Co-design Hardware/Software of Real time Vision System on FPGA for Obstacle Detection

Ali Alhamwi

Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :
l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue le *05/12/2016* **par :**

ALI ALHAMWI

Co-design Hardware/Software of Real time Vision System on FPGA for Obstacle Detection

---

JURY

MICHEL DEVY	Président du Jury
HICHEM SNOUSSI	Rapporteur
JULIEN DUBOIS	Rapporteur
JEAN FRANÇOIS NEZAN	Examinateur
BERTRAND VANDEPORTAELE	Examinateur

École doctorale et spécialité :
EDSYS : Systèmes embarqués 4200046
Unité de Recherche :
Laboratoire d'analyse et d'architecture des systèmes
Directeur(s) de Thèse :
Jonathan PIAT
Rapporteurs :
Hichem Snoussi et *Julien Dubois*

Remerciements

Ce travail est le fruit d'une thèse effectuée au laboratoire LAAS-CNRS Toulouse, les travaux de cette thèse n'auront jamais pu voir le jour sans le soutien de nombreuses personnes auxquelles je voudrais exprimer ma profonde reconnaissance.

Tout d'abord je tiens à remercier mon directeur de thèse, Jonathan Piat qui m'a soutenu constamment durant la préparation de ce doctorat. Un grand merci à Bertrand Vandeportaele pour les discussions fréquentes sur les travaux, sur les blocages, sur les outils disponibles, et pour les nombreuses corrections faites sur le manuscrit.

Je remercie chaleureusement Monsieur Julien Dubois, Maître de Conférences à l'Université de Bourgogne, et Monsieur Hichem Snoussi, Professeur à l'université de technologie de Troyes pour l'intérêt qu'ils ont bien voulu porter à ces travaux et pour les précieuses remarques qu'ils ont apportées.

Ensuite je tiens à remercier Monsieur Michel Devy, pour l'honneur qu'il me fait en acceptant de présider le jury de cette thèse. Je remercie également Monsieur Jean François Nézan, Professeur à l'université de Renne, pour l'intérêt qu'il a porté à ce travail en acceptant d'en être un examinateur.

Je tiens à remercier toutes les personnes que j'ai pu côtoyer au LAAS ainsi que tous mes collègues et amis. J'adresse un grand merci à mes chers parents, mes frères et sœurs, Nadine, Sabrine, et Rabee qui illuminent ma vie. Ils ont accepté ma mauvaise humeur sans savoir les raisons et ont toujours été présents quand j'avais besoin.

Abstract

Obstacle detection, localization and occupancy map reconstruction are essential abilities for a mobile robot to navigate in an environment.

Solutions based on passive monocular vision such as Simultaneous Localization And Mapping (SLAM) or Optical Flow (OF) require intensive computation. Systems based on these methods often rely on over-sized computation resources to meet real-time constraints. Inverse Perspective Mapping allows for obstacles detection at a low computational cost under the hypothesis of a flat ground observed during motion. It is thus possible to build an occupancy grid map by integrating obstacle detection over the course of the sensor.

In this work we propose hardware/software system for obstacle detection, localization and 2D occupancy map reconstruction in real-time. The proposed system uses a FPGA-based design for vision and proprioceptive sensors for localization. Fusing this information allows for the construction of a simple environment model of the sensor surrounding. The resulting architecture is a low-cost, low-latency, high-throughput and low-power system.

Contents

Chapter 1

Introduction

Making a robot see was something in higher difficulty levels at the mid-twentieth century. After fifty years, researchers have achieved outstanding theoretical and practical successes. A broad field, called Computer Vision, has emerged as a discipline strongly affiliated to mathematics and computer science. Great progress has been achieved in the description of the way the appearance of objects changes when viewed from different viewpoints, and the expression of these changes as functions of objects shape and camera parameters. In the field of geometric computer vision, good math methods were introduced to explain the relations between objects in images and objects in world. On the practical side, we can cite the possibility of guiding a car through regular roads, recognition of road signs, pedestrian detection, forward collision warnings, vision guided robots (VGR) system, and many other applications demonstrated around the world. These achievements wouldn't be realized without sophisticated mathematical methods.

Not only are computer vision realizations grateful to mathematical algorithms, but also to the emergence of powerful, low-cost, and energy-efficient electronics. Every day we interact with many tiny computers. These small chips are the infrastructure of our modern world. An embedded system merges different kinds of devices (mechanical, electrical, chemical, ...etc) into one small computer in order to perform dedicated functions. Embedded systems are every where, in our homes, our phones, our cars, our cities, and even embedded in our bodies [Harris 2015]. Hence, these small devices are shaping our world today. Nowadays, embedded systems solutions become more useful and cost effective, and growing consumer interest in robotics.

With these tiny chips, it has become possible to integrate practical computer vision capabilities into embedded systems. The term "Embedded vision" refers to the use of computer vision in embedded systems [Dipert 2014b]. This topic is the subject of intensive ongoing research and development. The main purpose of embedded vision research is to adequate between desktop prototyping of vision algorithms and operational embedded systems environments.

Today, embedded vision creates a lot of opportunities in the markets in which it is used [Bier 2013], [Alliance 2016]. For example, Kinect video game controller is considered one of the fastest-selling electronic products. This product takes advantage of embedded vision to track users movements without the need for

hand-held controllers, makes video games accessible to all people [Bier 2013], [Bier 2011]. Another interesting application of embedded vision is video content analysis which can generate and organize video contents.

In robotics, the most of successful applications are based on embedded vision technology. It is commonly used to guide robots during navigation tasks, perform quality control, assemble automotive, to name a few [Quevedo 2012], [Dopplinger 2012]. Another interesting application is the use of embedded vision in surgery and healthcare [Hartford 2013]. Through mobile phone applications, functions like monitoring skin lesions for danger signs can be successfully performed.

Embedded vision based security applications is considered as a promising technology in the security market [Dipert 2014b], [Fularz 2015]. Vision systems are deployed in airports for surveillance tasks. The improvements in hardware platforms (processors, sensors) and algorithms allow to perform sophisticated surveillance applications, such as alert generation when an object is leaved or removed, reading vehicle license plate, ...etc.

In automotive area, automotive safety becomes one of the most exciting topic today. ADAS (Advanced driver assistance systems) systems based on embedded vision are mainly used to provide different technologies, including objects (pedestrian and vehicle) detection, lane detection and traffic sign recognition [Dipert 2014a],[Toyota 2016]. One of the promising implementations of embedded vision is the driver monitoring. The purpose of this technology is to monitor the driver's condition to ensure alerts while driving, by analyzing head, eye, and body movements [Dipert 2014c], [Routray 2011]. Parking assistance is another embedded vision application which is widely deployed in modern vehicles.

Let's cite Augmented reality which uses embedded vision technology to view computer-generated graphics in physical real world [Meier 2014], [Wilson 2014]. Due to the emergence of powerful embedded platforms, complex vision algorithms can be performed by mobile devices providing a sufficient processing performance at low cost and power consumption.

Embedded vision technology can touch every aspect of daily lives. The integration of vision capabilities (incorporating gesture recognition, face detection, facial recognition and eye tracking, ...etc) into existing products will make these products more responsive and easier to use. The market research of embedded vision is highly grown during last years. Research departments of large companies predict a good annual revenue growth due to embedded vision applications.

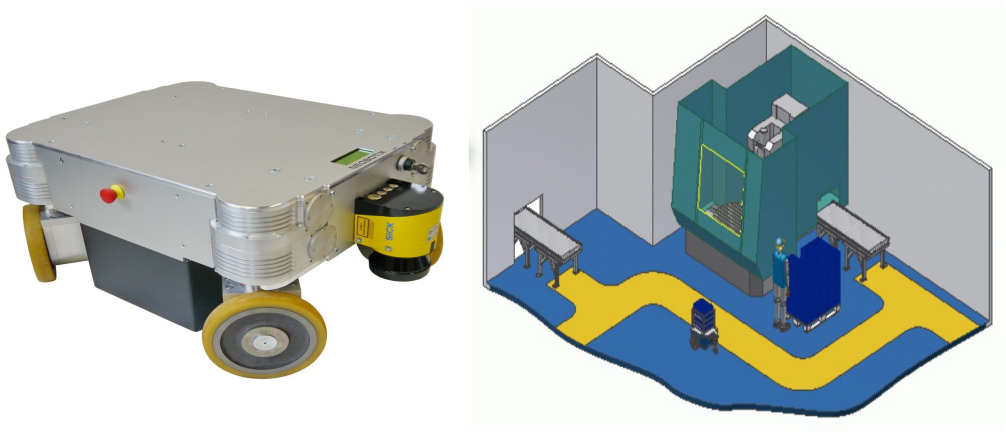


Figure 1.1: A typical transport robot, and an indoor environment model taken from [Neobotix 2016]

1.1 Thesis Context

The general context of this thesis is autonomous visual navigation of mobile robots in indoor environments. Intended applications are transport robots in factories or logistics centres. In research, the focus is in the exploration of unknown areas of buildings, and static environments modeling. For this purpose, different methods have been developed to achieve time-efficient exploration [Wettach 2016], [Arndt 2016]. For example, large ground floors in an office building can be detected and mapped, thus making easy to accomplish transport or monitoring tasks automatically. Autonomous robots navigating in such environments can be used to produce reliable exploration systems with reduced total cost, providing a complete description of the environment. Figure 1.1 shows a robot kit mainly used as a transport system for different scenarios in indoor environments. Such robots could react to the changes of its surrounding without danger of collision. Additionally, they are capable to autonomously explore unknown areas and build an occupation map. This latter serves as a reference for transport tasks.

Another important application is the use of indoor mobile robots for the care of elderly. For example, when an emergency situation is detected, a fast navigation (reduced latency, high throughput) in a typical apartment environment is required to perform emergency procedures or to support specific household tasks. Hence, the response time of a robot in such situations is a an important problem. With the help of visual navigation methods, the implementation of service tasks can be more efficient.

LAAS has participated to several projects devoted for indoor navigation.

Figure 1.2: On the left, Robot's and eldercare's future taken from [Smith 2013]. On the right taken from [Aging 2015]

Different hardware-software designs and implementations of well-known vision-based obstacle detection and identification have been developed. For example, an obstacle detection system was developed in the PhD of M.Ibarra Manzano [MANZANO 2011]. The system is performed using a belt of micro-cameras mounted on a robot. Obstacle detection task is accomplished by a method based on pixel classification from color and texture descriptors.
One of the key aspects of development is hardware prototyping on FPGAs (Field Programmable Gate Arrays) for which the algorithm-level and hardware-level optimizations are prevalent. As throughput and latency are regarded as limiting factors for the reactivity of robots, hardware prototyping on FPGAs is used to produce high throughput and low latency architectures. Different algorithms and architectures based on FPGA were developed in the PhD of D.Botero Galeano [Botero-Galeano 2012] for obstacle detection, and to perform localization using multi-spectral cameras.

This thesis is developed in the context of CAMERA BELT project. The project aims to build an embedded vision system multi-camera for obstacle detection and localization of a robot navigating in indoor environments. In this project, the hardware architecture will include 8 camera sensors mounted on a robot and connected to FPGAs. The 8 images acquired from different viewpoints allow to build a 2D occupancy map, thus making easy to accomplish navigation task in indoor environments. The complexity resulting from the use of an FPGA target to process multiple images acquired at the same instant imposes different constraints about the choice of obstacle detection method. This latter must be simple and reliable when it is implemented in a hardware architecture.

1.2 Thesis Objectives

The objective of this thesis is to design a real time vision system for a robot navigating in indoor environments. The proposed system is devoted to the obstacle detection and localization problems. Additionally, the designed system must facilitate navigation task by building a 2D occupancy map that represents obstacles localization in the ground plane.

In this thesis, we will propose the system parts, and design a hardware module for each part of the system (for image acquisition, distortion correction, homography transformation, Inverse Perspective Mapping, obstacles segmentation, obstacles localization and 2D occupancy map reconstruction).

In order to perform our system, we set the following objectives:

- Development and validation of an obstacle detection method based on visual information. A hardware module is then designed and implemented on FPGA.
- Development and validation of a method for obstacles segmentation. The method is then modelled in a hardware design and implemented on FPGA.
- Proposing a method for obstacles localization in the ground plane.
- Development and validation of the proposed method. A hardware module is then designed to accomplish the localization task.
- Development and validation of a method for free space detection based on localization module results. A hardware module is then deigned and implemented on FPGA.
- Development and validation of a method to build a 2D occupancy map that represents obstacles localization in the ground plane.

So, Our contributions are as follows:
In methodology level:

- Proposing a method for obstacles localization in the ground plane.
- Building a 2D occupancy grid map based on obstacles localization module results.

In hardware level:

- the hardware design of homography transformation is optimized, distortion correction and homography transformation are merged in one hardware module.
- A hardware architecture is designed for obstacles segmentation task. Different hardware modules are developed to perform Gaussian filter, binarization based on Otsu's algorithm, and morphological operations.

- Two hardware modules are proposed for obstacles localization in the ground plane. The first module is an implementation of a method proposed in the state of art while the second module is an implementation of our proposed method.
- A hardware architecture is designed for free space detection based on a method that exists in the state of art.
- For rapid system prototyping, a co-design hardware/software is developed using *Xillybus* core for 2D occupancy grid map reconstruction.

1.3 Document Organisation

The thesis is composed of six main chapters introduced as follows:

- **Chapter 2** introduces a brief presentation of obstacle detection methods that exist in the state of art. We present sensor types which can support obstacle detection methods. We then present a summary of obstacle detection methods based on visual informations.
- **Chapter 3** presents the theoretical background for the different algorithms evaluated during this thesis.
- **Chapter 4** explains, by and large, different embedded systems types. A brief presentation is introduced for different embedded systems platforms. Then, we explain in detail the embedded systems based on FPGAs architecture as our proposed architectures are based on FPGAs targets.
- **Chapter 5** introduces our contribution to the system methodology. Obstacles localization method, and 2D occupancy grid map reconstruction are described.
- **Chapter 6** presents the three proposed architectures and the hardware modules of each proposed system. We describe the design properties for each hardware module coded in VHDL, and the modules integration in the FPGA architecture.
- **Chapter 7** presents the results of the proposed architectures. Comparison to the state of art and the software implementation of the methodology are included in this chapter.

Finally, we present a conclusion and the perspective of thesis work.

Chapter 2

Obstacles Detection Methods

Contents

2.1 Introduction

Obstacle detection is a fundamental ability for a mobile robot to operate in a cluttered indoor environment and is essential to perform basic functions like obstacle avoidance and navigation. Traditionally, autonomous navigation systems are equipped with a set of sensors. By interpreting sensors readings, the required informations about obstacles distances in the environment around the robot are extracted. This allows to build a map that represents the occupied and free space of the environment around robot. This is a long studied problem in robotics and a lot of different methods are used to perform this task.

In this chapter, the existing architectures used for obstacle detection and 2D occupancy grid maps reconstruction are introduced. The first part presents obstacle detection systems based on SONAR (SOund Navigation And Ranging), RADAR (RAdio Detection And Ranging), and LIDAR (LIght Detection And Ranging) sensors. The second part will introduce obstacle detection methods based on vision sensors.

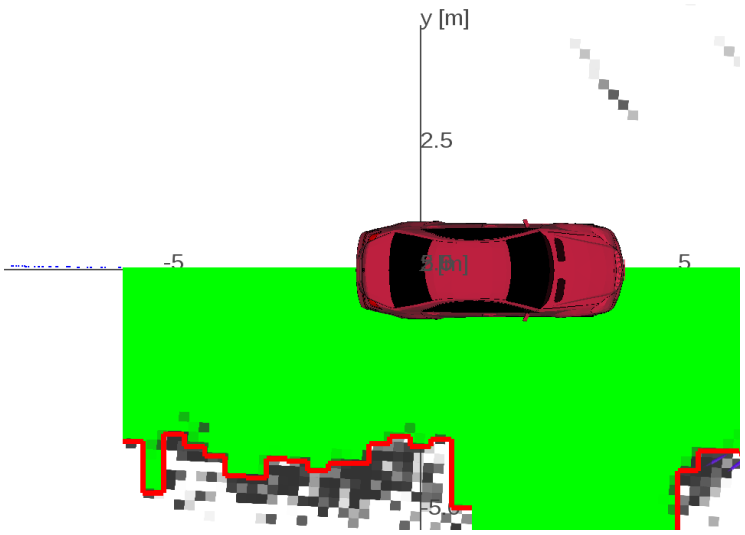


Figure 2.1: Detecting parking space using a system based on Radar sensors (taken from [Schmid 2011]).

2.2 Obstacle Detection based on Sonar and Radar sensors

Sonar based methods were used in very early research. Thanks to the low cost of ultrasonic sensors, and their good sensing range they have been widely used for obstacle detection and avoidance systems. Sonar methods are based on the computation of time differences between the emission and reception of a ultrasonic signal to compute the distance to a reflective surface. However, data measurements based on ultrasonic sensors are affected by air temperature and humidity. In addition, spatial position detection for obstacles edges is limited by obstacle distances and the angle between obstacle surface and the acoustic axis. Therefore, they prove to be unreliable and give imprecise depth information with low spatial resolution.

Radar based systems works by measuring properties of the reflected radio signals. In [Schmid 2011], a system based on three standard short range radar sensors is used in order to determine parking space in an indoor environment as pictured in 2.1. The drawbacks of Radar systems are that they provide noisy data with low-accuracy. In addition, the high angular uncertainty of radar sensors leads to decrease the accuracy in produced maps. So, Radar systems alone are insufficient to produce accurate terrain traversability maps. Recently, many researches focus in the fusion between these sensors and other active sensors in order to produce 2D occupancy grid maps with high resolution as depicted in [Ahtiainen 2015].

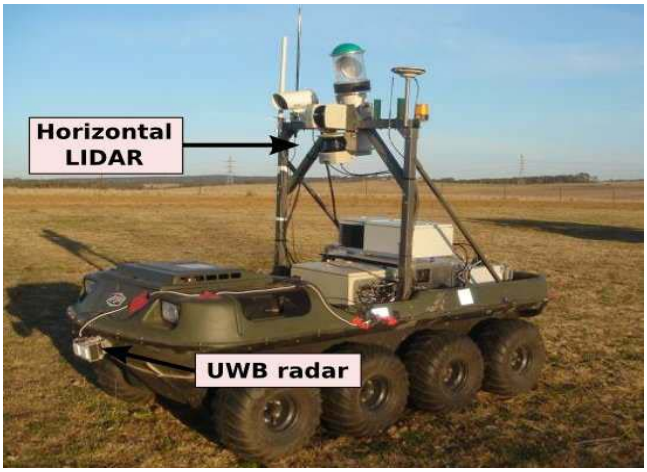


Figure 2.2: Argo robotic platform equipped with Radar and Lidar sensors in [Ahtiainen 2015].

2.3 Obstacle Detection based on Lidar sensors

Using Lidar devices for obstacle detection and reconstruction of 2D free space maps has been a standard in robotics for many years. LiDAR works by analyzing the reflected light when objects are illuminated by a laser beam. LiDAR sensors compute the distance from an obstacle by measuring the round-trip time of flight of an emitted pulse (or modulated wave) of light.

LiDAR sensors provide an accurate information, works independent of the ambient light, and offer detailed high rate data of the sensor proximity. This type of sensor gives a 2D information when used in planar or can be used to generate a 3D cloud in scanning mode. Compared to data received from vision sensors, data received from LiDAR scanners is less noisy, more accurate, long range, and precisely show the free space of a robot's environment. However, LiDAR scanners are generally expensive which makes them not suitable for low-cost robotic.

These sensors are also prone to mechanical damages because of their mirror assembly, and provide a performance with low level of vertical resolution. LiDAR may also generate a wrong estimate for black and shiny surfaces [Boehler 2003], and can't be compared to photogrammetric data when a high level of accuracy is required.
Beside, LiDAR measurements suffer from the lack of a rich information about an obstacle due to the sparse nature of data. Compared to vision sensors, LiDAR grids are less informative and more ambiguous.

A simple and practical system based on the 2-D LIDAR LMS511 from SICK is

Figure 2.3: 2D LiDAR SICK LMS511 taken from [Peng 2015].

introduced in [Peng 2015] and shown in figure 2.3. The proposed system uses an algorithm designed for obstacle detection and avoidance.
3D LIDAR scanners allow to build 3D maps of a robots environment. In [Shinzato 2014], a sensor fusion-based solution is proposed for obstacle detection. The proposed approach is based on perspective images acquired from a single camera and a 3D LIDAR.

2.4 Obstacle Detection based on vision sensors

Vision-based methods are becoming increasingly popular due to the dramatic cost reduction of cameras and associated computing architecture. Camera sensors also provide a rich information that can be used for a variety of heterogeneous tasks (obstacle identification, localization, tracking ...). A good vision-based obstacle detection system must be capable of detecting and localizing obstacles at high speed and low latency, avoiding false positive, and minimizing impact on overall system performance (system speed and power consumption).

Vision based algorithms like Optical Flow or SLAM have been widely adopted in robotic applications for obstacle detection task. However, these aforementioned algorithms are regarded as computational intensive methods. The low-cost and richness of the sensor comes with the needs of complex algorithms to extract information from the scene which can make them impractical.

In this section, a brief summary is given for obstacle detection techniques based on optical flow, stereo vision, pattern recognition, and inverse perspective mapping (IPM). These methods are performed using passive vision sensors. Then, techniques based on active vision sensors are introduced.

2.4.1 Methods based on Optical Flow

Optical Flow (OF) is a popular technique which is used in many areas [Beauchemin 1995] such as : object detection and tracking, image dominant

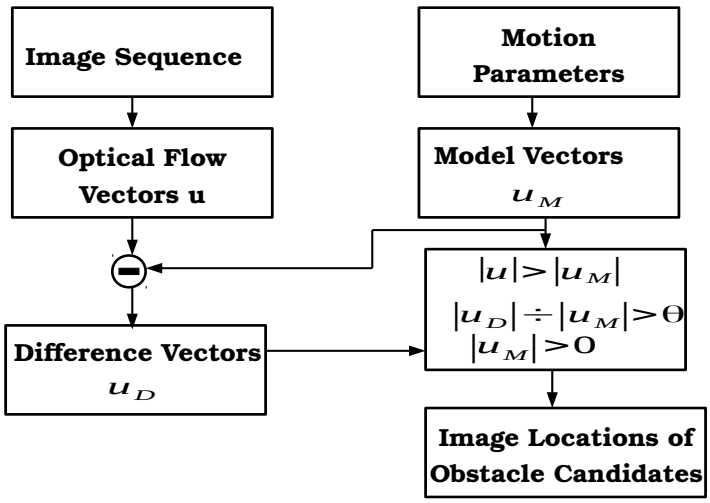


Figure 2.4: Obstacle detection approach based on optical flow method (taken from [Enkelmann 1991]).

plane extraction, motion detection, robot navigation and visual odometry. OF is based on the local motion of image elements in image sequences. It aims at computing the motion between two sequential frames. The estimation of optical flow vectors can be done using different approaches divided in two classes: features based methods [Horn 1980], [Smith 1997], [Hildreth 2003], [Castelow 1988] and analytical methods [Enkelmann 1991], [Nagel 1987], [Nagel 1983], [J.Heeger 1988].

The figure 2.4 illustrates the obstacle detection procedure based on the estimation of optical flow vectors. This approach consists of three steps [Enkelmann 1991].

1. The local motions (OF vectors) for each pixel of a given grid are measured from the image sequence.

2. The prediction of the optical flow model from the measured camera motion is performed.

3. The detection of the obstacles is then performed by comparing the optical flow vectors extracted from the images and the optical flow vectors estimated from the motion.

The obstacle detection based on optical flow requires a great computational effort because of the dense sampling it assumes. There is a large work in the literature presenting new solutions for efficient computation and high accuracy. The presented methods are compared to other approaches in terms of computing complexity and accuracy [Silar 2013]. As these methods aim at solving accuracy and computation complexity, the overall performance of a system based on such approaches tends to be lower in terms of real time requirements(frame-rate, latencies). In most cases, OF is a primary phase in more complex vision algorithms.

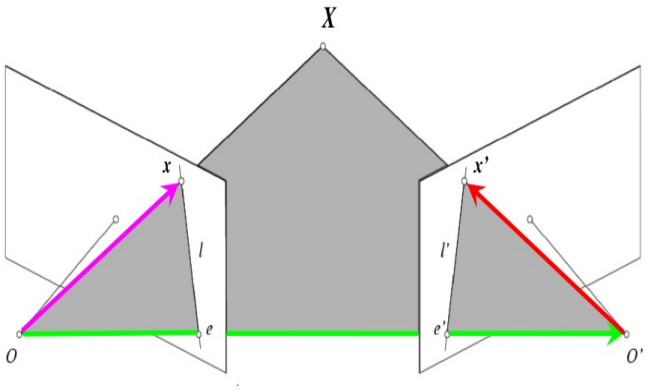


Figure 2.5: Epipolar Geometry [Hartley 2004] [Hoiem 2012]

.

Many considerations must be taken into account for the available computing resources.

Recent improvements of the computation efficiency have been proposed for optical flow methods by using newly available powerful computing platforms. Because software implementations of this kind of algorithms are not effective enough, some studies have proposed solutions based on dedicated hardware architectures. In [Bako 2015], a real time hardware implementation of the optical flow computation is proposed using an embedded FPGA based platform.

2.4.2 Methods based on Stereo vision

Thanks to the low cost of passive vision sensors, it is possible to integrate two camera to produce stereo vision systems. A system based on stereo vision sensors can extract 3D features from the perceived scene given some requirements on its texture and lighting.
Each camera provides a 2D representation of the environment. By processing two 2D images, one can map each 2D point represented in image coordinate systems to its correspondence in the 3D world coordinate system. The epipolar geometry provides some constraints on the 2D observations given the two camera poses.

Let us introduce some basic notations used in epipolar geometry as shown in figure 2.5.

- The baseline is a line connecting the two camera centers O and $\grave{O}$.
- The epipoles are the intersections of the baseline with the image planes in (e) and $(\grave{e})$.

- An epipolar plane is a plane containing the baseline $O\grave{O}$ and a given 3D point X.

- The epipolar lines are the intersections of a given epipolar plane with the image planes in (l) and $(\grave{l})$.

The camera matrices containing the intrinsic parameters of the two cameras are supposed known (K and $\grave{K}$). A 3D scene point (expressed in camera coordinate systems) X (resp. $\grave{X}$) is projected to the first (resp. second) camera to 2D pixel coordinates x (resp. $\grave{x}$) as follows:

$$x = K.X \qquad ; \qquad \grave{x} = K.\grave{X} \tag{2.1}$$

The first camera's coordinate system is chosen as the world coordinate system. R and T being the rotation and translation between the two camera frames, the 3D scene point X represented in the first camera coordinate system is related to $\grave{X}$ in the second camera system as follows:

$$X = R\grave{X} + t \tag{2.2}$$

The epipolar constraint enforces that the points X, $\grave{X}$ and the camera optical centers O and $\grave{O}$ lie in the same plane:

$$X.[t \times (R\grave{X})] = 0 \tag{2.3}$$

This is rewrote in Eq. 2.4 introducing the Essential matrix E [Hoiem 2012]:

$$X^T E\grave{X} = 0 \tag{2.4}$$

This relation can then be expressed in terms of image coordinates using the Fundamental matrix F:

$$F = K^{-T} E\grave{K}^{-1} \tag{2.5}$$

$$x^T F\grave{x} = 0 \tag{2.6}$$

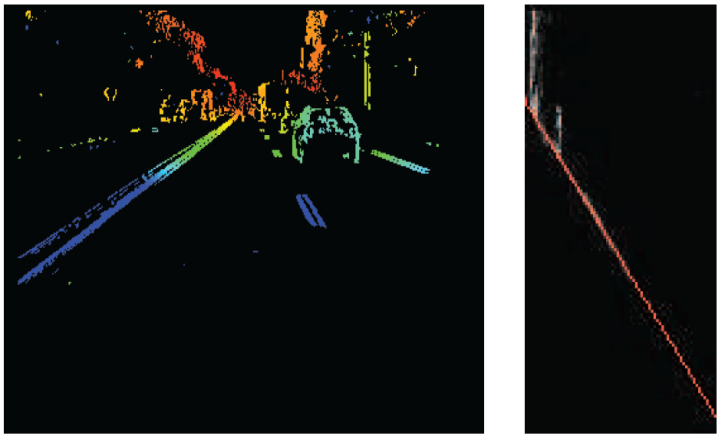


Figure 2.6: Results of the obstacle detection system based on V-disparity in [Labayrade 2002]. The disparity (resp. V-disparity) map is shown on the left (resp. right). The 3D points on the ground lie in a line in the V-disparity map whereas the 3D points on the obstacle (car) do not.

The stereovision configuration allows to constraint possibly matching points between the two images. It reduces the correspondence problem from two dimensions to one. The images from the two cameras can be rectified to obtain synthetic images that would have been acquired in a particular configuration where the epipolar lines are aligned with the horizontal lines of the image planes. In theese rectified images, the matching between pixel can be achieved individually on each line and a single value for each pixel indicates the matching pixel on the same lie in the other image. This value d is named disparity and is related to the depth of the corresponding 3D points in the scene[Miled 2007]. A disparity map is an image where each pixel encodes the corresponding disparity value.
The obstacle detection based on stereo vision has been studied in recent years. In this class of methods, one can cite "The V-disparity" approach in [Labayrade 2002] (see figure 2.6). A detection based on disparity map is performed to extract vertical objects on the scene. In [Ganoun 2009] (see figure 2.7), the authors propose to segment the ground plane and remove it from the disparity map. In [Tarel 2007], the ground is segmented using 3D reconstruction methods.

Different obstacle detection systems were developed in occupancy grid maps framework. In[Badino 2007], three models for occupancy grid map representation based on stereo measurements are presented.
In[Oniga 2010], Digital Elevation Map (DEM), a height based representation, is performed by transforming 3D stereo data measurements into a rectangular digital grid. The authors make use of two classifiers in order to mark DEM cells as road or obstacles. The proposed system is supposed to allow for real time execution due to specific optimizations applied to the software implementation.

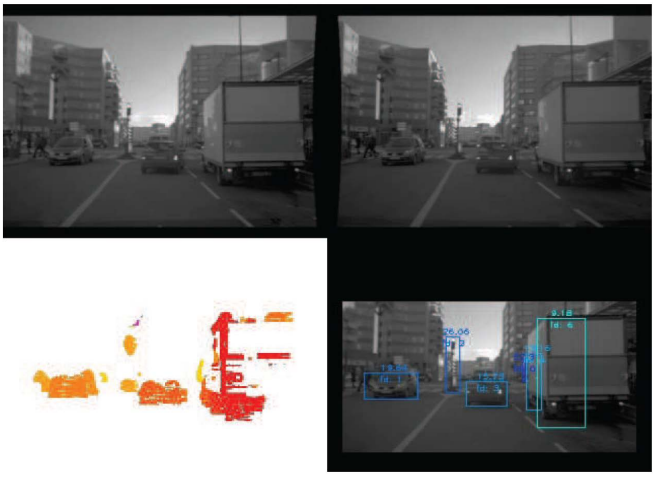


Figure 2.7: Tracking multiple targets using a system based on stereo vision (taken from [Ganoun 2009]).

In [Oniga 2015], a method based on a fusion between laser range finder and stereo vision is introduced. The authors claim that the proposed approach is simple and efficient in terms of computational resources, but the performance is limited by the accuracy of the intra-sensor calibration.

2.4.3 Methods based on Pattern Recognition

In the field of computer vision, the problem of linking semantics between high level concepts and low level features has been studied. Pattern recognition is a branch of machine learning that focuses on the recognition of patterns and regularities in data [Bishop 2006]. Classification methods used in computer vision are often based on the extraction of features that may include color, texture, shape, or spatial relation informations. Recently, different sophisticated feature extraction techniques have been proposed. An obstacle detection system based on color and/or texture feature algorithms requires an important amount of calculations. Some optimizations have been proposed in [Cervantes 2008] in order to reduce computation time by performing a color-based segmentation before applying a region based classification. The proposed approach is however not suitable for all applications. In [Manzano 2010], a hardware based architecture based on texture-color classification is proposed in order to detect specific objects on the scene. Classifier parameters are learnt offline from ground examples. Hence, the proposed system is able to detect obstacles on indoor environment as shown in figure 2.8.

Image features based on HOG (Histogram of Oriented Gradient) [Dalal 2005] and SIFT (Scale Invariant Feature Transform) [Lindeberg 2012], [Lowe 1999] have been

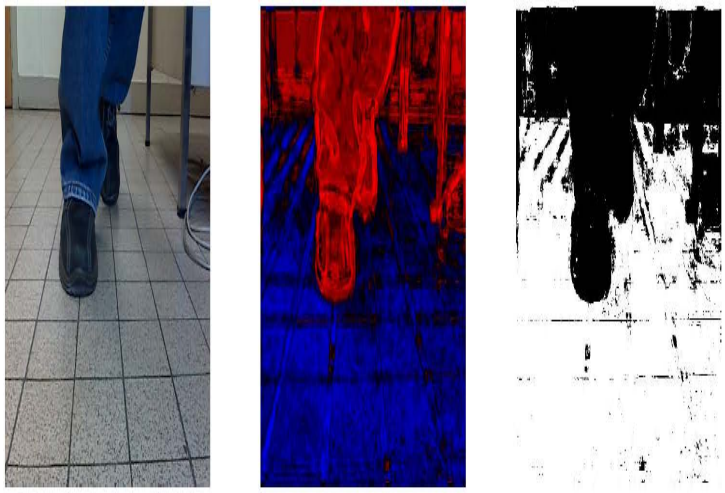


Figure 2.8: Obstacle detection based on terrain classification (taken from [Manzano 2010]).

widely used. Support vector machine (SVM) classifier [Shawe-Taylor 2000] with these features becomes one of the popular techniques used for obstacle detection. In [Lee 2015] an approach is introduced in order to reduce computing time of (SVM) by reducing the dimension of HOG feature. Authors claim that they can speed-up SVM classifier for vehicle detection by about three times while maintaining the original detection performance [Lee 2015].

2.4.4 Methods based on Inverse Perspective Mapping

Inverse perspective mapping IPM is a geometrical transformation where an initial image is used in order to produce a new image for the same scene from a different position. This method is essentially based on the angle of view of the objects in the scene and their distances from camera due to the perspective effect.

IPM transformation is influenced by the camera extrinsic parameters (position and orientation) and intrinsic parameters (focal, principal point and optical distortions). This transformation is computed for the ground plane in the scene. A subtraction is performed for each pixel between the transformed previous and current (acquired at the new position) images.

The obstacle detection based on IPM approach is proposed by [Bertozzi 1998b]. This method can be performed either by two cameras (Stereo IPM: SIPM) or one camera (Mono IPM: MIPM).The figure 2.9 shows results of obstacle detection based on IPM in [Bertozzi 1998b] where SIPM is performed. The two acquired images $(a), (b)$ are transformed to remapped views of the scene $(c), (d)$. Subtraction and threshold segmentation are performed between remapped images in (e) and

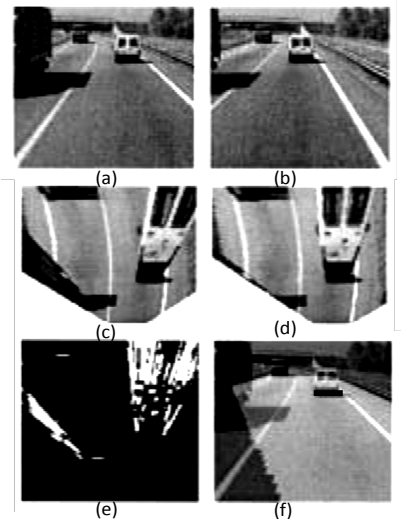


Figure 2.9: IPM method for obstacle detection in [Bertozzi 1998b]. The two acquired images $(a), (b)$ are transformed to remapped views of the scene $(c), (d)$. Subtraction and threshold segmentation are performed between remapped images in (e) and detected obstacle are shown in (f) (taken from [Bertozzi 1998b]).

detected obstacle are shown in (f).

2.4.5 Methods based on active vision sensors

Systems based on 3D active sensors have been adopted to solve traditional problems in vision systems such as tracking, recognition and feature extraction. An RGB-D sensor like Kinect produces synchronized color images and depth images [Totilo 2010]. Kinect makes use of an infrared (IR) projector which emits a textured light pattern. The disparities measured between expected and observed patterns acquired by a monochrome IR camera permit the estimation of the depth value for each pixel.

Another way of obtaining the depth image is to use Time-of-Flight (TOF) sensors [Castaneda 2011]. Their principle is to measure directly the distance from the camera to the scene for each pixel through the duration of the light round trip travel, the light being emitted by a source located on the camera.

In [Lee 2012], an obstacle detection system based on 3D-sensor includes two stages pictured in figure 2.10:

- Edge detection is firstly performed in the depth image in order to distinguish different objects. This process allows to segment objects in the scene as similar values of intensities in depth image may belong to the same object.

- The detection of the obstacles is performed by using the properties of the

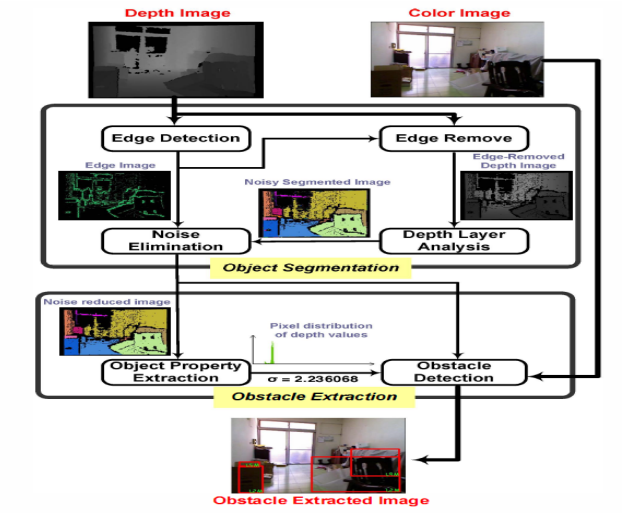


Figure 2.10: Obstacle detection system based on active vision sensor in [Lee 2012]. The system is composed of two stages. The first stage is object segmentation (Edge detection and depth layers analyzation). The next stage is obstacle extraction (taken from [Lee 2012]).

detected objects. The mean and standard deviation of depth values is computed for each object. As the distribution of pixels corresponding to floor in the depth map is scattered, its standard deviation is larger than those of the other objects, hence the floor object can be removed from the other detected objects.

2.5 Conclusion

In this chapter, different methods for obstacle detection were presented. A robot that uses optical flow based methods can find features (corners and edges) of obstacles. However, the algorithms that use optical flow encounter difficulties in large non-textured areas and for edges that suffer from the aperture problem. Obstacle detection based on stereo vision methods is widely used in robotics. However, stereo vision methods suffer from the long computation time required to compute stereo correspondences. In addition, the 3D reconstruction is also computationally prohibitive. Classification methods become increasingly popular. However, these methods need a prior knowledge about the environment. In addition, obstacle detection based on classification methods require a great computational load. Vision approaches based on IPM allow the detection of obstacles under the hypothesis of flat ground, this method is based on the perspective effect perceived from a scene when observed from two different points of view. This method is simpler in terms of computation load when it is compared to other methods. IPM method does not

require textured areas when it is compared to optical flow. IPM method doesn't need the computation correspondences between two sequential images for detecting obstacles. In addition, it doesn't require a prior knowledge about the environment when it is compared to classification methods. However, IPM method requires an accurate camera calibration. Additionally, the method could encounter problems with shiny surfaces and objects shadows.

Chapter 3

Theoretical Background

Contents

3.1 Introduction

In this chapter, the concepts and operators to define a vision-based obstacle detection method are introduced.
The proposed system is based on inverse perspective mapping for obstacle detection proposed in [Bertozzi 1998b] (see Figure 3.1). Since this method is based on a geometric transformation for the ground plane, the pinhole camera model is firstly introduced. Then a brief overview of the theory behind IPM is given. IPM only allows to compute a pixel-level classification of the image, hence a method for obstacle segmentation and post-processing operators are then presented. An obstacle localization method proposed by the authors of IPM method [Bertozzi 1998b] is finally described in this chapter.

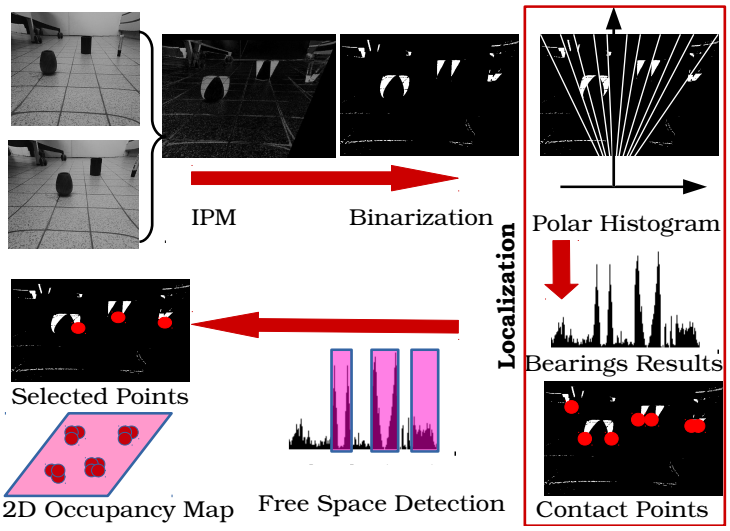


Figure 3.1: General overview of the proposed approach.

3.2 Pinhole camera model

A camera projects a 3-D environment to a 2-D image. The pinhole camera model achieves a central projection to a 2-D plane through an optical center (pinhole aperture). This model is widely used in computer vision due to its simplicity as discussed below. Some definitions of the different coordinate frames follows:

Let ω_{im} be a 2D Euclidean coordinate frame attached to the image plane of the camera with the culumn and line axes respectively denoted ω_{imu} and ω_{imv}.

Let ω_{cam} be a 3D Euclidean coordinate frame whose origin is the center of projection of the pinhole camera. Let ω_{camx} and ω_{camy} be the first 2 axes of this coordinate frame respectively parallel to the ω_{imu} and ω_{imv} axes of the image plane. The ω_{camz} axis is thus orthogonal to the image plane.

Let $\tilde{p_i} = \begin{bmatrix} p_u & p_v & p_w \end{bmatrix}^T$ be a projective point in projective space P^2. In that projective space, points can either represent positions or directions in the euclidian space R^2 using the coordinate frame ω_{im}, depending on the value of its last component p_w. If p_w is equal to 0, $\tilde{p_i}$ represents a direction $\begin{bmatrix} p_x & p_y \end{bmatrix}$. Oppositely, if p_w is not equal to 0, $\tilde{p_i}$ represents a position $\begin{bmatrix} p_u/p_w & p_v/p_w \end{bmatrix}$. Considering that $\tilde{p_i}$ defines a position $\begin{bmatrix} u & v \end{bmatrix}$, it is possible to express it up to a scale factor as:

$$\tilde{p_i} = \begin{bmatrix} s.u \\ s.v \\ s \end{bmatrix} \tag{3.1}$$

Let $\tilde{P_{cam}} = \begin{bmatrix} P_{camx} & P_{camy} & P_{camz} & 1 \end{bmatrix}^T$ be a point in projective space P^3 corresponding to the position $\begin{bmatrix} P_{camx} & P_{camy} & P_{camz} \end{bmatrix}$ in euclidian space R^3 using the coordinate frame ω_{cam}.

3.2.1 Intrinsic parameters

As pictured in figure 3.2, the center of projection is known as the camera or optical center. The line from the camera center perpendicular to the image plane is known as the principal axis or principal ray of the camera. The point where the principal axis meets the image plane is known as the principal point. Let K be the camera calibration matrix as defined in equation 3.2. This matrix contains the different intrinsic parameters: f_u and f_v are the focals in pixel unit and p_u and p_v are the principal point coordinates in the image plane in pixel unit.

$$K = \begin{bmatrix} f_u & 0 & p_u \\ 0 & f_v & p_v \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

The focal parameters in pixel unit are obtained from the focal f (in metric unit) and the dimensions of the pixel in u and v directions e_u and e_v (in metric unit per pixel), thus f_u and f_v differ if the pixels are not square:

$$f_u = f.e_u^{-1} \qquad f_v = f.e_v^{-1} \tag{3.3}$$

The projection of a 3D point $\tilde{P_{cam}}$ to the image plane of the camera is obtained by the equation 3.4. The 3×4 matrix $[I|0]$ is used to ensure size consistency between K and $\tilde{P_{cam}}$ and its effect is to omit the last component of $\tilde{P_{cam}}$. This omission renders the fact that all the 3D points along a line passing through the optical center (even at infinity) are projected to the same location in the image plane. Thanks to the homogeneous parametrization of the points, the central projection is expressed as a linear mapping between the homogeneous coordinates. This simplicity is the main cause of the wide use of the pinhole camera model.

$$\tilde{p_i} = K.[I|0].\tilde{P_{cam}} \tag{3.4}$$

3.2.2 Extrinsic parameters

In practice, the 3D points in space are often expressed in a different Euclidian coordinate frame known as the world coordinate frame ω_ω. These two coordinate frames are related via a rigid transform composed of a rotation and a translation as pictured in figure 3.3.

Let P_ω be the coordinates of a 3D point in the world coordinate frame, and P_{cam} denotes the same point in the camera coordinate frame. C being the coordinates of the camera center in the world coordinate frame, and R being a 3×3 rotation

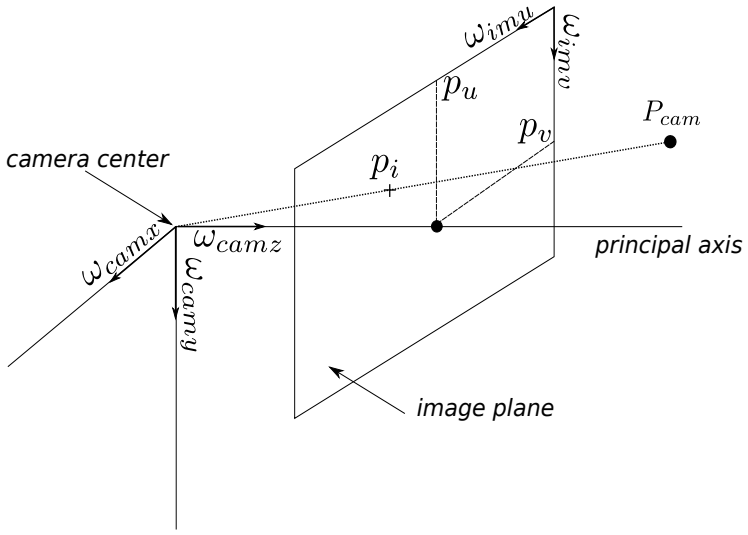


Figure 3.2: Pinhole camera model.

matrix which denotes the orientation of the camera coordinate frame. The equation 3.5 relates P_ω and P_{cam}:

$$P_{cam} = R.(P_\omega - C) \tag{3.5}$$

Considering homogeneous coordinates, the equation 3.5 can be rewritten as:

$$\tilde{P_{cam}} = \begin{bmatrix} R & -R.C \\ 0 & 1 \end{bmatrix} .\tilde{P_\omega} \tag{3.6}$$

The Eq. 3.4 and Eq. 3.6 can be combined to obtain directly the projection of a 3D point expressed in the world coordinate frame:

$$\tilde{p_i} = K.R.[I| - C].\tilde{P_\omega} \tag{3.7}$$

The projection matrix $\tilde{P}$ is used to express the projection of $\tilde{P_\omega}$ to $\tilde{p_i}$:

$$\tilde{p_i} = \tilde{P}.\tilde{P_\omega} \tag{3.8}$$

If one writes $t = -R.C$, the projection matrix expression is:

$$\tilde{P} = K.[I|0]. \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = K.[R|t] \tag{3.9}$$

3.2.3 Radial Distortion

The pinhole camera model assumes that a world point, its image and the optical center are collinear. Also, the world lines are mapped to lines in the images [Hartley 2004]. For real cameras made of lenses, this linear model may not hold.

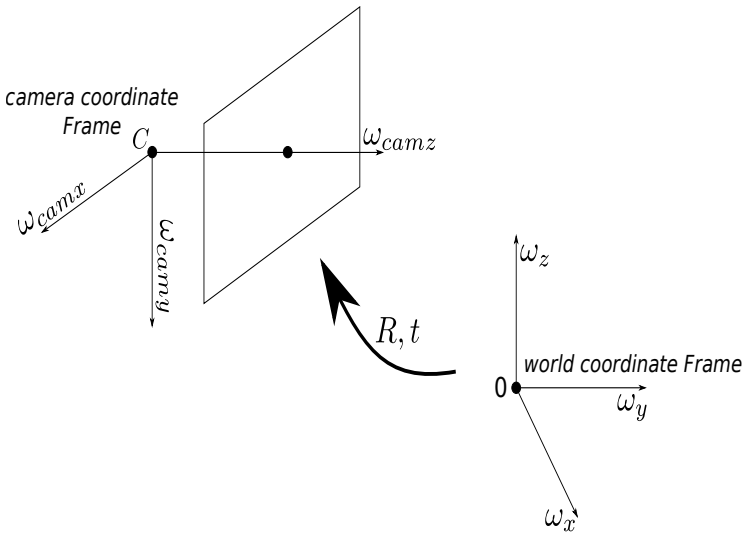


Figure 3.3: Extrinsic transformation.

The most important deviation is induced by radial distortions. They can induce "barrel" or "pincushion" distortion as pictured in figure 3.4. The modeling of the distortions aims at being able to correct the images in order to obtain images that would have been acquired without distortions, ie. where straight lines of the scene are imaged to straight lines.

Different models exist to model the distortions. A sixth order model is used in the next equations. This model is the one used in camera calibration with OpenCV [OpenCV 2014].

This model applies in the normalized image plane, which is a plane located at $\omega_{camz} = 1$ using metric unit for its axes ω_{nx} and ω_{ny}.

A point location $\begin{bmatrix} x_{distorted} & y_{distorted} \end{bmatrix}$ is mapped to $\begin{bmatrix} x_{corrected} & y_{corrected} \end{bmatrix}$ using the following equations:

$$x_{distorted} = x_{corrected}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.10}$$

$$y_{distorted} = y_{corrected}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.11}$$

$$r^2 = x_{corrected}^2 + y_{corrected}^2 \tag{3.12}$$

k_1, k_2, k_3 are the radial distortion coefficients. Higher-order coefficients are not considered in the implementation of the proposed hardware architectures because the added computational complexity is not justified.

In order to apply the radial distortion correction to the image, it is necessary to:

- Sample each pixel location of the corrected image to generate $\begin{bmatrix} u_{corrected} & v_{corrected} & 1 \end{bmatrix}^T$.

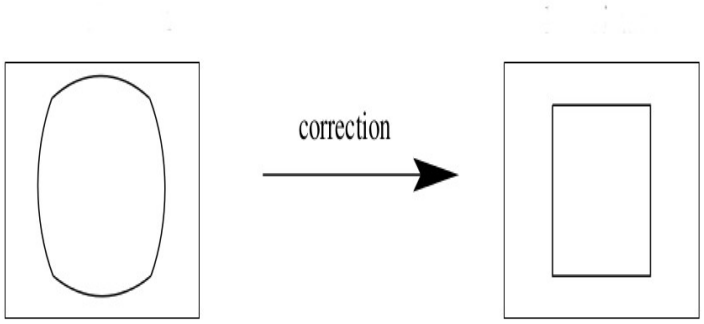


Figure 3.4: Example of the "barrel" distortion correction.

- Get the corresponding location in the normalized image plane by multiplying by the inverse of the K matrix (see Eq. 3.13).

$$\begin{bmatrix} x_{corrected} \\ y_{corrected} \\ 1 \end{bmatrix} = K^{-1}. \begin{bmatrix} u_{corrected} \\ v_{corrected} \\ 1 \end{bmatrix} \tag{3.13}$$

- Apply Eq. 3.10 and 3.11 to obtain the location of the corresponding point with distortion.

- Get the corresponding location in the image plane by multiplying by the K matrix (see Eq. 3.14).

$$\begin{bmatrix} u_{distorted} \\ v_{distorted} \\ 1 \end{bmatrix} = K. \begin{bmatrix} x_{distorted} \\ y_{distorted} \\ 1 \end{bmatrix} \tag{3.14}$$

- To determine the pixel value associated to the non integer position $\begin{bmatrix} u_{distorted} & v_{distorted} \end{bmatrix}^T$, one can achieve different kind of interpolation (nearest neighbor, bilinear...).

3.3 Inverse Perspective Mapping

The Inverse Perspective Mapping is a technique based on a geometric transformation applied on frames acquired with different point of view (either using multiple camera, or frames acquired at different time). This method belongs to the resampling effect family; an initial image is transformed to generate a view from a

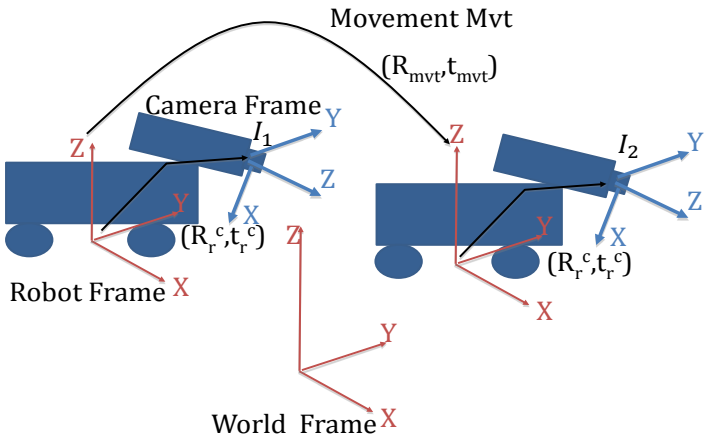


Figure 3.5: IPM method applied to robot.

different position.

Taking advantage of the perspective effect, this generated image is compared to a real image acquired from the new position, this comparison generates high differences for object sticking out of the ground plane. Detecting and localizing these differences in the ground plane allows to compute the object position relative to the camera.

In Mono Inverse perspective mapping [Botero 2012], a single camera is used, two frames are acquired at distinct instants t_n and t_{n+1}, as the robot moves. Odometry sensors are used as input to compute the homography matrix.

Homography matrix encodes the effect on the images of the relative motion of the robot between the two positions for a given plane in the scene which is the ground plane in our case. The camera is considered already calibrated; i.e., it's intrinsic parameters (focal, principal point and distortion coefficients) have been determined off line. Thanks to this knowledge, optical distortions can be removed and it is possible to consider the simple Pinhole camera model to perform IPM.
A 3D point of the scene (P_{camx}, P_{camy}, P_{camz}) in the world frame is projected to pixel coordinates (u ,v) in the pinhole image as already shown in the equation 3.9. It can be rearranged as follow:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = KR \begin{bmatrix} P_{camx} \\ P_{camy} \\ P_{camz} \end{bmatrix} + Kt \tag{3.15}$$

K is camera intrinsic matrix and (R, t) encodes the rotation and translation from the world frame to the camera frame. These former are named the camera extrinsic parameters. As IPM is intended to detect the ground pixels, the world frame which is the robot frame (see figure 3.5) is chosen such as the $P_{camx}P_{camy}$ plane is the ground plane. Therefore, for 3D points in the world frame laying in the ground plane, $P_{camz} = 0$ is applied to Eq. 3.15:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K(R \begin{bmatrix} P_{camx} \\ P_{camy} \\ 0 \end{bmatrix} + t) \tag{3.16}$$

Applying Algebraic properties to Eq. 3.16:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} P_{camx} \\ P_{camy} \\ 1 \end{bmatrix} \tag{3.17}$$

In the first acquisition, the robot frame is considered as the world frame. Therefore, each pixel coordinates are represented Eq. 3.18:

$$\begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} = K \begin{bmatrix} r^c_{r1} & r^c_{r2} & t^c_r \end{bmatrix} \begin{bmatrix} P_{camx} \\ P_{camy} \\ 1 \end{bmatrix} = H_1 \begin{bmatrix} P_{camx} \\ P_{camy} \\ 1 \end{bmatrix} \tag{3.18}$$

In the second acquisition, the position of the robot frame origin in the second acquisition is represented in the robot frame of the first acquisition Eq 3.19:

$$\begin{bmatrix} s_2u_2 \\ s_2v_2 \\ s_2 \end{bmatrix} = K \begin{bmatrix} r^c_{w1} & r^c_{w2} & t^c_w \end{bmatrix} \begin{bmatrix} P_{camx} \\ P_{camy} \\ 1 \end{bmatrix} = H_2 \begin{bmatrix} P_{camx} \\ P_{camy} \\ 1 \end{bmatrix} \tag{3.19}$$

The transformation (R^c_w, t^c_w) is computed from Eq. 3.20 as shown in the figure 3.5:

$$\begin{bmatrix} R^c_w & t^c_w \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R^c_r & t^c_r \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{mvt} & t_{mvt} \\ 0 & 1 \end{bmatrix} \tag{3.20}$$

From Eq. 3.18 and Eq. 3.19:

$$\begin{bmatrix} s_2u_2 \\ s_2v_2 \\ s_2 \end{bmatrix} = H_2H_1^{-1} \begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} = T_{ipm} \begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} \tag{3.21}$$

$$\begin{bmatrix} s_2u_2 \\ s_2v_2 \\ s_2 \end{bmatrix} = K \begin{bmatrix} r_{w1}^c & r_{w2}^c & t_w^c \end{bmatrix} \begin{bmatrix} r_{r1}^c & r_{r2}^c & t_r^c \end{bmatrix}^{-1} K^{-1} \begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} \tag{3.22}$$

Therefore, each ground point represented in the camera frame I_1 and represented with the coordinates (u_1, v_1) in the image frame will be presented in the camera frame I_2 with the coordinates (u_2, v_2) in the image frame. From Eq. 3.22:

$$H = T_{ipm} = K \begin{bmatrix} r_{w1}^c & r_{w2}^c & t_w^c \end{bmatrix} \begin{bmatrix} r_{r1}^c & r_{r2}^c & t_r^c \end{bmatrix}^{-1} K^{-1} \tag{3.23}$$

The transformation Eq. 3.23 is only correct for ground points. Therefore, the subtraction between $T_{ipm}[I_1]$ and I_2 will remove the ground points.

3.4 Obstacle segmentation

3.4.1 Gaussian filter

In the resulting image after IPM transformation, pixels of the ground plane have low absolute intensity values while pixels of objects sticking out of the ground plane have high intensity absolute values. Such image has a bimodal histogram; pixels intensities will be clustered around two well-separated values(peaks). Hence, a good threshold for separating these two groups can be found somewhere in between the two peaks in the histogram. Thanks to Otsu's algorithm which computes threshold value from a histogram of bimodal image.

Before Otsu's binarization is performed, Gaussian filter is applied to the resulting image after IPM transformation in order to remove noise. As segmentation methods are sensitive to noise, applying Gaussian Blur filter before Otsu's binarization aims to reduce noise and improve the results of Otsu's algorithm.

Gaussian Blur is a type of image-blurring filter that uses a Gaussian function. The equation of a Gaussian function in two dimensions is:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.24}$$

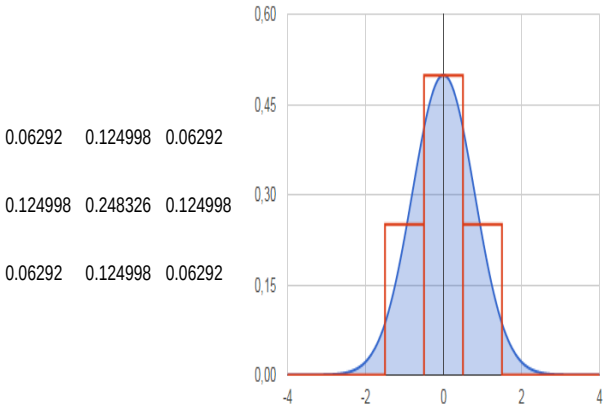


Figure 3.6: Gaussian kernel coefficients. Reprinted from Gaussian kernel calculator. Retrieved November 2015, from http://dev.theomader.com/gaussian-kernel-calculator/.

x denotes the distance from the origin in the horizontal axis, y denotes the distance from the origin in the vertical axis. σ being the standard deviation of the Gaussian distribution. This equation produces a surface whose contours are co-centric circles representing the Gaussian distribution from the center point.

Gaussian smoothing is done by convolution. Since the image is stored as a collection of discrete pixels, a discrete approximation kernel to the Gaussian function is produced before executing the convolution. The kernel weights are renormalized. Thus, the sum of all weights is one. The heaviest weight having the highest Gaussian value is given to the original pixel while smaller weights are given to neighbouring pixels.

The kernel used in the proposed system is a 3×3 kernel whose standard deviation is $\sigma = 0.8$. Figure 3.6 shows the weights of such kernel.

3.4.2 Otsu's binarization

In obstacle detection system, one of the important steps to extract obstacle pixels is the segmentation of binary image and thresholding is a fundamental tool for segmentation. Otsu's thresholding [Otsu 1979] is known as a good method, this method finds the optimal threshold by minimizing the mean square errors between original image and the resultant binary image. In order to segment obstacle points,

the image resultant from the Gaussian smoothing is segmented into two regions: obstacle objects and background region. In general, the gray-level histogram is normalized and regarded as probability distribution Eq. 3.25:

$$p_i = \frac{n_i}{N} \tag{3.25}$$

where N is the total number of pixels and n_i is the number of pixels at level i, the total number of gray levels is L. Pixels are classified into two classes background and obstacle objects by a threshold at level k. The class probability $\omega_{0,1}$ is computed as follows:

$$\omega_0 = \sum_{i=1}^{k} p_i, \qquad \omega_1 = \sum_{i=k+1}^{L} p_i \tag{3.26}$$

The class mean $\mu_{0,1}$ is then expressed as follows:

$$\mu_0 = \sum_{i=1}^{k} \frac{ip_i}{\omega_0}, \qquad \mu_1 = \sum_{i=k+1}^{L} \frac{ip_i}{\omega_1} \tag{3.27}$$

A threshold based on Otsu's algorithm is computed from Eq. 3.26 and Eq. 3.27:

$$\sigma_0^2(t) = \sum_{i=1}^{k} [i - \mu_0(t)]^2 \frac{p_i}{\omega_0} \tag{3.28}$$

$$\sigma_1^2(t) = \sum_{i=k+1}^{L} [i - \mu_1(t)]^2 \frac{p_i}{\omega_1} \tag{3.29}$$

The desired threshold corresponds to the minimum value of the weighted within class variance produced by Eq. 3.28, and Eq. 3.29. In practice, this problem of searching the optimal threshold can be reduced to search a threshold that maximizes the between-class variance which is computed as follows:

$$\sigma_B^2 = \omega_0 \omega_1 (\mu_1 - \mu_0)^2 \tag{3.30}$$

For quicker calculation and optimal performance in hardware implementation the equation 3.30 is used to find Otsu's threshold from the histogram extracted from gray-level image.

3.4.3 Morphological operators

In morphological image processing, erosion is one of two basic operators which are typically applied to binary images. When erosion operator is applied to a binary image, the boundaries of regions of foreground pixels (i.e. white pixels) are eroded away. Hence, areas of foreground pixels shrink in size, and holes within those areas become larger. Erosion operation is mathematically defined as follows [Fisher 2000]:

"Suppose that X is the set of Euclidean coordinates corresponding to the input binary image, and that K is the set of coordinates for the structuring element. Let Kx denote the translation of K so that its origin is at x. Then the erosion of X by K is simply the set of all points x such that Kx is a subset of X".

The erosion operator takes two inputs. The first input is the image which is to be eroded. The second input is a set of coordinate points known as a structuring element. The precise effect of the erosion on the input image is mainly determined by the structuring element.
Let the structuring element to be a 3×3 square with the origin at its center. Foreground pixels are represented by 1, and background pixels by 0.
To compute the output image by this structuring element, the structuring element is superimposed on top of the input image for each foreground pixel so that the origin of the structuring element corresponds to the input pixel coordinates. For every pixel in the structuring element, if the corresponding pixel in the image underneath is a foreground pixel, then no change is applied to the input pixel. On the other hand, if any of the corresponding pixels in the image is background, the input pixel is set to background value.

In our system, erosion is applied to the binarized image resulting from Otsu's algorithm. it uses a kernel of 3×3. Figure 3.7 shows the erosion operator impact using the 3×3 structure element [Fisher 2000].

3.5 Bird's eye transformation

This transformation is used in the state of art of IPM method for obstacle detection [Bertozzi 1998b]. Bird's eye transformation allows the distribution of obstacle information in all pixels and leads to an efficient implementation of polar histogram in order to localize obstacles.

The binarized image is projected on the ground plane in the robot frame as depicted in figure 3.8, up to a rotation around the vertical axis and a translation in XY. C_1 and C_2 represent camera frame as shown in figure 3.8. Any point on the ground plane P has a 3D position represented with respect to the camera $C_1$is r_{C1} [Hartley 2004] Eq. 3.31:

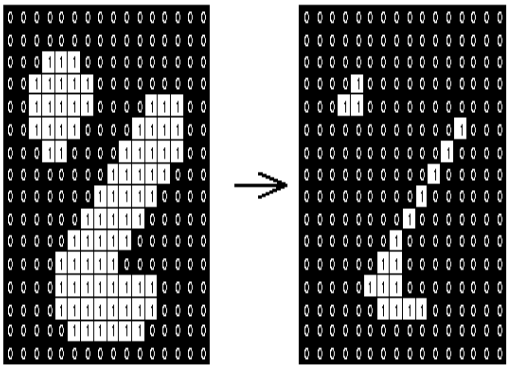


Figure 3.7: Erosion effect using a kernel of 3×3. Reprinted from [Fisher 2000].

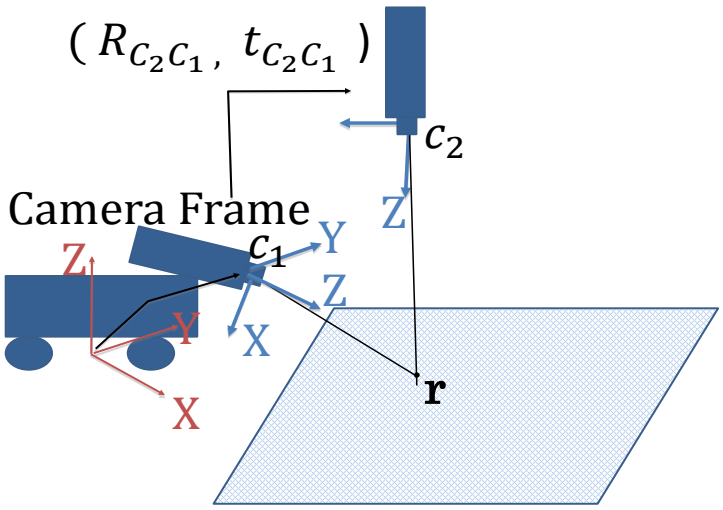
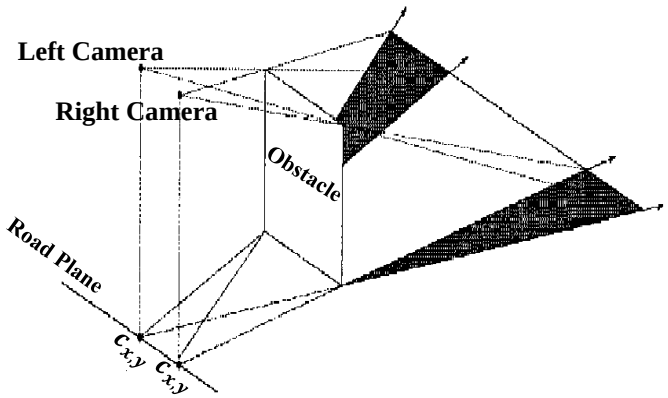
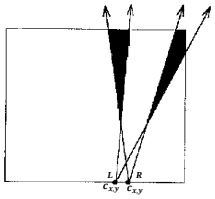


Figure 3.8: Bird's eye view transformation.



(a) Perspective view.



(b) Road plane view.

Figure 3.9: Relationships between obstacle and cameras position. Reprinted from "GOLD: Real time Stereo vision system for generic obstacle and lane detection" [Bertozzi 1998a]

$$\frac{n_{C_1}^T . r_{C_1}}{d_{C_1}} = 0 \tag{3.31}$$

n_{C_1} is the ground plane normal represented in camera C_1 coordinates and d_{C_1} is the distance of the ground plane from the origin of camera C_1,the position vector r_{C_2} of the same point represented in camera C_2 Eq. 3.32:

$$r_{C_2} = R_{C_2C_1}(r_{C_1} - t_{C_1}^{C_2C_1}) \tag{3.32}$$

where $R_{C_2C_1}$ is the rotation matrix from C_1 to C_2 and $t_{C_1}^{C_2C_1}$ is the translation from C_1 to C_2 presented in C_1 coordinates.

The required transformation from the original Camera C_1 to the virtual camera C_2 presenting bird's-eye view is Eq. 3.33:

$$r_{C_2} = H_{C_2C_1} \ r_{C_1} \tag{3.33}$$

$$H_{C_2C_1} = R_{C_2C_1} - \frac{1}{d_{C_1}} t_{C_2}^{C_2C_1} . n_{C_1}^T \tag{3.34}$$

By using Eq. 3.34, the homography matrix of bird's-eye view for the cameras is calculated. To use this matrix in image coordinates(pixels), camera intrinsic matrix K is required as shown in the equation (15):

$$H_{bird} = K(R_{C_2C_1} - \frac{1}{d_{C_1}} t_{C_2}^{C_2C_1} . n_{C_1}^T) K^{-1} \tag{3.35}$$

and the bird's eye image of the ground plane is generated.

3.6 Obstacle localization

Obstacles localization using stereo IPM is introduced in [Bertozzi 1998b]. The goal of this process is to determine the free space around robot without a complete recognition of obstacles. As depicted in figure 3.9, bird's eye operation transforms vertical edges of obstacles to lines passing through the projection $C_{x,y} = (l, d, 0)$ of the camera center on the road plane $z = 0$.

As the system used in [Bertozzi 1998b] is based on stereo IPM, two projection centers are obtained in the plane $z = 0$. The resulting image after IPM transformation in [Bertozzi 1998b] produces two triangle shapes of obstacle projected on the ground plane as depicted in figure 3.9. The triangle prolongation edges intersect at the projections $C_{x,y}^L$, $C_{x,y}^R$.

The midpoint point namely $focus$ is halfway between the two projections $C_{x,y}^L$, $C_{x,y}^R$. As polar histogram is used to detect the triangles in the produced image after applying IPM method in [Bertozzi 1998b], $focus$ point is used as the origin of polar histogram beam. The polar histogram is performed by a beam of lines originating from $focus$ and counting the number of over-threshold pixels per line.

As IPM method produces two triangle shapes of one obstacle as shown in figure 3.9, two peaks are produced in polar histogram. The position of peaks in the produced polar histogram refers to the bearing of obstacles. Peaks also have other characteristics such as width, amplitude, and sharpness.

3.7 Free space detection

From bearing measurements already performed, a polar histogram is produced. Peaks in this histogram represent the detected obstacles with respect to its orientation. An ideal obstacle shape with quasi vertical edges must produce two peaks. In real conditions, an obstacle produces two or more disjointed peaks. The goal of free space detection process is to address the detected peaks to their obstacles.

In practice, gaps are produced between peaks in the polar histogram. Some of these gaps represent a free space of a robot environment, while others represent the occupation of obstacle object in the space. Therefore, if two or more peaks are mapped to the same obstacle, the gaps between these peaks represent the occupation of obstacle in the space, while other gaps represent free space.

In figure 3.10, A_1 and A_2 are computed for every two sequential peaks in polar histogram. The ratio between A_1 and A_2 represents the depth of the valley between two sequential peaks. In most cases, the deeper is the valley between two peaks, the higher certitude that these two peaks belong to two obstacles and vice versa. Therefore, the ratio between A_1 and A_2 is compared to a threshold in order to determine whether these two peaks belong to one obstacle or not.

In algorithm 1, *polar* represents the extracted polar histogram (one-dimensional array of size l). Matrix indexes represent the bearings while matrix values represent the number of over-threshold pixels in a specific bearing.

Data: $polar[l]$
Result: A_1, A_2
$dens_1 = polar[0]$;
$i_1 = 0$;
for $i = 0 \rightarrow l - 1$ **do**
  if $polar[i] > polar[i-1]$ *and* $polar[i] > polar[i+1]$ **then**
    $i_2 \leftarrow i$;
    $dens_2 \leftarrow polar[i]$;
    $step \leftarrow (i_2 - i_1)/2$;
    for $j = i_1 \rightarrow (i_1 + step)$ **do**
      $A2_1 \leftarrow A2_1 + (dens_1 - polar[j])$;
      $A1_1 \leftarrow A1_1 + polar[j]$;
    end
    for $j = i_2 \rightarrow (i_2 - step)$ **do**
      $A2_2 \leftarrow A2_2 + (dens_2 - polar[j])$;
      $A1_2 \leftarrow A1_2 + polar[j]$;
    end
    $A1 \leftarrow A1_1 + A1_2$;
    $A2 \leftarrow A2_1 + A2_2$;
    $i_1 \leftarrow i$;
    $dens_1 \leftarrow polar[i]$;
  else
  end
end

Algorithm 1: Free space detection based on an extracted polar histogram

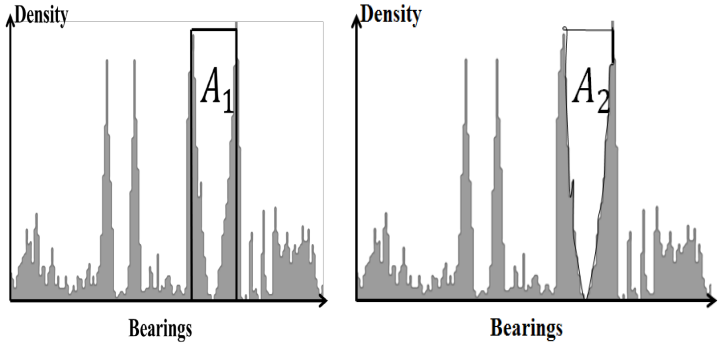


Figure 3.10: A_1, A_2 presentation in Polar histogram.

3.8 Obstacle distance

Polar histogram computed in section 3.6 and free space detection results are used to compute the distance of detected obstacles. Peaks produced in polar histogram are analysed in order to determine the localization of obstacle in the ground plane. For each peak detected in a specific bearing of the polar histogram, a radial histogram is computed by scanning this specific bearing in the resulting image from IPM. This computed histogram presents the number of over-threshold pixels with respect to distance from $focus$. As pictured in figure 3.11, the width a_i of the sector is determined as the width of the polar histogram peak. The radial histogram is normalized. A threshold is applied to the result, thus allowing the determination of the obstacle distance through a simple threshold.

3.9 Conclusion

In this chapter, the methodology of the proposed system was presented. The different algorithms that will be used to design the proposed system are introduced. These algorithms were already presented in the state of the art. Some of these works were done in the RAP team of LAAS. IPM method is adopted for obstacle detection. A Binarization based on Otsu's algorithm is used for obstacle segmentation. The state of the art of obstacle localization was also introduced. The proposed system is based on these algorithms to achieve the objectives of this thesis; a real-time system with a reduced computational time and a high frame-rate with low consumption of power.

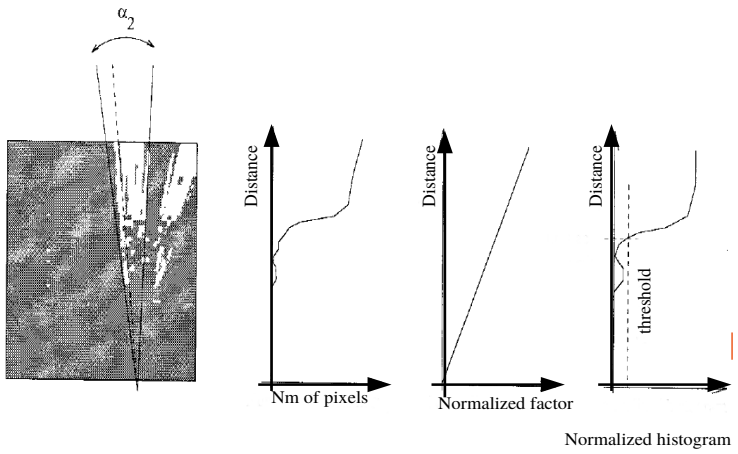


Figure 3.11: Steps for the Computation of normalized radial histogram for a peak (Reprinted from [Bertozzi 1998a]).

Chapter 4

Embedded Systems Platforms and Design

Contents

4.1 Introduction

Most computers in use today are General-Purpose computers. These computers are expected to perform a wide variety of tasks and must run a large number of different programs to accomplish these tasks. In contrast to a general-purpose computer, special-purpose computers are designed to meet specific requirements and most of the times their job is to solve one particular problem. The capability to solve a particular problem depends not only on the stored program but also on the design of the computer system [Harris 2015].

Embedded systems, special-purpose systems, are dedicated to perform single task over and over again and they are designed in order to meet only the needs of task. The design of this type of systems includes not only the software methodology but also the hardware architecture. An embedded system designer can control both

hardware and software development, including processor architecture, memory system design, compiler, and operating systems. Hence, great improvements (in terms of cost, size, and performance) can be made due to this flexibility in system design [Lutkebohle 2008] [Barr 2003].

Embedded systems often face real-time constraints. However, maintaining an efficient real time performance when executing complex algorithms is a traditional problem in embedded systems. This issue can be addressed using parallel programming methods and parallel computing platforms. Parallelism has been widely employed, mainly in high performance computing. Different options are available in order to build HPEC (High Performance Computing) systems using parallel computing platform. One can find General-purpose computing on graphics processing units (GPGPU), Application-specific integrated circuits (ASICs), multi-core processing systems, and Reconfigurable computing with field-programmable gate arrays (FPGAs).

In this chapter, an insight is given into parallel processing platforms which can be used to optimize the embedded performance when implementing image processing algorithms. As our proposed architectures are based on FPGAs targets, the common architecture of FPGAs and its properties are described.

4.2 General purpose Processing on Graphical Processing Unit (GPGPU)

Graphics processing units (GPUs) were introduced as special purpose accelerators for video games. These units are now used for high performance computing (HPC) applications in systems ranging from embedded systems to massive supercomputers. This has led to the emergence of GPGPUs field (General purpose Processing on Graphical Processing Unit).

In GPUs architecture the ALUs silicon area is expanded and the scheduling logic area that is used for instruction-level parallelism in CPUs is removed in GPUs. A thread-level parallelism is used to hide latency in GPUs, executing up to hundreds threads at once with each CPU. Threads execute in batches. Each batch consists of 32 threads called a warp. Each thread within a warp can be enabled or disabled independently, allowing to execute different parts of the program. The number of operations which are executed per cycle depends on the number of threads in an active warp. That is why GPU batching induces an important cost in terms of minimising thread divergence, which means that all threads take the same branch of conditional statements, and execute loops the same number of times. [Thomas 2009].

The use of multiple GPUs has different advantages [Fung 2004]:

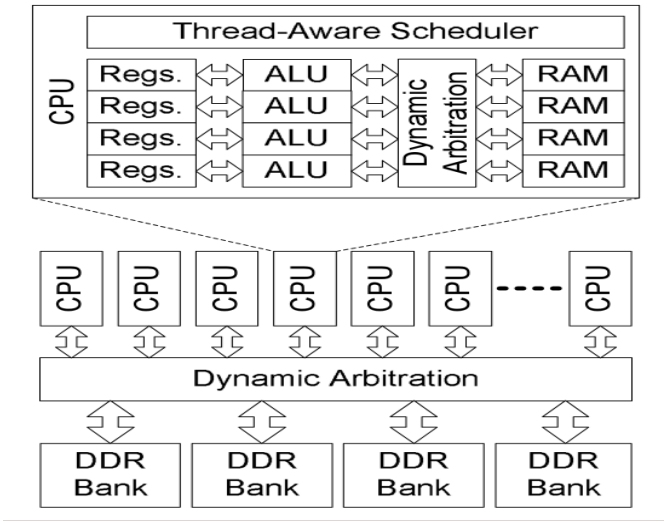


Figure 4.1: GPU architecture. Reprinted from "A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation" in [Thomas 2009].

- Each graphic card has its own RAM. Thus, they can access their memories in parallel which increases the overall memory bandwidth of the system [Fung 2004].

- GPUs do not contend with each other for access to a shared memory area. The fragment programs are stored locally on each graphics card, thus they can run relatively independently, requiring little supervision from the CPU.

GPU cores are native hardware floating-point processors. A GPU core can run hundreds of floating-point math operations every clock cycle, making GPUs a natural fit for floating-point-intensive signal and image processing applications [Adams 2014]. In addition, GPUs also provide good backward compatibility. If an algorithm changes, the new software can run on older chips [Adams 2014]. On the other hand, GPUs are regarded as power hogs. In contrast to hardware architecture based on FPGAs, GPUs execute algorithms in software. Instructions have to be fetched, results must be sent to memory...etc. GPUs based architectures need few inter-thread data dependencies and little data-dependent control.

4.3 Multi-Core CPUs

A multi-core processor is a processor including multiple processing units (attached for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks) on the same die. As single-core processors rapidly reaches the physical limits of possible complexity, the multi-core processing trends grows. The main difference between a single-core processor and

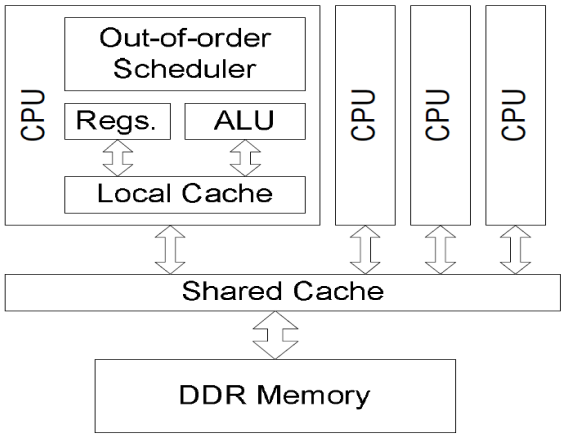


Figure 4.2: CPU architecture (taken from [Thomas 2009]).

multi-core processor is that a multi-core processor can execute multiple instructions at a time instead of executing one instruction stream in a single-core processor.

Most multi-core CPUs operate in the same way as single-processor. These cores communicate via a shared memory with a synchronization performed using a cache memory as shown in figure 4.2. One thread is executed by each core. The state of thread is stored using a set of registers [Thomas 2009]. A dedicated ALU is devoted to each thread, containing a number of functional units. A large unit is used for scheduling tasks, such as branch prediction, instruction ordering, speculative execution.

Multi-core processors are widely used across many applications in embedded systems. The improvement in performance produced by the use of a multi-core processor depends very much on the software algorithms used and their implementation. In particular, possible gains are limited by the fraction of the software that can run in parallel simultaneously on multiple cores. In the best case, speedup factors may be achieved near the number of cores. Even if the algorithm is split up to fit within each core, most applications are not accelerated so much.

The ARM Cortex A9 (a multi-core processor [ARM 2016]) is a popular choice for low-power cost-sensitive applications. Cortex-A9 processors are proven to offer highly effective outcomes in embedded systems. Additionally, they are able to implement multi-core designs that further scale the performance increase.
Compared to other processors, Cortex-A9 processor has different advantages [ARM 2016]:

- scalable performance and power efficiency for a wide range of applications.

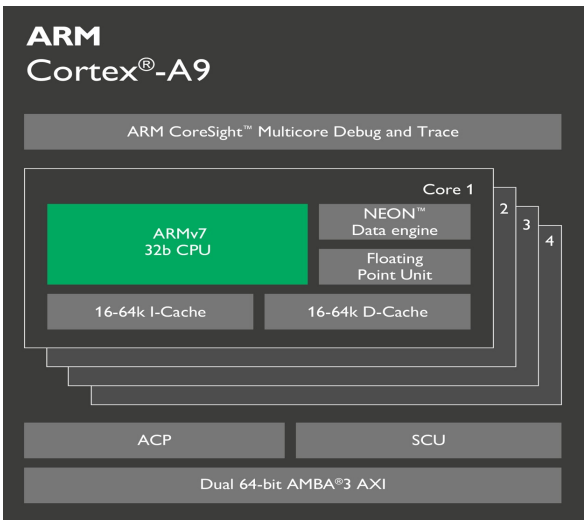


Figure 4.3: ARM Cortex A9 architecture. Reprinted from "Cortex-A9 Processor" in [ARM 2016].

- As Cortex-A9 is based on ARMv7-A architecture, the following benefits can be obtained:
  1. Dynamic length pipeline (8-11 stages).
  2. Highly configurable Level-1 (L1) caches.
  3. NEON technology can be implemented.
  4. Scalable multi-core configuration with up to 4 coherent cores.
- Support the wide 32-bit software eco-system.

4.4 Application-Specific Integrated Circuit

An ASIC is a microchip that is customized for a special application, rather than intended for general-purpose use. Mixed signal ASIC designs can incorporate both analog and logic functions. These mixed signal ASICs are particularly useful to build a complete system on chip (SoC). ASIC chips are designed in the following conceptual stages, although these stages overlap significantly in practice [Kommuru 2009]:

- To design a chip, an idea is required, and the first step to make the idea into a chip is to come up with the **Specifications**.
- The next step in the flow is to come up with **structural and Functional Description**, which means that a designer has to decide what kind of architecture to be used for the design.

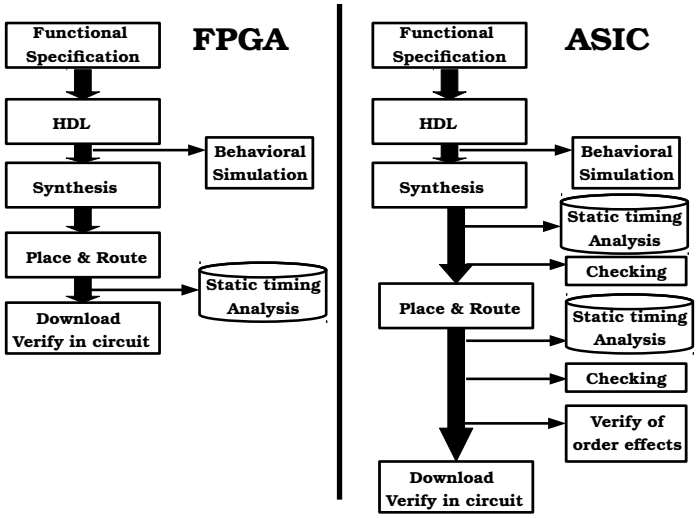


Figure 4.4: ASIC vs FPGA design flow (taken from [XILINX 2012a]).

- The sub systems are implemented using logic representation, finite state machines, combinatorial and sequential logic, schematics. This step is called Logic Design **Register Transfer Level**.

- Logic/RTL synthesis is done by synthesis tools such as Design Compiler (Synopsys), Blast Create (Magma). which takes an RTL hardware description and a standard cell library as input and produces a **Gate-Level Netlist** as output.

- The next step is the **Physical Implementation**. The Gate-Level Netlist is converted into geometric representation.

- The file produced by the previous step is the **GDSII** file which is used by the foundry to fabricate the chip.

Product cost can be reduced when ASIC technology is used. For low volume production, ASIC solutions are not efficient in terms of price. That's why ASICs are dedicated to meet special requirements of applications such high-performance or a high volume production.

An ASIC design needs more time to market and a harder design cycle when it is compared to FPGA design flow (Field Programmable Gates Array). In addition, ASICs are not reprogrammable while FPGA based products are reconfigurable designs. The ASIC design flow requires an important time-consuming floor-planning, place and route, and timing analysis On the other hand, an FPGA design flow can eliminate the previous cycles.

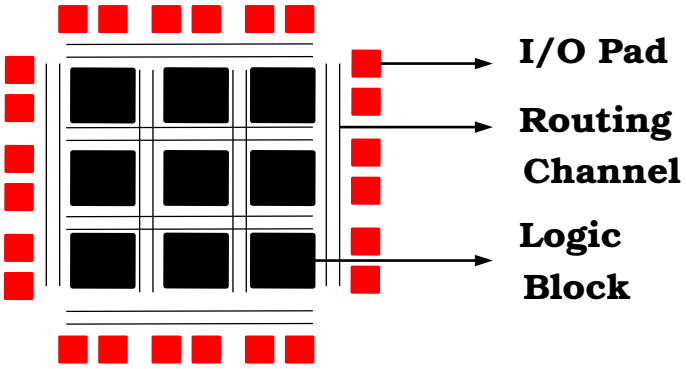


Figure 4.5: Basic FPGA structure.

4.5 Field Programmable Gate Array (FPGA)

Field programmable gate arrays (FPGAs) are digital integrated circuits (ICs) that include configurable blocks of logic along with configurable interconnects between these blocks (as pictured in figure 4.5). The **field programmable** portion of the FPGA's name denotes that the programming is done "in the field" in contrast to devices whose internal functionality is hard-wired by the manufacturer.
An FPGA can be used to solve any problem which is computable. FPGAs advantage lies in that they are significantly faster for specific applications due to their parallel nature and optimality in terms of the number of gates used for a specific process.

The core logic block from Xilinx is called a logic cell (LC) (shown in figure 4.6). An early LC is composed of : a 4-input LUT (Look Up Table), a multiplexer, Digital Flip-Flop (DFF) [Max-Maxfield 2004] (see Figure 4.6). The main role of the LUT is to design combinatorial functions. In addition, some vendors allow the cells forming LUT to be employed as a block RAM. This is referred as *distributed* RAM. LUT can sometime be formed implement shift registers. Hence, LUTs may be regarded as multifaceted. Most FPGAs also embed additional functions such as memory blocks or DSP blocks.

Nowadays, FPGA structure differs from a vendor to another. The most popular vendors of FPGAs are Xilinx, Altera, Lattice, Actel and Atmel. In Xilinx FPGA family, a slice includes two logic cells. A CLB (Configurable Logic Block) is divided into two slices [XILINX 2014a]. Each slice can be of two types :

- A SLICEL(L= logic) logic slice contains four 6-input LUTs, four D-latches, four DFF/D-latches and eight muxes with a strong local interconnect.

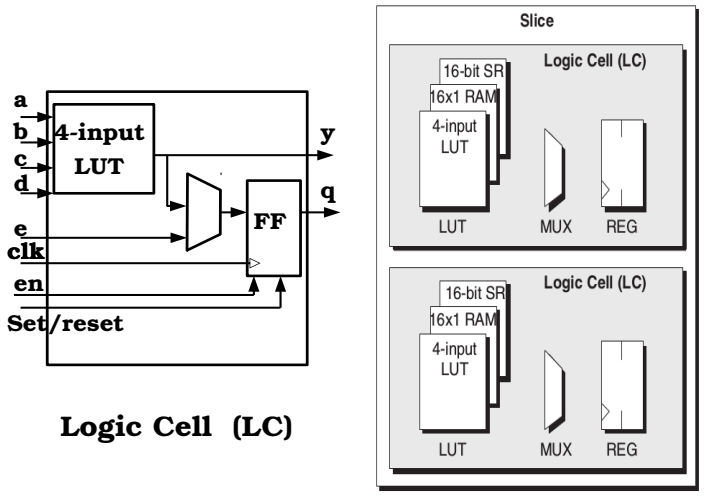


Figure 4.6: Left: Early structure of LC in Xilinx, Right: Slice having two LCs (taken from [Max-Maxfield 2004])

- SLICEM (M= memory) logic slices contains 6-input LUTs whose behavior can be configured to implement distributed memory or shift registers instead.

As many applications require the use of memory, FPGAs now contain relatively large chunks of embedded RAM called block RAM. In most architectures, these blocks are located at the periphery of the chip, scattered across the face of the chip in relative isolation, or organized in columns as shown in figure 4.8.
Multiple blocks of RAM can be combined together to implement larger blocks, or each block can be used independently. These blocks are implemented as standard single- or dual-port RAMs, first-in first-out (FIFO) functions, state machines etc...

Since some functions, like multipliers are required by a lot of applications, FPGAs incorporate special hard-wired multiplier blocks. These are typically located in close proximity of the RAM blocks.

The FPGA configuration is specified using a Hardware Description Language (HDL). This description is then translated into a connected set of logic elements (LUT, Mux, DFF) that is then placed on the FPGA existing structure and routed using the FPGA interconnect matrix. This process is usually refered-to as synthesis and result in a bitstream that can be loaded in the FPGA configuration memory.

4.6 FPGA Designs Properties

A circuit implemented on a FPGA has different characteristics such as latency, frequency, area, and power. FPGA designs are compared to other designs in terms of:

- Frequency

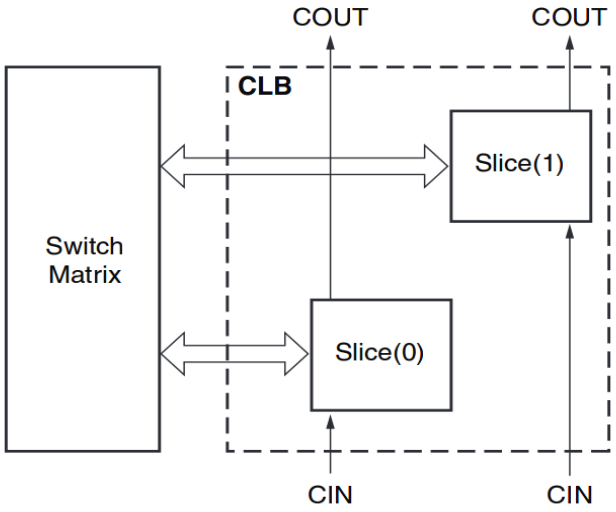


Figure 4.7: CLB containing two slices. Reprinted from [XILINX 2014a].

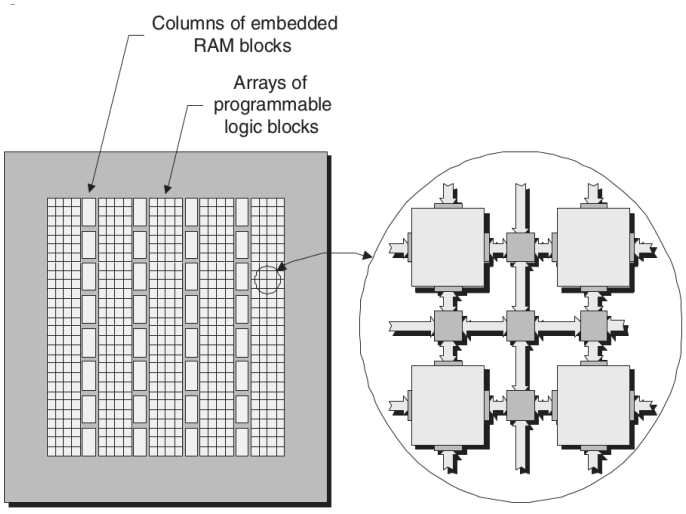


Figure 4.8: Top-down view of an FPGA chip with columns of blocks RAM. Reprinted from [Max-Maxfield 2004].

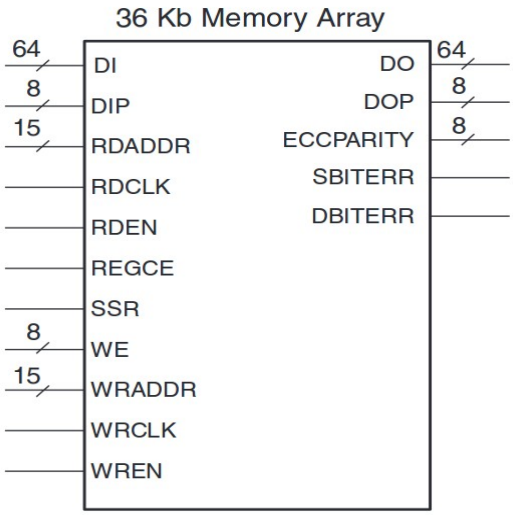


Figure 4.9: Simple Dual port RAM (taken from [XILINX 2014b]).

- Latency
- Cost and size
- Power consumption

4.6.1 Pipelined Designs

A pipeline is a systolic array where all data flow goes in one direction and there is no feedback [SCHMIT 2000]. In signal processing, a pipeline is a set of stages which are connected sequentially, the output of one stage is an input of the next. In synchronous circuits, a pipeline is characterized by a register-to-register delay path. This determines the maximum clock speed of a circuit. As pictured in figure 4.10, the normal design is implemented by connecting 3 multipliers in a cascaded fashion with a flip flop at the end stage while each multiplier output is connected to a flip flop in the pipelined design. The additional DFFs reduce the delay through the combinatorial logic. As result, pipelined design can operate at a higher frequency than the normal design at the cost of computing latency.

When one wants a design to reach the maximum speed, it requires to write a pipelined code. Pipelined code is harder to write since the designer must take into account synchronization between stages and may rethink the original application structure. In large projects, pipelined designs are required to optimize blocks that are considered as a bottleneck for the performance of the whole design.

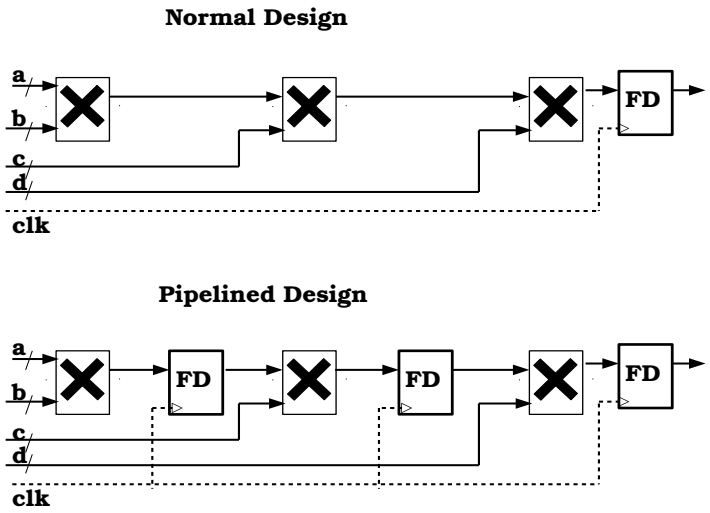


Figure 4.10: An example of pipelined design for $a \times b \times c \times d$.

4.6.2 Designs with reduced latency

Latency is defined as time elapsed to generate the output data for a given input data. In the context of an electronic system, latency is the time (clock cycles) it takes for a data to work its way through a device, or system. This parameter is computed by the product of the cycle time by the number of pipeline stages.
FPGAs are considered as a popular choice for applications that requires reduced latency [Morris 2009]. In practice, putting an algorithm in an FPGA based architecture shortens the trip of a signal to be produced. While latency takes a range between microseconds and many seconds in a software development, it is in the range of hundreds of nanoseconds in the hardware fabric.

4.6.3 Power consumption consideration

Power consumption is becoming an increasingly important variable, and it has always been a design consideration. FPGAs are a popular choice for reducing power consumption, and they can significantly aid the designer in reducing the challenges associated with power consumption. An FPGA design is impacted by four power components introduced as follows [Lattice 2009]:

- Pre-programmed static (Quiescent) device power consumption is the power consumed when a FPGA device is in a non-programmed state, already been powered.
- Inrush Programming Current represents the required power in order to program the FPGA device.
- Post-Programmed Static Power Consumption represents the power consumed with 'zero Mhz' frequency.

- Dynamic power consumption is the power consumed by a non-zero frequency module.

FPGAs vendors provide software tools to estimate the power consumption of a design. For example, Xilinx Power Estimator (XPE) tool can provide a detailed power and thermal information [XILINX 2016c], including estimated power breakdown between static and dynamic power, power consumption by device resource type (I/O, logic), and the total power consumption.

4.6.4 Cost and Size Requirements

FPGAs offer cost-effective designs due to their generic nature with different advantages of short time-to-market, no NRE (No Recurring Engineering) costs, reduced bill of materials (BOM) costs [Parnell 2004]. Nowadays, FPGAs have driven down die size and package costs in order to produce convenient custom processor alternative. The advanced capabilities of modern FPGAs allow designers to integrate discrete component functions into an FPGA reducing board and total component costs.

The Zynq-7000 All Programmable SoCs (Z-7010, Z-7015, and Z-7020) is a member of Xilinx Low-end Portfolio [XILINX 2015b]. This devices family is dedicated to meet cost-sensitive market requirements. The main advantage of Zynq-7000 is that it integrates the software programmability of an ARM-based processor with the hardware programmability of an FPGA, enabling hardware acceleration while integrating mixed signal functionality on a single device.

4.7 Zynq-7000 all programmable SoC

One challenge of today's technology is the capability to provide a design that combines different requirements, including low total cost, higher performance, flexible, low system power, and short time-to-market. Zynq based kits become a popular choice to achieve programmable systems integration where hardware and software (ARM Programmability + FPGA Flexibility in a Single Chip) can be programmed, creating a custom flexible SoC to meet the exact project requirements.
The system performance of a design based on Zynq kit can be increased due to the dual Core ARM Cortex A9's with NEON and vector floating point, the programmable logic with massive DSP processing, and the high throughput AMBA-4 AXI interconnect for fast data transfers. Hence, a hardware acceleration scales software performance in order to address different functionalities.

Another advantage of choosing Zynq based kit is the BOM cost reduction (as pictured in figure 4.12). As processors, PLDs, DSPs are all available on Zynq kit, one can reduce the number of devices per board and the PCB complexity

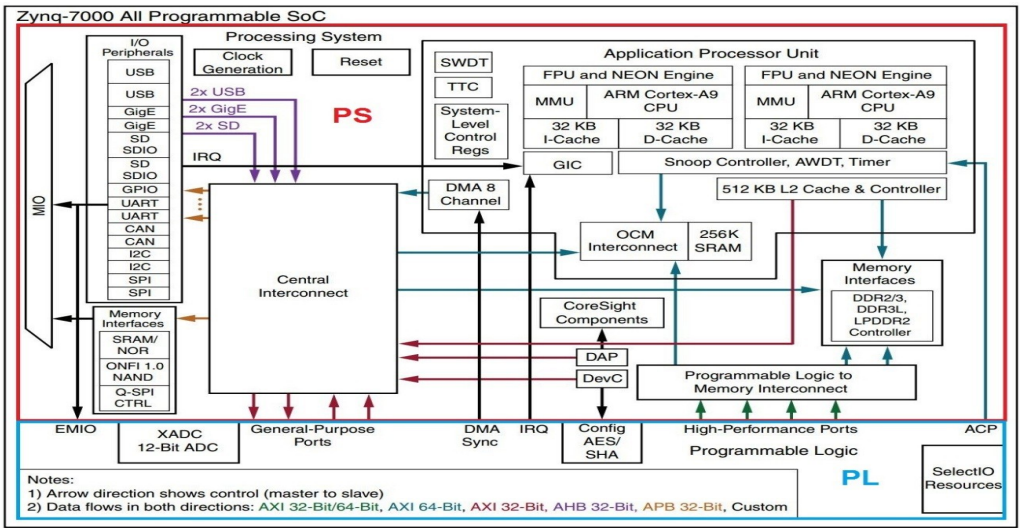


Figure 4.11: Zynq-7000 architecture. Reprinted from [XILINX 2015c].

Table 4.1: Some features of the Zynq-7000 [XILINX 2016b]

Zynq-7020		
	Part	Amount
Processing System	Processor Core	Dual core ARM Cortex-A9
	Processor Extension	NEON
	Maximum Frequency	667 Mhz
	L1 Cache	64 KB
	L2 Cache	512 KB
	On chip Memory	256 KB
	External Memory Support	DDR3, DDR2
	Peripherals	UART, CAN, I2C, SPI
Programmable Logic	LUTs	53200
	Flip-Flops	106400
	Extensible Block RAM	560 KB
	Programmable DSP Slices	220
	AXI Master	$2 \times 32 - b$
	AXI Slave	$2 \times 32 - b$
	AXI Memory	$4 \times 64 - 32b$
	Interrupts	16

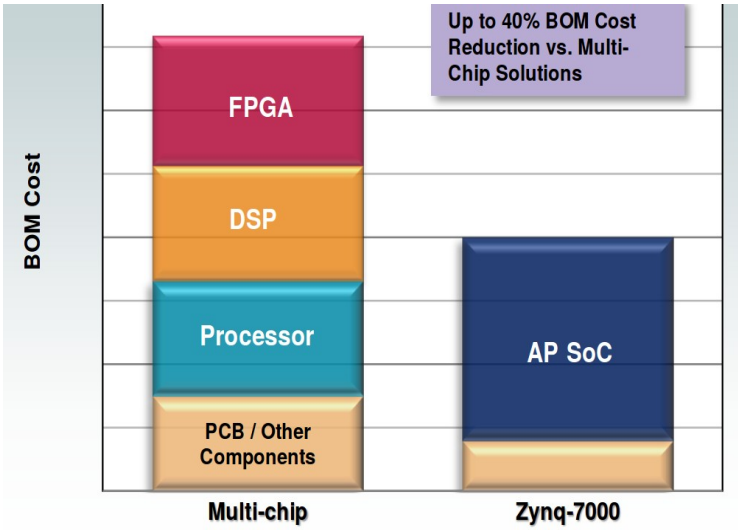


Figure 4.12: BOM cost reduction of Zynq-7000. Reprinted from [XILINX 2012b].

[XILINX 2012b]. In terms of power requirments, a Zynq based design is considered as flexible/tunable power envelope, adjusting the ARM processor speed, enabling low power states of ARM, and partial reconfiguration to reduce the amount of hardware resources.

4.8 Xillybus IP Core

Xillybus [Xillybus 2016c] is a DMA-based end-to-end turnkey solution for transporting signals between FPGA and a host running operating system (Linux, Microsoft Windows). Hardware and software designers interact with Xillybus through defined interfaces. In the hardware interface, the FPGA logic application is connected to Xillybus core through standards FIFOs [Xillybus 2016b]. In the software interface, the user performs file I/O operations using device files. Hence, the data is transferred between FIFOs in the FPGA logic and a device file in the host operating system. As depicted in figure 4.14, A logic module implemented on FPGA only needs to communicate with FIFOs. When data is written to one FIFO, Xillybus reads this data and sends it to the host. On the other side, Xillybus transmit the data through AXI bus, using DMA requests on the ARM core's bus. An application running in the software side interacts with device files that operate like pipes.

To transmit signals from the application logic to the host, the main signals are:

- *user-data* is an input signal which contains data during read cycles. the signal's width can be 8, 16 or 32 bits.
- $rd - en$ is a read enable signal

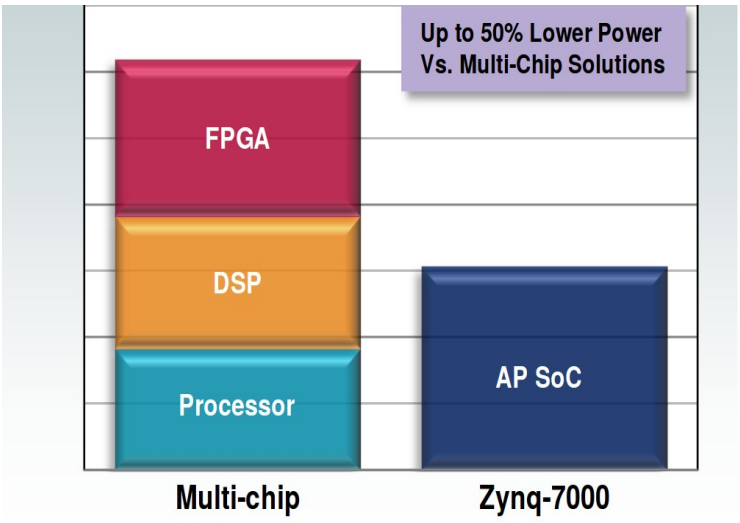


Figure 4.13: Power efficiency of Zynq-7000. Reprinted from [XILINX 2012b].

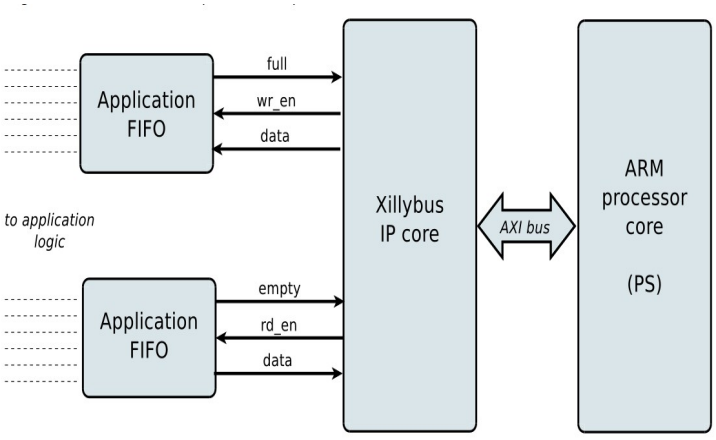


Figure 4.14: Xillybus IP Core. Reprinted from [Xillybus 2016c].

- *empty* is an input signal which informs the *Xillybus* core that no more data can be read.

To transmit signals from the host to the application logic, we mainly use:

- *user-data* is an output signal which contains data during write cycles. the signal's width can be 8, 16 or 32 bits.
- $wr - en$ is a write enable signal
- $full$ is an input signal which informs the *Xillybus* core that no more data can be written to FIFO.

Xillybus IP Core can be customized in order to satisfy application requirements [Xillybus 2016a]. Hence, designers can determine data transmission rate by fixing the number of streams to be sent. Also, data direction and other specs can be characterized. On the software side, Xillinux, a linux kernel, is a development platform used to run software applications in the ARM processing system. With basic programming skills, a complete co-deign Hw/Sw can be made due to the capabilities provided by Xillinux with its *Xillybus* core and driver.

4.9 Conclusion

In this chapter , different processing platforms used in embedded systems were introduced. Multi-core processor-based systems are widely used in embedded systems. However, the improvement in performance is limited by the fraction of the software that can run in parallel simultaneously on multiple cores. Architectures based on GPU platform provide a good pipeline performance but they don't meet power requirements. An FPGA solution can provide the best trade-off between power-consumption and pipeline for an embedded platform. As our objective is to build a real time system with reduced computational time, and high frame-rate with low consumption of power, an FPGA based solution is used in this work.

Chapter 5

Proposed Optimizations to System Methodology

Contents

5.1 Introduction

In this thesis, the main contribution is the design of hardware accelerators for machine vision applications. However, we propose some optimizations to the system methodology shown in figure 5.1. These improvements stand for obstacle localization, and the reconstruction of 2D occupancy map.

Bird's eye view transformation was used in the state of art for obstacles localization. The hardware implementation of this transformation induces a heavy cost in terms of latency time and hardware resources. Therefore, we propose to remove this module from the system architecture. Instead, we optimize polar histogram structure, and build a method for extracting contact points between obstacles objects and ground plane. By detecting quasi triangle shapes produced by IPM method, the vertex of these triangles represent contact points between obstacles and ground plane.

In the state of art, obstacle distance is measured by computing a radial histogram applied to specific bearings in the image (see 3.8). The implementation of this method induces a high level of noise, and it can not precisely determine obstacles localization in the ground plane. Additionally, the performance of the hardware implementation is costly due to the high latency induced by histogram computation. Therefore, when contact points are extracted and free space detection task is accomplished, we propose to select best points among extracted contact

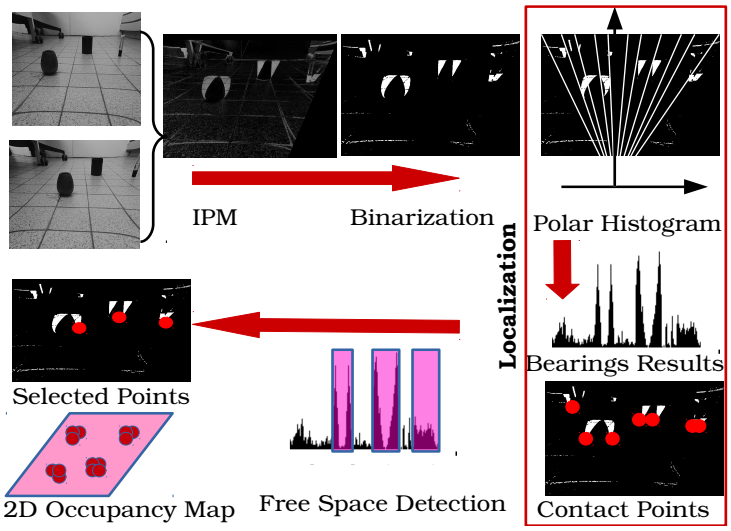


Figure 5.1: General overview of the system methodology.

points. The selected points are mapped and projected to a 2D occupancy grid map.

5.2 Obstacle bearings based on polar histogram

Obstacle clusters are produced in binarized images (see Obstacle segmentation 3.4 for details). An obstacle's vertical edges produces two quasi triangular shapes in the resulting images from IPM method [Bertozzi 1998b]. In real conditions, these disjointed shapes remain recognizable. Detecting these shapes and determining its localization in binarized images are accomplished using a polar histogram beam. This latter is defined as a beam of lines originating from a point called $focus$. The computed histogram denotes the density of obstacles with respect to orientation.

As already mentioned, typical obstacles have quasi vertical edges. These latter are straight lines perpendicular to ground plane. As these lines are parallel in the world frame, they have an intersection point at infinity plane called vanishing point. This vanishing point can be found and represented in the image frame. The intersection point between a ray parallel to vertical lines in the world frame (this ray is originating from camera center) and the image plane represents the vanishing point in the image frame as pictured in figure 5.2. π_∞ denotes infinity plane for vertical lines. This plane is represented using camera coordinate frame. X_∞ represents the intersection point of vertical lines at infinity plane. This point is also represented in the camera coordinate frame. In order to find the image of X_∞ in the image frame namely v Eq. 5.1:

$$v = PX_\infty \tag{5.1}$$

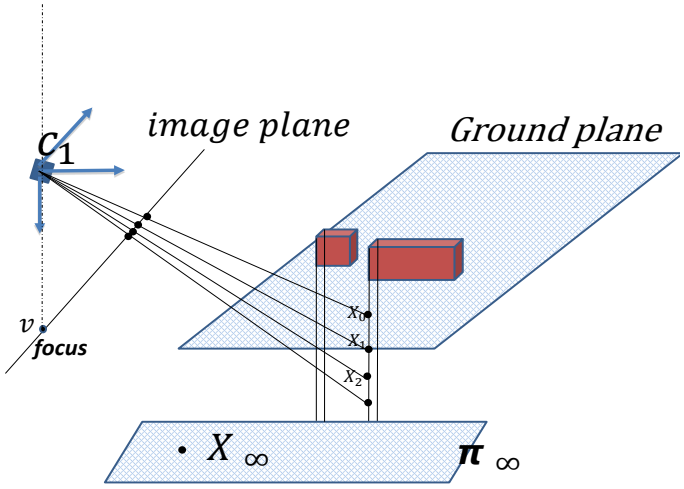


Figure 5.2: Conception of $focus$.

P denotes projection matrix from camera coordinate frame to image coordinate frame. Eq. 5.2:

$$P = K \left[I|0 \right] \tag{5.2}$$

X_∞ is represented in the camera frame using homogeneous coordinates Eq. 5.3:

$$X_\infty = \begin{bmatrix} d \\ 0 \end{bmatrix} \tag{5.3}$$

Hence, v is computed from Eq. 5.2 and Eq. 5.3 as follows:

$$v = K \left[I|0 \right] \begin{bmatrix} d \\ 0 \end{bmatrix} \tag{5.4}$$

$$v = Kd \tag{5.5}$$

v represents the vanishing point coordinates in the image frame. From $focus$ (the vanishing point), a beam of lines is made in order to compute polar histogram. Each line that exists in this beam represents a defined bearing. This latter is mapped to a specific sector in the scene. The density of obstacles located in a specific bearing is evaluated by computing the number of over-threshold pixels located in every line as shown in figure 5.3. Therefore, the peaks existing in a polar histogram represent obstacles density with respect to bearings. Hence, free space detection can be determined by identifying the available bearings where a robot can move. Bearings measurements process is a preliminary step in the proposed system and it is followed with other operations in order to maintain obstacles localization task.

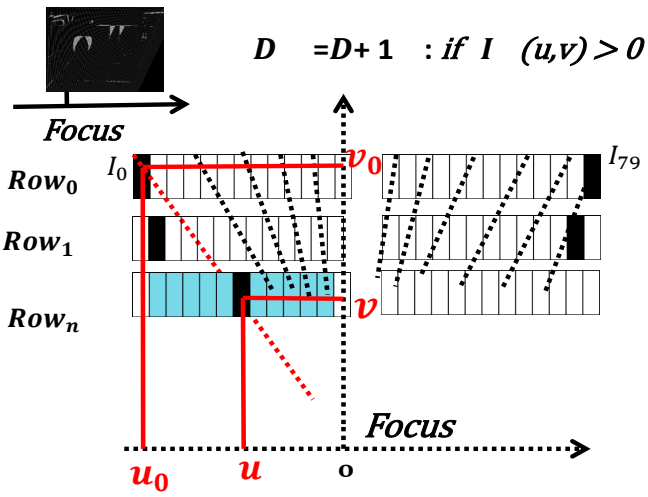


Figure 5.3: Bresenham's method implemented for a specific octant.

Data: u_0,v_0
Result: $Dens$
$dv = v_0$, $du = -u_0$, $D = 2du - dv$, $v = v_0$;
for $v = v_0 \rightarrow 0$ **do**
  if $D > 0$ **then**
    $u \leftarrow u + 1$;
    $D \leftarrow D + 2du - 2dv$;
  else
    $D \leftarrow D + 2du$;
  end
  if $I(u, v) > 0$ **then**
    $dens \leftarrow dens + 1$;
end

Algorithm 2: Bresenham's method implemented for a specific octant

As Bresenham algorithm determines the pixels that could be selected in order to form a close approximation to a straight line between two points, polar histogram beam can be made. In practice, two points coordinates are required to perform a line equation. Bresenham algorithm keeps track of one of the coordinates by maintaining an error bound. If this error becomes greater than zero, the concerned coordinate is incremented and the error is readjusted.

Each line that belongs to polar histogram beam is represented using two pixels coordinates. The first pixel exists in the first image row while the end pixel is $focus$. In addition, it is mandatory to define the octant drawer when applying Bresenham algorithm.

Algorithm 2 shows how to track pixels that belong to a line that exists in polar histogram beam. $I(u, v)$ is pixel value at the coordinates (u, v). (u_0, v_0) being the coordinates of a pixel that exists in the first image row. This pixel represents the first point of a line that exists in the octant $(u_0 < 0, v_0 > 0)$; the vertical projection $|v_0|$ is greater than the horizontal projection $|u_0|$ as depicted in figure 5.3.

The difference D computed in algorithm 2 is used to assess the error. Evaluating this latter allows to compute pixel coordinates. For example, if $I(u, v)$ is a pixel that located in a line that belongs to polar histogram beam, we can compute the next pixel coordinates as follows:

- If difference D is positive, the pixel $(u + 1, v + 1)$ is selected.
- If difference D is negative, the pixel $(u, v + 1)$ is selected.

This process is repeated until arriving $focus$. $Dens$ being the number of over-threshold pixels located in line. The implementation of the algorithm is generalized in order to produce and track all polar histogram lines in different octants.

5.3 Obstacles localization in the ground plane

The goal of this process is to extract contact points between obstacle objects and ground plane. These points represent the localization of obstacles on the ground plane. Obstacle shapes produced by IPM and binarization methods often have an isosceles triangular shape where the vertex corresponds to the contact point between ground plane and obstacle object. Detection of these points is done while polar histogram is calculated. As bearings measurements method aims to find over-threshold pixels and compute the number of these pixels located in each line that belongs to polar histogram beam, contact points extraction method is executed to find isosceles triangles crossed by polar histogram beam and extract the vertex of each isosceles triangle.

For each pixel $I(u, v)$ located in a defined line tracked by Bresengham's algorithm, a factor a is defined as a sum of neighbouring pixels located in the same image row r as shown in Eq. 5.6:

$$a_r = \sum_{k=-l}^{l} I(u + k, v) \tag{5.6}$$

$$f(a_r, a_{r+1}) = \begin{cases} a_{r+1} & \text{if } a_{r+1} \leq a_r \\ max & \text{otherwise} \end{cases} \tag{5.7}$$

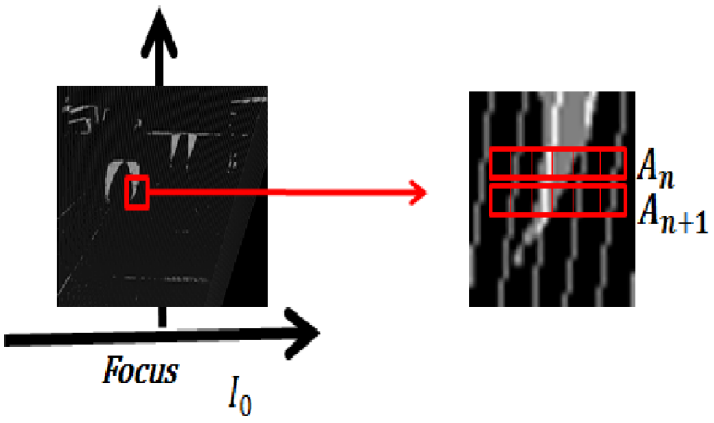


Figure 5.4: The proposed method for contact points extraction.

$$Sc = \begin{cases} 0 & \text{if } a_{r+1} > a_r \\ Sc + 1 & \text{otherwise} \end{cases} \tag{5.8}$$

max is the maximum possible value of a. l represents window width, and Sc is a score referring to the probability of the existence of an isosceles triangle. The computed score is compared to a threshold value. If the score is greater than the threshold, we decide that the line (which belongs to polar histogram beam) is crossing an isosceles triangle, and the pixel which has the minimum value of a is considered as the contact point between ground plane and obstacle object as depicted in figure 5.4.

We proposed this method in [Alhamwi 2015]. It works perfectly when over-threshold pixels form an ideal triangle as shown in figure 5.4. However, in real conditions, obstacle patterns produced by IPM method don't have a regular shape. In addition, binarized images may include some noise, false obstacles are segmented, especially when an indoor environment has floor tiles. Tiles edges don't disappear completely after binarization. Hence, if some tiles edges are parallel to the lines (that belong to polar histogram beam), false contact points are induced. Therefore, an optimization is proposed to contact points extraction method in order to deal with real cases.

$$a_r = \sum_{k=-l}^{l} I(u+k, v) \tag{5.9}$$

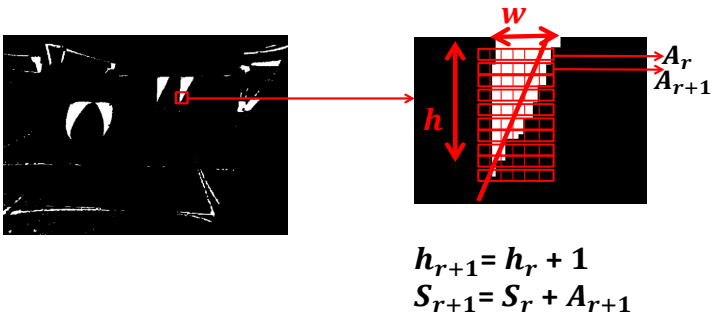


Figure 5.5: The optimized method for contact points extraction.

$$f(a_r, a_{r+1}) = \begin{cases} a_{r+1}, & \text{if } a_{r+1} \leq a_r \\ max, & \text{otherwise} \end{cases} \tag{5.10}$$

$$S_{r+1} = f(S_r, a_{r+1}, a_r) \tag{5.11}$$

$$f(S_r, a_{r+1}, a_r) = \begin{cases} S_r + a_{r+1}, & \text{if } a_{r+1} \leq a_r \\ S_r, & \text{otherwise} \end{cases} \tag{5.12}$$

$$h_{r+1} = f(h_r, a_{r+1}, a_r) \tag{5.13}$$

$$f(h_r, a_{r+1}, a_r) = \begin{cases} h_r + 1, & \text{if } a_{r+1} \leq a_r \\ h_r, & \text{otherwise} \end{cases} \tag{5.14}$$

$$w_{r+1} = \frac{S_{r+1}}{h_{r+1}} \tag{5.15}$$

$$Sc = \begin{cases} Sc - 1, & \text{if } a_{r+1} > a_r \\ Sc + 1, & \text{otherwise} \end{cases} \tag{5.16}$$

In the optimized method, S represents triangle surface, it computes the sum of over-threshold pixels which are detected over multiple rows and exists in a specific width l. h denotes the triangle height. Hence, the computed value w represents the width of the detected triangle as pictured in 5.5. Width value is used to evaluate detected triangles. The two computed values Sc and w are compared to thresholds. Hence, a decision can be made in order to discriminate between obstacle objects

and noise induced in binarized image.

Contact points produced by this method are not always perfect because the percentage of pixels that located in lines (which belong to polar histogram beam) is 45% of all image pixels. However, it extracts the closest point to the real point.

5.4 2D Occupancy Grid Map Reconstruction

5.4.1 Contact Points Selection

Contact points extracted by the previous method may include several points that belong to the same obstacle object. Therefore, we need to select best contact points which can represent the localization of obstacle in the 2D occupancy map. The presented method is mainly composed of two steps:

- Contact points are grouped into clusters. Each cluster is mapped to an obstacle object that exists in the scene.
- For each cluster, nearest contact points to robot are chosen as selected points.

In practice, occupied and free space are already determined from 3.7. Hence, selected contact points must be located in occupied bearing. Because of the perspective effect, an occupied bearing may represent one or more obstacles. The goal is then to determine nearest points to robot.

Data: $P[m,2]$,$C[n,3]$
Result: $S[q,2]$
$Temp[r,3]$, $max = 0$, $q = 0$, $k = 0$;
for $i = 0 \rightarrow m-1$ **do**
  if $(P[i,0] <> 1)|(P[i,1] <> 1)$ **then**
    for $j = 0 \rightarrow n-1$ **do**
      if $|P[i,0] - C[j,0]| \rangle th$ **then**
        $Temp[k:] \leftarrow C[j:]$;
        $k \leftarrow k+1$;
    end
  else
    for $l = 0 \rightarrow k-2$ **do**
      if $Temp[k,2] > max$ **then**
        $max \leftarrow Temp[k,2]$;
        $S[q:] \leftarrow Temp[k:]$;
        $q \leftarrow q+1$;
    end
  end
end

Algorithm 3: Selection of best contact points method

$P[m, 2]$ is peaks matrix which denotes occupied bearings in the scene. This matrix is produced by free space detection module. The first column represents the bearing of the occupied space while the second column represents the density of obstacles that exist in this bearing. $C[n, 3]$ is contact points matrix produced in 5.3. Similarly, first column denotes the bearing from which contact point is extracted. Second and third columns are the coordinates u and v of the contact point respectively. th is a threshold by which we decide that a contact point (that belongs to $C[n, 3]$) converges to an occupied bearing (in matrix $P[m, 2]$). $Temp[r, 3]$ is a temporary matrix in where candidates of best contact points are temporally stored.

5.4.2 Map Reconstruction

Selected contact points produced by the aforementioned method are represented in the image coordinate system. Below we describe how these points are mapped and projected to the 2D occupancy map.

Our camera is already calibrated for a given plane in the scene (ground plane). As contact points belong to ground plane, the coordinates of a contact point in the world frame are computed as follows:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K \begin{bmatrix} r_{r1}^c & r_{r2}^c & t_r^c \end{bmatrix} \begin{bmatrix} X_r \\ Y_r \\ 1 \end{bmatrix} = H \begin{bmatrix} X_r \\ Y_r \\ 1 \end{bmatrix} \tag{5.17}$$

$$\begin{bmatrix} sX_r \\ sY_r \\ s \end{bmatrix} = \begin{bmatrix} r_{r1}^c & r_{r2}^c & t_r^c \end{bmatrix}^{-1} K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{5.18}$$

Eq. 5.17 was presented in 3.3. K being intrinsic matrix and $[R_r^c, T_r^c]$ represents transformation matrix from camera coordinate frame to robot coordinate frame. Eq. 5.18 allows us to compute the coordinates of these points in the robot coordinate frame. Finally, world coordinates (which is the coordinate system of the occupancy map) are computed in order to be projected in the occupancy map.

$$\begin{bmatrix} sX_w \\ sY_w \\ s \end{bmatrix} = \begin{bmatrix} R_{mvt} & T_{mvt} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_r \\ Y_r \\ 1 \end{bmatrix} \tag{5.19}$$

By Eq. 5.19, all selected contact points are represented in the world coordinate frame.Figure 5.6 illustrates the mapping from robot coordinate system to world coordinate system. R_{mvt}, T_{mvt} represent rotation and translation from robot coordinate frame to world coordinate frame respectively. X_w , Y_w are the world axis (in the world coordinate system).

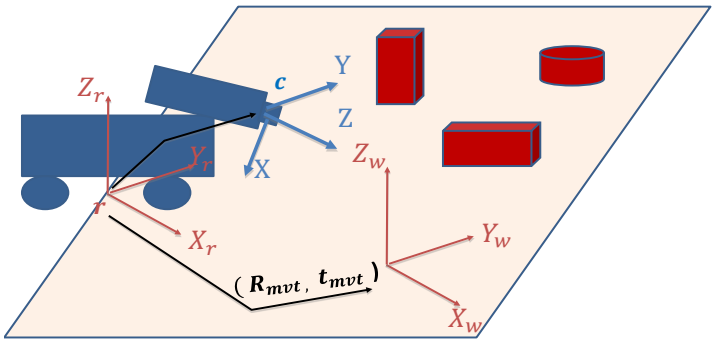


Figure 5.6: Occupancy map reconstruction and mapping of contact points.

Chapter 6

Hardware IP Cores

Contents

6.1 Introduction

Two Hardware architectures are proposed for obstacle detection and localization (pictured in figures 6.1, 6.2), and a hardware architecture is proposed for the reconstruction of 2D occupancy grid map (pictured in figure 6.3). In this chapter, we describe the design of each part of the hardware architecture. The proposed architectures are implemented using two different platforms Virtex6 and Zynq-7020.[1] Figure 3.1 shows the system methodology blocks. The proposed hardware modules are designed to perform this methodology. The three architectures are presented as follows:

- A hardware system is proposed for obstacle detection and localization. The proposed system is an implementation of IPM method proposed by [Bertozzi 1998b]. This system is named First hardware architecture.
- Some optimizations are proposed to IPM methodology in order to improve the performance of the hardware system. This system is named second hardware architecture.
- A hardware system is proposed for 2D occupancy map reconstruction. This system is composed of the aforementioned hardware system and some additional hardware/software modules to accomplish the map reconstruction task.

6.1.1 Obstacle Detection and Localization Systems

Figures 6.1, 6.2 show the general design of the two systems (namely: First and Second hardware architectures). We propose these systems in [Alhamwi 2015]. As already mentioned, first hardware architecture is an implementation of the IPM method introduced in the state of art. Figures 6.4, 6.5 show the differences between the two proposed systems. As depicted in these figures, bird's eye transformation is applied to eroded image in the first architecture while this transformation is not used in the second architecture.

Bird's eye transformation is excluded from the second architecture. In the case where this transformation is implemented, many constraints are imposed:

- Many block RAMs must be used to store a specific number of eroded image rows before bird's eye image is generated.
- A heavy cost of latency is induced by bird's eye transformation because many image rows must be stored in the memory before executing this transformation.
- As this transformation is regarded as a homography transformation, an important amount of hardware resources is required to design the module.

[1] Available resources are provided in [XILINX 2015a] and [XILINX 2016b]

6.1.2 2D occupancy Grid Map System

Figure 6.3 illustrates the proposed system for the reconstruction of 2D occupancy grid maps. We propose this system in [Alhamwi 2016]. The system is performed using an FPGA-based vision sensor and odometry sensors for obstacles detection, localization, and mapping. The fusion of these two sources of complementary information results to an enhanced model of robots environment. The proposed system has a reduced computational time, high frame-rate and low consumption of power.

When the proposed system is compared with the obstacle detection and localization system already introduced, specific differences are noted:

- In terms of methodology:
  - In this proposed system, an optimized method for extracting contact points (already introduced in 5.3) is implemented.
  - A method is also proposed for selecting the best contact points.
  - A mapping operation is required to represent the contact points in the coordinate system of the occupancy map (already introduced in 5.4).
- In terms of architecture:
  - A hardware implementation of the optimized contact points extraction method is presented in this system.
  - A hardware implementation of the free space detection method is presented.
  - A hardware implementation is presented for selecting the best contact points.
  - An embedded software module is proposed to map and project these selected points into the occupancy grid map.

For each part of the proposed architectures, an IP core is proposed in order to accomplish its task. The description of each module includes the following notes:

- An external view of each module is shown. This view illustrates input and output signals connected to this hardware module.
- An internal overview of each hardware module is shown. This view illustrates the internal hardware components of this hardware module.
- Hardware design and interconnections are then introduced. Each design describes how to perform arithmetic operations. Additionally, the proposed architectural designs may show delays, state machine design, and memory organisation.[2]

[2]Flip-flops are not shown in the figures

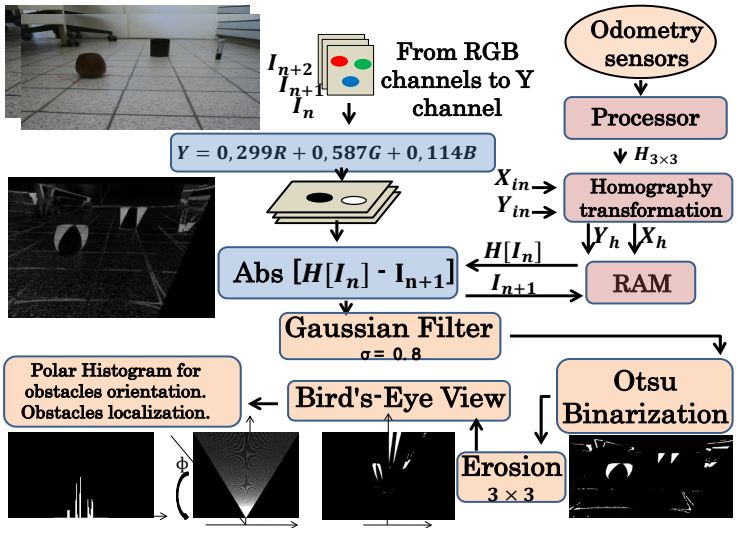


Figure 6.1: First Hardware Architecture.

- For each hardware module, a synthesis report is provided. This synthesis illustrates the performance of each hardware module in terms of hardware resources and the required latency to accomplish the task of the proposed module.

6.2 Homography transformation

Homography transformation is a pixel-level operation. The input image must be stored in the memory because image coordinates are changed after transformation. To produce the image after the transformation, two methods could be used:

- In the first method, sequential reads of the input image and non sequential writes of the output image are performed. The drawback of this method is that it produces holes (pixels coordinates without pixels intensity values) in the output image. Therefore, not all pixels have intensity value in the output image.

- In the second method, non-sequential reads of the input image and sequential writes of the output image are executed. Each output pixel is mapped to an input pixel.

An approximation is done for the non integer output coordinates. A null pixel value is assigned to the output coordinates that map input coordinates located outside of the image borders. The second method is adopted to perform the transformation. The drawback of this method is that it requires the computation of the inverse matrix produced by the equation 3.23. This inverse matrix is calculated in the software part. Figure 6.6 shows the external view of homography transformation module.

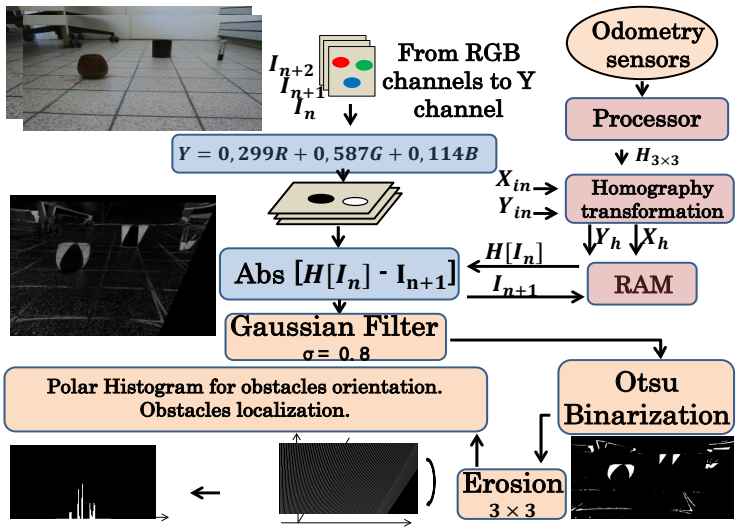


Figure 6.2: Second Hardware Architecture.

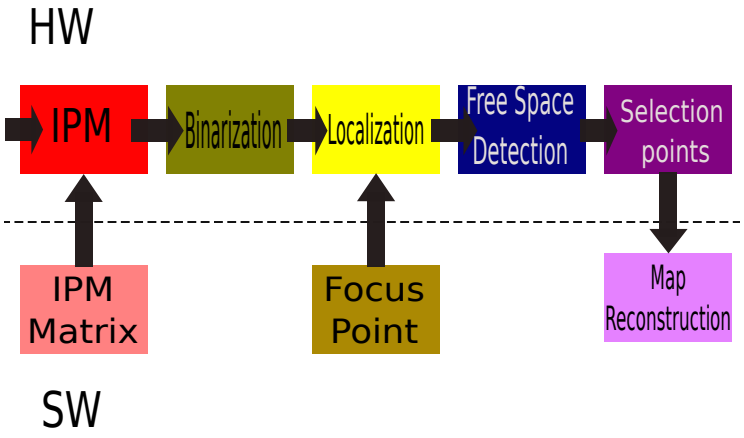


Figure 6.3: Hardware Architecture for the reconstruction of 2D occupancy grid maps.

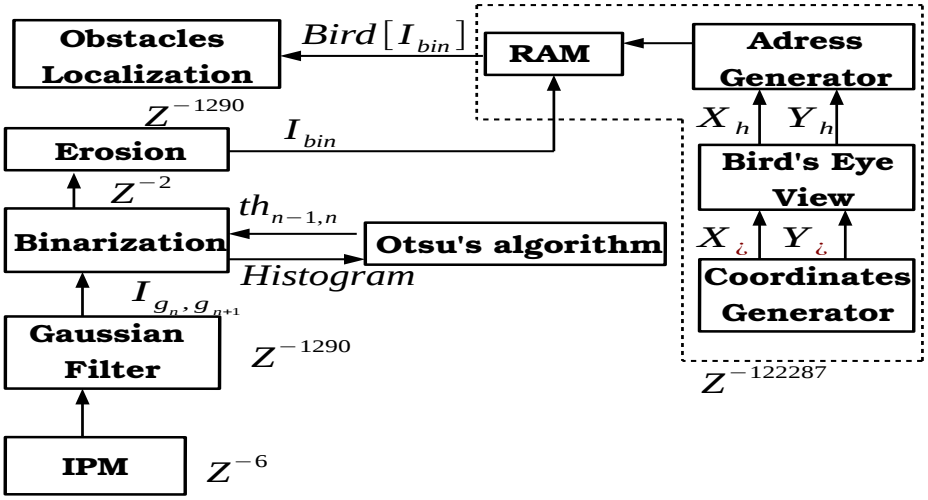


Figure 6.4: First architecture details.

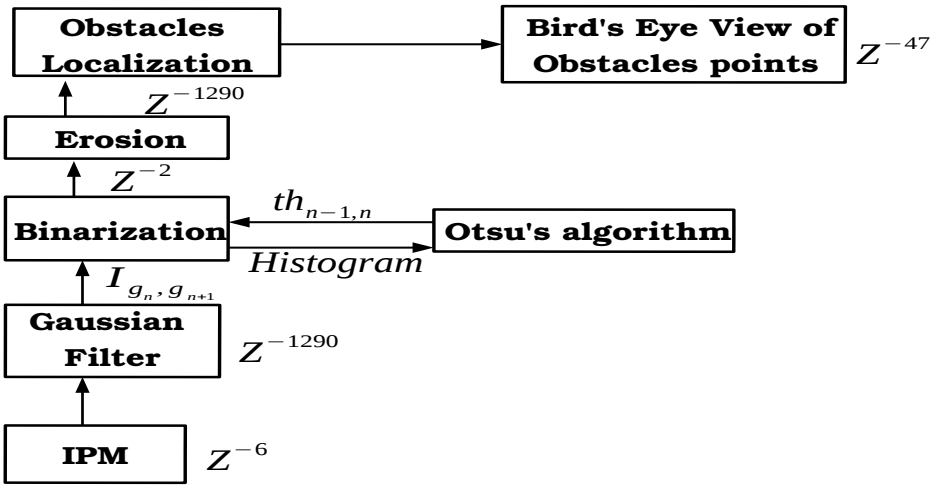


Figure 6.5: Second architecture details.

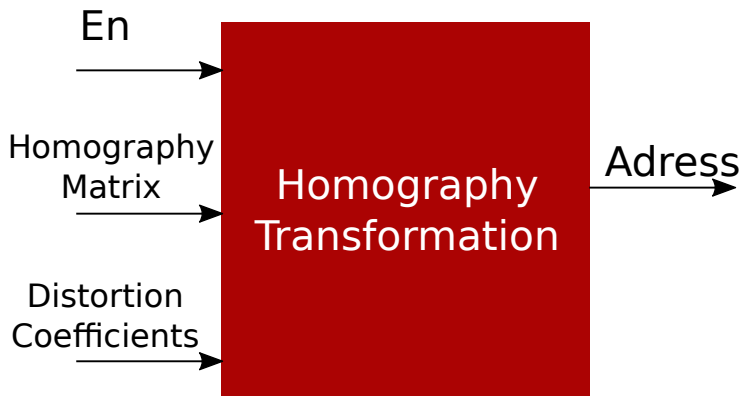


Figure 6.6: External view of homography transformation module.

Figure 6.7 illustrates the components of homography transformation module and the interconnections between these components. The homography transformation architecture includes *coordinates generator* , *homography*, and *Adress generator* components. These modules are presented in the sections below.

6.2.1 Coordinates Generator

The proposed hardware architectures are implemented for images of VGA resolution. Hence, this hardware component generates the pixel's coordinates (x, y) in the output image from $(0, 0)$ to $(639, 479)$. Pixel coordinates are represented using fixed point format. 10 bits are used to represent the decimal part, and 6 bits are used to represent the fractional part. Hence, the output signal of this module is 32 bits wide.

6.2.2 Distortion correction and Homography transformation

The coordinates produced by *ccordinates generator* goes to the *homography* module that computes the output coordinates (x_h, y_h) in the input image. This transformation includes two operations: distortion correction and the homography of an image stored in the memory. Therefore, this module also takes two components as two other inputs. The first component provides the intrinsic matrix and distortion coefficients while the second component provides the homography matrix. The homography matrix is updated in software and holds the change in pose between two frames.
In the processing pipeline pictured in figure 6.8,[3] the correction of radial distortion is performed first. In this case, the camera intrinsic matrix is required to transform

[3]DFFs are not shown in Fig 6.8

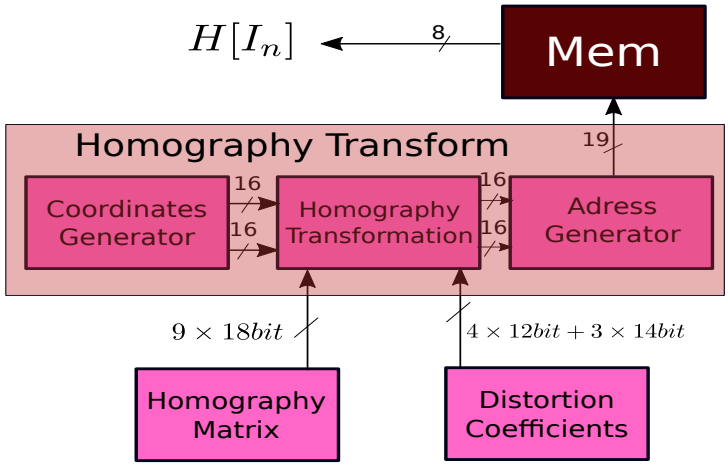


Figure 6.7: Hardware components of homography transformation and distortion correction.

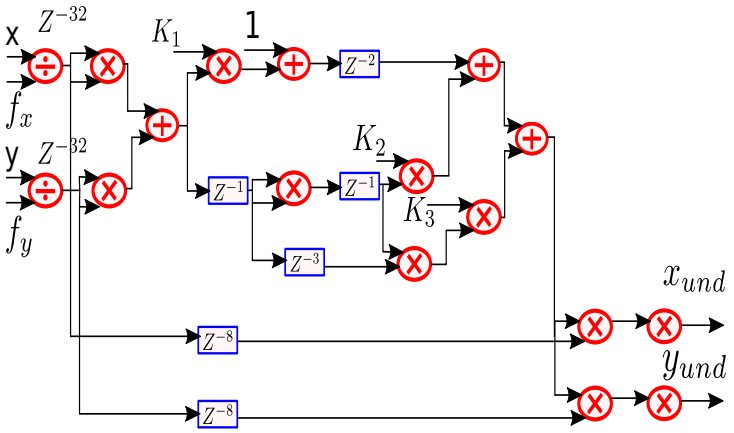


Figure 6.8: The pipelined design of distortion correction.

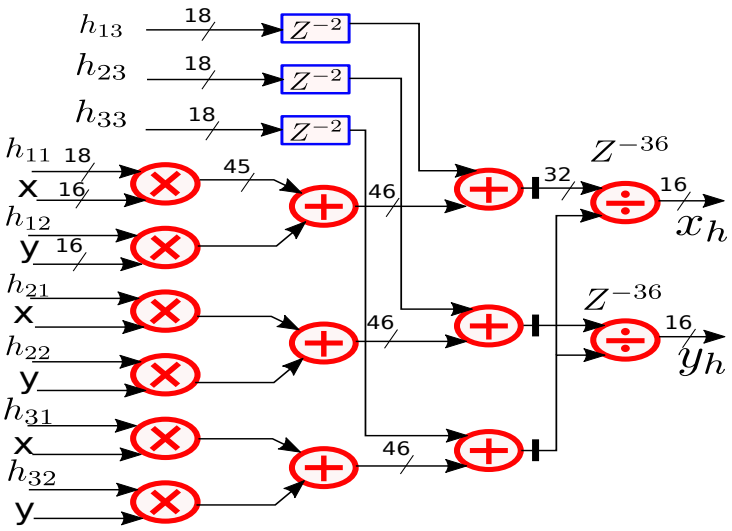


Figure 6.9: The pipelined design of homography transformation.

the coordinates represented in the image coordinate system to the camera frame system. Since a division operation is performed between the stream of coordinates and the focal distance, a 34 clock cycles latency is induced. The resulting values are then used to compute undistorted coordinates. Distortion coefficients K_1, K_2, K_3 are represented by 14 bits as a fixed point format; 11 bits are used for the fractional part of distortion coefficients. The representation of fractional part of distortion coefficients requires a high precision. Therefore, logic vectors of large width are required to represent these coefficients. In addition, the distortion correction process requires to perform complex arithmetic operations such as exponential functions. Hence, distortion correction operation requires an important amount of hardware resources. The whole process of distortion correction requires about 43 clock cycles in order to produce the first undistorted coordinates.

Homography transformation is executed after the distortion correction is computed. The homography matrix has 9 elements and each element is represented using 18 bits. The elements of the first and second columns in this matrix use 11 bits for the fractional part while the elements of the third column use 7 bits for the fractional part. This transformation is applied to undistorted coordinates represented in the image coordinate system using homogeneous coordinates. Figure 6.9 shows the pipeline stages of the proposed design.[4] This process requires about 42 clock cycles to output the first transformed coordinates.[5]
In the implementation of homography transformation, the division operation is the most critical operations; a 36 clock cycles latency is induced. The produced coordinates (x_h, y_h) refer to the pixel's coordinates in the input image (already stored in the memory). These coordinates are 32 bits wide. In some cases, these computed

[4]DFFs are not shown in Fig 6.9
[5]SRG in the figures denotes the shifting right or left of the input signal by n bits

coordinates (x_h, y_h) refer to pixels located outside of the image boundaries. In this case, we use an additional bit in order to label these coordinates, thus allowing to discriminate between the image pixels coordinates and other coordinates that have no correspondence in the input image.

6.2.3 Address Generator

The stream of transformed coordinates (x_h, y_h) pass to *Adress Generator* component to compute the memory addresses of theses coordinates. These addresses are expressed by the following equations:

$$x_h = x_h + 0.5 \tag{6.1}$$

$$y_h = y_h + 0.5 \tag{6.2}$$

$$adress = y_h \times w + x_h + adress_{offset} \tag{6.3}$$

with w being the image width. $adress_{offset}$ denotes the first pixel address of the stored image in the memory. An approximation is applied to the coordinates x_h, y_h as depicted in equations 6.1, 6.2. Thus, we remove the fractional part of these coordinates. As the resolution of our image is 640×480, 19 bits are used for the representation of the computed addresses. The memory takes these computed addresses as input in order to read the stored image.

6.2.4 Software part of Homography transformation

The software implements the computation of the homography matrix produced by equation 3.23. To compute this matrix, the following matrices are required: the intrinsic matrix and its inverse matrix, the extrinsic matrix (from robot coordinate frame to camera coordinate frame) and its inverse matrix, and the inverse matrix of the movement matrix. Intrinsic and extrinsic matrices are constant matrices, thus allowing to compute these matrices and their inverse matrices offline. Movement matrix is computed from odometry data produced by odometry sensors. In the experiment introduced in section 7.3, TurtleBot robot is used to provide odometry data. This includes x, y, θ. Hence, the inverse matrix of movement matrix can be expressed as follows:

$$r_{xy} = \sqrt[2]{x^2 + y^2} \tag{6.4}$$

$$\begin{bmatrix} cos(\theta) & sin(\theta) & 0 & r_{xy} \\ -sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

with x, y being the translation in the ground plane between two sequential frames. θ is the variation of the rotation angle around Z-axis between two sequential frames.

As shown in equations 6.4, 6.5, trigonometric and square root functions are used to produce the movement matrix. Two multiplications operations are per-

Table 6.1: Software part of homography module.

	Execution time
Movement matrix composition	$55\mu s$
The produced matrix by equation 3.23	$74\mu s$

Table 6.2: Comparison of the proposed homography module to [Botero 2012]

	LUTs	Slice Reg	Latency	Frame rate	Target
[Botero 2012]	3469	5907	1.237μ s	100 fps	Virtex-6
Ours	3897	6802	1.383μ s	200 fps	Zynq-7020

formed between matrices. Table 6.1 shows the execution time needed to produce homography matrix. These results are produced using Zedboard platform where a dual-core ARM, Cortex -A9 processor, is integrated with Zynq-7020.

6.2.5 Performance and Comparison with homography module in [Botero 2012]

Table 6.2 shows the comparison between our proposed homography module and the homography module proposed in [Botero 2012]. As illustrated in this table, the homography module in [Botero 2012] executes only homography transformation while our module merges two operation: homography transformation and distortion correction. The homography module presented in [Botero 2012] introduces two models for implementation:

- Homography module without cache memory, the proposed module uses SDRAM memory in order to store the input image. The frame rate is about 30 frame per second.
- Homography module with cache memory. The input image is written to a cache memory. The produced frame rate is about 100 frame per second.

Even if our hardware module consumes a greater amount of hardware resources, the frame rate of our proposed design is better than the frame rate achieved in [Botero 2012]. Since the pipeline stages are increased in our design to improve the frame rate, an additional cost in terms of hardware resources is induced.

6.3 Bird's eye transformation

Bird's eye transformation (already introduced in 3.5) is regarded as a homography transformation. The hardware design (pictured in figure 6.11) of this module is similar to the proposed design of homography transformation already presented in the previous section. Hence, the three components *coordinates generator*, *homography*, and *adress generator* are required to do the transformation.

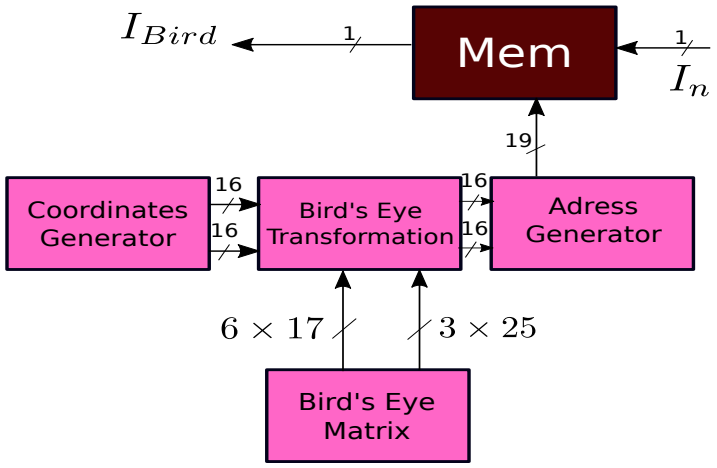


Figure 6.10: Hardware components of bird's eye module.

Table 6.3: Resources required for bird's eye transformation

	LUTs	Slice Reg	Latency	Target
Bird's eye transformation	3262	5491	2.108 ms	Virtex-6

Bird's eye transformation is only implemented in the first hardware architecture pictured in figure 6.1. This transformation was used in the literature as a preliminary step for obstacle localization in IPM method [Bertozzi 1998b]. As shown in figure 6.10, this transformation is applied to the binarized image. Before activating this module, a specific number of image rows must be stored in the memory. As a pixel of binarized image is 1-bit wide, BRAMs can be used for the storage of binarized image.

Bird's eye matrix is a constant matrix for a camera that does not rotate along its y or z axis. As a result, bird's eye matrix can be precomputed offline. The drawback of this transformation in hardware implementation is the high precision required to represent the fractional part of bird's eye matrix elements.

6.3.1 Resource Utilization and Latency

Table 6.3 illustrates the amount of hardware resources to perform bird's eye transformation. In the implementation of the first architecture (pictured in figure 6.1), 14 bits are used to represent the fractional part of each element in the bird's eye matrix. Each element of the third column in this matrix is 25 bits wide. Two division operations are executed in this transformation. Because of the high precision required for the representation of bird's eye matrix values, logic vectors of large width are used. Hence, this transformation consumes an important amount of hardware resources. Additionally, a heavy cost in terms of latency is induced when

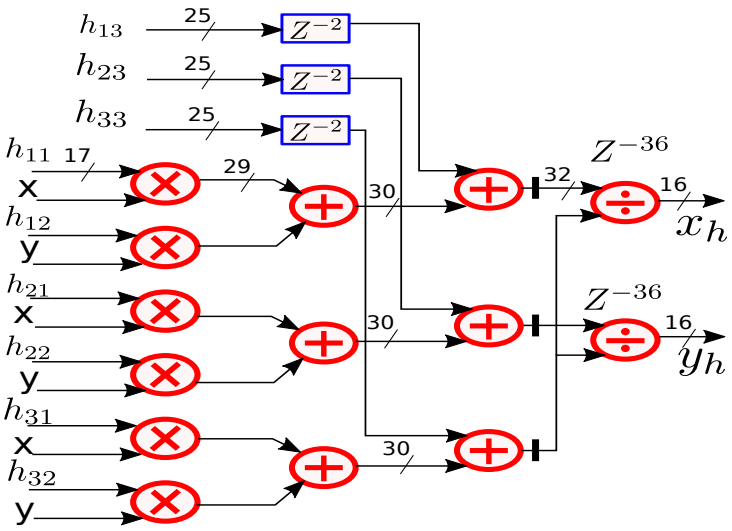


Figure 6.11: Hardware design of bird's eye transform.

this transformation is implemented on FPGA. In our implementation, a 122287 clock cycles latency is induced; 90 image rows must be stored in BRAM before activating this transformation.

6.4 IPM and Subtraction

The input signals of this module, as depicted in figure 6.12, are the stream of the transformed image $H[I_n]$ read from the memory and the stream of real image I_{n+1}. Each pixel in the two streams is represented using 8 bits. The output signals of this module are the resulting IPM image ($|H[I_n] - I_{n+1}|$) and the image stream I_{n+1} with its memory addresses as I_{n+1} will be written to the memory replacing I_n.

This module performs the absolute difference between the transformed image stream $H[I_n]$ and the image stream I_{n+1}. The critical operation in this module is the synchronization between the two streams I_{n+1}, $H[I_n]$. A FIFO component is used to synchronize the two streams. A careful attention is paid to the design of the FIFO component to perform real time image acquisition of I_{n+1}.

When a new frame is acquired, *vsync* is a control signal referring to a new valid frame, goes HIGH. In this case the homography module is activated. While homography module takes about 85 clock cycles in order to output the first pixel of the stream $H[I_n]$ from the memory, the new image frame I_{n+1} is written to the FIFO component.
hsync, a control signal that indicates a new image row, controls the writing of image stream I_{n+1} to the FIFO component. As depicted in figure 6.13, when *hsync*

Figure 6.12: External view of IPM module.

goes HIGH, the writing of image stream I_{n+1} to FIFO component is deactivated. *Rd-en* is a read enable signal that outputs from FIFO component. As shown in 6.13, writing process is slower than reading process in the FIFO component. Therefore, a careful attention is being paid to ensure that the process doesn't attempt to read from an empty FIFO and write to a full FIFO. If the same clock (Pixel Clock PCLK) is used for both reading and writing, the required depth of the FIFO component is computed as follows:

$$h \times a \leq FIFO - depth \tag{6.6}$$

with a being the number of clock cycles where *hsync* is HIGH. h represents the image height which is 480 for a VGA image.

The FIFO depth can be also adjusted using two different clocks for reading and writing. In this case, the frequency of writing clock to FIFO must be a bit greater than the frequency of reading clock from FIFO module. The FIFO depth is necessary for the design of IPM module in order to avoid reading from empty FIFO or writing to full FIFO. On the other hand, no FIFO component is used to control the stream of the transformed image $H[I_n]$.

In particular, *adress generator* component in the homography module (already introduced in section 6.2.3) generates the physical addresses in order to read the transformed image $H[I_n]$ from the memory. These addresses are also an input of the IPM module in order to recognize whether the address is mapped to a valid pixel in the input image stored in the memory or not. When the address is assigned to an invalid pixel (outside of the image boundaries), the pixel is assigned a value of 0 in the resulting IPM image.

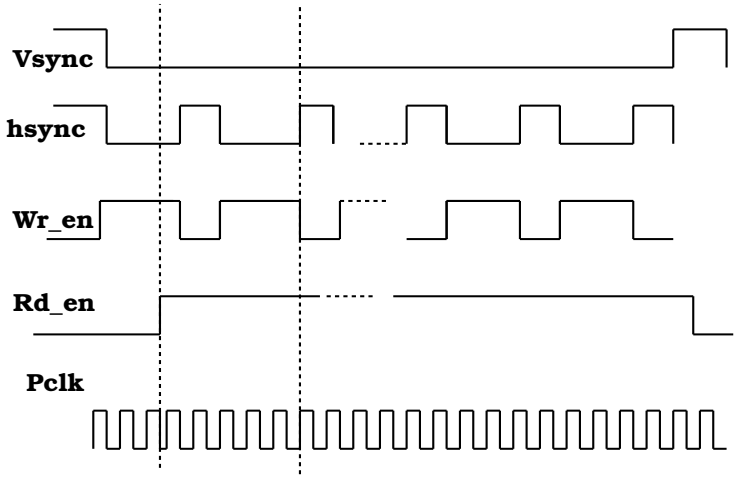


Figure 6.13: Reads-from/writes-to the FIFO component.

Figure 6.14 illustrates the hardware design that permits to output the stream of IPM image. An absolute subtraction is performed between the two streams I_{n+1}, $H[I_n]$. 9 bits are used for the representation of pixel value; the subtraction result is 8 bits wide. As 0 intensity pixel value is assigned to pixels having no correspondence in the input image, this 0 value can be also produced by a subtraction performed between two pixels having the same intensity value. Therefore, the MSB bit is used in order to know whether this pixel represents a subtraction value or a pixel having no correspondence in the transformed image.

The stream of image I_{n+1} is written to the memory replacing the image I_n. Reading operation of $H[I_n]$ and writing of I_{n+1} are carried out simultaneously. So, we have to pay attention when accessing the memory. If the same clock is used for both reading from and writing to the memory in which there are at least 50 image rows as a difference between reading and writing pointers, simultaneous reads-from and writes-to the same memory location will never occur.
The memory occupation space is composed of two parts:

- A major part contains the image I_n acquired at t_n.
- A minor part includes some image rows of I_{n+1} acquired at t_{n+1}.

The size of the major part is equal to the image size while the size of the minor part depends on the movement information encoded in the homography matrix. The longer is the distance performed between two sequential frames, the larger is the amount of memory needed to store the minor part.
A signal called offset address denotes the first memory location of the major part I_n. This signal is used by the component $adress-generator$ (introduced in section 6.2.3) in order to access the memory location of I_n, and it is also required for the

Table 6.4: Hardware resources for both IPM module and homography module

	Resources
Luts	3948
Slice Registers	6930
Target	Zynq-7020
Latency	6 clock cycle (Only IPM module)

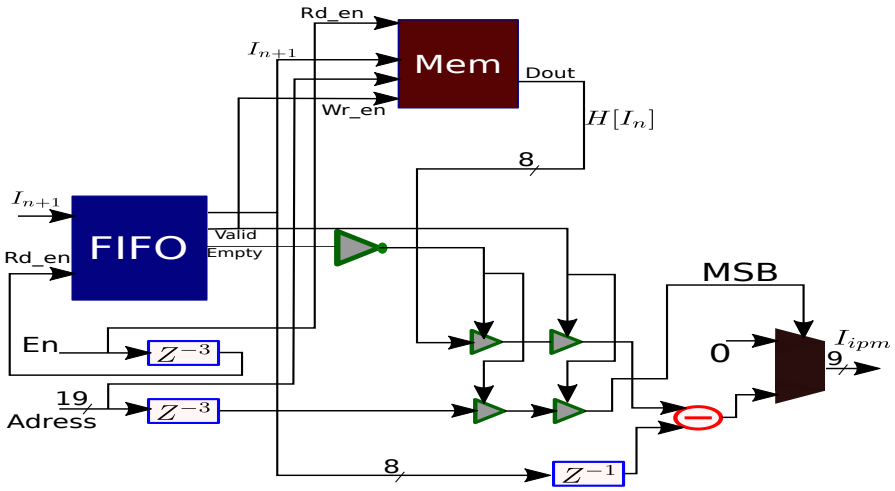


Figure 6.14: Hardware design of IPM module.

computation of the physical addresses of I_{n+1} when this stream is written to the memory.

6.4.1 Resource Utilization and Latency

Table 6.4 shows the amount of hardware resources to produce IPM image ($|H[I_n] - I_{n+1}|$). The two modules, homography module and IPM module, are required to produce this image. 91 clock cycles are induced as latency to produce IPM image; IPM module takes 6 clock cycles to output the first pixel of IPM image whenever a pixel of the transformed image is available on the input of IPM module. The other clock cycles are induced by the homography module.

6.5 Gaussian Filter

The resulting image after IPM module forwards to a Gaussian filter in order to remove noise and to ensure a good performance of Otsu's binarization. The input signal of this module is the stream of IPM image. The output signal is a blurred image of the IPM image.
Figure 6.15 illustrates the hardware components required to perform the filter. The

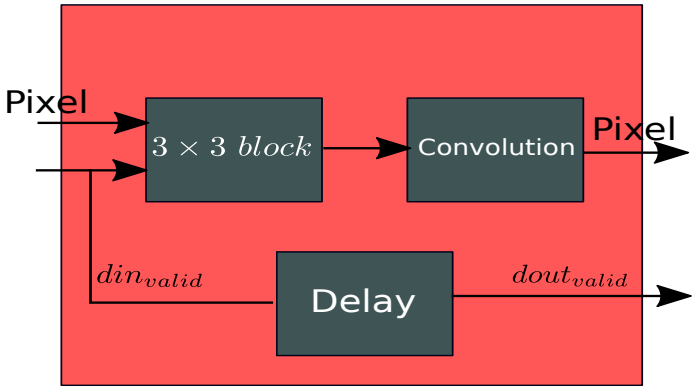


Figure 6.15: Hardware components of Gaussian filter.

3×3 *block* transforms the input stream of pixels into a stream of 3×3 blocks. A BRAM Block is needed in order to store three image rows (two image rows and two pixels). The stream of 3×3 blocks forwards to *Convolution* component where a convolution is performed between the stream of 3×3 block and a Gaussian kernel. The convolution process produces the output pixel. *Delay* component outputs an enable signal which is set to 1 whenever a valid pixel is available on the output.

6.5.1 Memory organization

An important cost of latency is paid before the first 3×3 block is produced. As the image resolution is 640×480, 1282 clock cycles are needed in order to produce the first block. The BRAM Block is divided into 3 parts. Each part stores one row of the input image. Hence, reads-from and writes-to memory can be easily performed. A state machine architecture with 3 states is implemented. As depicted in figure 6.16, I_0, I_1, and I_2 represent 3 image rows stored in the memory. The 3 states aim to produce the 3×3 blocks in good order. Each resulting block is composed of 9 pixels. Each pixel is 8 bits wide. en_0, en_1, and en_2 are used to recognize the order of image rows in the memory. The switch between these 3 states is done when a new image row is acquired.

6.5.2 Convolution and Delay

Figure 6.17 illustrates the hardware design of the *convolution* component where a convolution is performed between the Gaussian kernel and the 3×3 blocks resulting

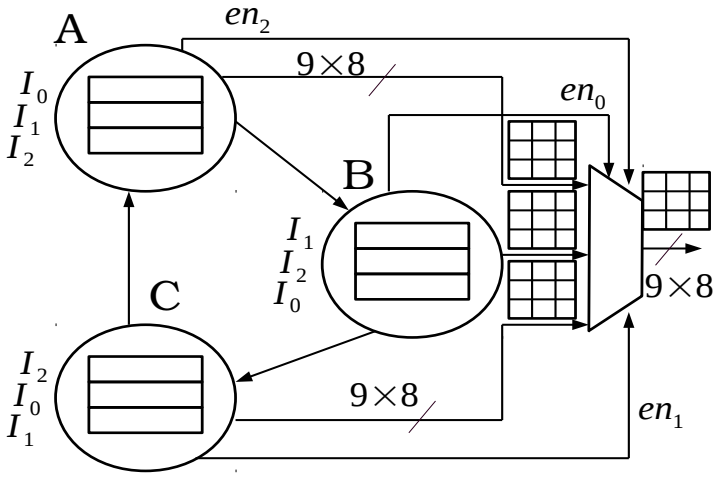


Figure 6.16: Memory organization to produce the 3×3 block.

Table 6.5: Hardware resources for Gaussian filter implementation

	Resources
Luts	160
Slice Registers	249
BRAMs	2 %
Target	Zynq-7020
Latency	1282 clock cycle

from the previous process. The Gaussian kernel used in this implementation is:

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix} \tag{6.7}$$

Gaussian kernel values are carefully selected. The proposed values allow to avoid critical arithmetic operations like division operation in hardware implementation. Hence, the shifting right by n bits on each signed signal of the 3×3 block will have the effect of dividing the values of a 3×3 block by (4, 8, 16). The input pixels are represented by 9 bits where the MSB bit denotes the identity of this pixel (already introduced in section 6.4). This MSB bit is delayed by 4 clock cycles and concatenated with the computed pixel resulting from convolution operation.
Delay component produces a valid signal referring to valid output pixels. 4 clock cycles are induced as latency in the *convolution* component, and 3 clock cycles are used in order to produce the 3×3 blocks.[6] Table 6.5 illustrates required hardware resources for Gaussian filter.

[6]Without taking into consideration the time required to store the first two rows of the image

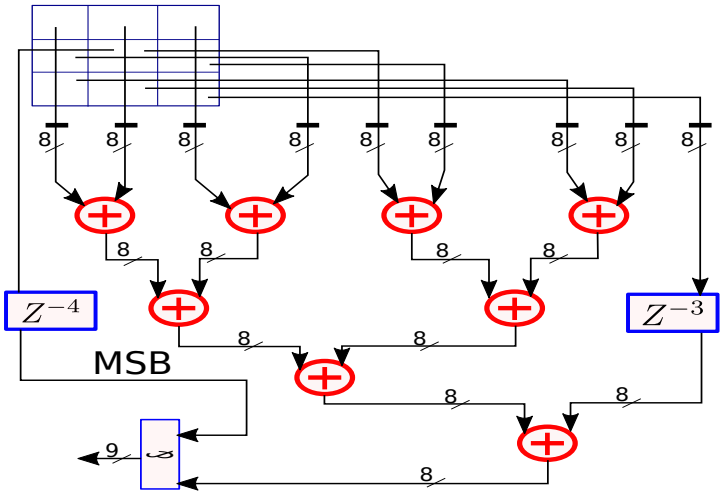


Figure 6.17: The hardware design of *Convolution* component.

Figure 6.18: External view of binarization module.

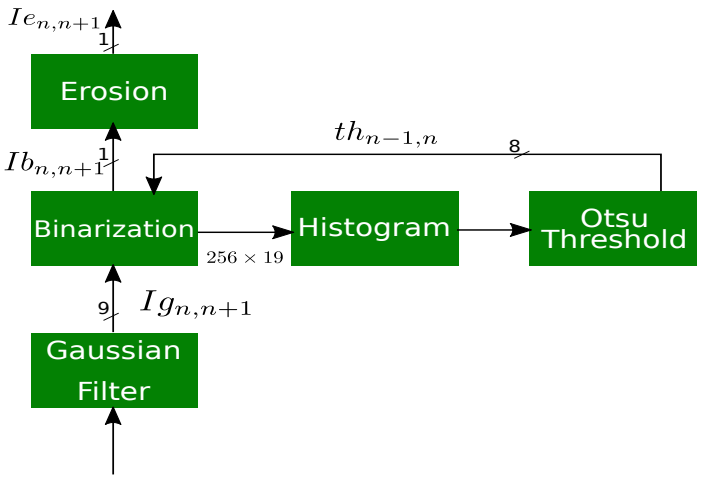


Figure 6.19: Hardware architecture for obstacle segmentation.

6.6 Binarization

The resulting image after Gaussian filter is binarized by Otsu's threshold. The binarization process aims to segment obstacle objects. As pictured in figure 6.18, the input signals of this module are the pixel stream and an enable signal which goes $HIGH$ whenever a valid pixel is available on the input. Each pixel in the input stream is 9 bits wide. The output signals of this modules are the pixel stream of the binary image and an enable signal which is set to 1 whenever a valid pixel is available on the output.

Figure 6.19 illustrates the proposed hardware components for obstacle segmentation. $Ig_{n,n+1}$ represents the produced stream from Gaussian filter. Indices $(n, n+1)$ denote the frame index (because IPM method needs two sequential images acquired at distinct moments (t_n, t_{n+1})). Also, $Ib_{n,n+1}$ denotes the pixel stream of the binary image. Each pixel is represented by one bit. The threshold $th_{n-1,n}$ used to binarize $Ig_{n,n+1}$ is already computed from the stream $Ig_{n-1,n}$. As histogram computation requires a heavy cost in terms of latency. This thresholding remains valid and allows to save on processing latency if there is not a high variation in intensities between two sequential frames. Figure 6.20 illustrates the hardware architecture for thresholding and histogram extraction. The hardware design in this figure is described as follows:

- The MSB bit of each input pixel is checked in order to recognize the identity of pixel. If the MSB bit is 0, the pixel value (represented by 8 bits) forwards to thresholding step. Otherwise, the pixel value is assigned a value of 0.

- In thresholding step, a comparison is performed between Otsu's threshold and the pixel stream (as pictured in figure 6.20). If Otsu's threshold is less than

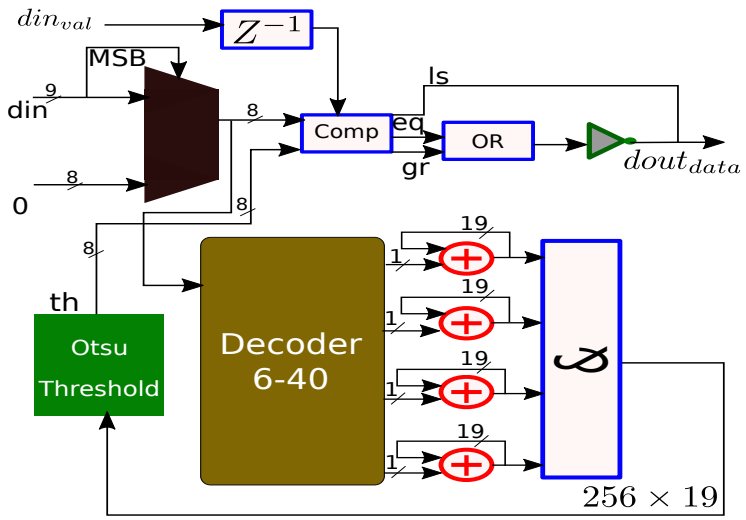


Figure 6.20: Hardware architecture for binarization and histogram extraction.

the pixel's intensity, the output pixel is set to 1. Otherwise, the output pixel is set to 0.

- Each input pixel also forwards to another hardware component in order to compute the histogram of the input image. Histogram is represented by a vector. Vector size is 256. Each element in the matrix is 19 bits wide. A 8:256 decoder is implemented for extracting the histogram. In this decoder, the input signal is 8 bits wide. This signal represents the index of the histogram value. Hence, the decoder output at this index is set to 1. A sum is then performed between the histogram value and this decoder output.

6.6.1 Hardware implementation of Otsu's method

The computed histogram forwards to *Otsu threshold* component which executes Otsu's algorithm in order to compute Otsu's threshold. Figure 6.21 shows the essential components to execute this algorithm in the hardware architecture; the proposed architectural design aims to perform the calculation of Eq. 3.30 already introduced in the theoretical background.
Otsu's algorithm is applied to the extracted histogram, and it is implemented as follows:

- The $gray-level$ generator is a 9-bit counter. The lower 8 bits of the counter are used to generate pixel intensity values from 0 to 255. The MSB bit of the counter output is used to control other components.

- Each pixel value resulting from $gray-level$ generator is used as an address in order to read the histogram value at each gray-level value.

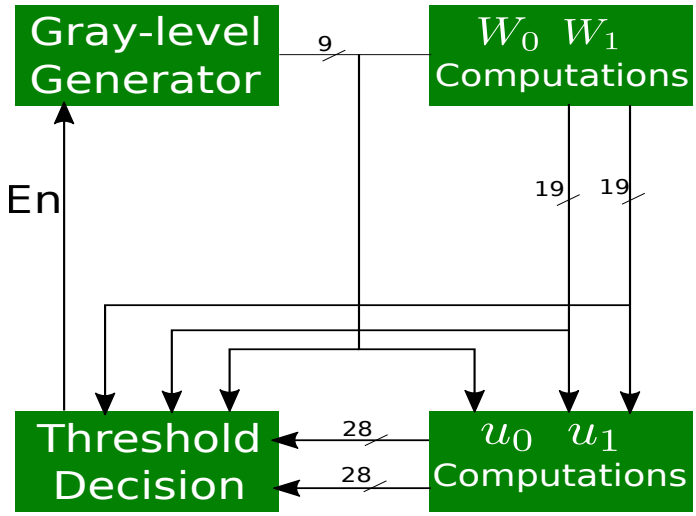


Figure 6.21: Hardware components for Otsu's algorithm implementation.

- The gray-level value and its histogram value are used to compute the class probabilities ω_0 and ω_1.
- These latter are then used in order to compute the class means μ_0 and μ_1.
- Finally, the computation of the between-class variance in Eq. 3.30 is performed.

A state machine design is implemented to execute the required computations (pictured in figure 6.21) in the good order. Figures 6.22, 6.23, and 6.24 illustrate the general hardware designs to perform the computation for each component shown in figure 6.21.[7]

Figure 6.22 illustrates the hardware design for the computation of ω_0 and ω_1. As illustrated in this figure, *Address Generator* component operates in like manner as $gray - level$ generator (pictured in figure 6.21). S_1 is the sum of the products of gray level values and its histogram values. When S_1 computation is done, the MSB bit of *Address Generator* is used to trigger $gray - level$ generator (illustrated in figure 6.22). LSB bit of $gray - level$ output enables the accumulation of histogram values which are mapped by the addresses from 0 to the current value of $gray - level$ counter, thus allowing to produce ω_0. *tot* is an input signal of *Otsu Threshold* module.[8] It represents the total number of valid pixels in the resulting image from IPM module. As already mentioned in section 6.4, the MSB bit of pixel value is used to recognize valid pixels. Hence, *tot* represents the number of pixels whose MSB bit is set to 0. A subtraction is performed between *tot* and ω_0 in order to compute ω_1.

[7] The figures show only the main operations

[8] *tot* signal is not shown Fig 6.20, but illustrated in Fig 6.22

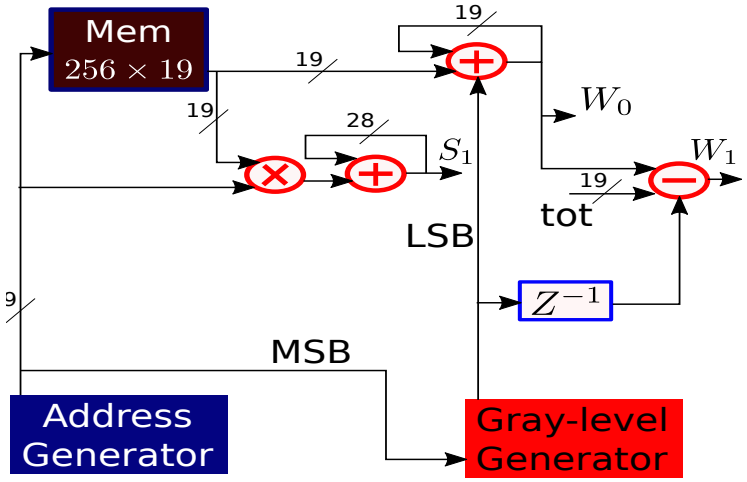


Figure 6.22: Hardware design for the computation of ω_0 and ω_1.

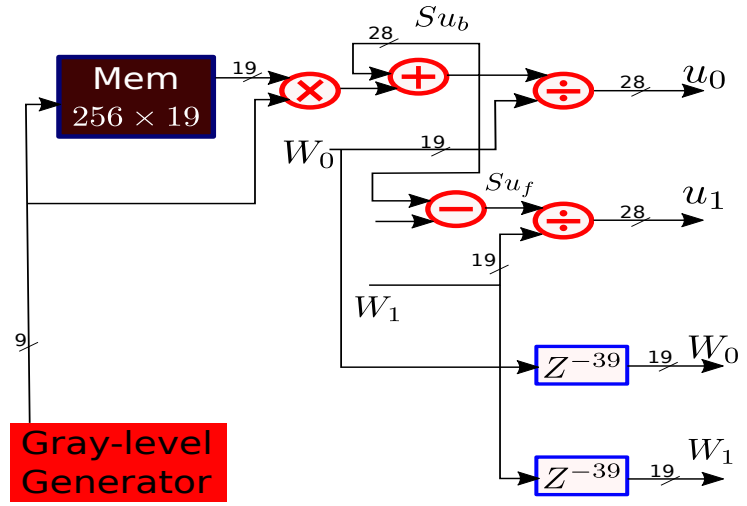


Figure 6.23: Hardware design for the computation of μ_0 and μ_1.

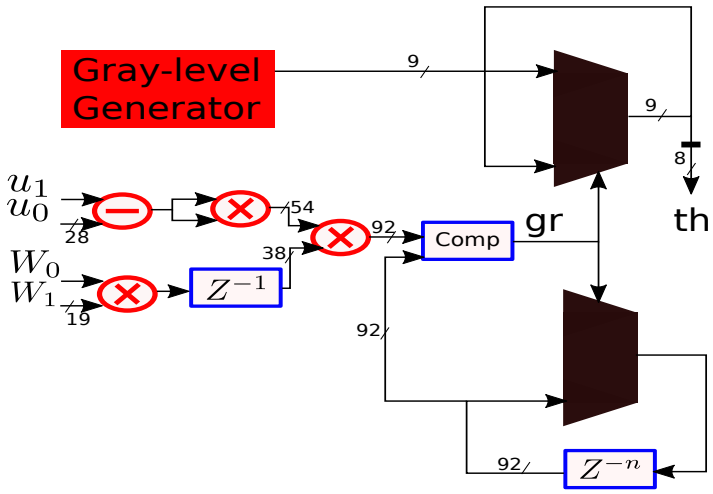


Figure 6.24: Hardware design for *threshold Decision* component.

Figure 6.23 illustrates the hardware design for the computation of μ_0 and μ_1. As already noted, $gray-level$ generator outputs gray levels in order to produce its histogram values. The product of histogram value and its correspondent gray level value is computed. This product is 28 bits wide. The sum of the product values is then performed to compute su_b. A subtraction is performed between su_b and S_1 to produce su_f. The two computed values su_b and su_f are each represented by 28 bits. Two division operations are then executed. 39 clock cycles are induced as latency to perform the two division operations.

- The first division operation is performed between su_b and ω_0 in order to produce μ_0
- The second division operation is performed between su_f and ω_1 in order to produce μ_1

Figure 6.24 illustrates the hardware design for *threshold Decision* component. Because of the latency induced by the two division operations, ω_0 and ω_1 are delayed by 39 clock cycles as shown in figure 6.23. A subtraction is performed between μ_1 and μ_0. The square of the subtraction result is then computed. This latter is 54 bits wide. The product of ω_0 and ω_1 is computed and delayed by 1 clock cycle. Finally, σ_B^2 in Eq. 3.30 is produced.
For each gray level value generated by $gray-level$ component, σ_B^2 is computed and compared to its previous value in order to find the maximum value of σ_B^2. The gray level by which the maximum value of σ_B^2 is produced represents Otsu's threshold.

Table 6.6: Hardware resources for binarization implementation on Zynq-7020

	Binarization module	Otsu algorithm
Luts	5233	3295
Slice Registers	15934	5320
Latency	2 clock cycle	12032 clock cycle

6.6.2 Resource Utilization and Latency

Table 6.6 illustrates required hardware resources for the binarization module, including the required resources for Otsu's algorithm implementation on Zynq-7020. A high consumption of hardware resources is induced because histogram is stored using slice registers. Additionally, division operations are performed between logic vectors of wide bits width. 2 clock cycles are needed for image thresholding. This doesn't include the clock cycles required for Otsu's algorithm implementation because thresholding uses an Otsu's threshold already computed. Otsu's algorithm implementation requires an large number of clock cycles since Otsu's algorithm is an iterative method that tries to find the optimal threshold.

6.7 Erosion operator

Erosion operation is applied to the binarized image produced by the previous process in order to remove noise resulting from binarization. The proposed hardware architecture in this process is similar to the hardware design of the Gaussian filter. Figure 6.15 illustrates the general hardware components. In the case of erosion operation a 3×3 *block* is used to transform the binarized stream pixels into a stream of 3×3 blocks since the binarized image is eroded by a 3×3 structuring element. Then, the produced blocks are convoluted with the structuring element.*Delay* component produces enable output signals referring to valid output pixels.
The memory organization is similar to that already described for the Gaussian filter (see 6.5.1 and figure 6.16 for more details). Table 6.7 shows the required hardware resources for erosion module. The produced latency is identical to Gaussian filter module latency. BRAM block size is an eighth of the memory size used for Gaussian filter, since erosion operation is applied to binarized images (1 bit per pixel).

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

If any of the neighbouring pixel in the 3×3 block is a background pixel, the output pixel is set to a background pixel. The *Delay* component produces a valid enable signals for valid output pixels after 4 clock cycles.

Table 6.7: Hardware resources for erosion filter implementation

	Resources
Luts	97
Slice Registers	75
BRAM	1 %
Latency	1282 clock cycle

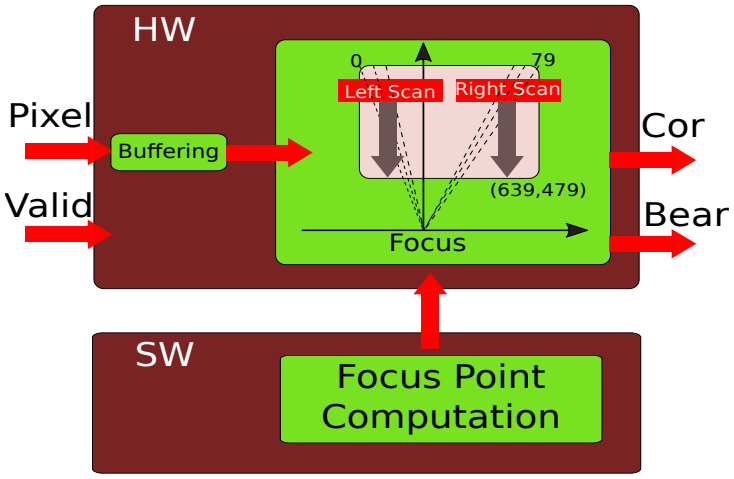


Figure 6.25: External View of Localization module.

6.8 Obstacles localization

The aforementioned operations aim to segment obstacle objects in the image frame. However, no information is produced about obstacles localization in the world frame. The proposed module in this section is devoted to accomplish the localization task. The module performs two functions:

- Obstacles bearings computations
- Contact points extraction

Figure 6.25 illustrates an external view of the localization module, including the main components of the hardware architecture. The input signals of this module are the pixel stream of binary image, an enable signal, and focus point coordinates. Each input pixel is 1 bit wide. Focus point coordinates are represented by 28 bits. The output signals are *Bear* and *Cor*. *Bear* denotes bearings measurements. It is one-dimensional array of 80 elements. Matrix indices denote defined bearings in the image while matrix values represent the number of over-threshold pixels per bearing. *Cor* is also one-dimensional array of 80 elements. Matrix values denote extracted contact point coordinates that exist in a defined bearing. Similarly, matrix indices in *Cor* represent defined bearings in the image.

Focus point coordinates are represented using 14 bits for each coordinate. The choice of the bit width is based on the fact that focus point is usually located outside image borders. As polar histogram beam is originating from this point, the localization of this point is an important issue. As already mentioned, IPM image is produced from two images: first image is a transformed image acquired at t_n, and second image is a real image acquired at t_{n+1}. The second image has a focus point called f_1. This latter can be precomputed offline from Eq. 5.5 because it depends on the orientation of camera with respect to the ground plane. The transformed image has a focus point called f_2. This point can be computed as follows using homogeneous coordinates:

$$f_2 = T_{ipm}^{-1} \times f_1 \tag{6.8}$$

$$f = \frac{f_1 + f_2}{2} \tag{6.9}$$

T_{ipm}^{-1} represents the inverse matrix of IPM matrix (Computed from Eq. 3.23). f is the midpoint between f_1 and f_2. Polar histogram beam is originating from f. Since IPM matrix is updated in real time on the embedded software part, the computations of f_2 and f are performed in the software part of the system.

$Buffering$ component stores the input stream while the other two components (namely Left/Right scan modules) perform the computations of $Bear$ and Cor.
These two components (Left/Right scan) operate independently. Each component scans a part of the image. As already mentioned in the theoretical background, polar histogram is performed using a beam of lines originating from focus. The number of lines depends on the image resolution. For VGA resolution used in our implementation, 80 lines are adopted in order to perform polar histogram beam. The choice of lines number is an important issue. The greater is the number of lines used to perform polar histogram, the higher is the accuracy of the results produced by this module. However, adopting a polar histogram beam with a great number of lines requires a huge amount of hardware resources. So, the number of lines in the histogram beam is a trade off between the amount of hardware resources and the accuracy of results.
Left scan module tracks 40 lines in the left side of image. Also, Right scan module does in the right side. The choice of the number of modules (which is two in our implementation) to track polar histogram beam depends on two criteria:

- Image resolution.
- The number of clock cycles required to compute the coordinates of each pixel by Bresenham's algorithm for a line in the polar histogram beam.

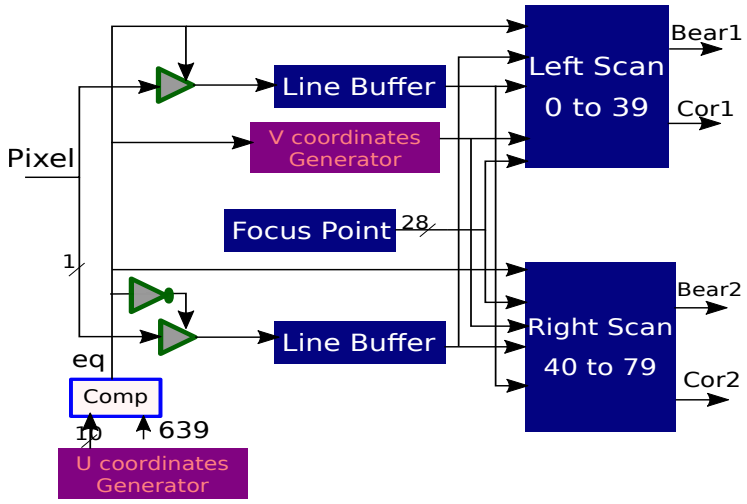


Figure 6.26: General hardware architecture of localization module.

Figure 6.26 shows the hardware architecture overview. It illustrates how the input stream is connected to the two components ($Left/Right$ scan). *Pixel* in this figure denotes a pixel in the input stream of the binary image. Two line buffers are used to control the flow of the input stream to the two components ($Left/Right$ scan). In details, these two buffers are operating as follows:

- One line buffer stores one image row.
- The other buffer is handled by the two components ($Left/Right$ scan).

Line buffer width is 1 bit, and line buffer depth is 640. Each line buffer is switched between storing an image row and being read by the two components ($Left/Right$ scan) when a new image row is received. *u Coordinate Genrator* is a 10-bit counter which produces u image coordinate from 0 to 639. Counter value is compared to a selected threshold (639). When the counter value is equal to 639, a control signal is generated in order to switch the storage of the input stream from one line buffer to the other line buffer as depicted in figure 6.26. *v Coordinate Genrator* is a 9-bit counter. This counter is used to count the number of received image rows. Therefore, this counter is enabled by the output of *comp*. This comparator outputs 1 when receiving 640 pixel. The counter value forwards to the components ($Left/Right$ scan) for further processing.

The outputs of the two components, ($Left/Right$ scan), are the parts of $Bear$ and Cor. These parts represent bearings measurements for $Bear$ and detected contact points for Cor. Each part is one-dimensional array of 40 element. A concatenation of these parts is done to produce $Bear$ and Cor outputs.

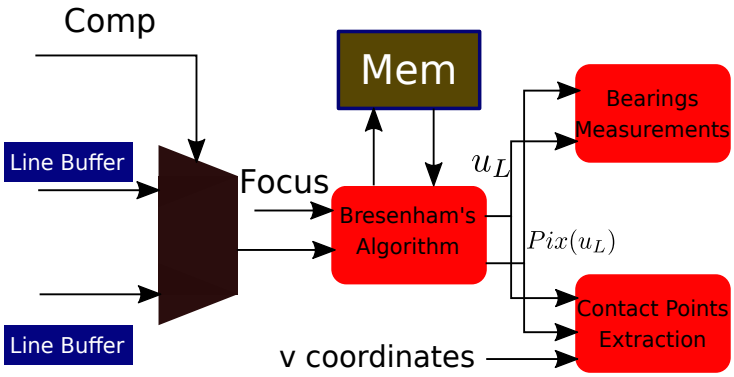


Figure 6.27: General architecture of Left and Right scan modules.

6.8.1 Left and Right Scan

As pictured in figure 6.26, the input signals are focus point coordinates, two line buffers, a control signal, and the output of *V Coordinate Generator*. Each module performs the following functions:

- Computing the coordinates of each pixel for each line in the polar histogram beam by the algorithm 2 (introduced in section 5.2).
- Performing bearings measurements, also included in algorithm 2.
- Extracting contact points between obstacles and the ground plane by the method already introduced in section 5.3.

The control signal (*Comp* output shown in figure 6.26) is used to select the good line buffer to be read. Line buffer and focus point coordinates forward to Bresenham's algorithm component (pictured in figure 6.27) which produces u coordinate for each line in polar histogram beam (named u_L in figure 6.27). The produced u_L coordinates with its pixel values $Pix(u_L)$ read from the line buffer forward to the two components *Bearings Measurements*, and *Contact Points Extraction*. This latter component takes *V Coordinate* as input in order to be concatenated to the computed u coordinate of detected contact points.
A state-machine design of 4 main states is implemented. The use of state machine design aims to organise the flow of data from Bresenham's algorithm implementation to contact points extraction and bearings measurements components.
Figure 6.28 illustrates the designed state machine for the proposed architecture. The functionality of these 4 states is described as follows:

- In state *IDLE*, Bresenham's parameters (du, dv, D, u_0, and u) are initialized for a defined line generated by *Line Counter* (pictured in figure 6.29),

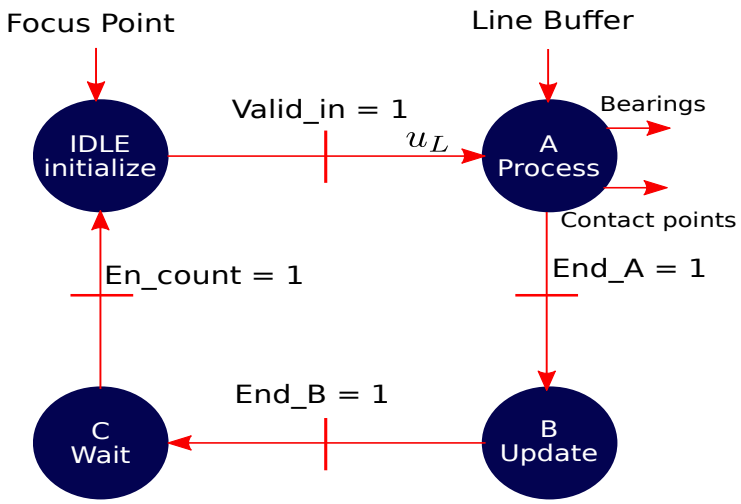


Figure 6.28: State-machine design for Polar histogram computation and contact points.

thus allowing to compute u_L coordinate for a generated line. u_L coordinate forwards to state A.

- In state A, contact points extraction method is implemented with bearings measurements. If any contact point is detected in this state, its coordinates (u_L,V) output in this state.
- In state B, the parameters D and u are updated in order to compute the next pixel coordinates u_L when a new image row is received. The updated values are stored in Mem[9]. In addition, some parameters used in contact point extraction process are updated in this state.
- In state C, *Line Counter* is enabled to generate all lines of the polar histogram, thus allowing to apply all previous operations to all these lines (from 0 to 39) for one image row stored in the line buffer. This state will be later a wait state until it is triggered by *comp* output signal (pictured in figure 6.26) which denotes an available image row in the other line buffer.

6.8.1.1 Bearing measurements

Figure 6.29 illustrates the hardware design of state $IDLE$. Bresenham's algorithm parameters (These parameters already introduced in 5.2) are initialized. Focus point is represented by 28 bit. *Coordinate Generator* (pictured in figure 6.29) generates the first coordinate (called u) of each line in the polar histogram beam. The upper 14 bits of focus point represent the parameter dv. A subtraction is

[9] Mem denotes that the values can be stored either in an array of storage elements such as individual registers or in RAMs

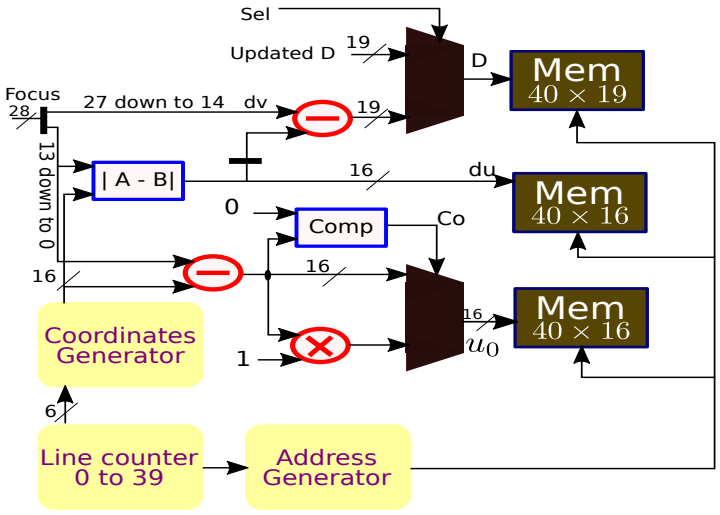


Figure 6.29: Hardware design in state $IDLE$.

performed between the lower 14 bits of focus point and *Coordinate Generator* output, thus allowing to produce I_2 represented using 16 bits. The absolute value of this subtraction is also computed, thus allowing to produce the parameter du using 16 bits.

I_2 is a signed logic vector which is compared to 0 in order to select the good sign of parameter u_0. This latter parameter is 16 bits wide. Parameter D is computed from the two parameters (du and dv). First, du is multiplied by 2 by shifting left by 1 bit. Then, a subtraction is performed between $2du$ and dv, thus allowing to initialize the parameter D by 19 bits. A multiplexer is implemented in order to choose the suitable value of D because this parameter is updated under specific conditions, it will be evaluated by another method. The parameters (du, u_0, and D) are stored in Mem whose depth is equal to the number of lines which is 40 in our implementation. The addresses are computed in *Address Generator*. In State $IDLE$, parameter u_L is computed and delivered to state A. This coordinate is computed from 3 parameters: u_0, u, and the lower 13 bits of focus point coordinates. u is initialized to 0 for each line in polar histogram beam. u_0 represents the coordinate of the first point of of each line in the polar histogram beam where the origin is focus point. Hence, the parameter u is the shifting from the first point of each line in the beam. This signal denotes the changes of the coordinate u when a line in the beam is tracked from the first point until focus point.

As depicted in figure 6.30, u_L is the sum of u_0, u, and the lower 13 bits of focus point with respect to the sign of u_0. This is because focus point is the origin (as shown in figure 6.25). As the left side of V axis represents negative values of u coordinates, the sign of u_L is functional to the localization of the line. If this line is in the left side, u_L is negative. Otherwise, it is positive. Therefore, a

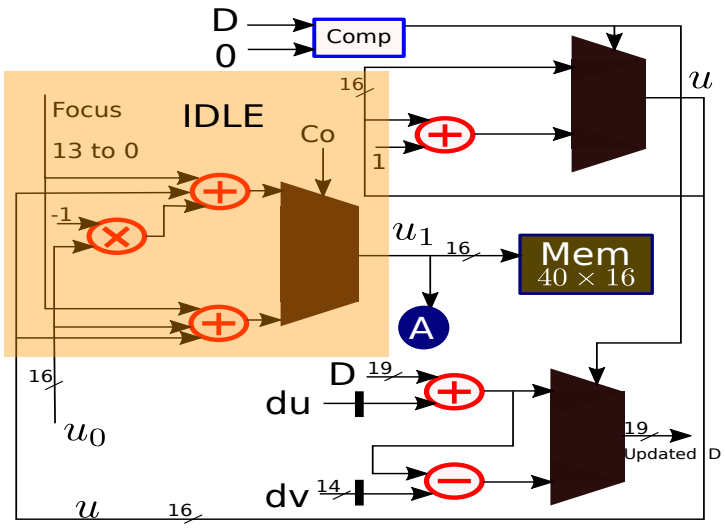


Figure 6.30: Hardware architecture in states $IDLE$ and B.

multiplexer is designed in order to choose the good value of u_L. Co controls the multiplexer output. u_L forwards to state A, and it is stored in Mem for each line in the histogram beam.

Figure 6.30 also shows the hardware design in state B. When a new image row is received, the parameters (u_L and D) are updated for each line in the histogram beam. Parameter D represents the evaluation of error in Bresenham's algorithm (already presented in section 5.2). D is compared to 0 in order to compute the shifting u. D itself is also updated in this state with respect to the sign of D as pictured in figure 6.30.

Figure 6.31 illustrates how obstacles bearings are produced in state A. u_L (which denotes the coordinate of a pixel that belongs to a line in the polar histogram beam) is used in order to read the pixel value from the line buffer. The pixel value (namely $Pix(u_L)$ in the same figure) is compared to 0 because each pixel assumes one of only two discrete values: 1 or 0 in the binary image. If the pixel value is equal to 1 (which means an obstacle pixel), $Comp$ generates an enable signal in order to pass $Line\ Counter$ value (represented using 6 bits) to a decoder (from 6 to 40). Hence, the correspondent value of element in the bearings matrix is incremented.

6.8.1.2 Contact points extraction

In state A, contact point extraction method is performed. Figure 6.32 illustrates the hardware implementation of the method. The following parameters (Sc, S, A, B, and h) are already introduced in the theoretical presentation (section 5.3). In the line buffer, each pixel is addressed using (u_L, V) coordinates. A decision is

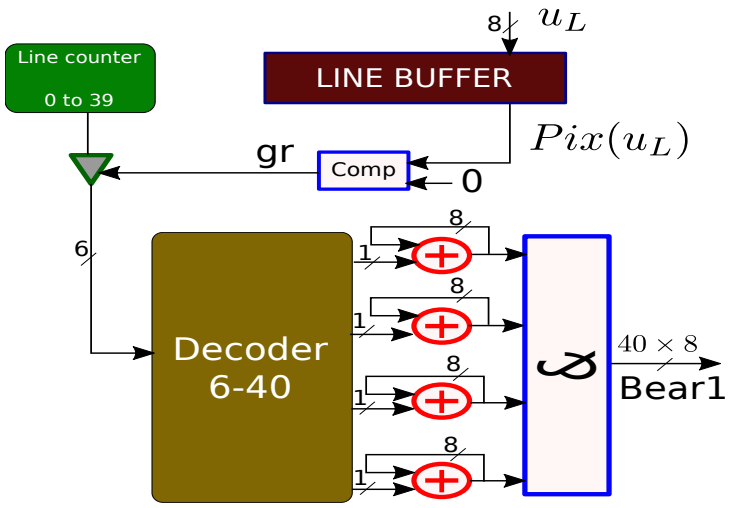


Figure 6.31: Hardware design for bearings measurements in state A.

made for the extraction of coordinates. A decision positive means that En_3 and En_2 signals enable the passing of the coordinates to be an extracted contact point. Otherwise, the coordinates are not extracted.

A sum is performed between a pixel in the line buffer and its neighbouring pixels. In the VGA implementation, 5 is selected as the window width. Hence, 10 pixels are set to the sum operation. The sum result is controlled by another signal called En_1. This signal represents the $An1$ gate output. $An1$ gate outputs 1 when a pixel value and all its neighbouring pixels values in a selected width are set to 1. Otherwise, $An1$ gate output is 0 and the sum result is not valid.

The sum result represents the parameter A. This latter is compared to another signal B (another parameter used in the method). Parameter A is 4 bits wide. Similarly, B is represented using 4 bits. B is initialized to the maximum value of sum operation. If the computed value of A is less than B, $Comp$ outputs 1. Hence, B is updated to the actual value of A and stored in Mem. In like manner, S and Sc parameters are updated. The decreasing of A of a defined line in the histogram beam will increase the likelihood of the existence of a triangular shape. In this case, Sc value is increased and the parameter S encodes the triangle surface by computing the number of over-threshold pixels of the defined histogram line across multiple image rows.

Once $An1$ outputs 0, an enable signal En_2 is set to 1. This enables the second passing of coordinates. Before this step, another control signal En_3 must be set to 1 in order to output the extracted coordinates. En_3 is $An2$ output. This gate is controlled by the outputs of two comparators. The first comparator output is set to 1 when the computed Sc is greater than a constant threshold defined for

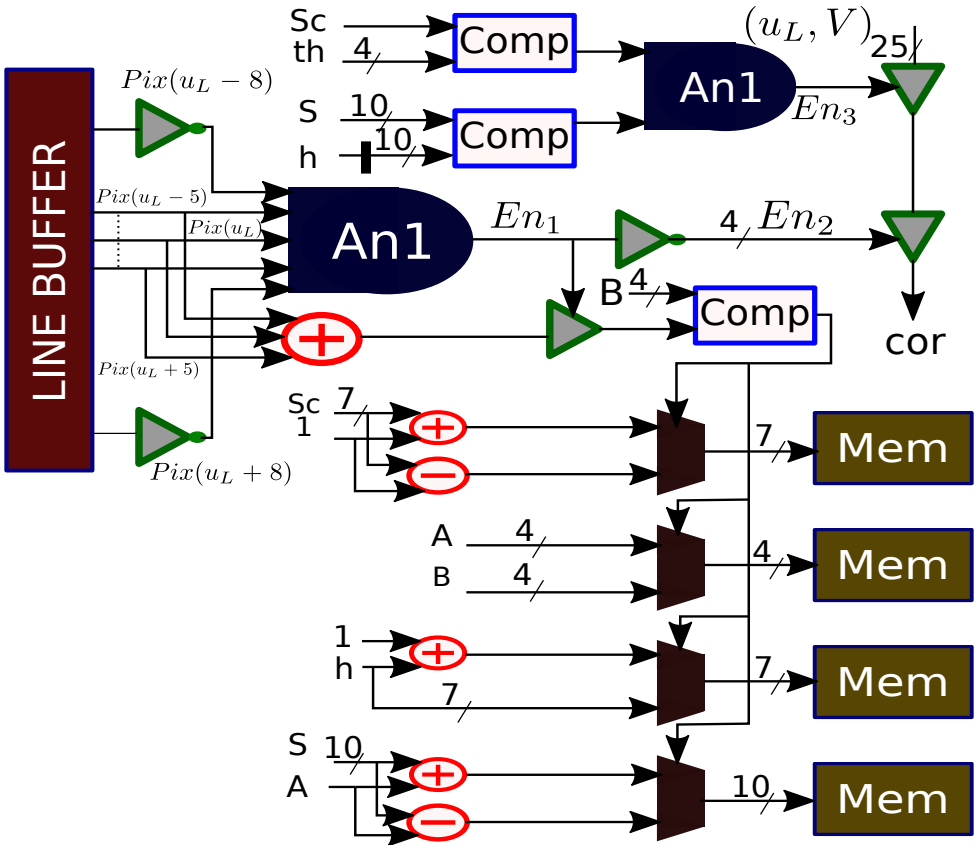


Figure 6.32: Hardware design for contact points extraction method in state A.

a specific environment. The second comparator outputs 1 if S is greater than a defined value computed by multiplying the computed h with a threshold. In this implementation, 4 is selected as a threshold.

Figure 6.33 illustrates the composition of extracted contact points. For a contact point of size 26 bits, 7 bits encodes the bearing, and 19 bits are used to represent contact point coordinates.

6.8.2 Resource Utilization and Latency

Table 6.8 shows the required hardware resources for the localization module implementation. Latency estimate relies on many constraints which must be defined. Generally, latency computation is impacted by the image resolution, the number of lines in the histogram beam, and the parameter h used in extraction contact points method. One image row must be stored in a line buffer before activating (Left and Right) scan module. Hence, latency linearly increases with horizontal resolution. The latency computation in table 6.8 is defined as follows:

- As the image resolution is 640×480, 640 clock cycles are required to store one image row.

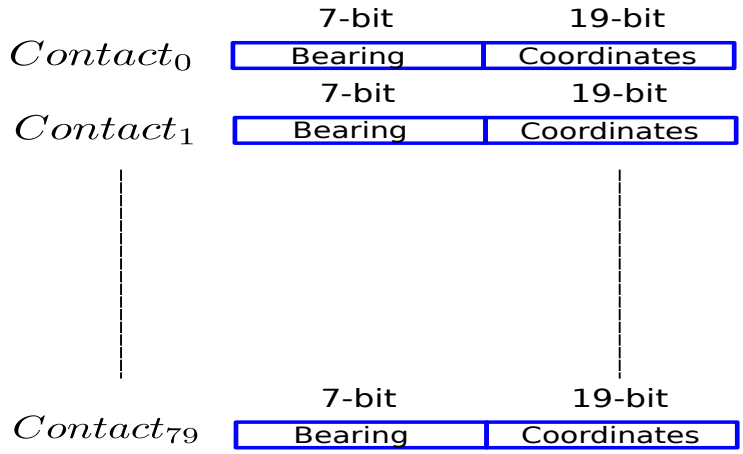


Figure 6.33: Composition of extracted contact points matrix.

Table 6.8: Hardware resources for localization implementation

	Resources
Luts	10433
Slice Registers	5642
Target	Zynq-7020
Latency	3646 clock cycle

- If the parameter h takes 5 as value, $5 \times 640 = 3200$ clock cycles are induced since it represents the minimal valid height of a detected triangle.
- After receiving 5 image rows, the first possible contact point could be produced after 6 clock cycles. So, total computed latency is 3646 clock cycles.

6.9 Free space detection module

Localization module takes *Bear* and *Cor* as outputs. Polar histogram matrix (namely *Bear*) is one-dimensional array of 80 elements mapped to the 80 selected bearings in our VGA image. Each matrix element is 8 bits wide. *Bear* matrix could be either stored in a block memory BRAM or in slice registers. If a block BRAM is chosen for storage, an additional cost is induced in terms of latency time. On the other hand, if slice registers are selected, an additional cost is paid in terms of hardware resources. In our design, slice registers have been selected in order to store the polar histogram *Bear* because the priority is given to reduce latency of the total system.

As pictured in figure 6.34, Free space detection hardware module takes po-

lar histogram matrix *Bear* as input and peaks matrix *Peak* as output. *Peak* matrix is one dimensional matrix composed of 32 elements. Each matrix element is 16 bits wide. The MSB bit among these 16 bits is used for mapping between peaks and obstacles. In other words, if MSB bit of a peak element in the peaks matrix is set 1, the peak element and the previous peak element in the peaks matrix belong to two different obstacles. On the other hand, if this bit is set to 0, the peak element and the previous peak element belong to the same obstacle object.

Figure 6.38 shows a peak vector decomposition. For a peak of size 16-bits, 7-bits are used in order to represent defined bearings in our image. As already noted, polar histogram beam is composed of 80 lines. These 80 lines are mapped to 80 bearings in a specific vision champ. The other 8-bits represent the peak amplitude. In other words, it represents the number of over-threshold pixels of a detected peak in a specific bearing.

As pictured in figure 6.35, the architecture of a 3-state machine is proposed in order to extract peaks from polar histogram. The state machine operates as follows:

- In the first state $IDLE$, *Bear* matrix elements are checked in order to seek peaks in this matrix
- state A is triggered when two sequential peaks are detected. A loop is executed to compute the two values A_1 and A_2(introduced in section 3.7).
- In state B, the two computed values (A_1, A_2) are compared to a threshold. Then a decision is made in order to decide whether the two sequential peaks are derived from the same obstacle object or not.

The output peak encodes the relation to neighbouring peaks by MSB bit. Additionally,This output includes peak bearing and amplitude. Figure 6.36 shows hardware design of state $IDLE$. $Count2$, and $Count1$ are bearings indices generator. Each generator value is represented using 8 bits. They are used to read the bearings measurement stored in polar histogram matrix. As state $IDLE$ outputs every two sequential peaks, $Count1$ and $Count2$ values denote these two sequential detected peaks. Among these two peaks, the first peak detected is addressed using $Count1$. Peaks are basically detected by comparing each bearing measurement $Pol(count2)$ to its neighbouring bearings measurements ($Pol(count2+1)$, $Pol(count2-1)$). If a new peak is detected, $An1$ gate outputs 1. Hence, the peak value $Amp1$ forwards to the next state. A subtraction is performed between $Count1$ and $Count2$ to compute the bearing difference. This value (namely def) represents the distance between the two detected peaks.

Figure 6.37 shows the hardware design of states (A, B). In state A, $A1$ and $A2$ quantities (already presented in 3.7) are computed for each two sequential

Figure 6.34: External view of free space module.

peaks. $Count3$ is bearings indices generator. This counter allows to scan bearings measurements values located between the two peaks (which are addressed by $Count1$, and $Count2$). The counter $Count3$ counts from 0 to $Step$. $Step$ represents the middle distance between the two peaks. As the counter value of $Count3$ is less than $Step$, bearings measurements values at u_1, u_2, and $Count2$ are read. These 3 values are used to compute the following sums:

- $A2$ is the accumulation of the sum of the subtraction result performed between the two bearings measurements $pol(count2)$ and $pol(u_1)$. $A2$ is 14-bits wide.
- $A1_1$ is the accumulation of the sum of bearings measurements $pol(u_1)$. $A1_1$ is 14-bits wide.
- $A1_2$ is the accumulation of the sum of bearings measurements $pol(u_2)$. $A1_1$ is 14-bits wide.

When $Count3$ value is equal to $step$. In this case, the following operations are executed in state B:

- The sum operation $A1$ is performed between $A1_1$ and $A1_2$ outputs. $A1$ is represented using 15 bits
- The output of the sum operation $A2$ is multiplied by a threshold th. The choice of th value is critical because the decision of joining peaks (Whether the two peaks belong to one or two obstacles) is made using th. The result of this product is represented using 19 bits.

This result is compared to A_1 output. If A_1 output value is greater than the aforementionnned result, MSB bit of the output peak vector is set to 1. Simultaneously, the lower 8 bits of this peak vector are set to $pol(Count2)$, and the other peak

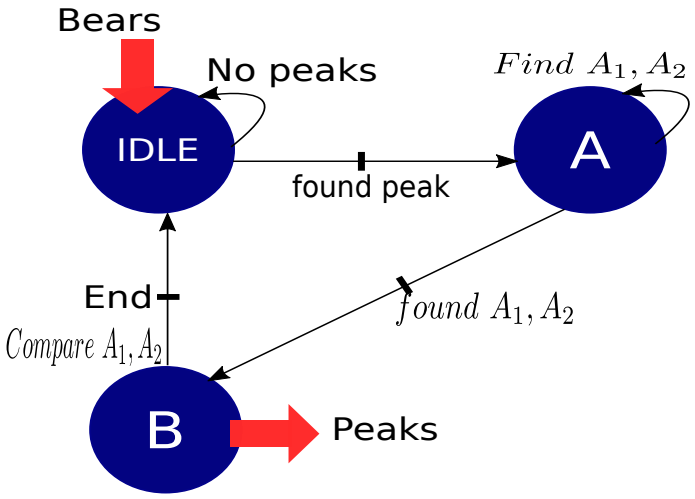


Figure 6.35: Free space detection module hardware design.

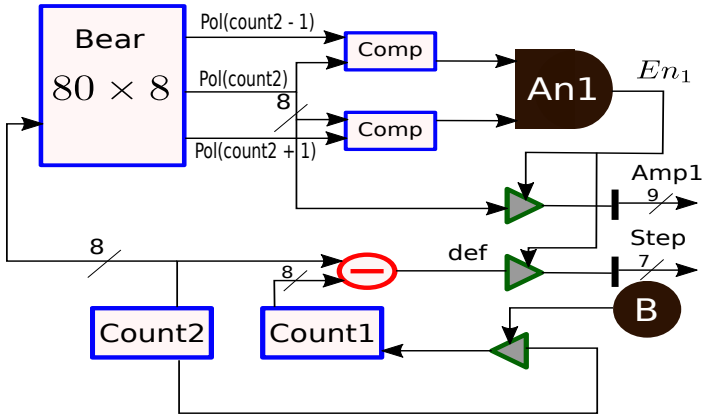


Figure 6.36: State *idle* hardware design.

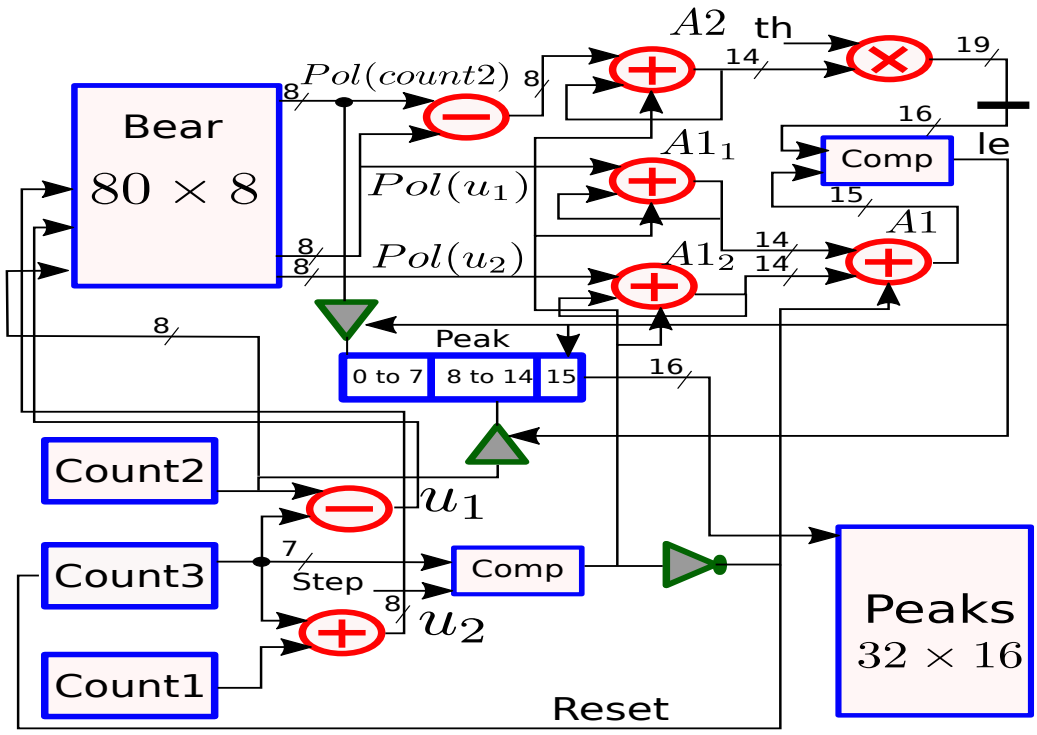


Figure 6.37: Hardware design of states A, B.

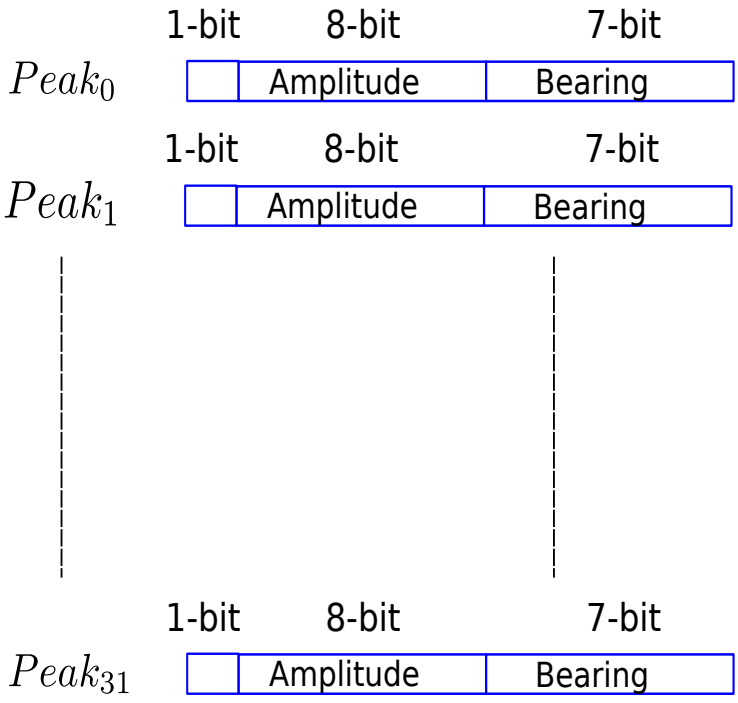


Figure 6.38: Composition of peaks matrix.

Table 6.9: Hardware resources of Free space detection module

	Resources
Luts	1630
Slice Registers	723
Target	Zynq-7020
Latency	16 clock cycle

vector bits are set to $Count2$. Peak vector is stored in the peaks matrix which is a 32-peaks wide.

6.9.1 Resource Utilization and Latency

Table 6.9 illustrates hardware resources used for the free space hardware module. Because registers are used to store peaks matrix and polar histogram, an additional cost in terms of hardware resources is induced. Latency cost of this block depends on the number of detected peaks and the distance between these peaks. The minimum detection time (latency) of the first possible free space between two sequential peaks is 16 clock cycles. The maximum execution time is 280 clock cycles per frame. This estimated value is computed under strict conditions such as a $Bear$ matrix includes 25 peaks (which represents the maximum number of detected peaks in our implementation).

6.10 2D Occupancy Map Reconstruction

6.10.1 Contact Points selection Module

Contact points selection module takes $Peaks$ matrix and $Contacts$ matrix as input. $Selected\ Points$ matrix is the output of this module as pictured in figure 6.39. A selection of best contact points is accomplished in this module. The selected points are used to build the occupancy grid map.

As depicted in figure 6.40, a state machine design of 4 states is implemented for this task. The functionality of this state machine is described as follows:

- In the first state $IDLE$, the peaks which mapped to the same obstacle object are read from the peaks array. This process is performed by checking the MSB bit of each peak element in the peaks array.

- In state A, we select the contact points candidates (from contact points matrix) which could be assigned to one or more peaks (peaks that mapped to one obstacle object) in the peaks matrix. The selection of contacts candidates is based on bearing index, as every peak and contact point uses 7 bits for bearings representation.

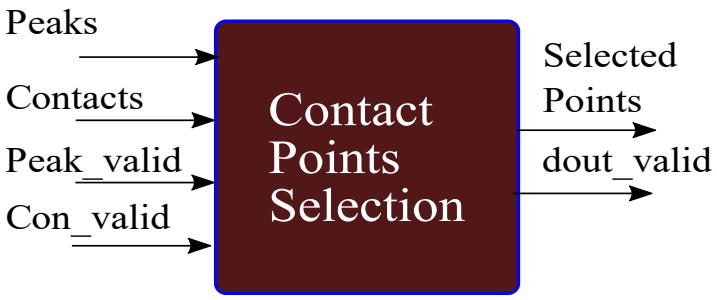


Figure 6.39: External view of contact points selection module.

- In state C, a comparison is performed between these two 7 bits in order to decide whether this contact point could be a candidate or not. The contact points candidates are then stored in a temporary matrix. The 3 states $IDLE$, A and C are repeated until a new obstacle peak is detected.
- When a read peak from peaks matrix refers to a new obstacle object, state D is executed. In this state, contact points candidates matrix is called in order to choose the best contact point. As noted in the theoretical presentation of this method, this selection is based on the nearest point to $focus$. A comparison is done between candidates points and the vanishing point $focus$ in order to choose the best point.

Figure 6.41 illustrates the hardware design of each state. $Peaks$ matrix is 32 peaks wide. Each peak is represented using 16 bits. For a peak of size 16 bits, 7 bits encodes the bearing. $Contacts$ matrix is composed of 80 point. Each contact point is 26 bits wide. Similarly, 7 bits are used to represent the bearing. $Count1$ generates the addresses to read $Peak$ matrix. $Count2$ also provides the addresses to read $Contacts$ matrix. These addresses are each 7 bits wide. Bearings values encoded in the peak matrix and contact matrix are extracted. In state C, a subtraction is performed between the two bearings values. The subtraction result is then compared to a threshold. If the subtraction result is less than a selected threshold, an enable signal is generated to store the contact point in a temporary matrix (contact points candidates matrix).

If the MSB bit of a peak matrix value is set to 1, state D is executed. In

Table 6.10: Hardware resources of contact points selection module

	Resources
Luts	349
Slice Registers	386
Target	Zynq-7020
Latency	16 clock cycle

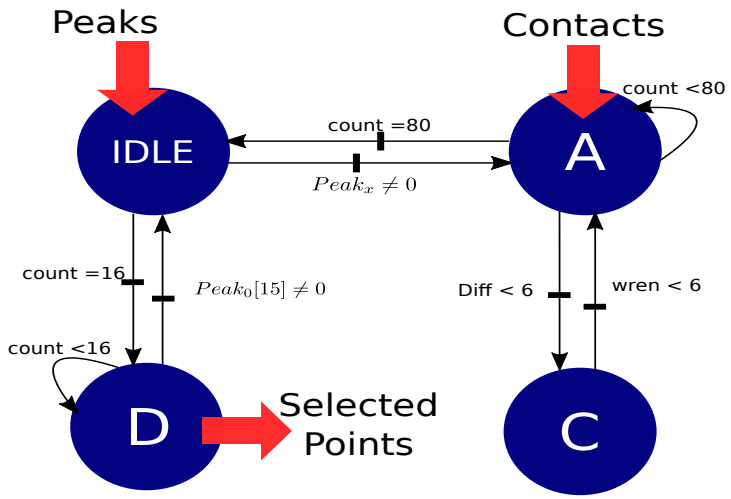


Figure 6.40: State machine design of contact points selection module.

this state, *Counter*4 generates the necessary addresses to read the temporary matrix. The selected contact points are produced from this state. The output matrix of selected contact points is an array of 32 elements.

6.10.1.1 Resource Utilization and Latency

Table 6.10 illustrates the amount of hardware resources for the selection of best contact points. The minimal latency induced to output the first possible selected contact point is 87 clock cycles. The maximal execution time is about 2100 clock cycles per frame. This latter time is an estimated time computed under strict conditions such as 25 detected peaks (The maximum number of peaks)and a *Cor* matrix includes 80 contact points (The maximum number of contact points).

6.10.2 Map Reconstruction

Selected contact points matrix is delivered to software part using *Xillybus* IP core (introduced in 4.8). In the simplest form, the *Xillybus* interface with application logic was designed for a direct connection with either a standard FIFO or RAM memory. In this application, a FIFO component is used to interface with the application logic. FIFO component works with data in widths of 8 bits, 16 bits

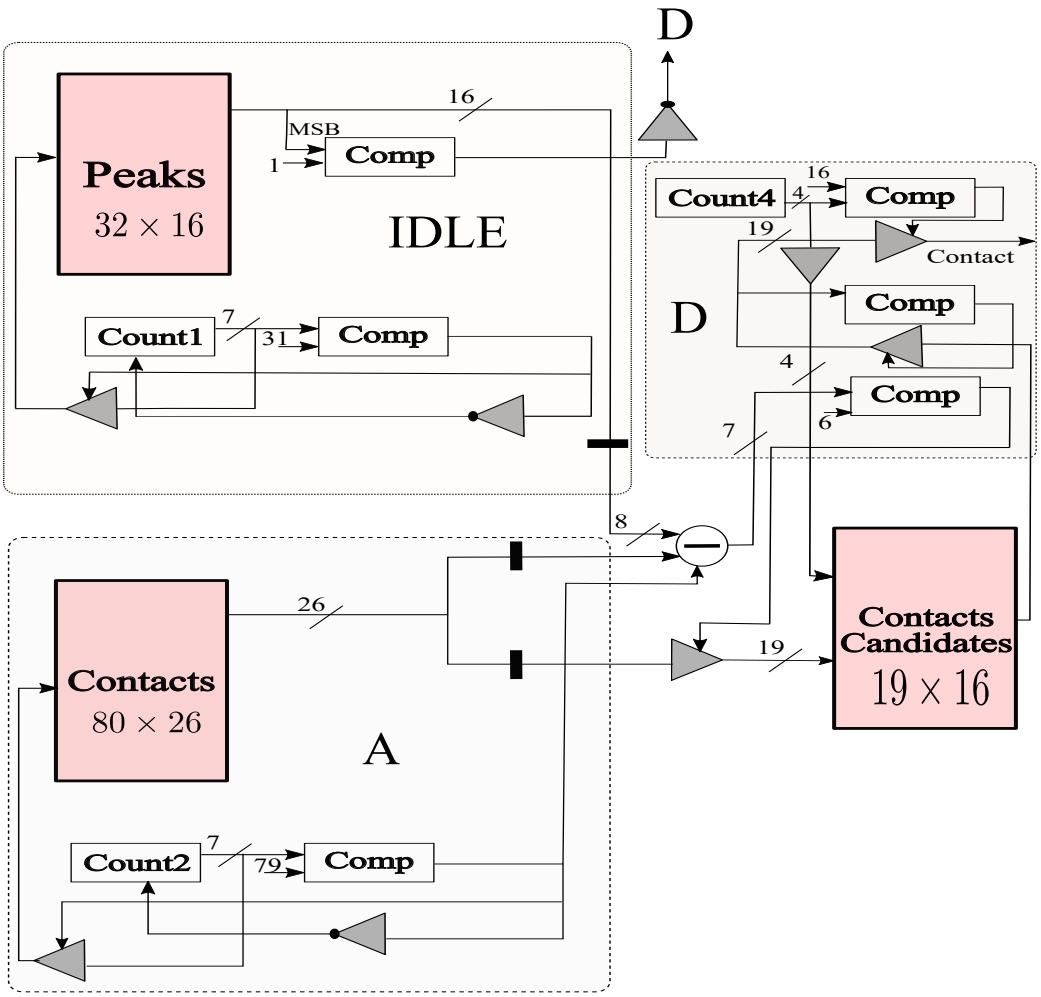


Figure 6.41: Hardware design of state machine.

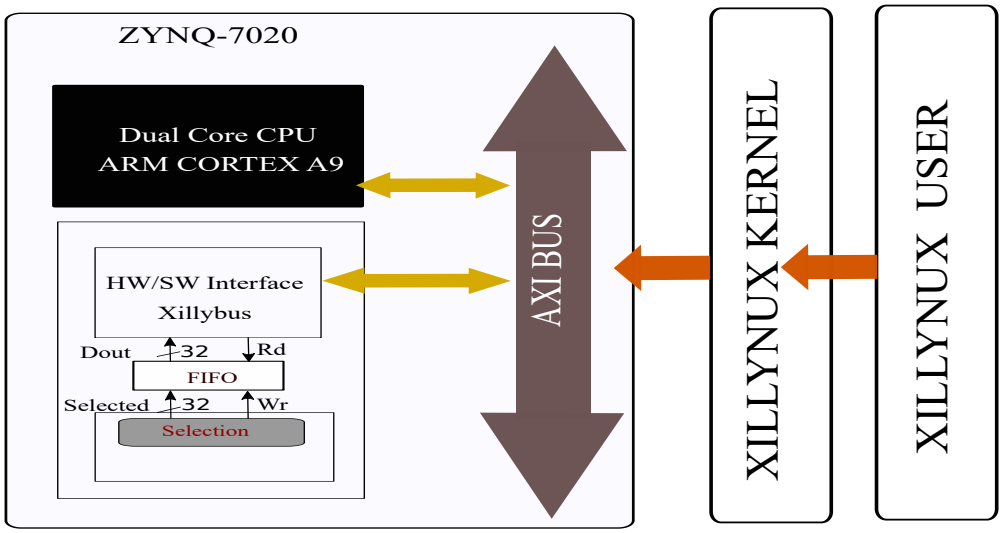


Figure 6.42: Selected points forward to the software part.

or 32 bits. Because each contact point is 19 bits wide, the logic vector will be zero-extended to 32 bits wide. The configuration of FIFO takes place during the formation of *Xillybus* core. FIFO component outputs the selected contact point (called *Dout* in figure 6.42). *Dout* is read by *Xillybus* core. Additionally, *Xillybus* core outputs an enable read signal *Rd* to FIFO. When *Rd* is set to 1, the core expects valid data to be present on the ARM core on the following clock. In the embedded Linux kernel, contact points are read from a device file.

A software module is developed in order to map contact points to the coordinate system of the occupancy grid. In details, the software code implements the equations 5.17, 5.18, 5.19. The execution time of the software part is $56\mu s$ in order to produce the coordinates of one selected point in the world frame of the occupancy grid. The maximum number of selected points is 32 points per frame. Hence, the maximum execution time of the software part is about $1.8ms$ per frame.

Chapter 7

Results and Experiments

Contents

7.1 Introduction

The previous chapter has introduced the three proposed architecture (Figures 6.1, 6.2, 6.3) and described the hardware design of each module in these architectures. An external view illustrating input and output signals is shown for each module. An internal view that represents the structural design of the component is shown. The architectural designs are analyzed and synthesized in terms of hardware resources, latency, and overall performance.

Two platforms are used to validate thesis work:

- Virtex-6 is used to implement first and second hardware architectures because the two architectures have consumed a huge amount of hardware resources.
- Zynq based kit (Zedboard) is used to implement the proposed architecture for the 2D occupancy map reconstruction.

In this chapter, we present the results of the tests done to validate the proposed systems. An experiment is presented to validate the proposed architecture for obstacle detection and localization with a comparison between hardware and software implementation. Another experiment is also presented for the reconstruction of 2D occupancy grid map. Finally, we present the integration of this later in the camera-belt platform.

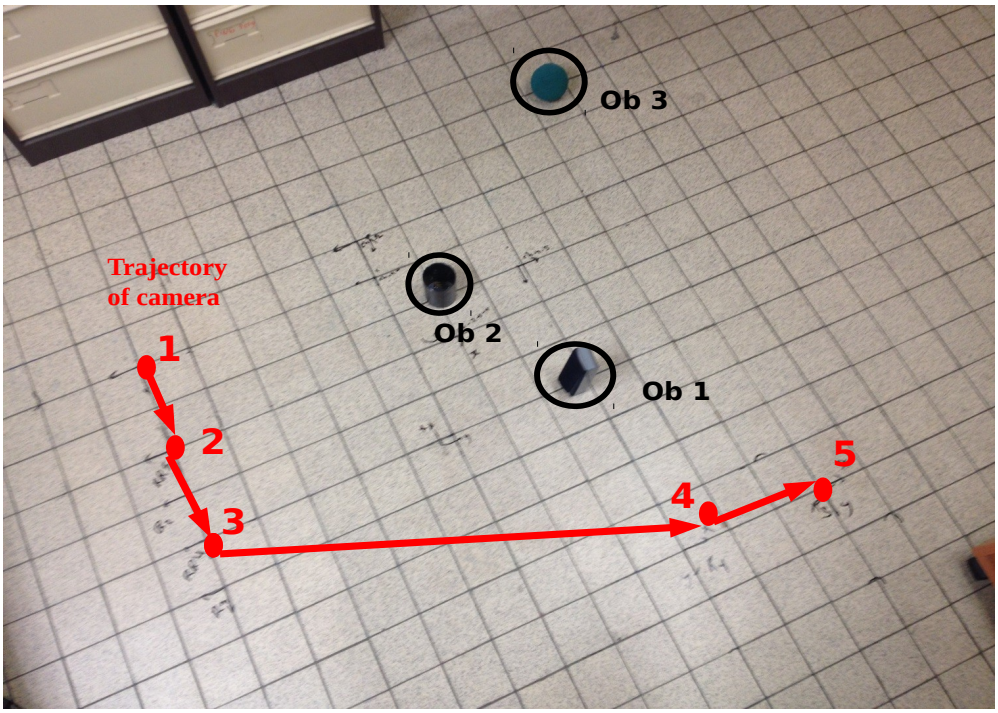


Figure 7.1: Overhead view of test environment.

7.2 Obstacle Detection and Localization System Experiment

The primary focus of this experiment is to test and validate the proposed system model (The following results and comparisons are produced by the second architecture except the comparison of the bird's eye transformation between software and hardware implementation). Figure 7.1 shows the general overview of the environment of test. As pictured in this figure, camera positions are preselected to draw a given trajectory. The selected indoor environment includes floor tiles which could be a source of noise in the resulting binary images.

A webcam camera is used for image acquisition. The acquired images are of VGA resolution. As pictured in figure 7.2, images (b),(c),(d) and (e) show selected obstacles objects. The dimensions of tested obstacles are as follows:

- Width : between 1.5 cm and 60 cm.
- Height : between 5 cm and 55 cm.

Figure 7.3 shows the acquired images and IPM results at 2, 3, 4, 5, positions. Figure 7.4 also shows the resulting binarized images, and detected contact points at these positions.

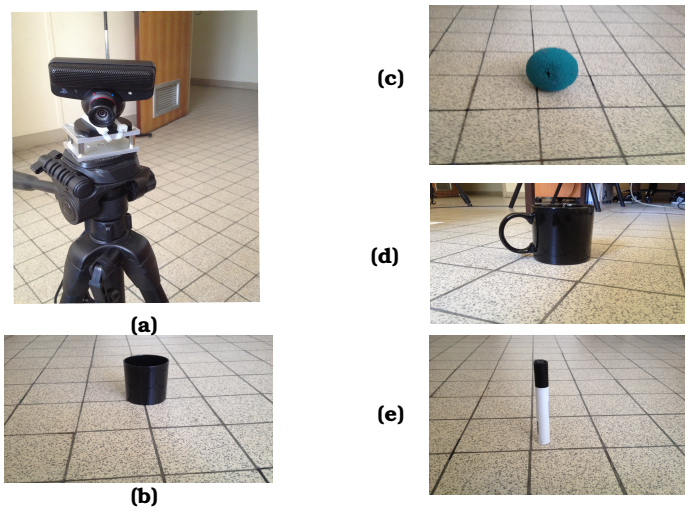


Figure 7.2: Camera and selected obstacles in the test.

7.2.1 Comparison of the results between hardware and software implementation

In this section, we compare the hardware implementation results with the software implementation of the proposed system. Figure 7.11 shows two images acquired at two distinct instances. Three obstacles are pictured at these two instances. Figures (7.5) (7.6) illustrate the image resulting after IPM method in the hardware and software implementation.

Figure 7.7 illustrates the resulting images from gaussian filter. The same Gaussian kernel is used for hardware and software implementation. In the hardware implementation, all arithmetic operations are executed by shifting because kernel values are divided by 2, 4, and 8. In the software implementation, double-precision floating-point representation is used for kernel values.
Figure 7.8 shows the produced binary images. As depicted in this figure, an important difference is observed in the results due to the difference of the Ots's threshold value between hardware implementation and software implementation.
In practice, fixed point format is used to store and manipulate the signals in arithmetic operations in the hardware implementation. When Otsu's algorithm is implemented, complex arithmetic operations are performed between logic vector of large width. Additionally, an approximation is done after each arithmetic operation (by removing a specific number of the least significant bits from the resulting signal). Rounding numbers during signal processing naturally yields quantization error, the deviation between actual values and quantized digital values. Since the gaps between adjacent numbers can be much larger with fixed-point format when compared to floating-point processing, round-off error can be much more pronounced.

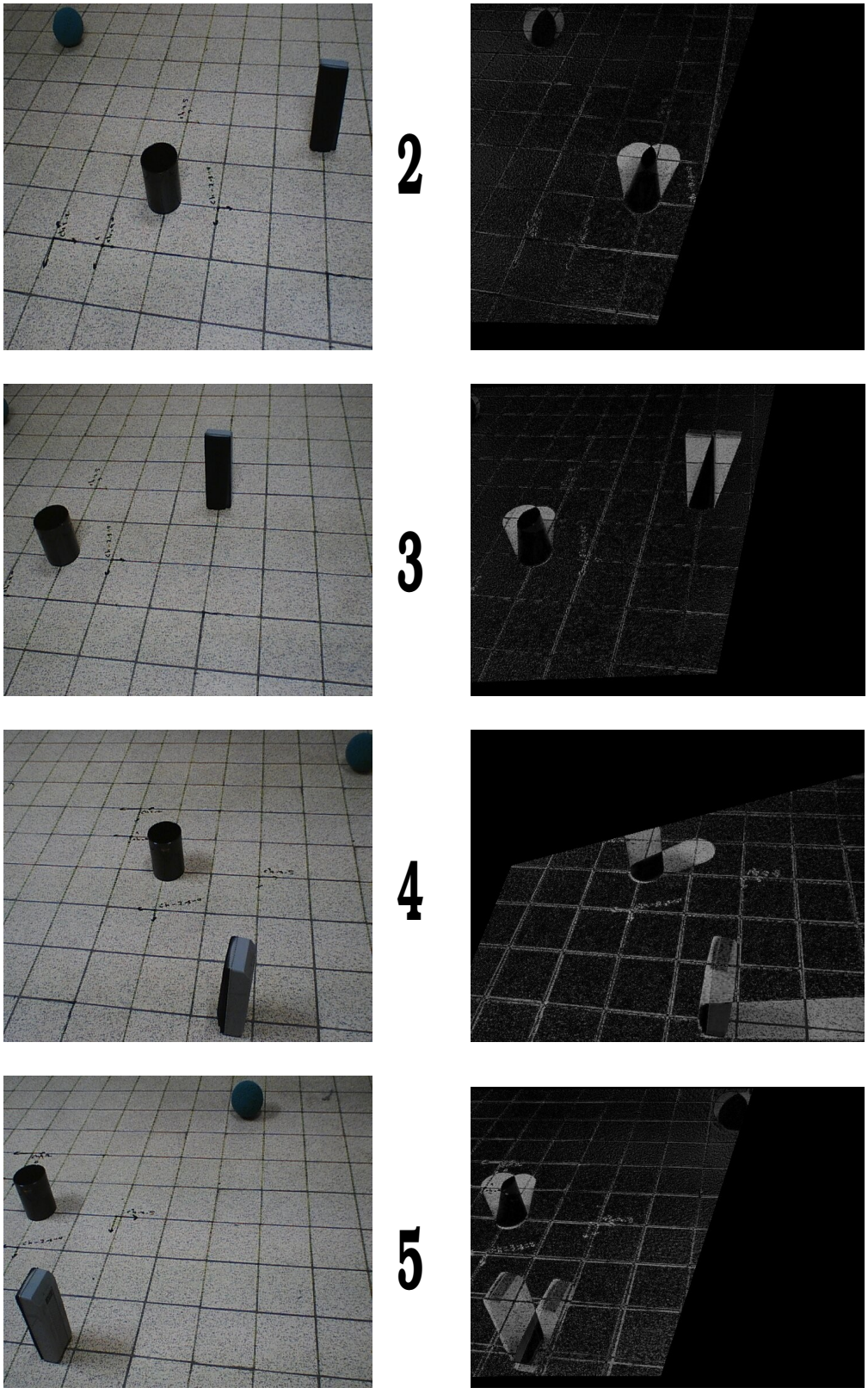


Figure 7.3: Acquired images and IPM results.

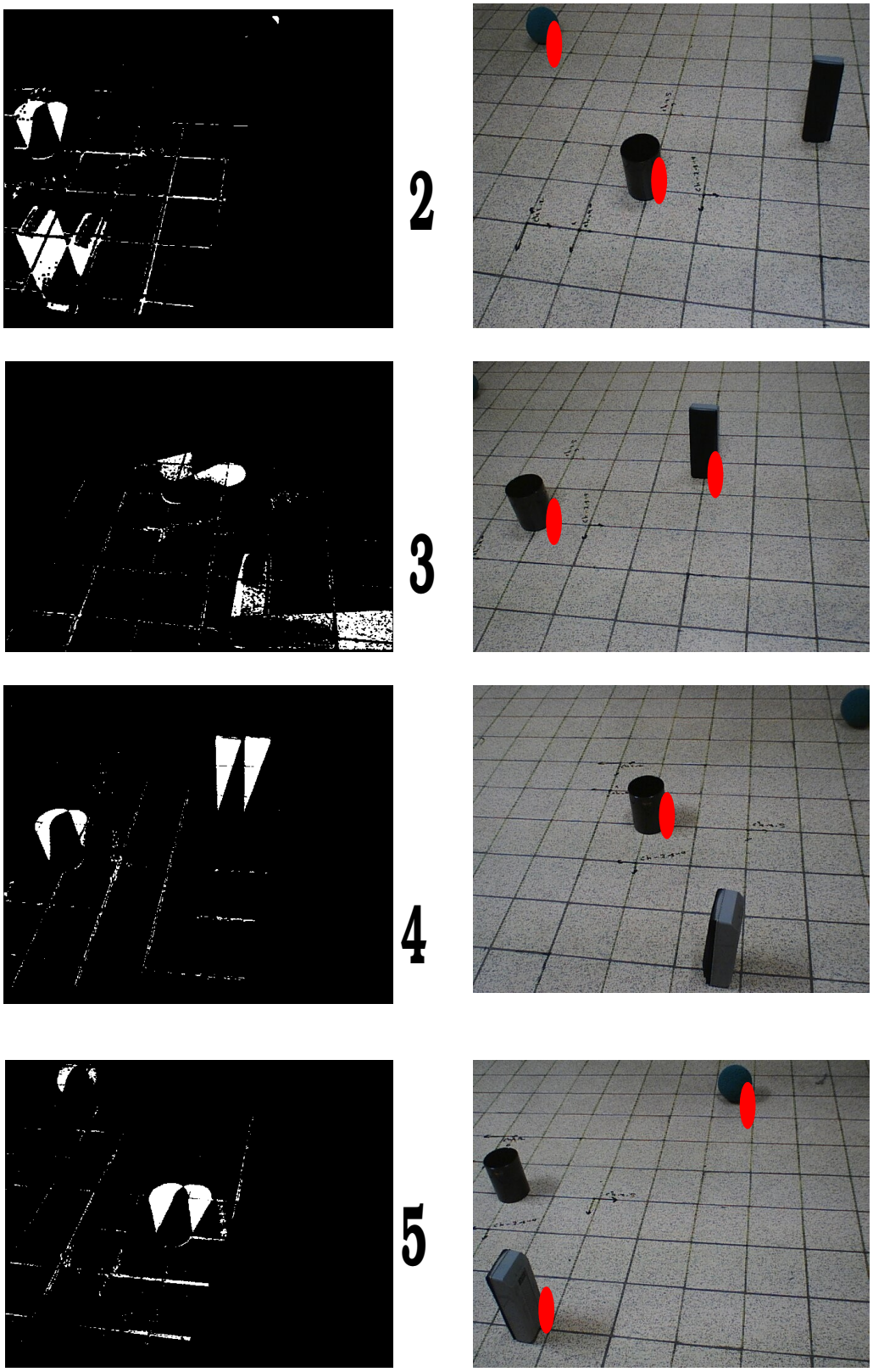


Figure 7.4: Results binarized images and contact points.

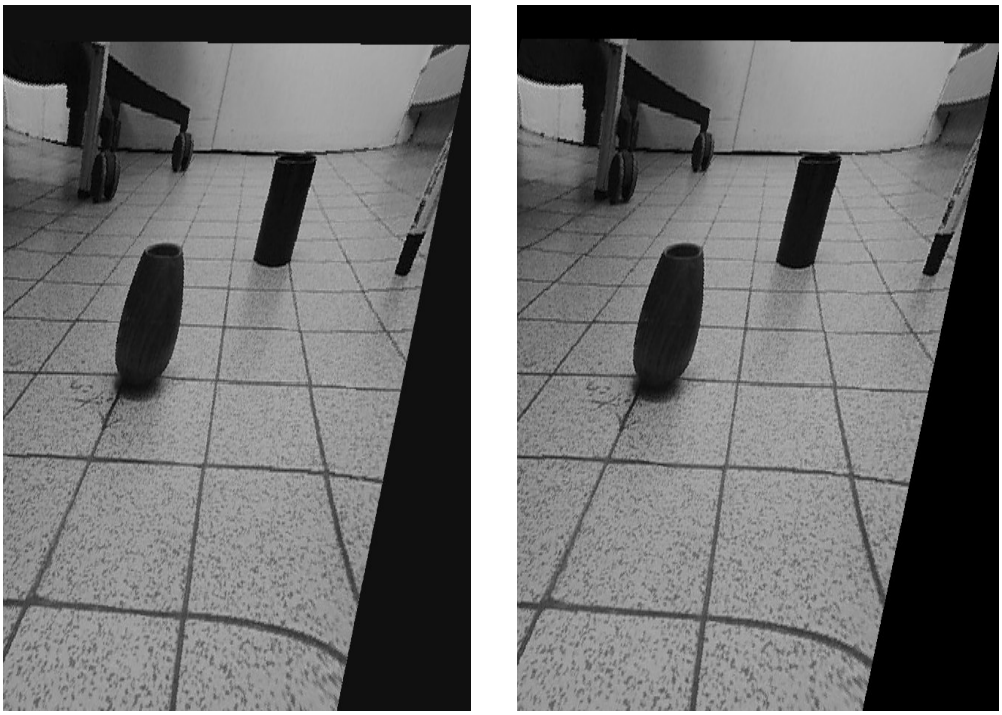


Figure 7.5: Transformed image (Left: Hw result, Right: Sw result) .

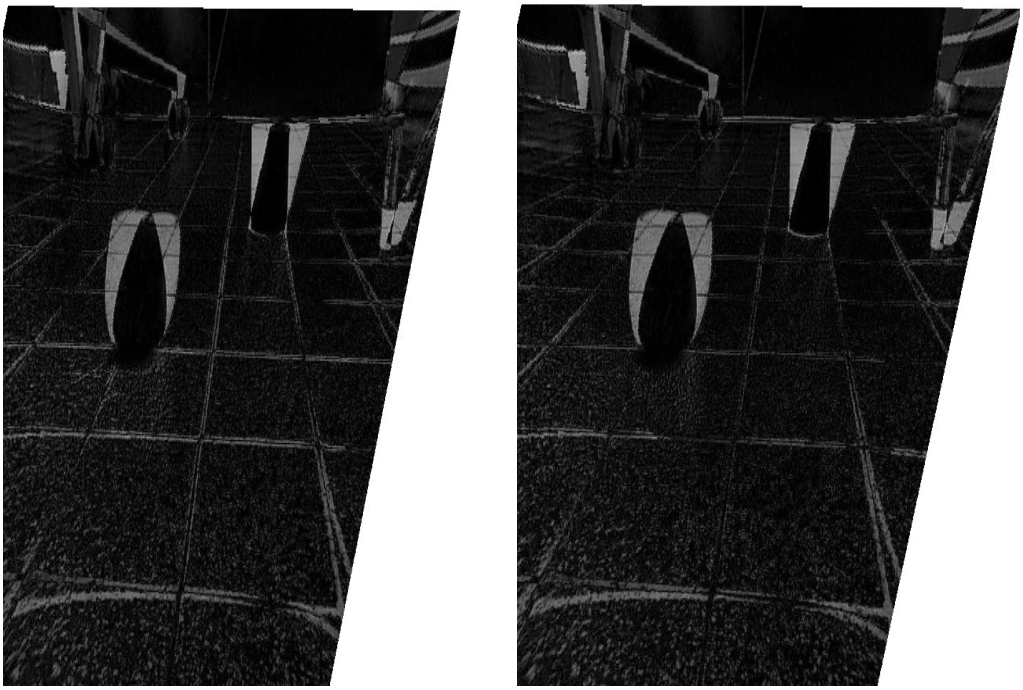


Figure 7.6: IPM image (Left: Hw result, Right: Sw result) .

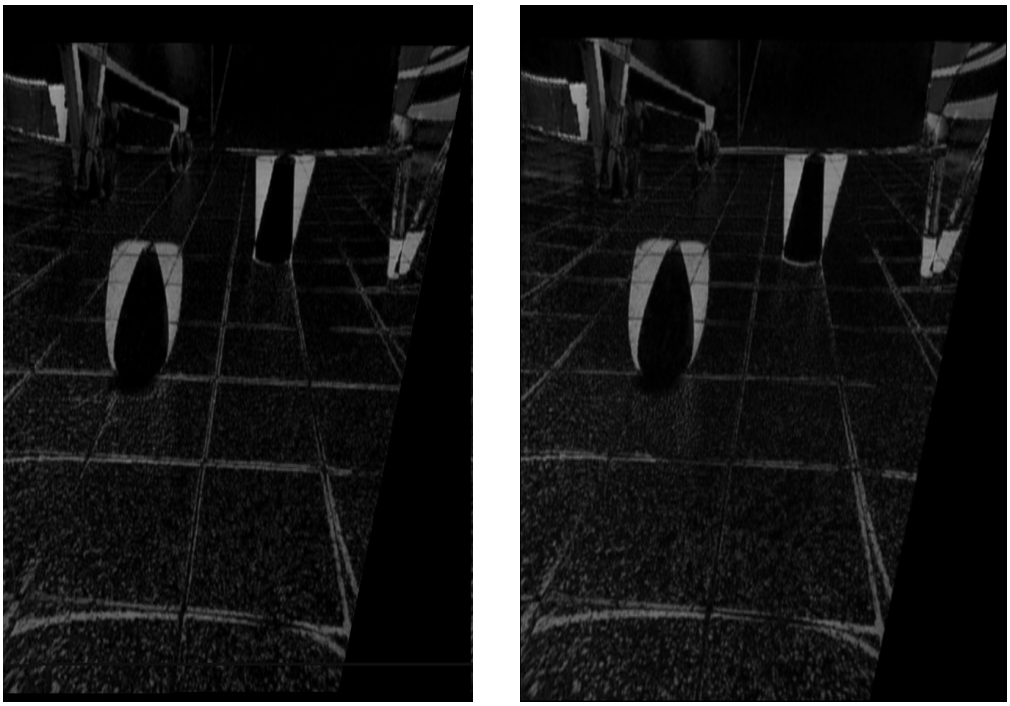


Figure 7.7: Gaussian output image (Left: Hw result, Right: Sw result) .

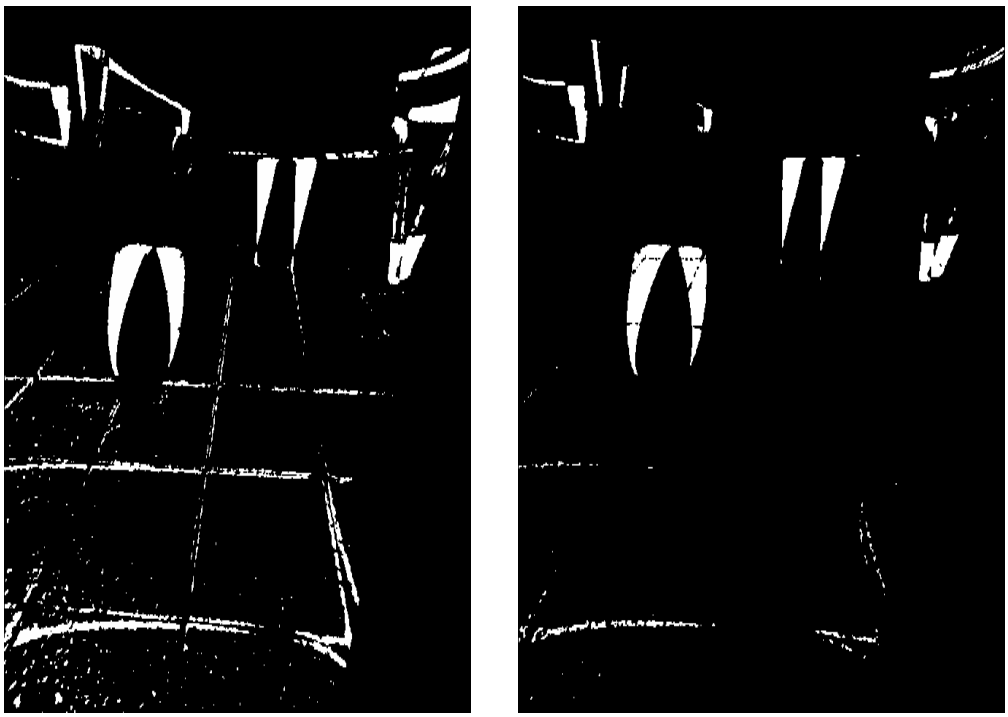


Figure 7.8: Binarized image (Left: Hw result, Right: Sw result) .

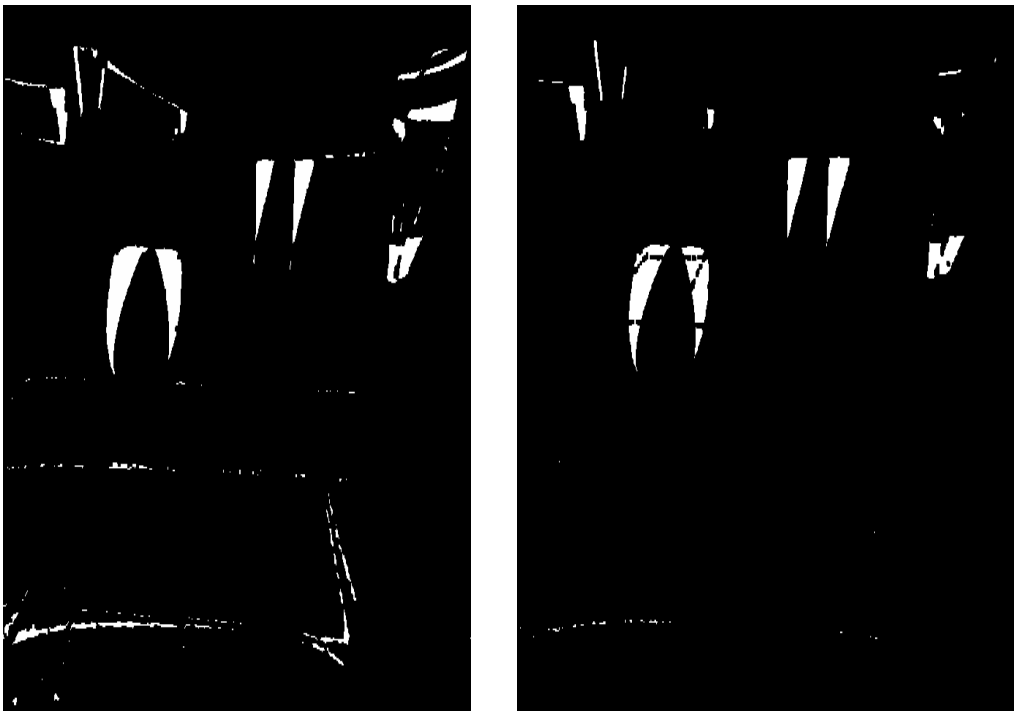


Figure 7.9: Eroded image (Left: Hw result, Right: Sw result) .

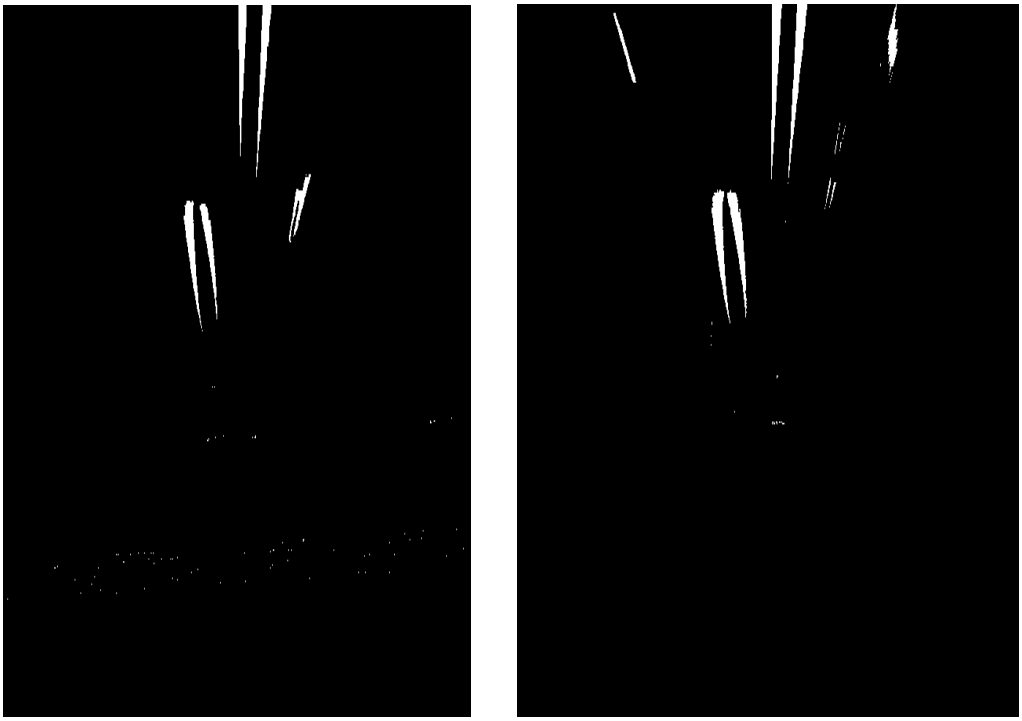


Figure 7.10: Bird's eye image (Left: Hw result, Right: Sw result) .

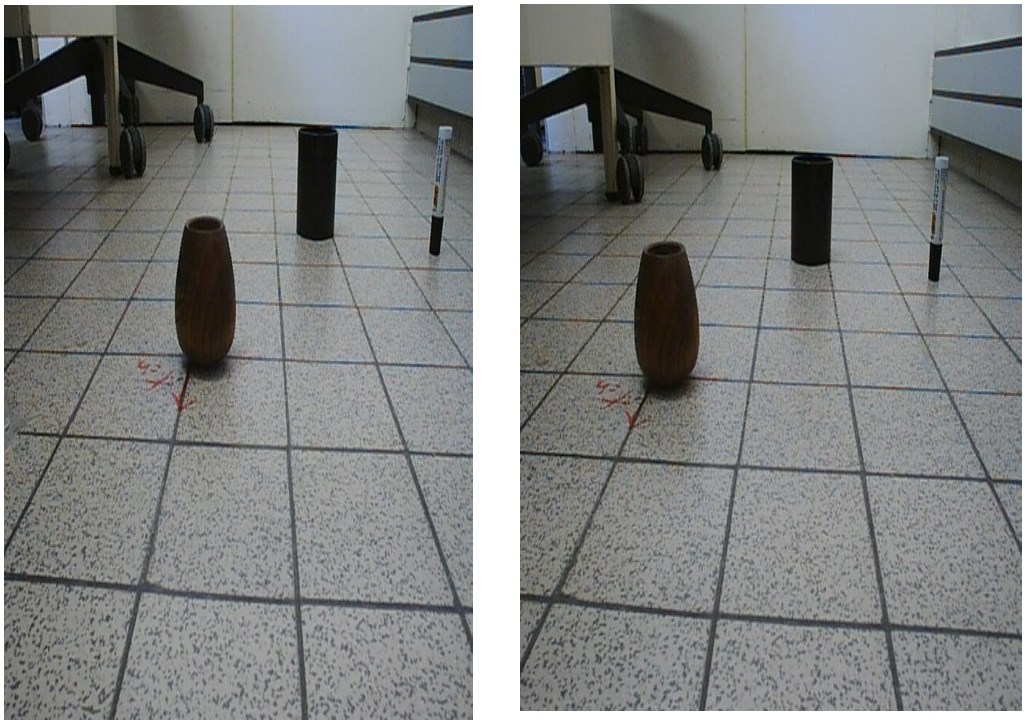


Figure 7.11: Acquired images at t_n and t_{n+1}.

Figure 7.10 shows bird's eye transformation results. The transformation matrix used in the hardware implementation is a an approximated matrix of the matrix used in the software implementation. Because a fixed-point format with 14 bits is used to represent the fractional part of bird's eye matrix values, an important difference is observed between hardware and software implementation. Since division operation is executed between two logic vectors of large width in bird's eye transformation, this produces less precise values. In order to produce higher precise values, a representation with 20 bits is required to represent the fractional part of bird's eye matrix values in the hardware implementation.

7.2.2 Comparison to the state of art

All the subsystems composing each proposed system (the first and second architecture) to validate are executed on the simulation environment (ISim) using Virtex-6 platform. The estimation of power consumption in our architecture is around 3.9 watt. This value is produced using Power Analyzer tool in ISE software (Integrated Synthesis Environment). The computational latency time is about 4610 clock cycles for the second architecture while 126897 clock cycles are required to perform the first architecture. Table 7.1 illustrates the comparison between our implemented system using the second architecture and other obstacle detection

systems that exist in the state of art.

In [He 2012], authors proposed a method to detect obstacles around vehicles by optical flow computation based on inverse perspective mapping. Authors claim that their method can produce good optical flow values with any type of trajectory. The presented system is a GPU-based implementation (Intel core i5-2400, NVIDIA GeForce GTX 550 Ti platform). Compared with our proposed system, we'll find:

- Optical flow is computationally intensive method when it is compared to the IPM method.
- Because of the heavy cost of the computation load, the system in [He 2012] is performed using a GPU-based architecture. The power consumption of the employed GPU platform is 12 *watt* (only GPU card) in idle state (No load) and the maximum power consumption value is 116 *watt* [Geforce 2016b]. It is clear that the power consumption of such platform is greater than the power consumption of an architecture based on FPGA.

In [Yankun 2011], a rear obstacle detection system was proposed using a single rear view camera. A hierarchical detecting strategy is used to achieve high detection rate and low false positives. Coarse detection is based on inverse perspective mapping method, and fine detection is based on a multi-scale integral image algorithm. Compared with our system, we'll find:

- In [Yankun 2011], three sequential frames are needed to perform the system while two frames are required in our system.
- The proposed system is limited to vertical edges of obstacles.
- The frame rate achieved in our system is better than the frame rate in [Yankun 2011] as depicted in table 7.1.

In [Cesar 2013], a GPU-based implementation of a method based on the height difference and "slope" between three-dimensional points (introduced in [Talukder 2002]) was proposed. The proposed implementation was performed using a stereo camera and an RGB-D sensor. Authors claim a significant gain in computational performance, reaching a speedup of almost 80 times in a specific instance. Compared with our system, we'll find:

- The proposed implementation requires a stereo camera and a RGB-D sensor while our proposed system requires only one monocular vision sensor.
- The obstacle detection method in [Talukder 2002] is a method based on 3D reconstruction where a higher computational cost is induced when it is compared to the IPM method.
- Because of the complexity of such method, GPU-based architecture was proposed in [Cesar 2013] to meet real time requirements. However, no power consumption analyse is reported. The GPU platform employed in [Cesar 2013]

Table 7.1: Other implementations of obstacle detection system

	platform	OD method	Frame rate
[He 2012]	CPU + GPU	optical flow	25 fps 640 × 480
[Yankun 2011]	PC 1.73 GHz	IPM coarse detection	30 fps 720 × 480
[Cesar 2013]	GPU	3D reconstruction	45.8 fps 640 × 480
[Bendaoudi 2012]	FPGA	stereovision	180 fps 640 × 480
ours	FPGA	IPM	201 fps 640 × 480

is the Intel i7 3770 CPU and the AMD RADEON 7950 GPU. The power consumption of GPU card in idle state (No load) is about 11 *watt*. This value can be increased up to 179 *watt* with maximum load [Techpowerup 2012]. As already noted, the estimation of power consumption of our design is about 3.9 *watt*.

- The frame rate achieved in our system is better than the frame rate produced in [Cesar 2013] as depicted in table 7.1.

In [Bendaoudi 2012], a hardware architecture is proposed for obstacle detection using stereo vision. The proposed system combines stereo vision algorithms to compute the disparity map, V-disparity image, and Hough transform for obstacle detection. The proposed architecture is implemented using Virtex-II FPGA based prototyping board. Compared with our system, we'll find:

- Hardware resources amount in [Bendaoudi 2012] is less than the hardware resources amount used in our architecture.
- The proposed system requires two cameras to perform a stereo vision system while one camera is needed in our system.
- The maximum frame rate achieved in our system is a bit better than the frame rate achieved in [Bendaoudi 2012].
- Latency time in [Bendaoudi 2012] (minimum detection time) is 5.5 ms while latency time (computed to produce contact points between obstacles and ground plane) of our architectures is 7.49 μs for the second architecture and 2.05 ms for the first architecture.

7.3 2D Occupancy Grid Map Reconstruction Experiment

In this experiment, TurtleBot (pictured 7.12), a low-cost personal robot kit, is used to carry the image acquisition. This robot consists of Yujin Kobuki base, a battery pack, a kinect sensor, an Asus 1215N labtop dual core, and a hardware mounting kit. Turtlebot combines all these aforementioned components into an integrated

Figure 7.12: TurtleBot kit [ROS 2016a]. Reprinted from "TurtleBot 2, Open-source robot development kit for apps on wheels", from http://www.turtlebot.com/.

development platform for ROS applications.

Turtlebot is provided with a Kinect sensor. This latter is composed of an RGB camera and a depth sensor. In our experiment, the RGB camera is only used for the image acquisition.
Basic software configurations must be loaded to ROS-based Turtlebot in order to make our data-set [ROS 2016b]:

- We start up the turtlebot using *roslaunch* package by which all processes can be controlled.
  - *roslaunch turtlebot$_b$ringup minimal.launch.*
- *roslaunch* tool is then used to launch remotely multiple ROS nodes via SSH application.
- Turtlebot is initialized to take the first position as the origin of the occupancy map.
  - *roslaunch currentposition tf.launch*
- *rosbag* package, a command line tool [ROS 2015], is used for working with bag files as well as code APIs for reading/writing bag files. Here, it is used to record odometry data during navigation.
  - *rosbag record /odom*
- Similarly, *rosbag* package is used for the storage of image stream via a bag file.

- *rosbag record /camera/rgb/image_raw*

The simulation is performed using ISIM tool from XILINX to validate the hardware architecture. The tested indoor environment is selected with strict conditions such:

- a reflective floor where objects shadows could be observed in the ground plane.
- a strong source of light which could produce shiny spots on the floor because of shiny surface.

Obstacles dimensions and shapes tested in this experiment (shown in figure 7.13) are introduced as follows:

- in terms of dimensions:
  - obstacles height between: 2 cm, 100 cm.
  - obstacles width between: 3 cm, 110 cm.
- in terms of shapes nature:
  - vertical and quasi vertical shapes ($ob1$, $ob2$,$ob7$).
  - objects with a large surface area, which comes into contact with ground plane ($ob4$).
  - objects having multiple contact points with ground plane($ob3$, $ob5$).
  - objects having no uniform shape ($ob6$).

Figures 7.14, 7.15 show some example results. These figures are introduced as follows:

1. The first column presents the images acquired at different positions. These images represent the second image acquired at t_{n+1}. Images acquired at t_n are not shown in these figures.
2. The second column presents the resulting binarized images. As pictured in these images, polar histogram beam is shown. The distance between detected contact points and the origin of the robot coordinate system should not exceed 2 meters.
3. The third column presents the field of view and the robot trajectory. Arrows represent robot translation in the ground plane while curves represent the rotation of robot about Z axis which is perpendicular to the ground plane.
4. The fourth column presents detected contact points and its localization in the image.

As depicted in these figures, good contact points are extracted even if the floor is reflective and contains shiny spots. The world is represented in the occupancy map pictured in figure 7.16 by Cartesian model. The point at which the two axes (X_w,

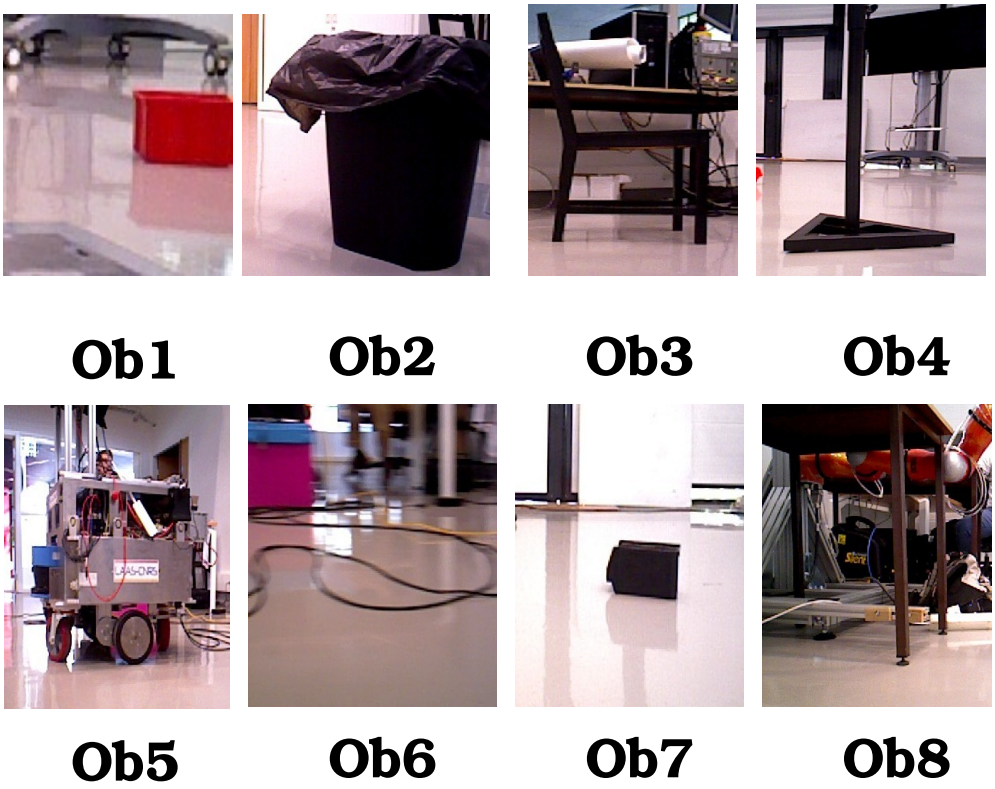


Figure 7.13: Obstacles objects used in experiment 2.

Y_w) intersect represents the origin of the occupancy map. This origin corresponds to the first position of the robot. Detected contact points in the image frame are mapped to and projected on the occupancy map. Projected points are each coloured with black circles whose diameter is 5 cm in the world coordinate system. This value also denotes the resolution of the resulting map.

The resulting map pictured in figure 7.16 is produced after hundreds of frames. The real occupancy of objects is coloured with red. Points circled in green represent false detected points. When analysing the results, we find:

- For *ob*5 and *ob*1, some detected points are located close to the objects because of the odometry errors. These sensors are subject to many sources of measurement error including wheel slip and gyro drift.
- For *ob*5, not all detected points in the upper side belong to this object. Some of these detected points belong to the cables located behind the object *ob*5.
- Similarly for *ob*6, detected points belong to these cables that exist in the indoor environments. Because cables have no quasi triangular shape, these objects produce some false points in some cases as pictured in this map.

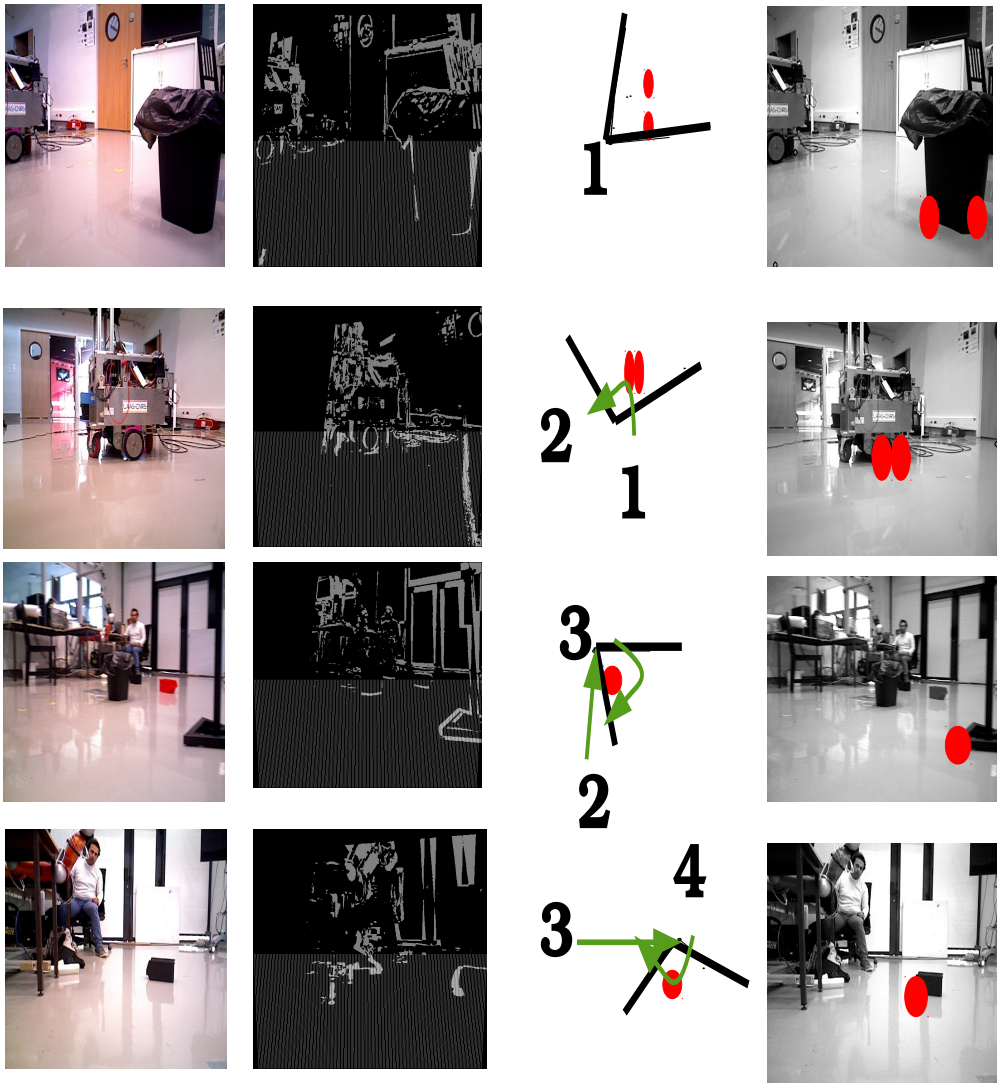


Figure 7.14: Examples of results 1.

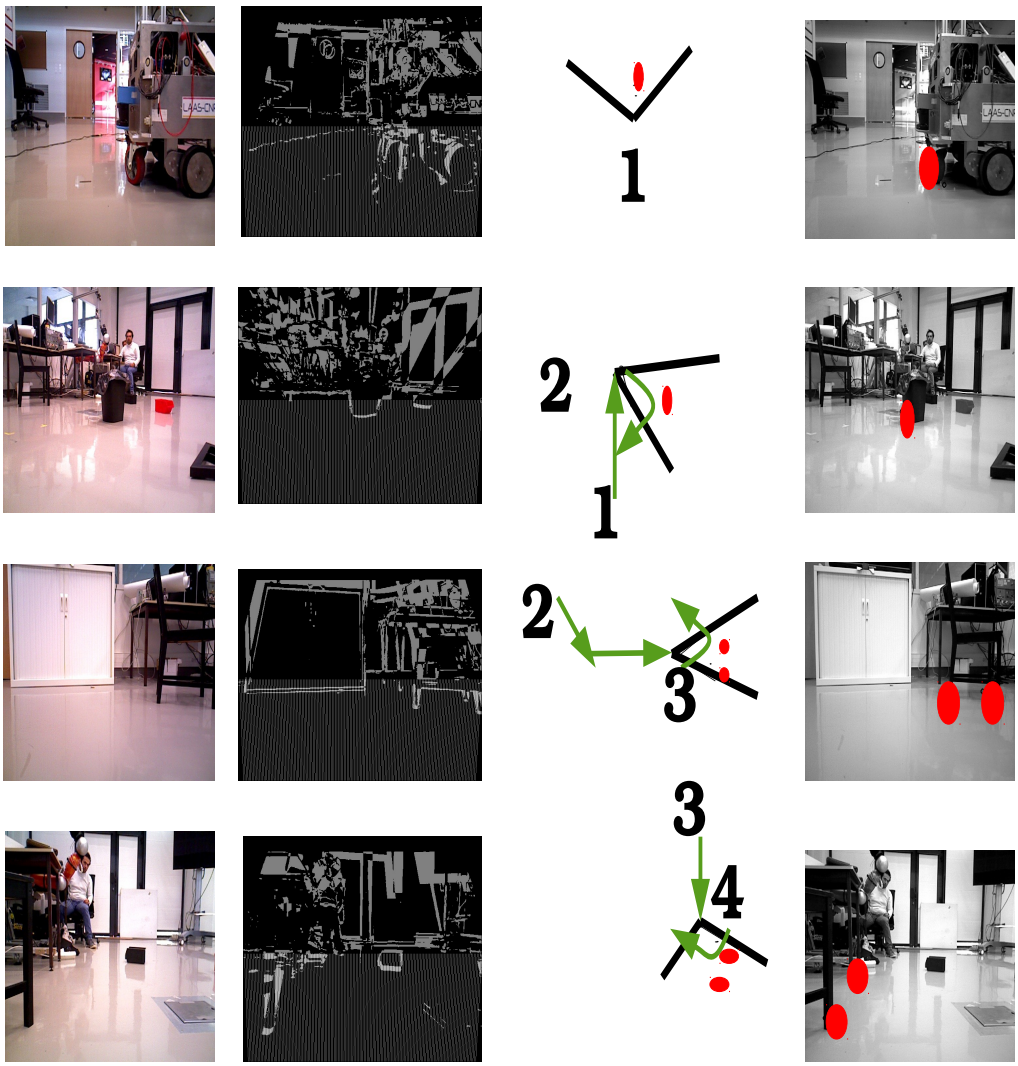


Figure 7.15: Examples of results 2.

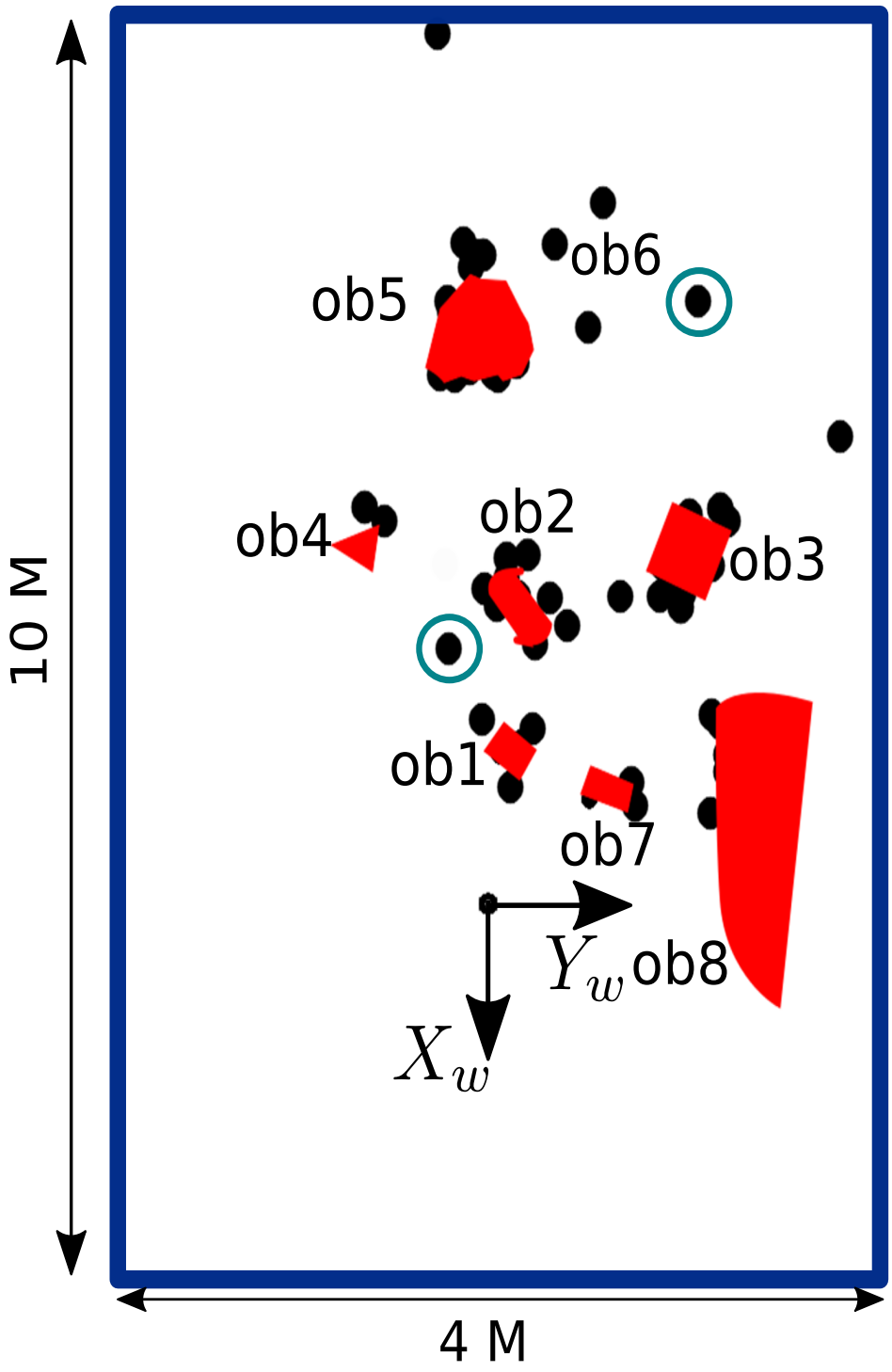


Figure 7.16: Map results.

Table 7.2: False detected points evaluation

	All false points	shiny surface	objects shadows
Hard conditions	16%	7%	9%
Normal conditions	8%	2%	6%

Table 7.3: Comparison to other architectures

	PLatform	Frame rate
[Peasley 2013]	CPU + GPU	30 FPS
[Benacer 2015]	FPGA	122 FPS
[Mattoccia 2014]	FPGA+ embedded ARM	20 FPS

- The false point detected between $ob2$ and $ob1$ is produced by the shadows of object on the reflective floor.

Figure 7.17 illustrates examples of false detected contact points. Two reasons lie behind these noisy points:

- Shiny surface such as the first image on the left side shown in figure 7.17
- Objects shadows on the floor such the other images in figure 7.17

7.3.1 Comparison to the state of art

Table 7.3 shows the comparison of the system implementation to other architectures in the literature. In [Peasley 2013], an approach is introduced for obstacle detection task based on 3D sensor. A 2D occupancy map is generated to determine the presence of obstacles. Compared with our system, we'll find:

- The proposed system is able to update the occupancy map at 30 FPS, while our system can update the occupancy map at 148 FPS.
- The platform used in [Peasley 2013] is Intel Core i7 processor with an EVGA GeForce 9800 graphics card. The power consumption of the GPU card is 105 *watt* (Full load) [Geforce 2016a] without considering the system power requirements while the estimation of power consumption in our design implemented on Zynq-7020 is 0.412 *watt*.
- An active 3D sensor is used to perform the system while a monocular passive vision sensor is used in our system.

In [Benacer 2015], an FPGA based architecture is introduced for obstacle and free space detection. The proposed method is based on stereo vision method. Authors claim the high efficiency and accuracy of the proposed system. Compared with our proposed system, we'll find:

- The proposed system runs at 122 frame per second while the hardware part of our system runs at 203 fps for the same resolution.
- The amount of LUTs resources used in [Benacer 2015] is less than the amount used in our system.
- The proposed system in [Benacer 2015] doesn't produce an occupancy map for the presence obstacles.

In [Mattoccia 2014], an embedded system based on 3D vision sensor is introduced for autonomous navigation task. Compred with our system, we'll find:

- The system requires two components: a stereo vision camera with FPGA processing and an embedded quad-core ARM board while our proposed system uses one board (Zynq based kit).
- A 3D camera is needed to perform the stereo vision system while one monocular vision sensor is used in our design.
- Obstacle detection task results in [Mattoccia 2014] seem to be more robust. However, the approaches used in [Mattoccia 2014] are more complex, thus resulting in 20 FPS as frame rate.

7.4 Camera-Belt Prototype

The thesis work is scheduled in the Camera-belt project. The goal of this project is to propose an embedded vision system, multi camera for obstacle detection task. An embedded development platform is made to test and implement the IP cores realized during this thesis. The heart of this platform is the Zedboard, a Zynq based kit. The hardware architecture of the occupancy map system is implemented on this platform using XILINX tools. The system prototype includes two levels of work: hardware level, and firmware level. The hardware level is achieved by LAAS-CNRS staff.

7.4.1 Hardware Level

Figure 7.18 shows the platform hardware parts. The platform is composed of:

1. The board of the vision sensor (Aptina MT9v034 [Aptina 2008] [Piat 2016]).
2. The interface card between the sensor and Zedboard [Piat 2016].
3. Zedboard card, the Zynq based kit [Avnet 2012].

The vision sensor is Aptina VGA CMOS image sensor. The camera board is made to output the pixel stream using LVDS interface. LVDS mode allows data transmission as a serial LVDS stream.

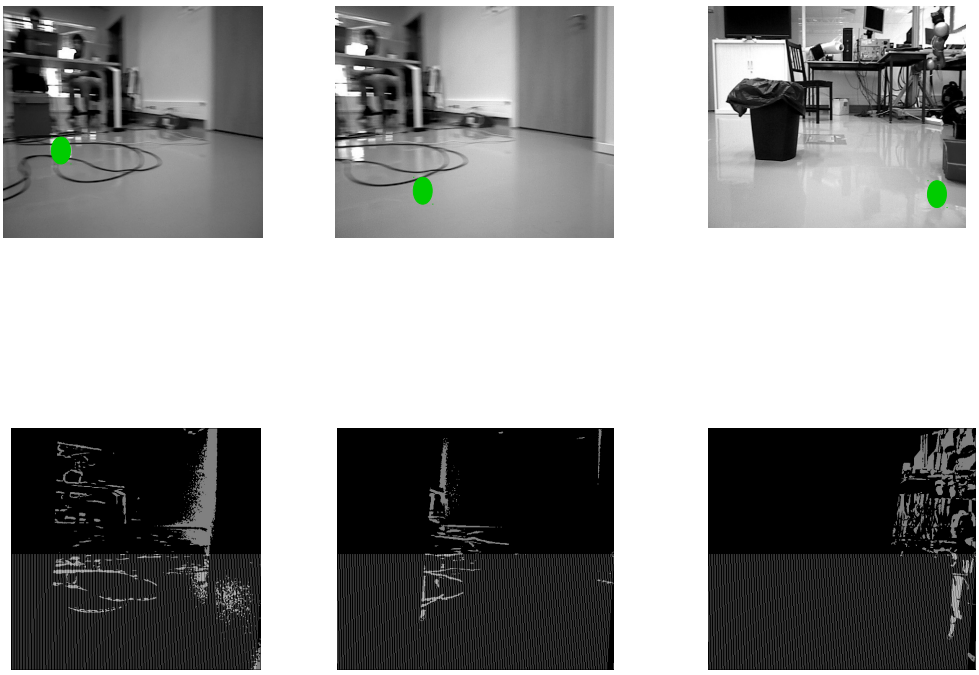


Figure 7.17: False contact points.

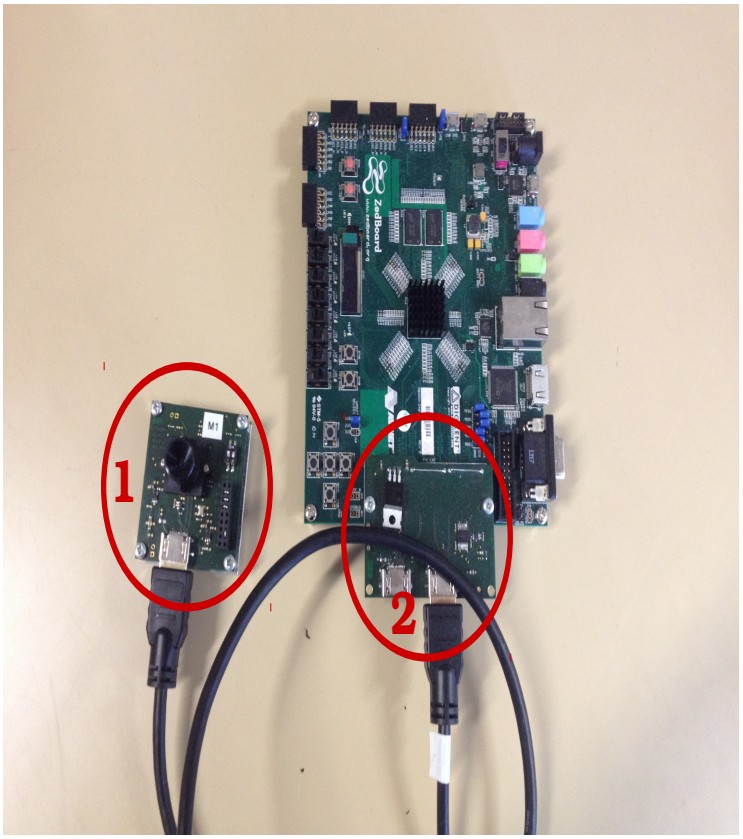


Figure 7.18: Hardware boards.

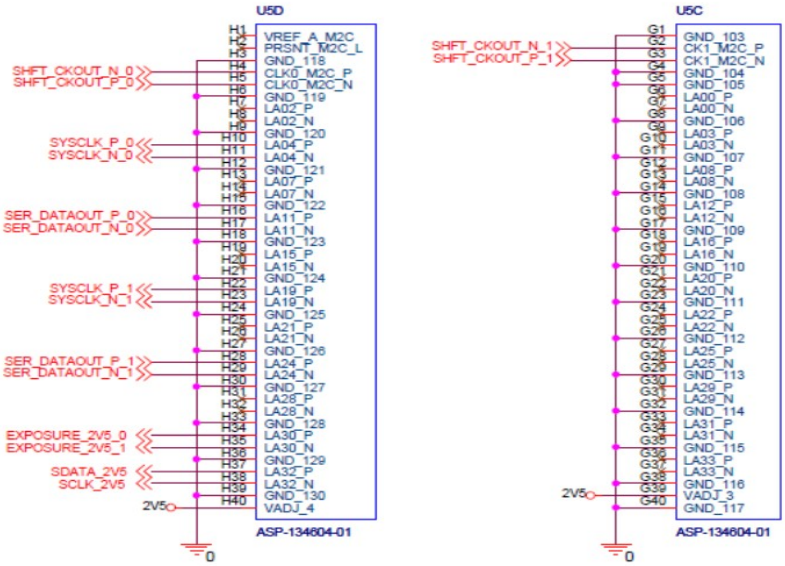


Figure 7.19: Interface to Zedboard [Piat 2016].

The interface card aims to convert a $3.3V$ input signal to a $2.5V$ output signal. In addition, this interface is used to transmit camera sensor signals from HDMI port to FMC port on Zedboard. Figures 7.19, 7.20 illustrate the input signals to the interface card and Zedboard. The following signals are connected to Zedboard:

- $Shft\text{-}CKOUT\text{-}N$, $Shft\text{-}CKOUT\text{-}P$ are LVDS signals which represent clock signals transmitted from the camera sensor.
- $SER\text{-}DATAOUT\text{-}N$, $SER\text{-}DATAOUT\text{-}P$ are LVDS signals which represent data signals transmitted from the camera sensor.
- $SDATA\text{-}2V5$, $SCLK\text{-}2V5$ are two wire serial interface for each clock and data (I2C signals). These signals are used for the configuration of camera registers.

7.4.2 Firmware Level

Figure 7.21 illustrates the general overview of firmware level. The top-level module is named $Top\ Cam\ Firm$. The input signals are the output signals of the hardware level (already mentioned in the previous section). The output signal is the selected contact points coordinates which forward to the software part of the reconstruction of 2D occupancy map.

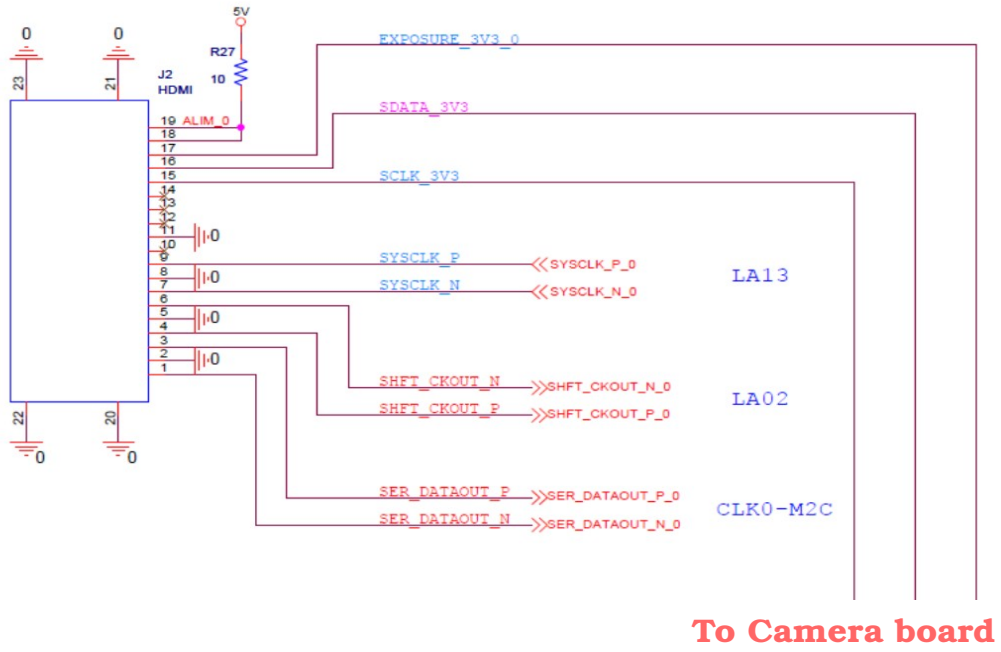


Figure 7.20: Input signals to the interface card [Piat 2016].

Figure 7.22 shows the parts of the top level module. As depicted in this figure, the first component is *Deserialization* which is implemented using *SelectIO* interface wizard from XILINX tools [XILINX 2016a]. This core assists in instantiating and configuring the components within the I/O interconnect block. Here, it is mainly used to deserialize the datapath (from 1 bit to 4 bit).
The second component is *Camera Signals Generator*. This component generates pixel data stream, and the control signals (*hsync*, *vsync*) line and frame valid signals respectively. The input signal *data-deser* is buffered and grouped into a packet of size 12 bit. In 8-bit pixel mode, the packet consists of a start bit, 8-bit pixel data, the line valid bit, the frame valid bit and the stop bit. This configuration is done to test the system in stand-alone mode. The configuration must be changed in stereoscopy mode when a second camera is connected.

Figure 7.23 illustrates the embedded system design. Most system functions are executed in the hardware fabric using the modules already explained in the previous chapter. Homography matrix, Focus point computations and projecting contact points on the 2D occupancy map are scheduled on ARM under Xillinux. The hardware modules are connected to Xillinux through Xillybus FIFOs. These FIFOs are accessed in the user space of the Xillinux kernel through device files. The software component $Config$ includes the registers values which are used for

Table 7.4: Breakdown hardware resources

Hardware Module	LUTs	Slice Reg	BRAM
IPM + Homo	8%	6%	87%
Binarization	10%	15%	3%
Localization	20%	5%	0%
Free space	3%	1%	0%
Selected contacts	1%	1%	0%
Total hardware architecture	41%	28%	90%
Zynq-7020 available resources	53200	106400	560 KB

the camera configuration using $I2C$ controller. Table 7.4 shows the amount of hardware resources for each module in the architecture. The produced system consumes a high percentage of BRAM blocks because an image is completely stored in BRAM memory in order to perform homography transformation. The hardware process can run at 203 frame per second for VGA resolution. The estimated latency of the hardware architecture is about 4.98 ms. The estimated power consumption of the hardware architecture is about $0.412W$. This value is computed using power analyse tool in VIVADO.

7.5 Conclusion

In this chapter, we have presented the results of the proposed architectures. These architectures are fully pipelined designs with a high frame-rate. The proposed architectures have advantages for power consumption and total system cost compared to other systems that exist in the state of the art. The proposed architectures fully implement the processing pipeline for obstacle detection and localization and the reconstruction of 2D occupancy grid maps. However, the detected contact points are not very accurate due to the use of odometry sensors for the movement information. In addition, the proposed system could produce false contact points induced by shiny surface, or shadows. Using stochastic methods could help improve the reconstructed occupancy map accuracy.

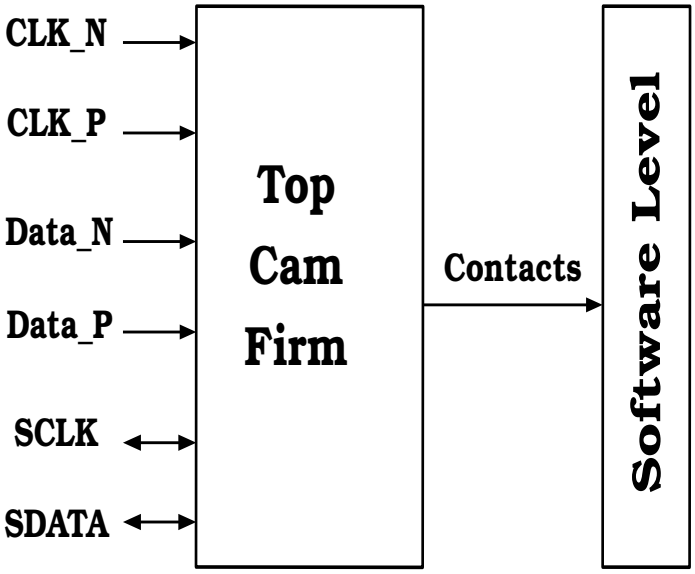


Figure 7.21: External view of Firmware level.

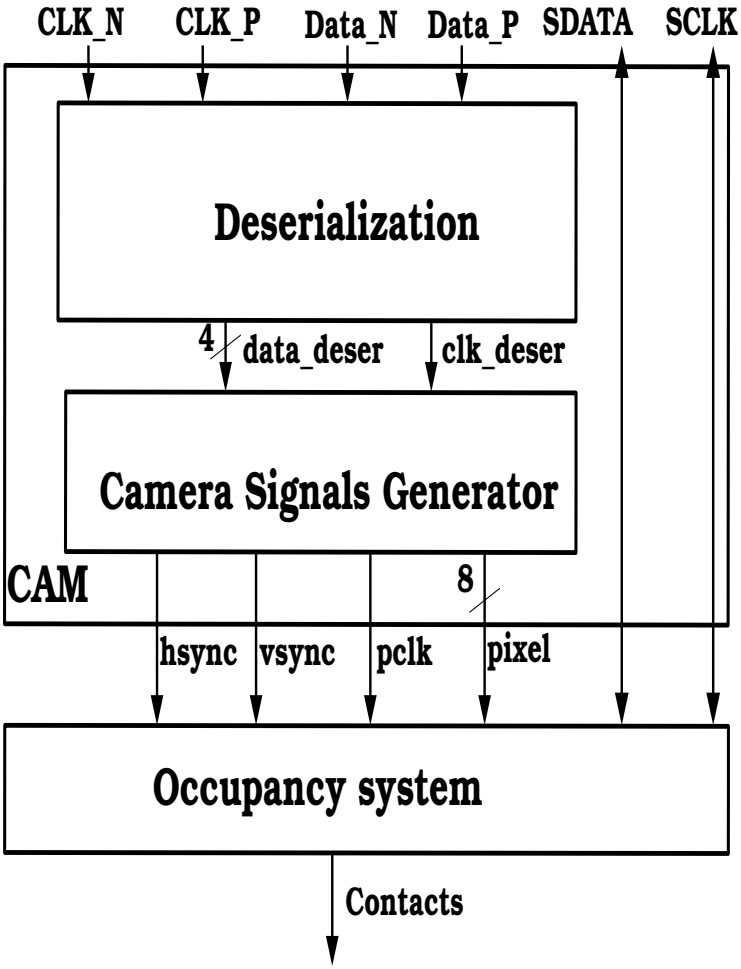


Figure 7.22: Top level module parts.

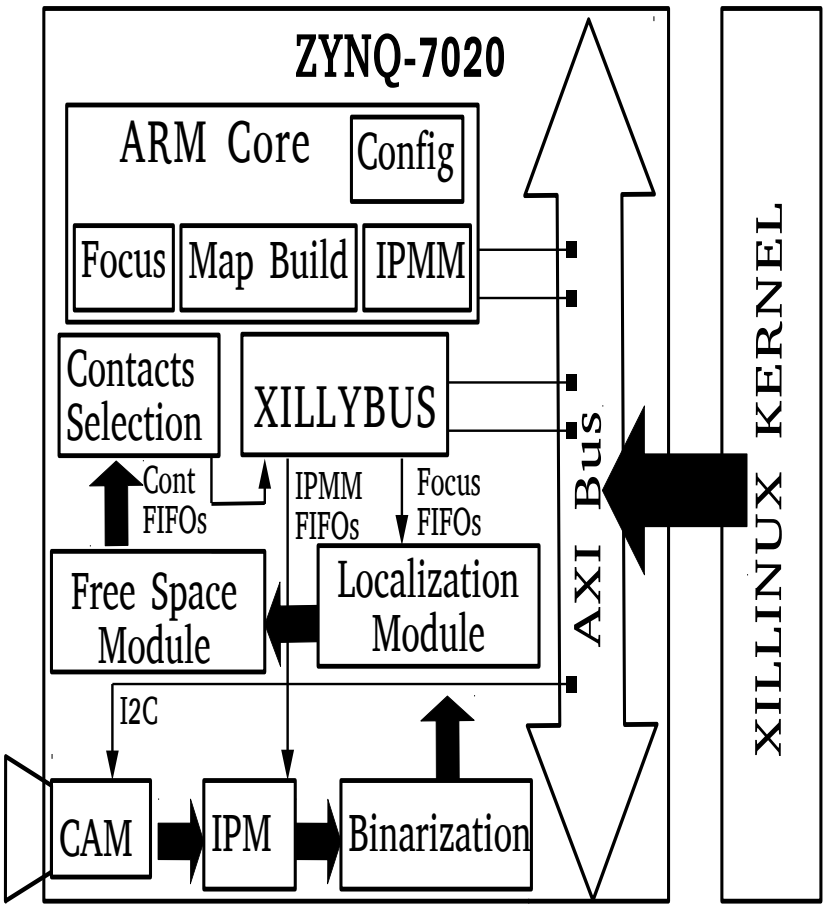


Figure 7.23: Occupancy system.

Chapter 8

Conclusion

The main interest of this thesis is the evaluation of vision based algorithms for robotic applications in indoor environments, and the design of application-specific architectures to meet strict constraints (such as low-priced, limited resources, low power consumption ...). The main problem discussed in this thesis is often refeered to as Algorithm Architecture Matching or hardware/software co-design.

The context of this work is the autonomous navigation of mobile robots in indoor environments. The main applications are transport robots in factories and logistics, exploration systems in buildings by producing a 2D occupancy grid map. For this latter task, we propose a hardware architecture that can update the occupancy map at 148 Hz. Another important application is fast navigations in emergency cases where low latency response and high throughput of detection is mendatory. In this case, the estimated latency of the hardware architecture is about 4.98 ms for the map update. If there is no need to build an occupancy map, obstacle detection latency induced by the hardware architecture is 7.49 μs and the achieved frame rate is as high as 201 FPS.

Taking advantage of parallelism of an FPGAs, efficient designs can be implemented to execute vision tasks. Let us cite our contributions and the future works for this thesis.

8.1 Proposed System Methodology

Since our objective is to build a real-time vision system for obstacles detection, localization, and mapping, we started by defining the system tasks flow.The obstacle detection task is performed using a geometric approach (namely Inverse Perspective Mapping). This method was well treated in the existing state of the art and was partially implemented on an FPGA-based architecture in [Botero 2012].
The obstacles segmentation, uses a binarization based on Otsu's algorithm. Gaussian filter and morphological operations are prerequisites to this task.
The obstacles localization is implemented using a polar histogram beam. This method allows to compute obstacles bearings relative to the camera. Additionally, an original method is introduced in order to extract contact points between obstacles and ground plane.
For free space detection, authors in [Bertozzi 1998a] proposed a method to perform this function using bearings measurements results.

Finally, an original method is proposed for the selection of contact points. these latter points are mapped and projected in a 2D occupancy grid

We introduced briefly the methodology of our proposed system. Let us recall our contributions to the system methodology in details:

1. We propose a method for obstacles localization. The approach takes binarized images produced by IPM method (which is used for obstacles detection) as input. The localization module produces obstacles bearings measurements, and contact points between obstacles and ground plane. Bresenham algorithm is implemented to track pixels that belong to polar histogram beam. The simplicity of Bresenham's algorithm allows to produce an efficient performance in the hardware implementation. While the pixels that belong to polar histogram beam are tracked, another method is performed to detect contact points between obstacle objects and ground floor. Taking advantage of quasi triangle shapes produced by IPM method, our proposed method for extracting contact points aims to find these triangles, and searches their heads which considered as the contact point.

2. We propose an approach to build 2D occupancy grid map. After contact points are extracted and free space is detected, a selection of best contact points is accomplished. These selected points are mapped and projected to a 2D occupancy grid. The method takes free space detection results and extracted contact points as input. A search operation is done to find the best points, and determine the nearest points to robot.

8.2 Proposed Hardware Architectures and Designs

Chapters 5-6 describe the positioning, design specifications, and advantages of the proposed hardware modules based on FPGA. Let us recall our contributions in terms of hardware modules:

- For obstacle detection.

    1. A hardware module is proposed for homography transformation. We optimize the homography module proposed by [Botero 2012]. our proposed component accomplishes distortion correction and homography transformation in single pass. The produced frame rate is 200 FPS for images of VGA resolution. Induced latency is $1.383\mu s$. The main disadvantage of this module is the high consumption of BRAMs to accomplish this transformation.
    2. A hardware module is proposed for the generation of IPM image. A subtraction is performed between a transformed image (produced by the previous module) acquired at t_n and a real image acquired at t_{n+1}, thus allowing to remove the ground plane. Additionally, the image acquired

at t_{n+1} must be written to the memory replacing the image acquired at t_n. Induced latency is 6 clock cycles. The main disadvantage of this module is the configuration of the FIFO depth. FIFO component is used to synchronize the stream of real image to the stream of transformed image.

- For obstacle segmentation

  1. Gaussian filter is implemented in a hardware design. This filter is a prerequisite to perform Otsu's binarization. Latency induced by this module is 1282 clock cycles. Two image rows are minimally required before the module runs.
  2. A hardware module is proposed to perform Otsu algorithm. This latter is used to compute Otsu's threshold which is employed to binarize the filtered image. 2 clock cycles of latency are induced by this module. An important amount of hardware resources is employed to execute Otsu algorithm. The critical point of this binarization is that Otsu's threshold is correct if there is not a high variation in intensities between two sequential frames.
  3. Erosion operator, a morphological operation, is implemented using a hardware module. This hardware accelerator aims to remove noise produced in some indoor environments which contain floor tiles. 1282 clock cycles of latency are induced in this module.

- For obstacle localization:

  1. Bird's eye transformation is implemented in a hardware design. This transformation was used by authors in [Bertozzi 1998b] as a preliminary step for obstacles localization. Due to the heavy cost of latency produced by the hardware implementation of this transformation, we propose another methodology for obstacles localization.
  2. Our proposed method for obstacles localizations is implemented in a hardware design. This module produces bearings measurements and contact points between obstacle objects and ground plane. 3646 clock cycles of latency are induced by this accelerator. An important amount of LUTs is also employed.

- For free space detection, a hardware accelerator is proposed. This module is an implementation of a method that exists in the state of art.

- For 2D occupancy grid map reconstruction:

  1. A hardware module is proposed for selecting best contact points. These latter are mapped and projected to the occupancy grid. 16 clock cycles are minimally required to produce the first selected point.

2. A software module is developed in order to map the selected points to the 2D occupancy map. The execution time of this software module is $56\mu s$. This is an estimated time to produce the coordinates of one selected point in the world frame.

These aforementioned hardware modules are employed to build 3 architectures summarized as follows:

8.2.1 Obstacles detection and localization architecture with bird's eye view transformation

The first architecture is an implementation of an obstacle detection method that exists in the state of art [Bertozzi 1998b], [Bertozzi 1998a]. This architecture combines Mono IPM method for detection and Otsu's method for segmentation, plus Bird's eye view transformation and Bresenham's algorithm for localization. Because bird's eye transformation is implemented in this architecture, the required latency time is about $2.05ms$. 126897 clock cycles of latency are induced. Hence, this is disadvantageous in hardware implementation; bird's eye transformation requires a high latency time, many BRAMs to store a specific number of eroded image rows, and a large amount of hardware resources. Therefore, a large FPGA platform (such as Virtex-6) is needed to implement the architecture. On the other hand, polar histogram beam implemented in this architecture is simpler than the beam used in other architectures.

8.2.2 Obstacles detection and localization architecture with our localization method

We propose a second architecture for obstacles detection and localization. This architecture combines Mono IPM method for detection and Otsu's method for segmentation, plus our proposed method for localization. Bird's eye transformation is not implemented. Hence, latency induced by this architecture is about $7.49\mu s$ which is clearly less than one induced by the first architecture. 4610 clock cycles are minimally required to output the first contact point. The resulting power consumption is about 3.9 watt. This value is estimated using Power Analyze tool in ISE tool. The architecture produces a pipelined design with a high frame rate. This latter is about 201 for images of size 640×480.

8.2.3 2D occupancy grid map reconstruction architecture

We propose an architecture for 2D occupancy map reconstruction. This architecture combines the second architecture (already mentioned), free space detection accelerator, plus a module for building the occupancy grid. This latter combines a method for selecting the best contact points and a developed approach to map and project selected points to the 2D occupancy map. This architecture is implemented on Zynq target which is tightly coupled to ARM processing system. This

implementation can build the 2D map at high frame rate,148 FPS for images of VGA resolution. This is an estimated value without taking into consideration the delay induced by *Xillybus* FIFOs which are used to interface hardware cores with software part. Induced latency by this architecture is about $4.98ms$ where $1.8ms$ is induced by the software part. The architecture can maximally produce 32 contact points per frame. The resulting power consumption of the hardware architecture is about 0.412 watt. This value is computed using power analyse tool in VIVADO. The main disadvantage of this implementation and the two previous implementations is the high consumption of BRAMs because an image is completely stored in BRAMs to perform homography transformation.

8.3 Future Works

Due to the limited time of the thesis, we did not widely contribute to methodologies. We only propose some optimizations to existing methods. Here are some short-term perspectives:
Concerning IPM method which is adopted for obstacles detection, the main hypothesis of this latter is the flat ground. Hence, the applications are limited to indoor environments. IPM method can be improved to work with uneven ground. For example, a valid auto-calibration of the camera can help to determine the orientation of ground plane with respect to camera position. The problem could be also simplified to compute the slope of ground plane in the horizontal direction. This improvement will permit to extend thesis applications to outdoor environments. Hence, our architectures can support ADAS systems (Advanced driver assistance systems).

Concerning obstacles localization method which is dedicated to IPM method, the efficiency of this localization approach depends on the existence of quasi vertical edges. As polar histogram beam is originating from one point $focus$ which is the vanishing point of vertical lines represented in the world frame, we can use two or more beams originating from two or more $focus$ points if we know obstacles edges orientation. This improvement allows to localize obstacle objects which do not have quasi vertical edges. As a result, false contact points percentage will be reduced.

Concerning the hardware architecture, an external memory (SDRAM) will be devoted for the storage of acquired image instead of using BRAMs. In this case, a lower frame rate of the pipelined design is delivered because of the high latency induced by read/write access to external memory. Additionally, adopting external memories permits to implement high resolution images to our architectures.
In this thesis, a hardware accelerator is designed to perform bird's eye view transformation. The number of bits used to represent the fractional part of bird's eye matrix elements is not sufficient. Because bird's eye view transformation

is a coarse case of homography transformation, higher resolution is required to represent the fractional part.

Now let us present some long-term perspectives, mainly inspired from the context of Camera-belt project. The proposed system will be extended to multi-camera system in order to build the occupancy grid map in all directions.
LAAS participated to a work package devoted to ADAS systems. An application of a hw/sw co-design methodology on visual 3D EKF SLAM application is developed in the PhD of Daniel Tortei and Francois Brenot. Our proposed architectures for obstacle detection and localization will be integrated to the SLAM application, resulting to an embedded application which can support ADAS systems and meet their requirements.
In this thesis, odometry sensors were adopted to provide movement informations. The estimation of robot's pose can not be produced without errors as the robot slips. We will exploit visual odometry methods to compute the robot position instead of relying on odometry sensors.
In our research team in LAAS, PhD students must study obstacles identification based on classification approaches. For example, human detection based on HOG (Histograms of Oriented Gradients) descriptor and SVM (support vector machine) classifier could be efficiently coupled to our architecture. The advantage of such implementation is that the region of interest will be widely reduced due to obstacles localization results delivered by our architecture.

Bibliography

[Adams 2014] Charlotte Adams. *FPGA or GPU? - The evolution continues.* In defense.ge-ip.com, page 1. mil-embedded.com/articles/, September 2014. (Cité en page 41.)

[Aging 2015] Global Health Aging. *How Robot Technology is Caring for the Elderly.* https://globalhealthaging.org/2015/03/20/how-robot-technology-is-caring-for-the-elderly/, 2015. Accessed: 2016-08-23. (Cité en page 4.)

[Ahtiainen 2015] Juhana Ahtiainen, Thierry Peynot, Jari Saarinen, Steven Scheding et Arto Visala. *Learned Ultra-Wideband RADAR Sensor Model for Augmented LIDAR-based Traversability Mapping in Vegetated Environments.* In International Conference on Information Fusion, pages 953–960. IEEE, July 2015. (Cité en pages 8 et 9.)

[Alhamwi 2015] Ali Alhamwi, Bertrand Vandeportaele et Jonathan Piat. *Real time Vision System for Obstacle Detection and Localization on FPGA.* In 10th International conference on computer vision systems, pages 80–90. Springer, July 2015. (Cité en pages 60 et 66.)

[Alhamwi 2016] Ali Alhamwi et Jonathan Piat. *FPGA implementation for 2D occupancy map reconstruction.* In Conference on Design and Architectures for Signal and Image Processing. https://ecsi.org/dasip, October 2016. (Cité en page 67.)

[Alliance 2016] Alliance. *The Embedded Vision Alliance, The Promise of Embedded Vision.* http://www.bdti.com/Resources/EmbeddedVisionAlliance, 2016. Accessed: 2016-08-23. (Cité en page 1.)

[Aptina 2008] Aptina. 1/3-inch wide-vga cmos digital image sensor mt9v034. www.aptina.com, 2008. (Cité en page 127.)

[ARM 2016] ARM. *Cortex-A9 Processor.* http://www.arm.com/cortex-a9.php, 2016. Accessed: 2016-08-21. (Cité en pages 42 et 43.)

[Arndt 2016] Michael Arndt. *Artos: Autonomous Robot for Transport and Service.* pages 1–1. agrosy.informatik.uni-kl.de/en/robots/artos/, July 2016. (Cité en page 3.)

[Avnet 2012] Avnet. Zedboard, zynq evaluation and development. Zedboard.org, 2012. (Cité en page 127.)

[Badino 2007] Hernan Badino, Uwe Franke et Rudolf Mester. *Free Space Computation Using Stochastic Occupancy Grids and Dynamic Programming.* In Programming," Proc. Int'l Conf. Computer Vision, Workshop Dynamical Vision, 2007. (Cité en page 14.)

[Bako 2015] L Bako, C Enachescu et S.-T. Brassai. *Design and validation of a low resource-cost video data processing method for embedded implementation of optical flow extraction.* In Carpathian Control Conference (ICCC), 2015 16th International, pages 13–18. IEEE, May 2015. (Cité en page 12.)

[Barr 2003] Michael Barr et Jack Ganssle. *Embedded Systems Dictionary.* In Embedded Systems Glossary. CMP Books, 2003. (Cité en page 40.)

[Beauchemin 1995] S. S. Beauchemin et J. L. Barron. *The computation of optical flow.* In Journal ACM Computing Surveys (CSUR), pages 433–466. ACM New York, NY, USA, September 1995. (Cité en page 10.)

[Benacer 2015] Imad Benacer, Aicha Hamissi et Abdelhakim Khouas. *Hardware design and FPGA implementation for road plane extraction based on V-disparity approach.* In Circuits and Systems (ISCAS), pages 2053 – 2056. IEEE, May 2015. (Cité en pages 126 et 127.)

[Bendaoudi 2012] Hamza Bendaoudi, Abdelhakim Khouas et Brahim Cherki. *FPGA design of a real-time obstacle detection system using stereovision.* In Microelectronics (ICM), 2012 24th International Conference on, pages 1–4. IEEE, December 2012. (Cité en page 119.)

[Bertozzi 1998a] Massimo Bertozzi et Alberto Broggi. *GOLD: A Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection.* In TRANSACTIONS ON IMAGE PROCESSING,. IEEE, 1998. (Cité en pages 34, 38, 137 et 140.)

[Bertozzi 1998b] Massimo Bertozzi, Alberto Broggi et Alessandra Fascioli. *Stereo Inverse Perspective Mapping: theory and applications.* Image and Vision computing, vol. 16, no. 8, pages 585–590, May 1998. (Cité en pages 16, 17, 21, 32, 35, 56, 66, 76, 139 et 140.)

[Bier 2011] Jeff Bier. *My vision for embedded vision.* In Embedded vision collection, pages 1–1. embedded.com, July 2011. (Cité en page 2.)

[Bier 2013] Jeff Bier. *Embedded Vision: Systems that See and Understand.* In Embedded.com, pages 1–1. Synopsys.com, February 2013. (Cité en pages 1 et 2.)

[Bishop 2006] Christopher Bishop. Pattern recognition and machine learning. Springer, New York, 2006. (Cité en page 15.)

[Boehler 2003] W. Boehler, M. Bordas Vicent et A. Marbs. *INVESTIGATING LASER SCANNER ACCURACY.* In CIPA SYMPOSIUM. www-group.slac.stanford.edu, October 2003. (Cité en page 9.)

[Botero-Galeano 2012] Diego Botero-Galeano. Development of algorithms and architectures for driving assistance in adverse weather conditions using fpgas. HAL, Robotics, INSA de Toulouse, 2012. (Cité en page 4.)

[Botero 2012] Diego Botero, Jonathan Piat, Pierre Chalimbaud et Michel Devy. *FPGA implementation of mono and stereo inverse perspective mapping for obstacle detection.* In Design and Architectures for Signal and Image Processing (DASIP), pages 1–8. IEEE, October 2012. (Cité en pages vi, 27, 65, 75, 137 et 138.)

[Castaneda 2011] Victor Castaneda et Nassir Navab. *Time-of-Flight and Kinect Imaging.* http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/, 2011. Accessed: 2016-08-10. (Cité en page 17.)

[Castelow 1988] D.A. Castelow, D.W. Murray, G.L. Scott et B.F. Buxton. *Matching Canny edgels to compute the principal components of optic flow.* In Image and Vision Computing, pages 129–136. ELSEVIER, May 1988. (Cité en page 11.)

[Cervantes 2008] J.G. Avina Cervantes, M. Estudillo Ayala, S. Ledesma Orozco et M.A. Ibarra-Manzano. *Boosting for image interpretation by using natural features.* In MICAI, pages 117–122. Seventh Mexican International Conference, October 2008. (Cité en page 15.)

[Cesar 2013] Caio Cesar, Teodoro Mendes, Fernando Santos Osorio et Denis Fernando Wolf. *An Efficient Obstacle Detection Approach for Organized Point Clouds.* In Intelligent Vehicles Symposium, pages 1203–1208. IEEE, June 2013. (Cité en pages 118 et 119.)

[Dalal 2005] Navneet Dalal et Bill Triggs. *Histograms of Oriented Gradients for Human Detection.* In Conference on Computer Vision and Pattern Recognition, pages 1–8. INRIA, January 2005. (Cité en page 15.)

[Dipert 2014a] Brian Dipert, Tim Droz, Stephane Francois et Markus Willems. *Improved Vision Processors, Sensors Enable Proliferation of New and Enhanced ADAS Functions.* In John Day's Automotive Electronics News, pages 1–1. www.embedded-vision.com, January 2014. (Cité en page 2.)

[Dipert 2014b] Brian Dipert, Jacob Jose et Darnell Moore. *Vision-Based Artificial Intelligence Brings Awareness to Surveillance.* In EE Times Embedded.com Design Line, pages 1–2. Embedded.com, May 2014. (Cité en pages 1 et 2.)

[Dipert 2014c] Brian Dipert, Tom Wilson, Tim Droz, Ian Riches, Gaurav Agarwal et Marco Jacobs. *Smart In-Vehicle Cameras Increase Driver and Passenger Safety.* In John Day's Automotive Electronics News, pages 1–1. embedded-vision.com, October 2014. (Cité en page 2.)

[Dopplinger 2012] Alexandra Dopplinger et Brian Dipert. *20/20 Embedded Vision for Robots.* In Electronic Products Magazine, pages 1–1. www.embedded-vision.com, July 2012. (Cité en page 2.)

[Enkelmann 1991] Wilfried Enkelmann. *Obstacle detection by evaluation of optical flow fields from image sequences.* Image and Vision Computing, vol. 9, no. 3, pages 160–168, June 1991. (Cité en page 11.)

[Fisher 2000] Robert Fisher, Simon Perkins, Ashley Walker et Erik Wolfart. *Morphological operations Erosion Description.* http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm, 2000. Accessed: 2016-08-29. (Cité en pages 32 et 33.)

[Fularz 2015] Michał Fularz, Marek Kraft, Adam Schmidt et Andrzej Kasinski. *The Architecture of an Embedded Smart Camera for Intelligent Inspection and Surveillance.* In Advances in Intelligent Systems and Computing, pages 43–52. Springer, April 2015. (Cité en page 2.)

[Fung 2004] James Fung et Steve Mann. *Using Multiple Graphics Cards as a General Purpose Parallel Computer : Applications to Computer Vision.* In Pattern Recognition, ICPR 2004. Proceedings of the 17th International Conference on, pages 805–808. IEEE, August 2004. (Cité en pages 40 et 41.)

[Ganoun 2009] A. Ganoun, T. Veit et D. Aubert. *Tracking multiple targets based on stereo vision.* In International Conference on Computer Vision Theory and Applications, pages 470–477. Springer, February 2009. (Cité en pages 14 et 15.)

[Geforce 2016a] Geforce. *GeForce 9800.* http://www.geforce.com/hardware/desktop-gpus/geforce-9800gt/specifications, 2016. Accessed: 2016-08-21. (Cité en page 126.)

[Geforce 2016b] Geforce. *GeForce GTX 550 Ti specification.* http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-550ti/specifications, 2016. Accessed: 2016-08-21. (Cité en page 118.)

[Harris 2015] Ian Harris. *What are embedded systems.* https://www.coursera.org/learn/iot/, 2015. Accessed: 2016-06-29. (Cité en pages 1 et 39.)

[Hartford 2013] Jamie Hartford. *The Embedded Vision Revolution.* In Medical imaging, pages 1–1. http://www.mddionline.com, April 2013. (Cité en page 2.)

[Hartley 2004] Richard Hartley et Andrew Zisserman. Multiple view geometry in computer vision. Cambridge University Press, UK, 2004. (Cité en pages 12, 24 et 32.)

[He 2012] Chin Yi He, Chao Tse Hongand et Rong Chin Lo. *An Improved Obstacle Detection Using Optical Flow Adjusting Based on Inverse Perspective*

Mapping for the Vehicle Safety. In Intelligent Signal Processing and Communications Systems (ISPACS), 2012 International Symposium on, pages 85–89. IEEE, November 2012. (Cité en pages 118 et 119.)

[Hildreth 2003] Ellen C. Hildreth. *Computations underlying the measurement of visual motion.* In ARTIFICIAL INTELLIGENCE, pages 309–354. ELSEVIER, February 2003. (Cité en page 11.)

[Hoiem 2012] Derek Hoiem. *Epipolar Geometry and Stereo Vision.* http://dhoiem.cs.illinois.edu/, 2012. Accessed: 2016-08-10. (Cité en pages 12 et 13.)

[Horn 1980] Berthold Horn et Brian Rhunck. *Determining Optical Flow.* In ARTIFICIAL INTELLIGENCE, pages 185–203. Laboratory, Massachusetts Institute of Technology, Cambridge, March 1980. (Cité en page 11.)

[J.Heeger 1988] David J.Heeger. *Optical flow using Spatiotemporal Filters.* In International Journal of Computer Vision, pages 85–117. Kluwer Academic Publishers, January 1988. (Cité en page 11.)

[Kommuru 2009] Hima Kommuru et Hamid Mahmoodi. Asic design flow tutorial using synopsys tools. San Francisco State University, USA, 2009. (Cité en page 43.)

[Labayrade 2002] R. Labayrade, D. Aubert et J.P. Tarel. *Real time obstacle detection in stereo vision on non-flat road geometry through v-disparity representation.* In Intelligent Vehicle Symposium, pages 646–651. IEEE, 2002. (Cité en page 14.)

[Lattice 2009] Semiconductor Lattice. *POWER CONSIDERATIONS IN FPGA DESIGN.* In White Paper, pages 1–16. www.latticesemi.com, February 2009. (Cité en page 49.)

[Lee 2012] Chia Hsiang Lee, Yu Chi Su et Liang Gee Chen. *An Intelligent Depth-Based Obstacle Detection System for Visually-Impaired Aid Applications.* In Image Analysis for Multimedia Interactive Services. IEEE, 2012. (Cité en pages 17 et 18.)

[Lee 2015] Seung Hyun Lee, MinSuk Bang, Kyeong-Hoon Jung et Kang Yi. *An Efficient Selection of HOG Feature for SVM Classification of Vehicle.* In International Symposium on Consumer Electronics. IEEE, 2015. (Cité en page 16.)

[Lindeberg 2012] Tony Lindeberg. *Scale Invariant Feature Transform.* http://www.scholarpedia.org/article/SIFT, 2012. Accessed: 2016-08-10. (Cité en page 15.)

[Lowe 1999] D. G. Lowe. *Object recognition from local scale-invariant features.* In Computer Vision, The Proceedings of the Seventh IEEE International Conference, pages 1150–1157. IEEE, September 1999. (Cité en page 15.)

[Lutkebohle 2008] Ingo Lutkebohle. *Embedded Computer Systems.* http://www.ece.ncsu.edu/research/cas/ecs, 2008. Accessed: 2016-06-29. (Cité en page 40.)

[Manzano 2010] Mario Alberto Ibarra Manzano, Michel Devy et Jean-Louis Boizard. *REAL-TIME CLASSIFICATION BASED ON COLOR AND TEXTURE ATTRIBUTES ON AN FPGA-BASED ARCHITECTURE.* In DASIP, pages 250–257. IEEE, October 2010. (Cité en pages 15 et 16.)

[MANZANO 2011] Mario IBARRA MANZANO. Vision multi-cameras pour la detection d'obstacles sur un robot de service. HAL, LAAS-CNRS, 2011. (Cité en page 4.)

[Mattoccia 2014] Stefano Mattoccia, Paolo Macrí, Giacomo Parmigiani et Giuseppe Rizza. *A compact, lightweight and energy efficient system for autonomous navigation based on 3D vision.* In Mechatronic and Embedded Systems and Applications (MESA), 2014 IEEE/ASME 10th International Conference on, pages 1–6. IEEE, september 2014. (Cité en pages 126 et 127.)

[Max-Maxfield 2004] Clive Max-Maxfield. The design warrior's guide to fpgas. ELSEVIER, USA, 2004. (Cité en pages 45, 46 et 47.)

[Meier 2014] Peter Meier. *Embedded Vision Challenges for Implementing Augmented Reality Applications.* In IEEE Embedded Vision Workshop, pages 1–1. cvisioncentral.com, June 2014. (Cité en page 2.)

[Miled 2007] Wided Miled, Jean-Christophe Pesquet et Michel Parent. *Robust Obstacle Detection based on Dense Disparity Maps.* In Eleventh International Conference on Computer Aided Systems Theory-EUROCAST 2. HAL, February 2007. (Cité en page 14.)

[Morris 2009] Gareth W. Morris, David B. Thomas et Wayne Luk. *FPGA accelerated low-latency market data feed processing.* In Symposium on High Performance Interconnects, pages 83–89. IEEE, February 2009. (Cité en page 49.)

[Nagel 1983] Hans-Hellmut Nagel. *Displacement vectors derived from second-order intensity variations in image sequences.* In Computer Vision, Graphics, and Image Processing, pages 85–117. ELSEVIER, January 1983. (Cité en page 11.)

[Nagel 1987] Hans-Hellmut Nagel. *On the estimation of optical flow: Relations between different approaches and some new results.* In Artificial Intelligence, pages 299–324. ELSEVIER, November 1987. (Cité en page 11.)

[Neobotix 2016] Neobotix. *Neobotix - Robotics & Automation*. http://www.neobotix-robots.com/, 2016. Accessed: 2016-08-23. (Cité en page 3.)

[Oniga 2010] Florin Oniga et Sergiu Nedevschi. *Processing Dense Stereo Data Using Elevation Maps: Road Surface, Traffic Isle, and Obstacle Detection*. In TRANSACTIONS ON VEHICULAR TECHNOLOGY, pages 1172–1182. IEEE, March 2010. (Cité en page 14.)

[Oniga 2015] Florin Oniga et Sergiu Nedevschi. *Ground or Obstacles? Detecting Clear Paths in Vehicle Navigation*. In International Conference on Robotics and Automation, pages 3927–3934. IEEE, May 2015. (Cité en page 15.)

[OpenCV 2014] OpenCV. *Camera calibration With OpenCV*. http://docs.opencv.org/2.4/doc/tutorials/calib3d/, 2014. Accessed: 2016-08-10. (Cité en page 25.)

[Otsu 1979] NOBUYUKI Otsu. *Threshold Selection Method from Gray-Level Histogram*. IEEE TRANSACTIONS ON SYSTEMS., vol. SMC-9, no. 1, pages 62–66, January 1979. (Cité en page 30.)

[Parnell 2004] Karen Parnell et Roger Bryner. *Comparing and Contrasting FPGA and Microprocessor System Design and Development*. In White Paper, pages 1–32. XILINX, JuLY 2004. (Cité en page 50.)

[Peasley 2013] Brian Peasley et Stan Birchfield. *Real-Time Obstacle Detection and Avoidance in the Presence of Specular Surfaces Using an Active 3D Sensor*. In Robot Vision (WORV), pages 197–202. IEEE, January 2013. (Cité en page 126.)

[Peng 2015] Yan Peng, Dong Qu, Yuxuan Zhong, Shaorong Xie, Jun Luo et Jason Gu. *The Obstacle Detection and Obstacle Avoidance Algorithm Based on 2-D Lidar*. In International Conference on Information and Automation, pages 1648–1653. IEEE, August 2015. (Cité en page 10.)

[Piat 2016] Jonathan Piat et Bertrand Vandeportaele. *Cartes MT9V034 + HDMI*. http://homepages.laas.fr/bvandepo/wiki/, 2016. Accessed: 2016-08-21. (Cité en pages 127, 130 et 131.)

[Quevedo 2012] Miguel Cazorla Quevedo et Jose Garcia-Rodriguez. Robotic vision. IGI Global, USA, 2012. (Cité en page 2.)

[ROS 2015] ROS. *rosbag*. http://wiki.ros.org/rosbag, 2015. Accessed: 2016-08-21. (Cité en page 120.)

[ROS 2016a] ROS. *TurtleBot*. http://wiki.ros.org/Robots/TurtleBot, 2016. Accessed: 2016-08-21. (Cité en page 120.)

[ROS 2016b] ROS. *TurtleBot*. http://wiki.ros.org/Robots/TurtleBot, 2016. Accessed: 2016-08-21. (Cité en page 120.)

[Routray 2011] Aurobinda Routray et Sukrit Dhar. *Creating a Real-Time Embedded Vision System to Monitor Driver's Visual Attention*. In Report from Indian Institute of Technology, Kharagpu, pages 1–1. National Instruments, October 2011. (Cité en page 2.)

[Schmid 2011] Matthias R. Schmid, S. Ates, J. Dickmann, F. von Hundelshausen et H.-J. Wuensche. *Parking Space Detection with Hierarchical Dynamic Occupancy Grids*. In ntelligent Vehicles Symposium, pages 254–259. IEEE, June 2011. (Cité en page 8.)

[SCHMIT 2000] HERMAN H. SCHMIT, SRIHARI CADAMBI et MATTHEW MOE. *Pipeline Reconfigurable FPGAs*. In Journal of VLSI Signal Processing Systems, pages 129–146. Kluwer Academic Publishers, February 2000. (Cité en page 48.)

[Shawe-Taylor 2000] John Shawe-Taylor et Nello Cristianini. Support vector machines and other kernel-based learning methods. Cambridge University Press, UK, 2000. (Cité en page 16.)

[Shinzato 2014] Patrick Y. Shinzato, Denis F. Wolf et Christoph Stiller. *Road Terrain Detection: Avoiding Common Obstacle Detection Assumptions Using Sensor Fusion*. In Intelligent Vehicles Symposium, pages 687–692. IEEE, June 2014. (Cité en page 10.)

[Silar 2013] Z. Silar et M. Dobrovolny. *The obstacle detection on the railway crossing based on optical flow and clustering*. In Telecommunications and Signal Processing (TSP), 36th International Conference on, pages 755–759. IEEE, July 2013. (Cité en page 11.)

[Smith 1997] Steve Smith. *The Feature-Based Approach*. http://users.fmrib.ox.ac.uk/, 1997. Accessed: 2016-08-10. (Cité en page 11.)

[Smith 2013] Barbara Peters Smith. *THE HOME CARE REVOLUTION: Robots and Eldercare's Future*. http://newamericamedia.org/2013/06/the-home-care-revolution-robots-and-eldercares-future.php, 2013. Accessed: 2016-08-23. (Cité en page 4.)

[Talukder 2002] A. Talukder, R. Manduchi, A. Rankin et L. Matthies. *Fast and reliable obstacle detection and segmentation for cross-country navigation*. In Intelligent Vehicle Symposium, 2002. IEEE, pages 1–4. IEEE, June 2002. (Cité en page 118.)

[Tarel 2007] J.P. Tarel, S.S. Ieng et P. Charbonnier. *Accurate and robust image alignment for road profile reconstruction.* In International Conference on Image Processing, pages 365–368. IEEE, 2007. (Cité en page 14.)

[Techpowerup 2012] Techpowerup. *Power Consumption.* https://www.techpowerup.com/reviews/AMD/, 2012. Accessed: 2016-08-21. (Cité en page 119.)

[Thomas 2009] David B. Thomas, Lee Howes et Wayne Luk. *A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation.* In Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, pages 63–72. ACM, February 2009. (Cité en pages 40, 41 et 42.)

[Totilo 2010] Stephen Totilo. *Natal Recognizes 31 Body Parts, Uses Tenth Of Xbox 360 Computing Resources.* http://kotaku.com/, 2010. Accessed: 2016-08-10. (Cité en page 17.)

[Toyota 2016] Toyota. *Lexus and Toyota Will Make Automated Braking Standard on Nearly Every Model and Trim Level by End of 2017.* In Toyota-USA Newsroom, pages 1–1. pressroom.toyota.com/, March 2016. (Cité en page 2.)

[Wettach 2016] Jens Wettach. *Experimental Evaluation of Some Indoor Exploration Strategies.* In International Conference on Informatics in Control, Automation and Robotics, pages 1–1. IEEE, July 2016. (Cité en page 3.)

[Wilson 2014] Tom Wilson, Brian Dipert, Marco Jacobs et Tim Droz. *Augmented Reality: A Compelling Mobile Embedded Vision Opportunity.* In Electronic Engineering Journal, pages 1–1. embedded-vision.com, April 2014. (Cité en page 2.)

[XILINX 2012a] XILINX. *FPGA vs. ASIC.* http://www.xilinx.com/fpga/asic.htm, 2012. Accessed: 2016-08-21. (Cité en page 44.)

[XILINX 2012b] XILINX. *Zynq-7000 All Programmable SoC Product Overview, The SW, HW and IO Programmable Platform.* http://gongterr.co.kr/makus/seminar121212/down/, 2012. Accessed: 2016-08-21. (Cité en pages 52 et 53.)

[XILINX 2014a] XILINX. *7 Series FPGAs Configurable Logic Block.* http://www.xilinx.com/support/documentation/, 2014. Accessed: 2016-08-21. (Cité en pages 45 et 47.)

[XILINX 2014b] XILINX. *7 Series FPGAs Memory Resources.* http://www.xilinx.com/support/documentation/, 2014. Accessed: 2016-08-21. (Cité en page 48.)

[XILINX 2015a] XILINX. *Virtex-6 Family Overview.* http://www.xilinx.com/support/documentation/, 2015. Accessed: 2016-08-06. (Cité en page 66.)

[XILINX 2015b] XILINX. *Zynq-7000 All Programmable SoCs Product Tables and Product Selection Guide.* http://japan.xilinx.com/support/documentation/selection-guides/, 2015. Accessed: 2016-08-21. (Cité en page 50.)

[XILINX 2015c] XILINX. *Zynq-7000 All Programmable SoCs Technical Reference Manual.* http://www.xilinx.com/support/documentation/user-guides/ug585-Zynq-7000-TRM.pdf, 2015. Accessed: 2016-08-21. (Cité en page 51.)

[XILINX 2016a] XILINX. Selectio interface wizard. Xilinx.com, USA, 2016. (Cité en page 131.)

[XILINX 2016b] XILINX. *Zynq-7000 All Programmable SoC Overview.* http://japan.xilinx.com/support/documentation/, 2016. Accessed: 2016-08-21. (Cité en pages 51 et 66.)

[XILINX 2016c] XILINX. *Zynq-7000 All Programmable SoCs Technical Reference Manual.* hhttp://www.xilinx.com/products/technology/power/xpe.html, 2016. Accessed: 2016-08-21. (Cité en page 50.)

[Xillybus 2016a] Xillybus. *Getting started with the FPGA demo bundle for Xilinx.* http://xillybus.com/downloads/doc/, 2016. Accessed: 2016-08-21. (Cité en page 54.)

[Xillybus 2016b] Xillybus. *Xillybus FPGA designer's guide.* http://xillybus.com/downloads/doc/, 2016. Accessed: 2016-08-21. (Cité en page 52.)

[Xillybus 2016c] Xillybus. *XILLYBUS, IP core product brief.* http://xillybus.com/downloads/, 2016. Accessed: 2016-08-21. (Cité en pages 52 et 53.)

[Yankun 2011] Zhang Yankun, Hong Chuyang et Weyrich Norman. *A Single Camera Based Rear Obstacle Detection System.* In Intelligent Vehicles Symposium, pages 485–490. IEEE, June 2011. (Cité en pages 118 et 119.)

Résumé :
La Détection, localisation d'obstacles et la reconstruction de carte d'occupation 2D sont des fonctions de base pour un robot navigant dans un environnement intérieure lorsque l'intervention avec les objets se fait dans un environnement encombré. Les solutions fondées sur la vision artificielle et couramment utilisées comme SLAM (Simultaneous Localization And Mapping) ou le flux optique ont tendance à être des calculs intensifs. Ces solutions nécessitent des ressources de calcul puissantes pour répondre à faible vitesse en temps réel aux contraintes. Nous présentons une architecture matérielle pour la détection, localisation d'obstacles et la reconstruction de cartes d'occupation 2D en temps réel. Le système proposé est réalisé en utilisant une architecture de vision sur FPGA (Field Programmable Gates Array) et des capteurs d'odométrie pour la détection, localisation des obstacles et la cartographie. De la fusion de ces deux sources d'information complémentaires résulte un modèle amélioré de l'environnement autour des robots. L'architecture proposé est un système à faible coût avec un temps de calcul réduit, un débit d'images élevé, et une faible consommation d'énergie.

Mots clés :
Implementation FPGA, Détection d'obstacles, Traitement d'images en temps réel, accéleration hardware, vision robotique.