-term navigation scenario test

The long-term navigation scenario test measures the performance of the navigation algorithm in a scenario of outdoor vision-based longterm autonomous navigation. The test focuses on the ability of the navigation algorithm to correctly establish heading of the robot relatively to the intended path in substantially changing environment. The evaluation method is based on the article [29] and is described in the article [10]. The histogram voting mechanism is used to estimate the correct heading of the robot.

The dataset used for evaluation contains 12 images per 5 locations covering seasonal changes of the Stromovka forest park in Prague throughout the year. Figure 29 shows the seasonal changes on one of the five locations. Figure 30 shows the rest of the locations. Individual images at given location vary in seasonal changes and slightly in the viewpoint. Altogether 660 comparisons were made, i.e. 12(months)$\times$11(months)$\times$5(locations).

The test was performed directly on the module which observed the individual dataset images on the computer screen. This makes the whole scenario rather difficult because of the image details and contrast loss in such set-up. An estimation of the heading was considered as correct if matched the course of ground truth heading values provided with the dataset. Table 9 lists the matching success rates between individual months. Approximately 180 features per image were used for histogram voting.

The results indicate that the module is capable of successful heading estimation even in a long-term navigation scenario. The reliability of the method decreases when

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Jan	100	80	60	80	60	100	40	100	100	80	100	80
Feb	60	100	100	100	80	80	100	80	100	100	60	80
Mar	80	100	100	80	40	80	100	60	100	100	40	100
Apr	100	80	80	100	80	100	80	80	100	80	100	60
May	80	80	100	80	100	80	80	60	80	80	100	100
Jun	60	60	80	60	80	100	60	80	40	60	80	40
Jul	80	100	100	80	80	100	80	100	80	100	100	60
Aug	80	80	100	100	20	80	80	100	100	80	80	80
Sep	100	80	100	100	80	100	80	100	100	80	100	100
Nov	60	60	60	100	80	60	100	60	80	100	80	60
Oct	80	80	100	100	100	60	80	100	60	80	100	60
Dec	80	100	80	100	80	60	60	80	100	80	80	100

Table 9: Heading estimation success rates.

matching images with and without foliage. Note that due to the ratio-test matching scheme, the Table 9 is not symmetric.

Although the test was influenced in a negative way by the set-up and therefore it used softer criteria for heading estimation, results are comparable with other feature extraction methods. The overall error rates and computational times for 200 extracted features are listed in the Table 10. The proposed method with genetically adapted GRIEF descriptor performed second best after the BRIEF feature descriptor however better than the other feature extractors. The reason why GRIEF performed worse than the BRIEF descriptor is probably due to the test setup and utilisation of different feature detector.

Extractor	DE0-Cam	BRIEF	ruSIFT	uSIFT	ORB	uSURF	STAR	BRISK
Stromovka[%]	17	10	41	44	24	59	44	50
Time[ms/image]	-	5	21	22	24	5	5	12

Table 10: Error rates and computational times for 200 extracted features. Data for other extractors taken with permission from [56].

4.4 Comparison with other embedded platforms

This section compares the module properties with other recognised implementations targeting constrained mobile robotic platforms. The platforms were already briefly described in the beginning of the section 3.1. Table 11 lists the most important properties of individual modules. Comparison of the developed DE0-Cam board with the PX4Flow optical flow sensor [8], SCHVAB minimodule [18], semi-direct visual odometry [65] module based on the Odroid-U2 platform and FPGA based board for dense stereo reconstruction [73] is presented.

	PX4Flow	SCHVAB	Odroid-U2	FPGA 3D	DE0-Cam
CV algorithm	Optical flow	Feature extraction	Feature extraction	Dense stereo	Feature extraction
Nav algorithm	-	-	Visual odometry	Collision avoidance	SURFNav
Dimensions(mm)	$45 \times 35 \times 25$	$120{\times}80{\times}20$	$48 \times 52 \times 15$	$76 \times 46 \times 25$	$75 \times 50 \times 50$
Power consum.(W)	0.6	6	10	5	1.82
Camera resolution	64×64	1024×768	752×480	752×480	752×480
Camera frame rate	250fps	10fps	55fps	60fps	60fps
Price	150$	1175$	89$	$\tilde{5}$00$	180$(123$)
FPGA utilisation	-	95%	-	91%	28%

Table 11: Comparison of modules for mobile robot navigation.

From the results it can be noted that while utilising the same camera hardware as 4 out of 5 proposed modules the presented module exhibits best price-performance ratio. Hence implementing the whole real-time visual based teach-and-repeat navigation algorithm while in contrast to other modules still featuring a lot of performance reserves for further improvements.

Part 5

Conclusion

This thesis presented a complete hardware and software design of an embedded computer vision module based on the FPGA platform capable of real-time mobile robot navigation. Although the main objectives of the work on the project were concerning only one particular hardware and implementation, three years of experience and continuous research in the fields of FPGA prototyping, computer vision and mobile robotics resulted in a proposal of complete methodology and framework for acceleration of computer vision applications on the FPGA platform which became the core component of the implementation part of this thesis. The framework proposed an FPGA architecture comprising of set of basic building blocks with unified interfaces for simplifying the development of computer vision applications on the FPGA platform as well as providing a tool for building high level applications on a tightly constrained hardware.

The developed embedded module is more a case study for verification of the methodology rather than a single purpose hardware. The modules dimensions are $75 \times 50 \times 50$mm and weights only 75 grams. It consumes only 1.82 Watts and it can be operated on a variety of input voltages ranging from 5V USB to 40V. It features both the FPGA and CPU platforms for efficient co-design of embedded applications and provides variety of direct interfacing options ranging USB, CAN bus, UART, microSD card or dedicated VGA output board. While the price of the module does not exceed 180$.

The FPGA architecture of the module was developed with respect to versatility and scalability. The suitability of the solution for the FPGA platform can be best expressed by the performance validation and the device resources utilisation. The method implements all stages required for visual-based teach-and-repeat mobile robot navigation, i.e. detection of FAST feature points in the image, description of their surroundings, establishing their correspondence with a known map and calculation of the robot's steering commands based on the histogram voting. All of this implemented in the FPGA fabric and running at the frame rate of 60 frames per second with a minimum latency. The architecture occupies only 28% of total logic elements and 36% of total embedded memory resources of a low-end FPGA chip which provides plenty of space for further extending the possibilities of the module.

All these implementation results are complemented with thorough verification of the module abilities. Although the module was not yet extensively tested on the actual mobile robot, the steering commands provided by the module were evaluated in several real-world robotic scenarios. These scenarios include tasks of heading estimation, position stabilisation and path following. The overall usefulness for mobile robotics was demonstrated by verification of all parts of the navigation stack as well as experimental verification on challenging outdoor datasets collected by small mobile robot.

In comparison with the most relevant implementations of embedded navigation modules used in mobile robotics the presented solution exhibits best price-performance ratio. Hence implementing the whole real-time visual based teach-and-repeat navigation algorithm while in contrast to other solutions still featuring a lot of performance reserves for further enhancing of the architecture.

Therefore I am convinced that the main goals of this thesis were achieved.

In the future, we will extend the presented methodology to allow automatic synthesis of computer vision systems for FPGA platforms and provide a modular FPGA architecture for mobile robotics capable of sensor fusion and direct mobile robot control. In the near future we plan to add obstacle avoidance capabilities to the navigation algorithm and test it with a micro-robotic platform [78].

References

[1] Hana Szücsová. Computer Vision-based Mobile Robot Navigation. Master's thesis, CTU, Faculty of Electrical Engineering, 2011.

[2] Christoph Vogel, Stefan Roth, and Konrad Schindler. View-Consistent 3D Scene Flow Estimation over Multiple Frames. In *Proceedings of European Conference on Computer Vision. Lecture Notes in, Computer Science*, Berlin, 2014. Springer.

[3] Pablo De Cristóforis, Matías A Nitsche, Tomáš Krajník, and Marta Mejail. Real-time monocular image-based path detection. *Journal of Real-Time Image Processing*, pages 1–14, 2013.

[4] Chunpeng Wei, Qian Ge, S. Chattopadhyay, and E. Lobaton. Robust obstacle segmentation based on topological persistence in outdoor traffic scenes. In *Computational Intelligence in Vehicles and Transportation Systems (CIVTS), 2014 IEEE Symposium on*, pages 92–99, Dec 2014.

[5] Collective of authors. Vicon Industries, Inc., @ONLINE. `http://www.vicon.com/`.

[6] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends® in Computer Graphics and Vision*, 3(3):177–280, 2008.

[7] Collective of authors. NVIDIA, Corp.@ONLINE. `http://www.nvidia.co.uk/object/tegra-automotive-uk.html`.

[8] 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.

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

[10] 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].

[11] Chris Urmson. The self-driving car logs more miles on new wheels. *Google Official Blog*, 2012.

[12] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, 2007.

[13] Nathan Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The GRASP Multiple Micro-UAV Testbed. *Robotics Automation Magazine, IEEE*, 17(3):56–65, 2010.

[14] Sol Pedre. *A new co-design methodology for processor-centric embedded systems in FPGA-based chips*. PhD thesis, University of Buenos Aires, Faculty of Exact and Natural Sciences, Institute of Calculations, 2013.

[15] Collective of authors. Xilinx, inc.@ONLINE. `http://www.xilinx.com`.

[16] Earl Fuller, Michael Caffrey, Anthony Salazar, Carl Carmichael, and Joe Fabula. Radiation testing update, SEU mitigation, and availability analysis of the Virtex FPGA for space reconfigurable computing. In *IEEE Nuclear and Space Radiation Effects Conference*, pages 30–41, 2000.

[17] Jan Šváb. FPGA-based Computer Vision Embedded Module. Master's thesis, CTU, Faculty of Electrical Engineering, 2011.

[18] Tomáš Krajník, Jan Šváb, Sol Pedre, Petr Čížek, and Libor Přeučil. FPGA-based module for SURF extraction. *Machine vision and applications*, 25(3):787–800, 2014.

[19] Bonin-Font, Francisco and Ortiz, Alberto and Oliver, Gabriel. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263–296, 2008.

[20] Guilherme N DeSouza and Avinash C Kak. Vision for mobile robot navigation: A survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(2):237–267, 2002.

[21] Jesus Martinez Gomez, Antonio Fernandez Caballero, Ismael Garcia Varea, L Rodriguez Ruiz, and Cristina Romero Gonzales. A taxonomy of vision systems for ground mobile robots. 2014.

[22] 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.

[23] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[24] 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.

[25] Kai M Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2, 2010.

[26] Nils Bore, Patric Jensfelt, and John Folkesson. Querying 3d data by adjacency graphs. In *International Conference On Computer Vision Systems (ICVS)*, 2015.

[27] 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.

[28] Karel Košnar, Tomáš Krajník, and Libor Přeučil. Visual topological mapping. In *European Robotics Symposium 2008*, pages 333–342. Springer, 2008.

[29] T. Krajník, J. Faigl, M. Vonásek, V. Kulich, K. Košnar, and L. Přeučil. Simple yet Stable Bearing-only Navigation. *J. Field Robot.*, 2010.

[30] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[31] Zhichao Chen and Birchfield, S.T. Qualitative vision-based mobile robot navigation. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2686–2692, 2006.

[32] Davison, Andrew J and Reid, Ian D and Molton, Nicholas D and Stasse, Olivier. MonoSLAM: Real-time single camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[33] Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

[34] Rodney A Brooks. Intelligence without reason. *The artificial life route to artificial intelligence: Building embodied, situated agents*, pages 25–81, 1995.

[35] A. Miranda Neto, A. Correa Victorino, I. Fantoni, and J.V. Ferreira. Real-time estimation of drivable image area based on monocular vision. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 63–68, June 2013.

[36] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart. Mav navigation through indoor corridors using optical flow. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3361–3368, May 2010.

[37] Borenstein, Johann and Everett, HR and Feng, Liqiang and Wehe, David. Mobile robot positioning-sensors and techniques. Technical report, DTIC Document, 1997.

[38] H Haddad, Maher Khatib, Simon Lacroix, and Raja Chatila. Reactive navigation in outdoor environments using potential fields. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1232–1237. IEEE, 1998.

[39] Milan Sonka, Václav Hlaváč, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[40] E Roy Davies. *Computer and machine vision: theory, algorithms, practicalities*. Academic Press, 2012.

[41] Joseph K Kearney, William B Thompson, and Daniel L Boley. Optical flow estimation: An error analysis of gradient-based methods with local optimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):229–244, 1987.

[42] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[43] Antoine Beyeler, Jean-Christophe Zufferey, and Dario Floreano. Vision-based control of near-obstacle flight. *Autonomous robots*, 27(3):201–219, 2009.

[44] Lenka Mudrová, Jan Faigl, Jaroslav Halgašík, and Tomáš Krajník. Estimation of mobile robot pose from optical mouses. In *Research and Education in Robotics-EUROBOT 2010*, pages 93–107. Springer, 2011.

[45] Haiyang Chao, Yu Gu, and M. Napolitano. A survey of optical flow techniques for uav navigation applications. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pages 710–716, May 2013.

[46] Yung Siang Liau, Qun Zhang, Yanan Li, and Shuzhi Sam Ge. Non-metric navigation for mobile robot using optical flow. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4953–4958, Oct 2012.

[47] A. Talukder and L. Matthies. Real-time detection of moving objects from moving vehicles using dense stereo and optical flow. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 4, pages 3718–3725 vol.4, Sept 2004.

[48] Pablo De Cristóforis, Matias Nitsche, Tomáš Krajník, Taihú Pire, and Marta Mejail. Hybrid vision-based navigation for mobile robots in mixed indoor/outdoor environments. *Pattern Recognition Letters*, 2014.

[49] Mark Fiala. ARTag, a fiducial marker system using digital techniques. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 590–596. IEEE, 2005.

[50] Tomáš Krajník, Matías Nitsche, Jan Faigl, Petr Vaněk, Martin Saska, Libor Přeučil, Tom Duckett, and Marta Mejail. A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76(3-4):539–562, 2014.

[51] Yuji Ayatsuka and Jun Rekimoto. Active cybercode: a directly controllable 2d code. In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*, pages 490–495. ACM, 2006.

[52] L. Naimark and E. Foxlin. Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In *Mixed and Augmented Reality, 2002. ISMAR 2002. Proceedings. International Symposium on*, pages 27–36, 2002.

[53] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[54] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[55] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767, 2004.

[56] Tomáš Krajník, Pablo de Cristóforis, Matias Nitche, Keerthy Kusumam, and Tom Duckett. Image features and seasons revisited. In *European Conference on Mobile Robotics (ECMR)*, 2015. in review.

[57] Min Meng and Avinash C Kak. Mobile robot navigation using neural networks and nonmetrical environmental models. *Control Systems, IEEE*, 13(5):30–39, 1993.

[58] M. Fiala. Designing highly reliable fiducial markers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(7):1317–1324, July 2010.

[59] Dibyendu Mukherjee, Q.M. Jonathan Wu, and Guanghui Wang. A comparative experimental study of image feature detectors and descriptors. *Machine Vision and Applications*, 2015.

[60] 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.

[61] M.E. Angelopoulou and C. Bouganis. Feature selection with geometric constraints for vision-based unmanned aerial vehicle navigation. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 2357–2360, Sept 2011.

[62] Civera, Javier and Grasa, Oscar G and Davison, Andrew J and Montiel, JMM. 1-Point RANSAC for extended Kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[63] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[64] Kurt Konolige, Motilal Agrawal, and Joan Solà. Large-scale visual odometry for rough terrain. In Makoto Kaneko and Yoshihiko Nakamura, editors, *Robotics Research*, volume 66 of *Springer Tracts in Advanced Robotics*, pages 201–212. Springer Berlin Heidelberg, 2011.

[65] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22, May 2014.

[66] Carlos Estrada, José Neira, and Juan D Tardós. Hierarchical SLAM: real-time accurate mapping of large environments. *Robotics, IEEE Transactions on*, 21(4):588–596, 2005.

[67] 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.

[68] S. Se, D. Lowe, and J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 2051–2058 vol.2, 2001.

[69] Kostas E Bekris, Antonis A Argyros, and Lydia E Kavraki. Exploiting panoramic vision for bearing-only robot homing. In *Imaging beyond the pinhole camera*, pages 229–251. Springer, 2006.

[70] 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.

[71] Jianchang Mao and Anil K Jain. Artificial neural networks for feature extraction and multivariate data projection. *Neural Networks, IEEE Transactions on*, 6(2):296–317, 1995.

[72] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[73] Dominik Honegger, Helen Oleynikova, and Marc Pollefeys. Real-time and Low Latency Embedded Computer Vision Hardware Based on a Combination of FPGA and Mobile CPU. *International Conference on Intelligent Robots and Systems, Chicago, Illinois, USA*, 2014.

[74] Collective of authors. Altera, corp.@ONLINE. `http://www.altera.com`.

[75] S. Pedre, T. Krajnik, E. Todorovich, and P. Borensztejn. 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.

[76] Tomáš Krajník. *Large-scale Mobile Robot Navigation and Map Building*. PhD thesis, CTU, Faculty of Electrical Engineering, 2011.

[77] Tomas Krajnik, Sol Pedre, and Libor Preucil. Monocular navigation for long-term autonomy. In *Advanced Robotics (ICAR), 2013 16th International Conference on*, pages 1–6. IEEE, 2013.

[78] Farshad Arvin, John Murray, Chun Zhang, Shigang Yue, et al. Colias: an autonomous micro robot for swarm robotic applications. *International Journal of Advanced Robotic Systems*, 11(113):1–10, 2014.

CD content

Table 12 lists names of all root directories on CD together with their content.

Directory name	Description
`/mt`	Master thesis in pdf format.
`/sources`	Source codes.
`/documentation`	Documentation of the embedded module.

Table 12: CD content.