UNIVERSITÉ DE FRIBOURG SUISSE
UNIVERSITÄT FREIBURG SCHWEIZ

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Master Of Science Thesis

Stereo Vision Library for Obstacle Avoidance Applications

University of Fribourg

Project by:
Biologically Inspired Robotics Group, EPFL

Switzerland

Author:
Elia Palme

elia.palme@gmail.com

Supervisor:
François Fleuret
Professors:
Auke Ijspeert
Rolf Ingold

BIOLOGICALLY INSPIRED
ROBOTICS GROUP (BIRG)

October 3, 2007

Abstract

With this report we want to propose a new stereo algorithm, which is especially conceived to be used for robot navigation applications. It is fast and very robust, and allows to suppress the need of empirically set threshold values. The stereo library is self adapting and is able to manage environment luminosity modifications. A texture quality filter is used to drop those image portions which probably yield to a mismatch. This technique increase the algorithm performance and its reliability. We also propose a new consistency check which is especially conceived to allow close objects detection. To overcome the algorithm limits a detection model based on neural networks is proposed. Neural networks are used to identify very close objects which are out of range, and no more detectable by the stereo algorithm. A floor detection filter is also introduced, its purpose is to filter out all floor depth measurement which can confuse the obstacle avoidance algorithm. In this report we also explain the problematic of doing dynamic stereo vision having a passively moved binocular vision system. How we conceived the hardware and it related drivers. We explain the need to have two perfectly synchronize frames (left and right cameras images) and how to achieve it. Finally in the report the reader can find a description of all utility tools we developed for this project.

Acknowledgements

I would like to thank Matteo De Giacomi for his great collaboration. We had a lot of fun. Thanks Matteo for helping me in any situation, you provided a precious support.

Many thanks to François Fleuret, he provided a lot of great ideas and a huge mathematical support. He was my supervisor and we had many amazing meetings, where I got the opportunity to learn a lot of useful notions.

A special thank also goes to Alessandro Crespi, he was my hardware wizard.

Thanks to Prof. Auke Ijspeert, he proposed me this project and followed it with a lot of enthusiasm. You have bean an ideal leader.

Thanks to André Guignard for building us the stereo vision cameras support.

Thanks to Prof. Pascal Fua, which dedicated us some time and for his precious advices.

Thanks to Prof. Rolf Ingold, my supervisor at the University of Fribourg. Thanks Rolf for accepting this project and for supervising me with interest.

A big thank to Yerly, Terreaux, Bodmer, Anli, Sebastien, Matteo and the whole BIRG lab. We had a lot of great times together, thanks guys!

And least but not last I would like to thank my family and my friends which are always there to make me happy.

Contents

1 Introduction **4**
- 1.1 Objectives . . . 4
- 1.2 Project description . . . 5
- 1.3 Planning . . . 6

2 Theory **7**
- 2.1 Introduction to computational stereo . . . 7
- 2.2 Overview of correlation methods . . . 7
- 2.3 Calibration and Rectification . . . 8
  - 2.3.1 Pinhole cameras . . . 9
  - 2.3.2 Image distortion . . . 9
  - 2.3.3 Image rectification . . . 10

3 Further Analysis **11**
- 3.1 Real-time correspondence algorithms . . . 11
  - 3.1.1 Crucial image features for correlation algorithms . . . 11
- 3.2 Hardware requirement for real-time stereo . . . 12
  - 3.2.1 Commercial cameras solution . . . 12

4 Tools **15**
- 4.1 Calibration Tool . . . 15
  - 4.1.1 Calibration Tool Architecture . . . 16
- 4.2 Synchronized Frames Grabber . . . 16
  - 4.2.1 Specification . . . 17
  - 4.2.2 Architecture . . . 18
  - 4.2.3 Conception . . . 19
  - 4.2.4 Remarks . . . 20
- 4.3 Test and Tuning GUI . . . 20

5 Correspondence Algorithm **21**
- 5.1 Introduction . . . 21
- 5.2 Dynamic Programming Algorithm . . . 23
- 5.3 Block matching Algorithm . . . 24
  - 5.3.1 Stereoscopic Machine Vision Algo v.1 . . . 24
  - 5.3.2 Stereoscopic Machine Vision Algo v.2 . . . 26
  - 5.3.3 Stereoscopic Machine Vision Algo v.3 . . . 27
  - 5.3.4 Stereoscopic Machine Vision Algo v.4 . . . 34

6 Stereo Head (Hardware) 36
6.1 Cameras and they Support . . . 36
6.2 Lenses . . . 38
6.3 External Trigger . . . 38

7 Tests 39

8 Neural Networks 42
8.1 Too Close Object Detection . . . 42
8.1.1 Training . . . 43
8.2 Floor Detection . . . 44
8.2.1 Training . . . 44
8.2.2 Application . . . 45

9 Conclusion 46

A Dynamic Programming Case Study 49

B Distance at Disparity measure 51

C MatLab code to test the difference-of-Gausian filter 54

D MatLab code to generate the .h file containing the difference-of-Gaussian filter constants 57

E XML calibration file example 61

F Trigger micro controller C code 63

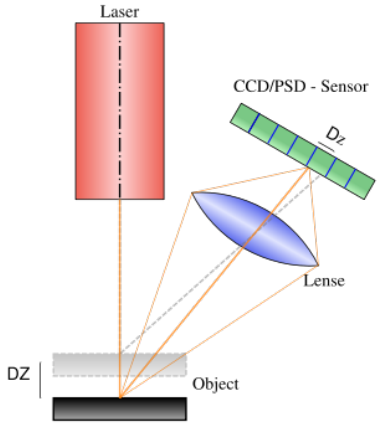
G BibTeX reference 65

Chapter 1

Introduction

1.1 Objectives

AmphiBot II and the Salamander robot are two biologically inspired robots developed at the EPFL's Biologically Inspired Robotics Group. Their locomotion system is the subject of multiple papers [1, 2, 3]. Thanks to a numerical model simulating the salamander's spinal cord these robots are capable to swim and walk like a salamander (AmphiBot II does not have legs, it crawls). This model was implemented as a system of coupled nonlinear oscillators running on a micro-controller on board of the robots.
The goal of our project is to provide information about the geometry of the surrounding environment, so that this information can be further treated by an obstacle avoidance application allowing the robots to explore their ambient.
We briefly compare some technique to acquire 3D informations of the environment. 3D Laser



(a) Laser Scanning Configuration (Image source: Wikipedia)

(b) SICK LMS 200 a laser scanner

Figure 1.1: Laser Scanning

scanning is actually the most reliable method, it can reach a precision of 0.05 mm. It consists of a laser source which irradiate the environment. Laser rays are reflected by the objects in front of the laser source. An optical sensor is placed with a view angle slight different respect to the laser source as showed in figure 1.1(a). From the difference between the expected point of incident and the measured one (where the laser ray hit the optical sensor) the depth information is evaluated. Four our scopes this system has an important drawback. O. Wulf and B. Wagner

[4] have recently developed a fast 3D scanning method, they where able to scan an apex angle of 180°x180° in 1.6 seconds. For a view angle of 60°x50° (which is comparable to an optical device view field) the scanning time is about 0.15s. This is just the acquisition time without any processing, during this time the robot should not move otherwise measures will result distorted. Such a constraint is not acceptable, therefore more sophisticated scanners has been developped. Unfortunately their size are definitively out of range respecting our needs. There is only few solutions allowing to do dynamic 3D scanning (not from a fix point). As example we report the LMS200/291 figure 1.1(b), it is 4.5 kg heavy and consume approximately 20W. Ultrasonic depth sensors are also employed to avoid obstacle. They have a small size and a low consummation, unfortunately their resolution is too low to reconstruct a dense depth map of the surrounding environment. To provide a 3D information of the ambient with ultrasonic sensors, a scanning head composed of multiple sensors and motors is needed. The most practice solution is stereo vision, it is reliable enough for obstacle avoidance purpose and can produce dense depth maps. Stereo vision systems do not need complex hardware, two coupled video cameras are the minimal requirement. We can build our own binocular vision system, and fit its size to our constraints. Stereo vision is also consistent with the laboratory philosophy since it is a biologically inspired solution. The last reason is that the depth information is not directly measured thought the hardware but it has to be extrapolated from the binocular images. Stereo vision is primarily algorithmic, the hardware produce the data, and thanks to an intermediate software layer the information is treated and the depth is computed. This solution gives us a bigger research space, and the possibility to easily propose interesting innovations.
Our days there are numerously algorithms to compute depth maps with different characteristics and for different scopes. Some algorithms are more specific for statical image analysis they have high reliability and produce dens depth maps. Such algorithms (e.g. graph-cut, Bayesian diffusion) are computationally too expensive to treat an image stream and are not adapted for obstacle avoidance purposes. Our choice is reduced to that algorithms which have a real-time performance. An important aspect we want to underline is that our vision system is mounted on a moving robot, doing stereo vision with non static points of view introduce noise and arbitrary hazards like motion blur, illumination variation, etc. More sophisticated vision systems have actively moved cameras, thanks to apposite motors they can change their cameras view angles and also try to stabilize them. Since our binocular vision systems is simpler, cameras are passively moved by the robot itself, therefore the stereo algorithm has to be much more robust and especially conceived to handle noisy inputs.

1.2 Project description

The aim of the project is to provide stereoscopic vision to AmphiBot II and the Salamander robot. More precisely we want to conceive an extremely robust stereo vision system which provide a simple and reliable depth information. The output format is especially conceived to be treated by obstacle avoidance applications, let me take this opportunity to introduce the work of Matteo De Giacomi [5]. He developed a biologically inspired controller to handle the robots behaviors. His project is also based on our stereo vision system, he use the depth information we provide him to avoid obstacles. The stereo vision system is composed by a binocular vision hardware and a dedicated software able to reconstruct the depth information. The whole problem can be divided in four consecutive processes.

1. Image grabbing, the binocular vision hardware has to be interfaced trough a dedicated driver which provides perfectly synchronized frames.

2. Rectification, the aim of this process is to rectify captured images to correct lens distortion

and align epipolar lines to the coordinate axis.

3. Correspondence, the task of this process is to identify correspondences between the left and the right image and establish a disparity map of the two images.

4. Reconstruction, by knowing the geometry of the vision system the disparity map is used to reconstruct the true depth value. This process also tries to identify particular situations such as too close objects or an undesired floor detection.

1.3 Planning

We have decomposed the project in 4 consecutive steps (state of the art is not included) and fixed three milestones:

- 23-Apr-07: Complete the image rectification method.
- 28-May-07: Complete the disparity map library.
- 16-Jul-07: Complete the stereo vision system.

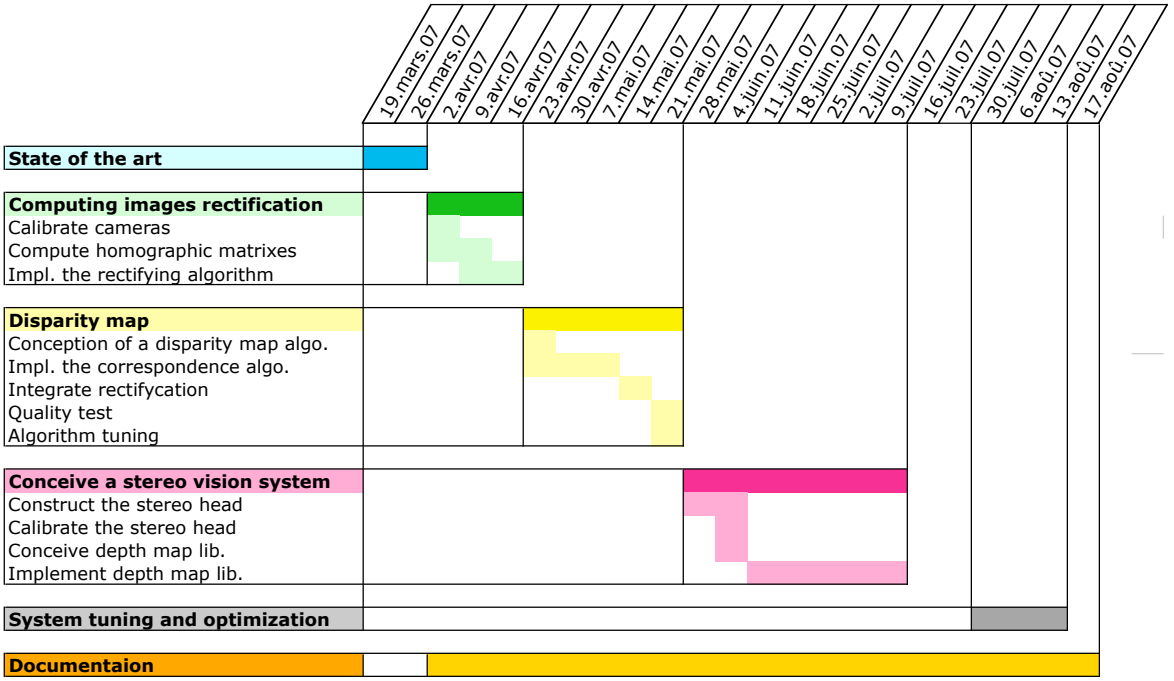


Figure 1.2: Detailed project planning

Chapter 2

Theory

2.1 Introduction to computational stereo

The computational stereo problem is based on the physical fact that a single three-dimensional object has multiple projections depending on the point of view. Having at least two images of the same subject taken from two distinct points of view, the computational stereo approach tries to reconstruct the 3D scene. The points of view should be parallel and not too far away. The computational stereo problem can be divided in three consecutive stages:

1. Calibration/Rectification: The calibration process determines the internal and external geometry of the binocular system. With the knowledge provided by the calibration process captured images are undistorted and rectified such that corresponding epipolar lines lie along the horizontal axe.

2. Correspondence: Determine the disparity[1] of a physical point in multiple point of view projections. By repeating this process for all points of the 3D scene the correspondence phase computes a disparity map.

3. Reconstruction: From the disparity map and the known camera system geometry the reconstruction phase determines the depth of all points.

2.2 Overview of correlation methods

In this section we briefly introduce different algorithms and methods to compute disparity maps from two or more images taken from distinct points of view. We can distinguish two basic types of correlation algorithms: feature-based and area-based.
Feature based algorithms extracts features (e.g., edges[6], curves[7], etc.) from images and try to match them in two or more views. They are very efficient but as drawback they produce poor depth maps. For further information interested reader are addressed to a review by Dhond and Aggarwal [8].

Area-based algorithms compare the grey level of pixels and determine the correspondence between pixels of more views. We can further categorize area-based algorithms in local correspondence methods and global correspondence methods as introduced in section 3.3 of [9] or by [10]. Local correspondence methods try to match a small region surrounding a pixel in more views, they are more sensitive to local ambiguity but computationally efficient. The main

[1]The term Disparity is a component of the process of Stereopsis. It is the difference in images from the left and right eye that the brain uses as a binocular cue to determine depth or distance of an object (Wikipedia)

principle is to compare each pixel of the left image with all pixels of its relative epipolar line on the right image within a certain range. The range is defined by the maximum disparity we want to reach, the more the disparity range is high the more we can detect close objects. Using rectified images the comparison process becomes more performing, since the correlation is made by matching pixels with the same Y coordinate. To better explain how the correlation is performed, we define a pixel in a two dimensional space as a function $I(x, y)$ returning the pixel intensity. Intensity value is an integer going from 0 to 255. There are multiple correlation methods, the simplest we can imagine is comparing the intensity of one pixel in to the left image with all pixels of the right image and retain the comparison with the lowest difference. Clearly such a method will produce an arbitrary matching, since an image has multiple pixels having the same intensity. To diminish the ambiguity rather than comparing only two pixels at time we compare a sequence of pixels, a portion of the image called window. Multiple methods to compute the similarity of two windows exist, and again we can distinguish more categories. Basically there are normalizing methods and not. The Sum of absolute difference (equation 2.1) or the Sum of square difference (equation 2.2) are two examples of non-normalizing methods, the cross correlation (equation 2.3) normalizes instead the comparison by subtracting the mean window value.

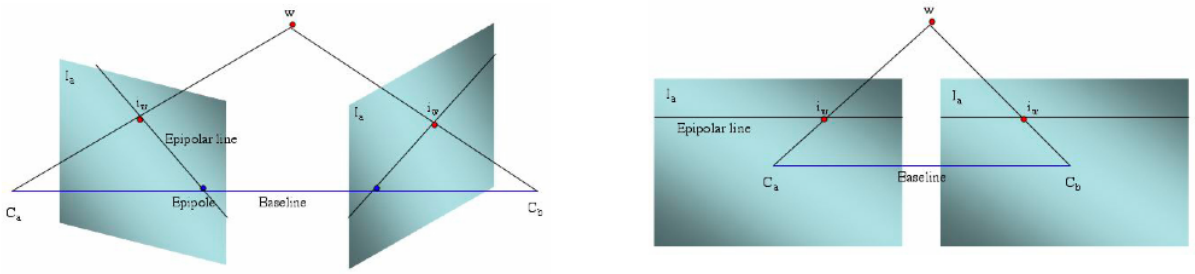
$$ASD(x, y) = \sum_{j=0}^{n}\sum_{i=0}^{n} abs(I1(x+i, y+j) - I2(x+i+d, y+j)) \tag{2.1}$$

$$SSD(x, y) = \sum_{j=0}^{n}\sum_{i=0}^{n} (I1(x+i, y+j) - I2(x+i+d, y+j))^2 \tag{2.2}$$

$$CC(x, y) = \frac{\sum_{j=0}^{n}\sum_{i=0}^{n}[(I1(x+i, y+j) - mx).(I2(x+i+d, y+j) - my)]}{\sqrt{\sum_{j=0}^{n}\sum_{i=0}^{n}(I1(x+i, y+j) - mx)^2}.\sqrt{\sum_{j=0}^{n}\sum_{i=0}^{n}(I2(x+i+d, y+j) - my)^2}} \tag{2.3}$$

Where n is the window size, I1 and I2 the left and right images pixel intensities, d the disparity, mx and my are the means of the corresponding windows.

2.3 Calibration and Rectification

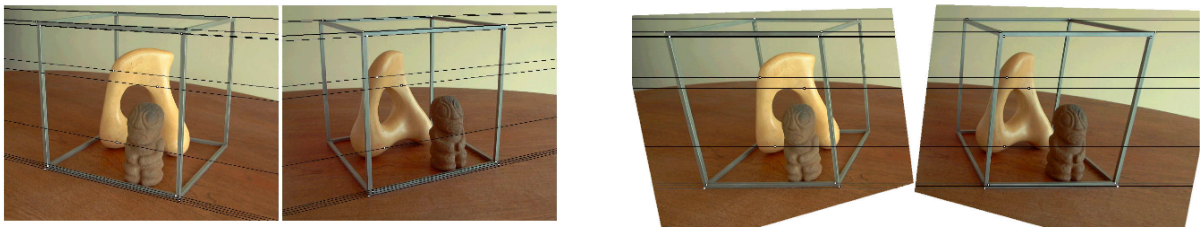


(a) Epipolar Geometry

(b) Rectified Epipolar Geometry

Figure 2.1: Epipolar Geometry Rectification, images taken from [11]

In this section we briefly introduce rectification techniques and their associated calibration methods. Rectification methods are well known and have been meticulously studied by years.



(a) Non rectified image

(b) Rectified image

Figure 2.2: Image rectification, images taken from [12]

The aim of these techniques is to adjust captured images to simplify their manipulation. When we capture an image with an optical device the resulting image differs in some way from the real world geometry. There are basically two factors that have to be adjusted in stereo applications, the image distortion and the image epipolar geometry, see figures 2.1 and 2.2.

2.3.1 Pinhole cameras

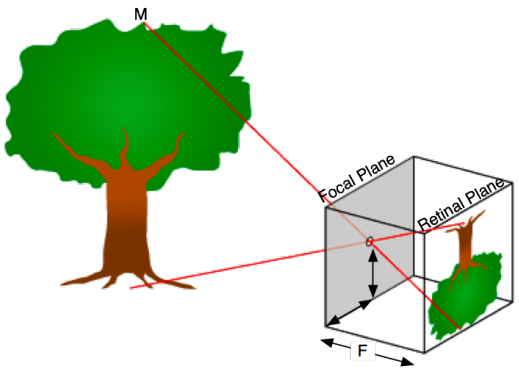


Figure 2.3: Principle of a pinhole camera (modified image taken from Wikipedia).

This section describes pinhole cameras, which is the type of camera used in our project. The pinhole camera is the simplest projective camera that maps a three dimensional point to a two dimensional image. A pinhole camera consists of two planes, the retinal and the focal one. On the retinal plane the image is formed, the focal plane is parallel to the retinal one on a distance F called focal length. A three dimensional point M from the real world is mapped to the two dimensional image via a perspective projection. Pinhole cameras are characterized by two sets of parameters. Internal or intrinsic parameters, describe the internal geometry and the optical characteristics of the camera. Extrinsic or external parameters describe the camera position and orientation on the real world. To compute a comparison between two images captured from two different cameras, intrinsic and extrinsic parameters are fundamental.

2.3.2 Image distortion

Optical devices use lenses to converge images on the image sensor, therefore acquired images result to be distorted since they have to pass trough a lens. Distortion factors are specific

to each camera and together with the focal length and the centre point they compose the intrinsic parameters of a camera. With the aid of calibration systems intrinsic parameters can be estimated and an undistortion projective matrix is easily computed. As calibration tool reference we advise the Camera Calibration Toolbox for Matlab [13].

2.3.3 Image rectification

To rectify an image we need to apply a 2D projective transformation called homography, obtained from the essential or fundamental matrix. For more information on homographies, the reader is addressed to [12]. Different solutions are proposed to achieve an image rectification. If the intrinsic parameters of the images (focal length, distortion, etc.) are know we can normalize it and work with the essential matrix. Otherwise the fundamental matrix can be estimated by calibrating the binocular system. The aim of the calibration process is to retrieve intrinsic parameters used to undistort images and extrinsic parameters used to rectify images. Both intrinsic and extrinsic parameters compose the fundamental matrix. To provide an accurate result two approaches are known: photogrammetric calibration and self-calibration. The first technique efficiently estimates the parameters by observing a well know calibration object (calibration rig) and estimating the vision system geometry. This method is adapted to stereo systems not changing their geometry during the runtime. If a stereo system using zooms or changing its cameras angles is needed, the self-calibration technique is more adequate. The self-calibration method does not need any calibration rig, it runs permanently but provides less accurate results. For further details about calibration methods the reader is addressed to [14] and [15].

Chapter 3

Further Analysis

3.1 Real-time correspondence algorithms

Stereo matching has been one of the most explored research area in computer vision during the last years. There are actually many different approaches to solve this problem. S. Scharsteun and R. Szeliski [9] have well measured performances of the most common stereo correspondence algorithms. From their work it emerges that only few correspondence approaches are suitable for real-time[1] application on a common computer architecture[2]. Best performances are provided by correlation based algorithms. In [16] a similar approach obtains comparable performances, but still not enough for our purposes, since at their best rate it may take a few seconds to process a single frame. Therefore our further works are essentially focused on correlation based algorithms. The first well documented real-time effort with a correlation based algorithm was made by the INRIA institute using a dedicated hardware [17]. In the last years many researches have been made to increase real-time correspondence algorithms accuracy and performance [18] [19] [20] [21]. Some commercial implementations[3] have reached a frame rate of more than 15 fps on common personal computer architecture. We have tested the Videre Design Small Vision System in our labs on a 3GHz computer, and it turned out to be able to process up to 40 fps. Actually the most employed correspondence algorithms for real-time stereo matching applications are correlation based. They only differ in their implementation and in some pre and post processing phases. In the following sections we better study correlation algorithms and crucial image features to care about for maximizing the disparity map quality.

3.1.1 Crucial image features for correlation algorithms

Correlation based algorithms try to establish a correspondence between windows of the left camera image and windows of the right camera image. Different size of windows are used, the size of the window determine the depth map resolution. As described in section 1.1 of [21] the correlation method assumes that the depth is equal for all pixels belonging to a window. Therefore small details are lost and object borders are blurred depending on the correlation window size. On the other hand, as shown by Nishihara and Poggio [22], the probability of a mismatch decreases as the window size increases. Multiple correlation functions (e.g. SSD, cross correlation, etc.) measure the correspondence between windows (see section 2.2). However the quality of the disparity map for all correlation methods are strictly related to the content of the correlation windows: the more the texture of a window differs from others, the less is

[1] At least 10 fps

[2] Personal computers mounting a Pentium III processor or higher, running with at least a 700 Hz clock

[3] Point Grey and Videre Design

the probability to obtain an erroneous matching. The image quality plays an important role in algorithm precision. A picture with a good contrast, a correct luminosity, a proper color scale and good borders definitions generates more reliable correspondences. As described by H. Hirschmuller [21] the correlation curve is not only used to determine the correspondence but also to determine the probability of a false matching. An other technique reduce the probability of having a false matching by applying the symmetric consistency check[23]. This approach compares the place of highest similarity obtained by computing the correlation from left to right and vice versa. If the two resulting disparities coincide the consistency is respected. By resuming we can assert that a low window texture quality (too homogeneous) is one of the main factor involved in errors generation. This facts will be taken into account to profile our hardware requirements.

3.2 Hardware requirement for real-time stereo

As explained in the previous section, the quality of the images plays a crucial role to obtain a precise disparity map. Different problems can affect the quality of the resulting image.

- Luminosity balance: to avoid this problem the cameras shutter time[4] and the gain level should be variable.
- Low luminosity: to handle low luminosity situations the camera should have a highly sensitive sensor.
- Motion blur: this phenomena appears when moving the camera during the image capture process. To reduce this effect the shutter time should be as small as possible. A mix of good quality lenses and cameras sensor helps to reduce the exposure time and as consequence the shutter time.

Cameras view vectors of a stereo vision system are hardly perfectly parallel, to partially try to settle this defect a calibration can be performed. In chapter 6 we further explore this problematic and how to minimize its troubles. Since the calibration process is made once in a while the cameras view vectors should not change. The stereo head has to be very solid, not susceptible to vibrations or small chocks. Another important aspect is the synchronism between the two cameras. Images should be captured at the same instant. An external trigger is suitable to solve the synchronism problem and allows us to employ the stereo head on mobile robots.

3.2.1 Commercial cameras solution

In the following section we compare some commercial cameras solutions to select a suitable candidate for our stereo vision system. We also briefly expose some ready to use stereo heads, which directly compute depth maps, and we further investigate for wireless solutions. Unfortunately for our purpose, all stereo heads we found were out of size. We choose as examples three binocular systems produced by Point Grey, Videre Design and Focus robotics (figure 3.1). This stereo heads are delivered with their specific software. Such a ready to use product has multiple advantages. First user do not have to care about calibration and images rectification. Also, these stereo heads are able to produce real-time depth maps thanks to a dedicated hardware or stereo engines.As mentioned, we also investigated the wireless domain. Wireless is a very practical solution especially for our robots, unfortunately we did not find any applicable product. Basically there are two kinds of wireless cameras, analogical ones that "broadcast"

[4] exposure time

(a) Videre Design (b) Point Grey (c) Focus robotics

Figure 3.1: Raedy to employ Stereo heads.

the raw image on a specific frequency, and IP cameras that digitalize images and transmit them on a wireless IP network. We focus our investigation on the last category, since we need a bidirectional communication with the camera in order to handle it. We found only few cameras with this characteristic and a reasonable size which could match with the mobile robots:

- AXIS Network Camera: 85x55x34 mm / 177g / No trigger
- Smart IP video system: 132x41x157 mm / - / No trigger
- Panasonic BB-HCM311: 100x100x73 mm / 300g / No trigger

The most important lack in these models is the absence of anything comparable to a trigger allowing us to synchronize the cameras capture.
Since we could not find any usable stereo heads or applicable wireless solutions, we focus our research on standalone pinhole cameras. A big variety of digital cameras are available on the market, to narrow our research we fixed some essentials criteria.

size Our stereo head has to fit with the robot body. The stereo head will be composed of two pinhole cameras whit a global size not bigger as 45x75x90 mm.

weight The stereo head must not be too heavy, otherwise it will affect the robot agility. We estimate a maximum weight of 100g.

exposure The shutter time should be modifiable to better avoid motion blur.

gain The gain should be used to compensate a shutter time reduction.

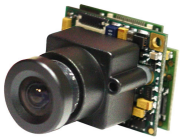
synchronization Cameras should have a mechanism to synchronize the image capture.

Table 3.1 summarize our investigation and compare some common features.

Final consideration

Cameras with self-synchronization offer a minimal advantage respect to triggered cameras, since an external trigger is easily built. Thanks to the CMU 1934 driver or cameras producers SDK toolkits it is possible to conceive a multi thread application that waits for the trigger signal to acquire perfectly synchronized images. As already mentioned the weight and the size are two fundamental aspects that we cannot ignore, therefore we reduced our choice to: Dragonfly,Firely MV , Videology and the Fire-i. Finally the Poin Grey/Firely MV has bean chosen because of its documentation, which is clear, because of its accessible price, and the size perfectly fitting our needs.

Table 3.1: Digital Camera comparison table

Image	Manufacture/Model	Size (mm)	Weight (g)	Resolution	Colors	Trigger	Shutter	Syncro	Interface
	Unibrain/Fire-i	59x53.5x19.5	35	640x480	Yes	No	> 0.3ms	No	IEEE-1394a
	Videology/Board Cam.	22x26x32	-	768x494	No	No	> 0.1ms	No	USB 2
	Poin Grey/Firely MV	25x40x19.9	8 (wl[a])	640x480	Yes	Yes	> 0.12ms	No	IEEE-1394a
	AVT/Guppy	48.2x30x30	< 50 (wl)	752x480	Yes	Yes	> 0.02ms	No	IEEE-1394a
	Prosilica/EC640	33x46x28	77 (wl)	659x493	Yes	Yes	> 0.01 ms	No	IEEE-1394a
	Point Grey/Dragonfly	64x51x25	38	640x480	Yes	Yes	> 0.12 ms	Yes	IEEE-1394a
	Point Grey/Flea 2	30x29x29	58 (wl)	648x488	Yes	Yes	> 0.01 ms	Yes	IEEE-1394a

[a]without lens

Chapter 4

Tools

In this chapter we introduce all libraries, applications and any kind of tool we conceived to help us during the project.

4.1 Calibration Tool

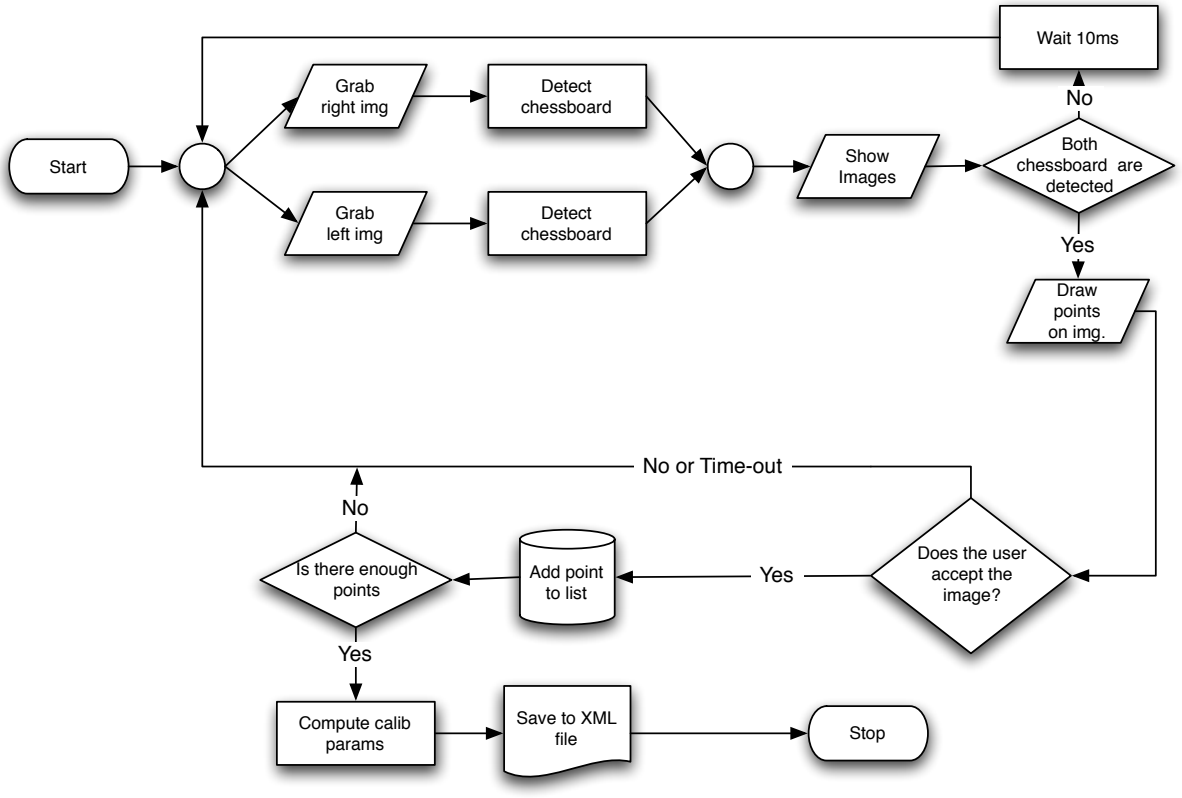


Figure 4.1: Semi automated calibration work flow

We chose to conceive a semi-automated calibration tool allowing us to rapidly detect the calibration rig and capture the images. As the aim of this project is not focused on calibration and stereo images rectification, a big portion of our calibration system will be based on existing implementations. The conception of a calibration system is largely influenced by the bounded

hardware. In our case we chose two identical cameras without zoom or any variable optics. Cameras are mounted on a rigid support so that their distance and orientations are fixed. Having a stereovision system with fixed intrinsic and extrinsic parameters allows us to take advantage from the photogrammetric calibration method. The Photogrammetric calibration result to be more precise than self calibration methods as motioned by Z. Zhang in [14]. Using a photogrammetric method, calibration is only performed once, unless the stereovision system is altered. Since intrinsic and extrinsic parameters do not change, they can be stored and further used to rectify images. This approach is less adaptive than the self-calibration one, on the other hand the photogrammetric method is more accurate, and increases the stereo computation performance. This is possible since the calibration values are already known. Nowadays the most widely useed and evolved calibration library is the Camera calibration Toolbox for MatLab. This library is based on Zhang's algorithm [15] and implemented by Jean-Yves Bouguet. Unfortunately the Matlab version of the calibration toolbox is not flexible enough for our purposes, therefore we took the OpenCV C++ implementation and built around it a semi-automated calibration application.

4.1.1 Calibration Tool Architecture

The semi-automated calibration tool is a standalone C++ application capable of computing and store on a XML file the calibration parameters. It also stores the images so that calibration parameters can be estimated with other applications. As second solution to estimate the calibration parameters we used the Videre Design SVS application [24]. The calibration using the SVS was obtained using the images provided by our semi-automated calibration tool. The schema 4.1 illustrates the images acquisition workflow. The application is composed by three external libraries: our synchronized frame grabber (section 4.2), the OpenCV R 1.0 and XMLParser v2.23 of Frank Vanden Berghen (a simple and small XML parser). To better allow reutilization, extension and portability of our application we chose to store calibration parameters in a proper XML format, annex E shows an example. The innovation in our calibration application is that we added a new decision layer establishing if the acquired calibration frame (image) is valid or not. Instead of taking a still picture at input we grab a video stream in which the calibration rig is continuously searched. The layer itself is extremely simple, but offers multiple advantages. Wrong frames not containing the full calibration pattern are automatically dropped and new ones are grabbed from the stereovision camera system. The decision layer start acquiring images from both cameras. Once on both images (left, right) a chessboard is found and corners are correctly detected the image acquisition stops. The user is asked to accept or refuse the last captured image as valid. A list of accepted images is established. Once the required numbers of images are accepted, the calibration parameters are computed and saved in to a XML file. The calibration application is especially conceived for binocular system, our project has a well defined configuration therefore the calibration application is not suitable for stereo vision systems with variable configuration. Although we have implemented the stereo application so that it can work with an arbitrary number of cameras, user needs to change a constant value and recompile the application in order to modify the number of input cameras.

4.2 Synchronized Frames Grabber

For synchronized frames grabber we refer to an application or library able to simultaneously grab several frames generated by a multi-camera system. Each frame for each camera should be acquired at the same instant. More precisely, the cameras sensor exposition should be synchronized, whereas no importance is given to the data transfer. A proper synchronized

fames grabber utility provides multiple advantages. If your stereovision system is conceived to be displaced (e.g. your cameras are mounted on a moving robot), the cameras synchronization is essential to determine a correct depth map. Without synchronized cameras the geometry linking their points of view varies during time according to the cameras system displacement. Imagine that you are a robot with two cameras; now with your left camera you take a picture, you make one step forward and with your right camera you take a second picture. Your captured images will not correspond and the computed depth map will be wrong. A simple method to avoid this phenomenon is to ensure that all frames are captured at the same time.

4.2.1 Specification

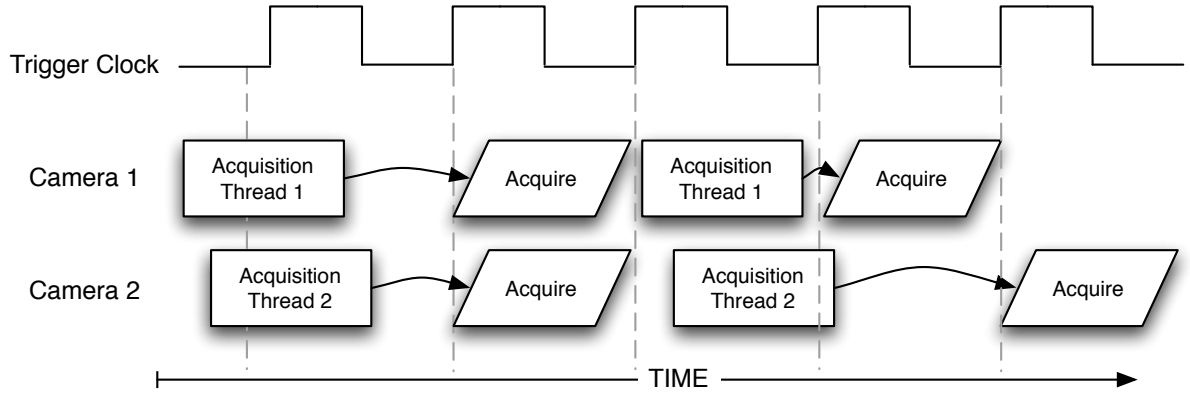


Figure 4.2: The two rectangles are acquisition threads, initializing cameras for a new image acquisition. The two rhombuses are the hardware acquisition processes initialized by the acquisition threads and started by the trigger signal. This schema shows that if the trigger frequency is too high, the acquisition threads cannot follow the rhythm and the image acquisition is no more synchronized.

A scientific high quality camera usually provides multiple ways to synchronize its image acquisition, trough an external trigger signal, or with some drivers providing software triggers. Our aim is to conceive a precise and robust synchronized frames grabber. Therefore we dropped the software trigger variant since it is not reliable. Indeed, to synchronize two cameras via software you need at least a multithreading application, but only one thread at time can be executed. How can two software trigger signal be synchronized if only one thread at time can send a trigger pulse? Even if you are running on a multi CPUs computer you should need two separated IEEE 1394 busses to send two simultaneous signals, and nothing will guarantee us that triggers signals will be sent simultaneously. An optimal software solution would be, sending a single trigger signal to one of the two cameras and use its output trigger signal to synchronize the second camera. Usually the camera output trigger is used to synchronize strobe pulses. This last solution has a minimal phase shift, but needs a more sophisticated hardware. Some cameras are able to synchronize themselves on the IEEE 1394 bus clock, but unfortunately only few cameras have this capability. To conceive our synchronized frames grabber library we used two external trigger signals generated by a single source, a PIC micro controller (see section 6.3). A single trigger source is connected with two cables. Since cables have approximately the same length the trigger pulse will reach the two cameras with a negligible delay. Most external triggers send trigger pulses with a fixed frequency, which has an important drawback. If the trigger frequency is to high the camera acquisition process may not be able to cope with

the speed, secondarily some hazard during the software execution may cause a phase shift as shown in figure 4.2. Decreasing the trigger frequency minimizes the probability of a phase shift situation but on the other hand a lower trigger frequency slows down the whole application. Our solution is an external trigger pulse generator controlled via software, figure 4.3 illustrate its workflow. Each time the camera driver is ready to capture a new image the synchronized frames grabber sends a signal via the RS-232 interface to an external micro controller, which will send two perfectly synchronized trigger pulses to both cameras. This solution avoids phase shifts since the trigger signal is sent only once both acquisition threads are ready to capture a frame.

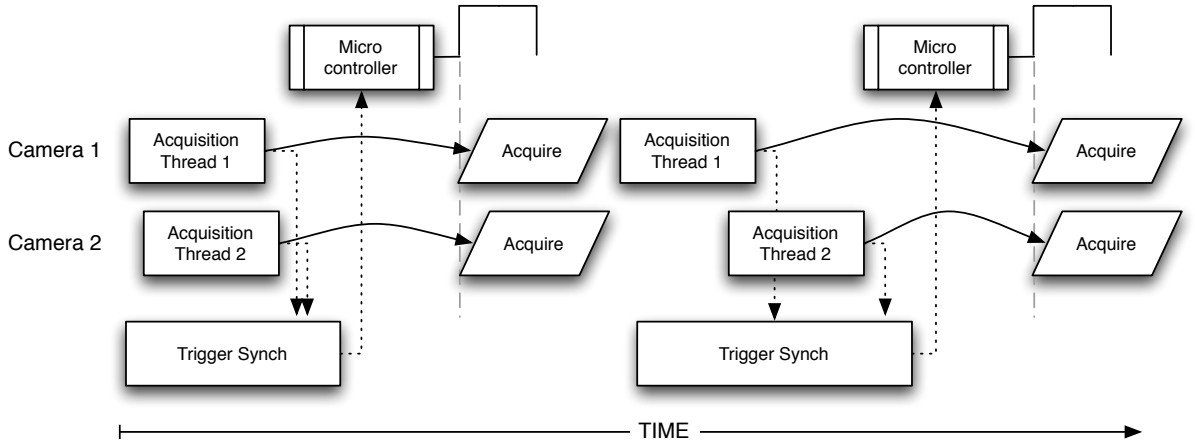


Figure 4.3: The two rhombuses represent the hardware acquisition processes. They are initialized by the acquisition threads and started by the trigger signal generated form an external micro controller. The Trigger synchronizer waits for acquisition threads. Once they are ready, it sends a signal to the micro controller. Also with largely out of phase acquisition threads this architecture always guarantees a synchronized images acquisition.

4.2.2 Architecture

The architecture of the synchronized frames grabber library is pretty simple. There is one main process and two threads which are responsible to grab cameras frames. The main process mainly coordinates threads and the external micro controller. It also provides some functionalities like image undistortion, rectification, cameras initialization and luminosity balancing. The first version of the library was designed to profit of the OpenCV library to rectify the images, in our final version we abandoned this solution and we employed part of the Videre Design Small Vision System [24], since its rectification method produces better results. The schema 4.3 show the synchronized frame grabbing workflow. Each time we want to acquire an image the main process wakes up both acquisition threads. Once they are ready to receive a new image, the main process communicates to the external micro controller, which sends a trigger signal to both cameras simultaneously. The following architecture guarantees a perfectly synchronized images grabbing also when acquisition threads are not temporary aligned. The cameras initialization phase ensure that all parameters are equally set for both cameras. The cameras luminosity balancing method is periodically called, it ensures a correct exposure time and more important it sets the same shutter time for both cameras. The frame grabber library not only synchronizes the frames acquisition but also ensures an equal exposure time.

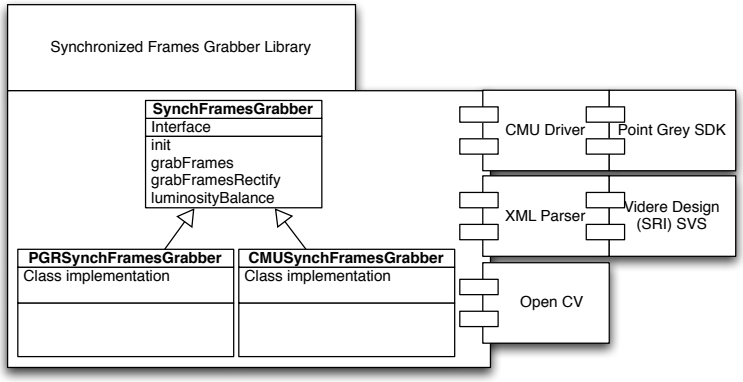
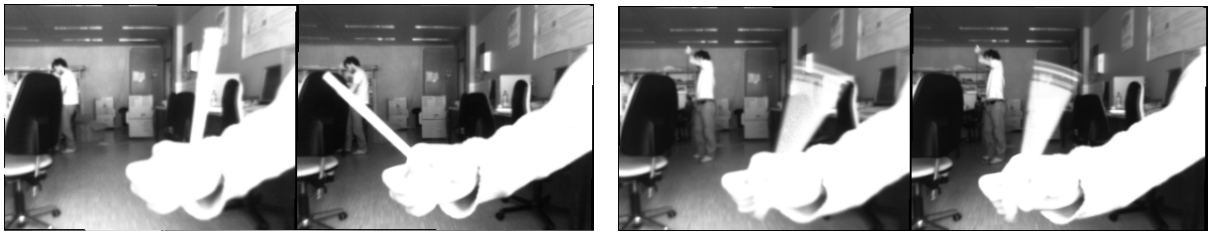


Figure 4.4: The synchronized frame grabber library package structure.

4.2.3 Conception

In order to develop an extensible synchronized frames grabber library and a flexible stereo vision tool kit we defined a C++ interface. The advantage of defining such an interface is that the stereovision system makes abstaction from the employed hardware. We defined the output that a synchronized frames grabber library has to produce, the initialization method and all kind of communication between the library and the employing application. For each kind of cameras, operating system or driver you want to employ a class implementing our interface has to be conceived. Therefore the binding is easily done since the stereo vision toolkit interacts with the synchronized frames grabber library whitout caring about its specific implementation and the type of connected hardware. As a practical example in the scope of our project we have implemented two classes, employing the CMU 1394 Digital Camera Driver and the Point Grey FlyCapture SDK. The CMU Driver works whit all cameras that comply with the 1394 Digital Camera Specification, it makes of it our most portable solution. The Point Grey FlyCapture SDK implementation is restricted to the Point Grey cameras, but it allows a better control and the acquisition of their raw image format. To demonstrate the effectiveness we made a video (http://www.elia.ch/svs/synchDemo.avi) and extracted two significant frames to report here. We have over exposed the images to better emphasize the phenomena. While the video capture we have moved a pencil in front of the cameras from left to right with a fast rhythm. Figure 4.5(a) is the result without cameras synchronization and figure 4.5(b) was obtained by activating the cameras synchronization feature.



(a) Left and right cameras images acquired without cameras synchronization

(b) Left and right cameras images acquired with cameras synchronization

Figure 4.5: An acquisition without and with the synchronization feature

4.2.4 Remarks

During the development we lost some hours around a bug that should be located on FireFly cameras or in the CMU driver. During the initialization phase some registers are not correctly set. To surround this problem we initialize both cameras, after that we set all parameters, we reinitialize both cameras twice but this time without reseting them to their default options.

4.3 Test and Tuning GUI

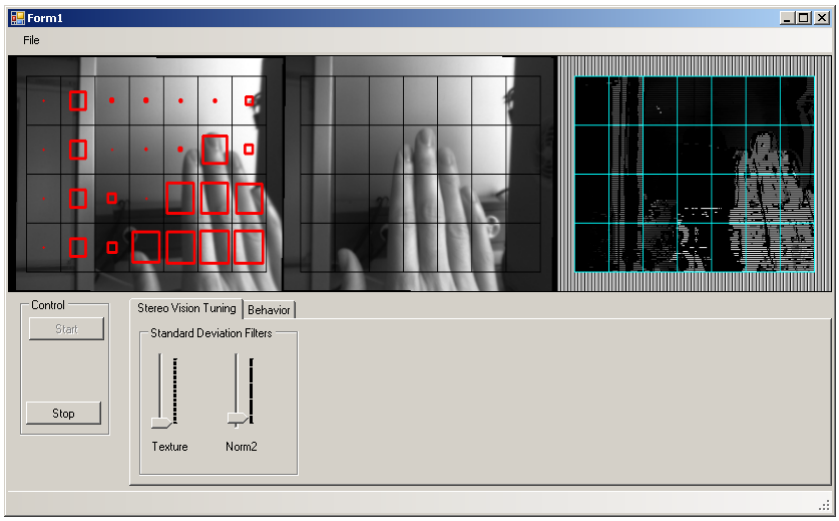


Figure 4.6: A screen shot of the Testing and Tuning GUI

To help us during the development to test our innovations and managing the correspondence algorithm parameters in real-time, we conceived a stand alone graphical user interface. The interface was developed in C#, since it allows to quickly conceive and adapt the graphical user interface thanks to its effortless development tool, Visual Studio .NET. As our stereo vision libraries are completely developed in C++ we conceived an intermediate dynamic library (dll) which is loaded by the graphical user interface. During the proceed of our project this interface becomes the point of conjunction between our work and the one provided by Matteo De Giacomi [5], therefore the interface has been extended for managing some parameters of the robots behavior. The further development of this tool has been abandoned since it was replaced by a more adapted architecture, which fully integrates this project and the work provided by Matteo De Giacomi.

Chapter 5

Correspondence Algorithm

5.1 Introduction

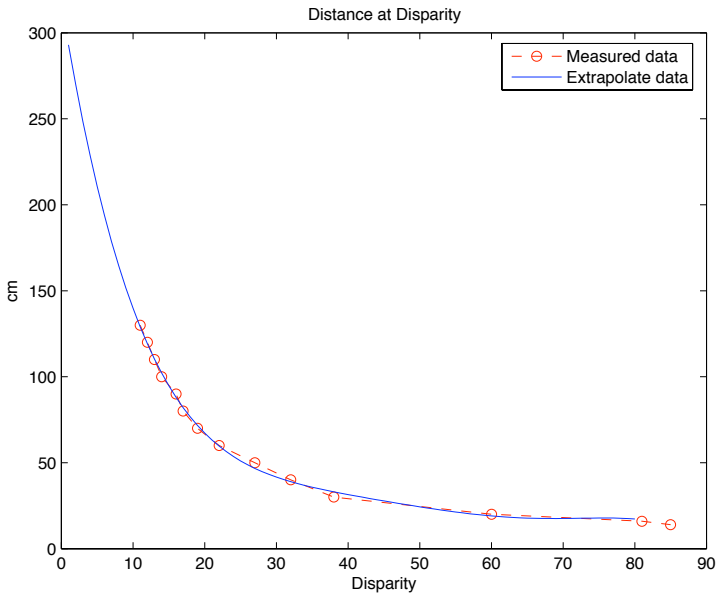


Figure 5.1: Extrapolated distance for disparity

In this chapter we explain the different correspondence algorithms we have tried or conceived with the aim of identifying the most adapted one for vision guided robotic applications. Different constraints limit our choice, the algorithms must be:

1. Portable: With the intent to port the algorithm on different platforms like Sparc architectures, any kind of micro controller or even on FPGAs; the algorithm must be simple without any special instruction (e.g. MMX instructions).

2. Performance: To let robots move fluently the algorithm must guarantee real-time performance or at least a frame rate of 5-7 fps. Combined with the portability constraint the algorithm must be computationally light to ensure good performance even if running on relatively slow architectures.

3. Precision: To avoid obstacles a complete information about the object in front of the robot is not needed, we just want to know if there are some obstacles and their distance from the robot.

We can distinguish two basic types of correspondence algorithms: feature based and area based algorithms. This last category can be further divided into local and global correspondence methods. A more complete description is given in chapter 2.2. Our research is addressed to area based algorithm, since feature based ones do not generate dense depth map and have difficulty to match smooth surfaces. The main challenge of adopting an area based correspondence algorithm is its computational cost. An area based algorithm produces a dense depth map, which means that for each pixel of an image the algorithm tries to find its mutual pixel on the other views. This process is quite computationally expensive, but profiting on optimization techniques and by preprocessing pixels we can reach a good compromise between the depth map density and its computational cost.

The final output we want to produce is slightly different from usual depth map functions.

(a) Stereo image (only the left one) (b) Disparity map (c) Depth grid

Figure 5.2: From source to output.

Our library is conceived for robot vision guided applications, therefore it do not return the depth value for each pixel but the median value at each sector. The field view is decomposed in multiple sectors as showed in figure 5.2(b). From the rectified input images to the depth grid output the algorithm passes trough three consecutive phases:

1. The correspondence phase is the heaviest process, it finds the correlation between the pixels of the two images (left/right) and produces the disparity map, figure 5.2(c).

2. From the disparity map, a disparity median grid is easily computed by estimating the disparity median value for each sector.

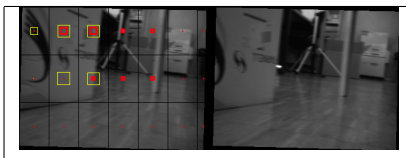
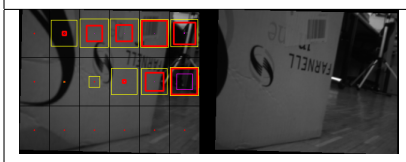
3. The depth values of each sector is calculated form the disparity median value thanks to a conversion function, obtained by calibrating our stereo vision system. To calibrate the stereo vision system we have placed a target at a distance of 130 cm and we saved the computed disparity every 10 cm while approaching the target. These measures allow us to establish a relation between the real obstacle distances (cm) and their corresponding disparity values (0 to 80 for our algorithm). This relation takes the form of a conversion function computed by interpolating the performed measures in order to obtain a polynomial function. This allows us to extrapolate the missing values. Figure 5.1 illustrates the interpolated conversion curve.

Measures and MatLab code to interpolate the measures and extrapolate the missing values are available in annex B. Each measure is the median of the values of at least three sectors where the target was present.

In the next sections of this chapter we present five correspondence algorithms. We will describe our effort and the road map needed to find out the algorithm which we believe to be the most adapted for vision guided robotics applications.

5.2 Dynamic Programming Algorithm

Table 5.1: Case study

	Normal situation
	A small escape
	Low luminosity

For our first try we decided to test an existing implementation provided by OpenCV. The largely employed Intel® vision library, includes a binocular stereo matching function. This is a good starting point for testing our hardware and rectification methods. After some tests it was clear that the dynamic programming approach employed by OpenCV was not adapted to our goals. The dynamic programming tries to minimize the global error by mean of non optimal local assumption. The algorithm produces ineffective results when it has to manage unusual environment situations. We have analyzed some unusual situations and the related algorithm behavior, to document our hypothesis. Results are presented in form of images, the depth information is given as median value for each cell of a 6x3 grid. Images represent the left and the right camera image already undistorted and rectified. We have superposed a 6x3 grid on the left image, inside the grid a red square represents the median depth value. The square size corresponds to the computed disparity for that sector, the bigger the square, the nearer the obstacle. A yellow and a violet square are also present. The first represents the 20st percentile whereas the latter represents the 80st percentile.
This three values are useful to estimate the certitude of the median value for a determinate sector, the more these three values are close, the higher is the probability for the median to represent the true depth value in that image region.
Table 5.1 illustrates some situations we chose as example. Presented images have been extracted from a video we made to test the algorithm. The video was produced by mounting the stereo head on the Amphibot II robot and registering in real-time the algorithm output during a 2

minute robot promenade. Afterwards we extracted some typical and atypical situations from the video to examine the algorithm behavior. The first image (Normal situation) shows what we expect to obtain. The following two examples are to demonstrate the bad assumptions which the algorithm mades to minimize the global error. In the first case a small escape door on the right is totally missed. In the second case we notice that an homogeneous texture produce a completely wrong depth map. As conclusion of this test (whose results are partially presented on this report) we decided to abandon the OpenCV implementation and start developing our own correspondence algorithm. More case studies are available in appendix A.

5.3 Block matching Algorithm

Block matching algorithms establish the disparity for a given block called window, by analyzing the result of the correlation function. The matching model is simple. Unlike the dynamic programming the decision is taken independently from any higher level information, since the only relevant data are relative to the current window. For every window a similarity value is returned for all disparities. These values are produced by the correlation function. The number of maximal disparities to check is predefined. The more the maximal disparity value is high the more the computation is slowed down, whereas the more this value is small the more the minimal detected distance is limited. In our case we fixed the maximal disparity to 80 pixels, which with our stereo vision system allows us to detect objects from a minimal distance of 15 cm. As correlation function we chose the Sum of Square Differences:

$$SSD(x) = \sum_{i=0}^{n} (I1(x+i) - I2(x+i+d))^2 \tag{5.1}$$

The SSD is the second lightest function in terms of computational cost, but is more discriminant than the lass expensive one, the Absolute Sum of Differences. We should point out that our SSD function is one-dimensional, which allows us to better optimize the code. As consequence the comparison window has a one-dimension as well. In chapter 7 we compare our algorithm with the Videre Design SVS [24], and we observed the a one-dimensional window does not compromise the algorithm quality. The Small Vision System is an efficient implementation of the SRI stereo algorithm, that was recently purchased by the Videre Design company. The SRI stereo algorithm is hold secret but we know that it adopt a block matching technique merging matching at multiple resolutions. Thanks to some filters they are able to increase the algorithm reliability. A good correlation function itself is not enough to establish a correct correspondence, therefore we decide to enrich our matching model by adding a cost function. The cost function transforms linearly the computed correlation values by penalize high disparity values. This is done since an image window representing close objects is less probable.

$$cost(x) = x * (d * 0.16 + 1.0) \tag{5.2}$$

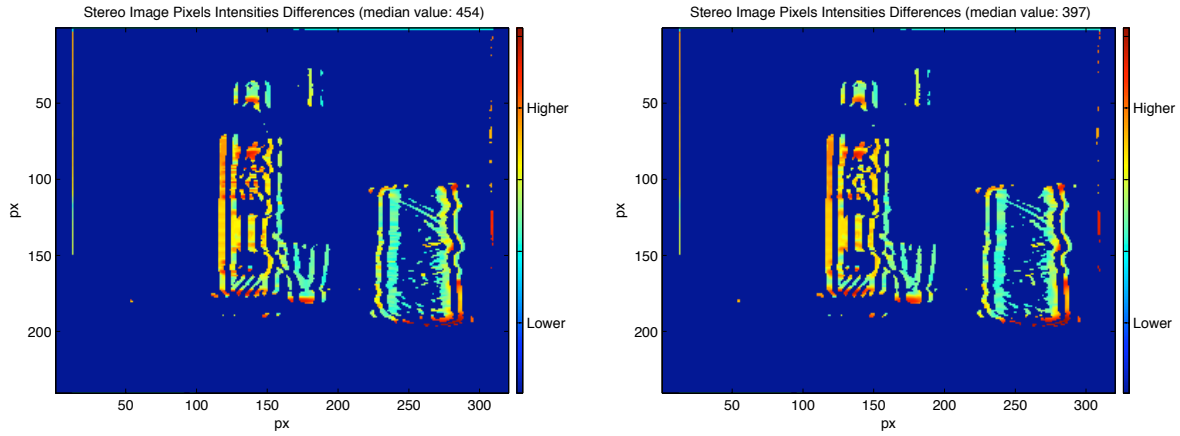
where d is the actual disparity and x is the SSD value obtained for that d. We have empirically chose 0.16 as cost factor since we have observed a maximal increase of the disparity map quality. For our first versions as consistency test we adopted the symmetric check [23][25]. The test defines as valid a match obtaining the same correlation for a given pixel by computing the correspondence from the left to the right image and vice versa.

5.3.1 Stereoscopic Machine Vision Algo v.1

Our correspondence algorithm is based on the simplest correspondence model, a scanning method and a one to one correlation function. We compute the correlation between one window

of the left image and multiple windows of the right one. The correspondence having the highest similarity score is selected. We believe that for vision guided applications using the most simple correspondence model is gaining, since the noise is specific to each pixel (not distributed over a scan line, etc.). The output is a raw depth grid (presented in section 5.1) where each cell is the median value of a region of pixels. Noisy pixels can therefore be dropped and only the ones having an high confidence value can be taken into account. The number of pixels selected to compute the median determines the confidence measure. If the number of dropped pixels for a specific region is too hight the depth value will not be representative. Behind a certain threshold we state the depth value is unknown. The threshold value has been empirically fixed at 20%. No tests has been made to determinate its optimal value since the algorithm v.1 is a draft.
The correlation function we adopted is the Sum of Square Differences already introduced at the begin of this chapter. The scanning method is the same of all area-based correspondence algorithms. For each pixel of the left image the similarity is tested by computing the correspondence with each pixel of the right image being on the same Y coordinate and within a range defined by the maximal disparity. For further details about the correlation functions and scanning methods see section 2.2.

Vertical Smoothing



(a) SSDPI without smoothing, median value 454

(b) SSDPI with smoothing, median value 397

Figure 5.3: Difference of pixels intensities while computing the correlation with and without vertical image smoothing. The color indicates the Sum of Square Differences of Pixels Intensities value for each correlated pixel.

In chapter 6 we describe the difficulty of obtaining vertically aligned stereo images. As we could not obtain perfectly aligned stereo images we decided to apply a vertical smooth on input images to partially overcome this lack. The smoothing effect is achieved by convoluting each vertical scan-line of the image whit a Gaussian, where σ determines the smoothing extension. To measure the effectiveness of the filter we computed the correlation (test data are generated using the Algo v.3 correlation algorithm, see section 5.3.3) with and without smoothing and we compared the median value of the SSDPI[1]. Pictures 5.3 prove that a better disparity map is

[1]Sum of Square Differences of Pixels Intensities

obtained by applying a vertical smoothing. To better understand why we take the SSDPI as quality measure for the computed disparity map reader is addressed to section 5.3.3.

Error Filtering

We decided to implement the error filtering described by H. Hirschmüller in section 4 of [21]. A match is identified as uncertain and dropped if the difference of the two highest minima of the correlation function is under its threshold value. A function with severals minima, means that multiple good matching are found. This is symptom of a too homogenous texture or a repetitive pattern. After computing the correspondence of a pixel we examine its correlation function to determine if hold or not the computed match. As second criteria to filter bad matches we adopted the symmetric check [23]and [25]. This technique is later explained in section 5.3.3.

Remarks

The algorithm is excessively slow since we do not have implemented any optimization. There are three threshold values to set:

- minimum required positive matches
- minimum difference between the two highest correlation minima
- symmetric check sensibility

This makes the algorithm less robust, since these values have to be adapted if the environment changes. The error filtering is too naive, since it does not take in to account the peaks distribution (e.g if the two minima have a close disparity, they are most likely representative). A more sophisticate filter should be able to distinguish an untextured region from a repetitive pattern. The vertical smoothing produces better results, it overcomes a possible bad rectification or cameras misalignment making the algorithm more robust.

5.3.2 Stereoscopic Machine Vision Algo v.2

From its progenitor we retain the correlation function, the scanning method, the symmetric check and the vertical smoothing. What we have improved in the second version is the error filtering. We have also added a preprocessing phase, the texture quality filter. To increase performances and reduce false positive matches we developed a technique to measure the texture quality of a correlation window to filter the pixels probably yielding to a mismatch. Applying the texture quality filter before the correlation process allows us to increase performances, since the correlation does not have to be computed for all pixels.

Mismatch Error Filtering

Instead of simply testing the difference of the two highest minima of the correlation function, we compute the standard deviation of the whole function. The standard deviation is a statistical information telling us how much the similarities are distributed over all possible disparities. If the standard deviation has a low value, good matches (points with an high similarity) are concentrated in few disparities, which is synonym of high reliability. On the other hand if the correlation yields to an uncertain situation (e.g. two points of high similarity with a far disparity) the standard deviation is affected and increases. Finally to determine if the correspondence is reliable or not, the standard deviation of its correlation function has to be lower than a certain threshold.

Texture Quality Filter

The correlation process computes the similarity between two windows by taking into account the difference of their pixels intensities or other factors depending on the correlation function it self. There are multiple correspondence functions e.g. cross correlation, absolute sum of difference, sum of square difference and so on. Anyway all these functions have a common peculiarity, the more a window content differs from other ones the more the resulting difference will be marked. This trivial observation leads us to the following question: what happens if we try to compute the disparity of an image region with an homogeneous texture (e.g. a completely white region)? The computed disparity will tend to be wrong, because the windows intensities differences are not sufficient to discriminate the correct match from other ones. The innovation we propose is to preprocess the comparison windows to skip those not having a sufficiently heterogeneous texture, instead of computing the correlation for each pixel dropping those not producing a discriminant intensities differences. Like for the mismatch error filtering, we use the standard deviation as confidence measure, where the window pixels intensities is the function. To establish if a window is sufficiently textured the standard deviation of its pixels intensities is computed. An homogeneous window contains pixels with about the same intensity, therefore the standard deviation results lower comparing to a heterogeneous window. Only the windows having a standard deviation higher than a certain threshold will be taken into account to compute the correspondence.

Remarks

The texture quality filter is a good solution to minimize the computational cost since it allows to skip the windows probably yielding to a mismatch. Unfortunately the standard deviation is not a good criteria to filter the untextured windows since it is influenced by the scene luminosity. If the ambient illumination decreases also the standard deviation value will decrease by consequence. Therefore the threshold has to be readjusted. The mismatch error filtering results to be redundant, the texture quality filter already drops those windows yielding to a mismatch. By precisely tuning the texture quality filter threshold, the mismatch error filtering is almost useless.
A second problem we noted is that the symmetric check error filtering has some troubles when the obstacle is too close to the cameras, since the points of view are too different.

5.3.3 Stereoscopic Machine Vision Algo v.3

Stereoscopic Machine Vision Algo v.3 is our final result, it focalizes all our effort to find the most adapted stereo algorithm for vision guided robotic applications. To reach the highest robustness degree we retained all valid solutions and exciting innovations we found during this project. The correspondence model has not changed, to determinate a match we still use the SSD as a correlation function combined with the linear cost transformation.
Otherwise the Stereoscopic Machine Vision Algo v.3 largely differs form its predecessors, it was conceived to be fast on any architecture (like its previous versions) and extremely robust to the environment transformations. To achieve the first objective we adapted the texture quality filter technique introduced in version 2, which allows to ignore those pixels probably yielding to a mismatch. The second objective, the robustness, was achieved by modifying the filtering technique which is now fewer influenced by the scene luminosity. We also slightly adapted the correlation process in the way that it was able to manage dysfunctional behavior (e.g. misaligned cameras view vectors). The most important difference in comparison to its predecessors is that the Stereoscopic Machine Vision Algo v.3 does not use the symmetric

validity test proposed by Pascal Fua [23][25], since it doubles the computational cost. To cover the lack we propose an alternative validity test which partially allows to detect occlusions and possible mismatches. Next in this section we will describe the new texture quality filter, the new validity test, modifications we made to the correspondence process and the code optimizations.

Texture Quality Filter

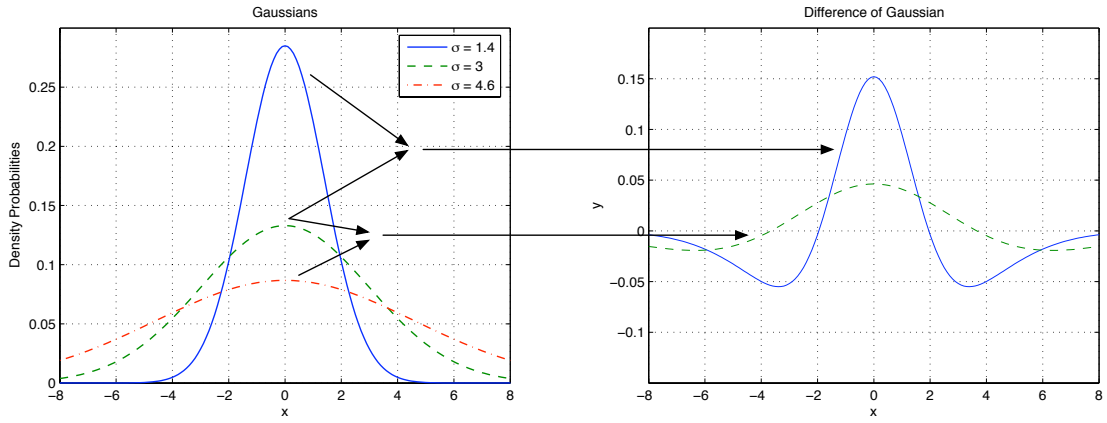


Figure 5.4: Difference of Gaussian functions.

For the third version we abandoned the standard deviation as measure of the texture quality. This because of its proportionally dependence of the pixels intensity average and the consequent need to adjust its threshold. To substitute the standard deviation as texture quality measure we adopted a method usually employed to discover the points of interests, the difference-of-Gaussian.
To determine if a pixel resides in a textured image zone, we convolute the window surrounding the pixel with two difference-of-Gaussian functions at a different scale. The firs scale is more sensible on shrunken intensity variations. It returns higher values if there is a significant pixels intensity variation concentrated in the proximity of the window center. The second difference-of-Gausian function has a larger scale (bigger σ), therefore it produces higher values if the pixels intensities are more homogeneous. The function sensibility depends from the two chosen σ values used to produce the Gaussians and they difference. An initial σ value is chosen and increased by a constant factor to determine the higher scales. David G. Lowe in [26] has demonstrated that if the σ values used to produce multiple scales of difference-of-Gausian functions are increased by a constant factor there is no need to normalize they result values. To define our two difference-of-Gausian functions we have chosen an initial σ and an increment factor Δ, from this two parameters we have obtained three Gaussian with they respective σ values equal to: σ, $\sigma + \Delta$, $\sigma + 2\Delta$. From this three Gaussian we produce two difference-of-Gausian functions by subtracting the two nearer Gaussian functions, like illustrated by figure 5.4. The convolution kernel size depends of the Difference-of-Gaussian domain.
The main advantage using the difference-of-Gaussian as texture heterogeneity measure is that it is less influenced by the pixels intensity average. This is explained by the fact that the decreasing of the pixels intensity average affects both scales. We provide in annex C a MatLab code to generate a .h file which contains the constant values used in our library code.
Figure 5.5 illustrate an example of the image filtering output, we have extracted a pixels row from an image and identified those regions having a too homogeneous texture. The image was

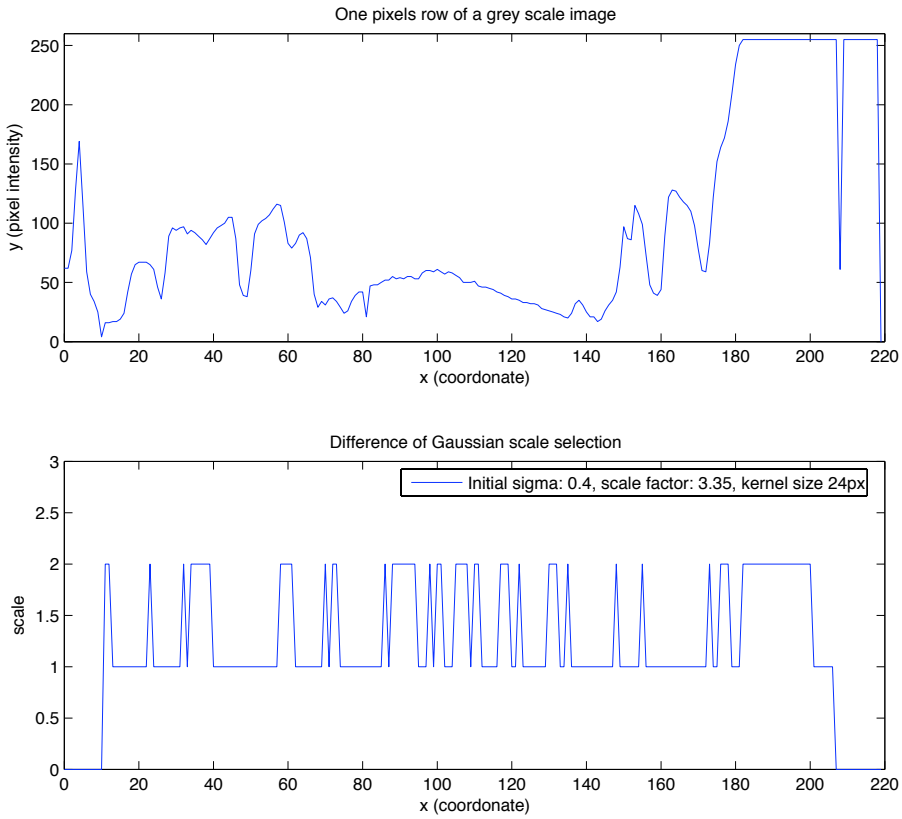


Figure 5.5: Texture quality filtering results of an image row. A window of 24 pixels is convoluted with two difference-of-Gaussian functions at a different scale. The scale producing the highest value is taken for that coordinate. Scale 2 identify those image regions which has a too homogeneous or smoothed texture.

produced using the MatLab code in annex D.

Validity Test

The aim of the validity test is to identify and drop those pixel correlations which we estimate to be wrong. A well known and largely exploited validity test is the symmetric consistency check [23][25]. It identifies as valid a match that has the same disparity values when computing the correlation from left to right and vice versa. We will show that the symmetric check is not the most adapted validity test for our needs. The main advantage of the symmetric consistency check is to identify occluded regions and improve the objects edges precision. This feature is not essential since, as our final output is a raw depth grid, we do not need to compute a precise pixel correlation. We decided not to adopt the symmetric consistency validity test because of its drawbacks which are more important compared to the advantages that it provides. The symmetric test is computationally expensive since we need to compute the correlation twice. The second important disadvantage is that the symmetric consistency check rejects the correlated regions which do not exactly correspond in both images (left, right) e.g. occluded regions. This behavior is acceptable, even correct, if your goal is to establish a precise depth map, not if

you want to avoid a close obstacle. As we will demonstrate later, the more an object is close to the cameras the more the two captured images will differ. For this reason the symmetric consistency validity test will suppress too many correspondence matches.

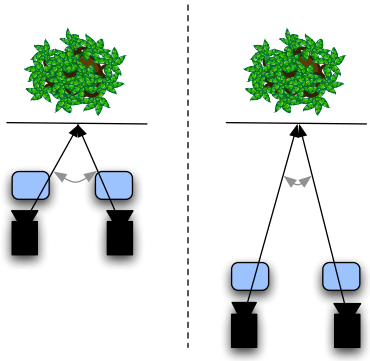
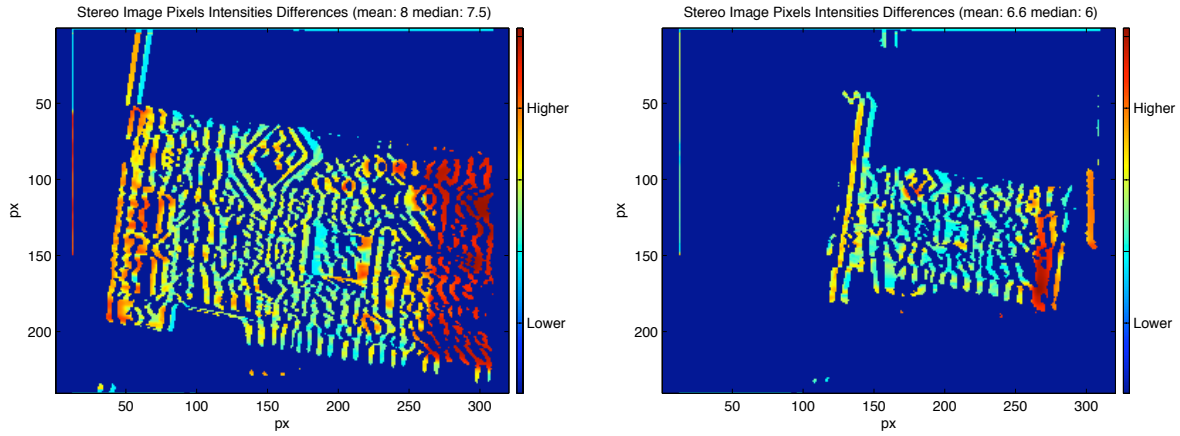


Figure 5.6: The angle between the two points of view increase by approaching the object.

The fact that the closer there is an object the more the two images (left, right) differ, is theoretically explained by the increase of the angle between the points of view and the consequent stretching of a same physical point on its projections (left, right) e.g. see figure 5.6.



(a) A PCI card close to cameras.

(b) The same PCI cars more distant.

Figure 5.7: Comparison between two stereo scene with the same subject at different distances. (The color indicate the SSDPI value for each correlated pixel) Lenses distortion induce to increase the SSDPI value on image edges.

As already explained in previous section 5.3 to establish the correlation between pixels of the left image and those of the right one, we adopted the Sum of Square Difference as correlation function:

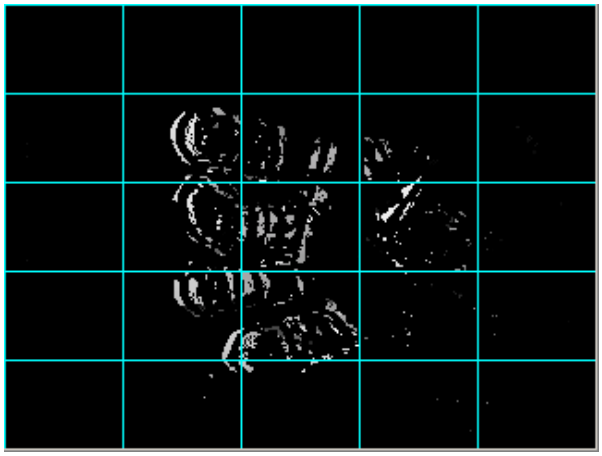
$$SSD(x) = \sum_{i=0}^{n} (I1(x+i) - I2(x+i+d))^2$$

where n is the window size, d is the disparity and I(x) the intensity of pixel x. Once the best correspondence is found we store the corresponding SSD value as a matching confidence measure. More precisely each match has its own Sum of Square Difference of Pixels Intensities value, which in the following of this report we refer as SSDPI.
To demonstrate in practice that the closer there is an object the more the two images differ, we chosen two scenarios. We taken an object and we computed the correlation once grabbing the images close to the object and once far away. Then we plotted the computed SSDPI for the two scenarios. If the two images (left, right) are exactly equal all correlated pixels will have a SSDPI value equal to zero, the more the two images differ the more the SSDPI value will increase. In our scale the SSDPI goes from 0 to 12 (12 is the *log* of the maximum SSDPI we obtained for that experience). Figure 5.7 shows how the SSDPI median and mean value is higher for closer objects.
To overcome this problematic we propose a new validity test which identifies as valid the correlations that have the SSDPI[2] value lower than a dynamic threshold. The threshold is computed by trying to correlate the left image with it self and choosing the minimum SSDPI as reference. Consequently the threshold will be higher (less discriminant) for textured zones and lower in proximity of homogenous regions. An other advantage of fixing a threshold is that we can partially filter arbitrary matches like occluded regions. Our validity test is less discriminate in presence of close objects as our measures will prove.
To measure the efficacy of our method we computed the correlation of a scene containing a close object, then we plotted the SSDPI value for all dropped correlation by the symmetric check and by the self check validity test. Figure 5.8 shows the test configuration and the obtained depth

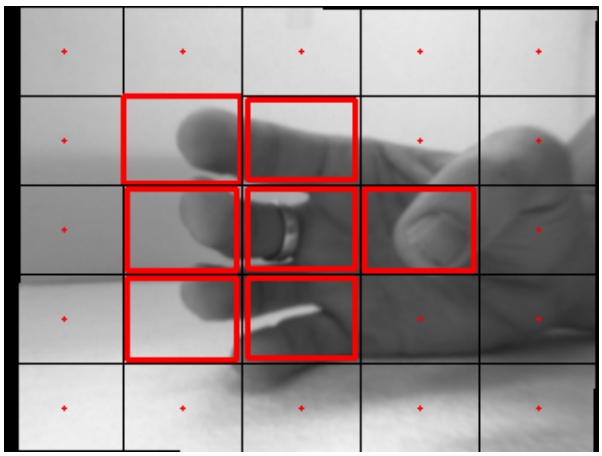
[2]Sum of Square Difference of Pixels Intensities

(a) The left image, Matteo's hand.

(b) After texture quality filtering.



(c) Depth map.



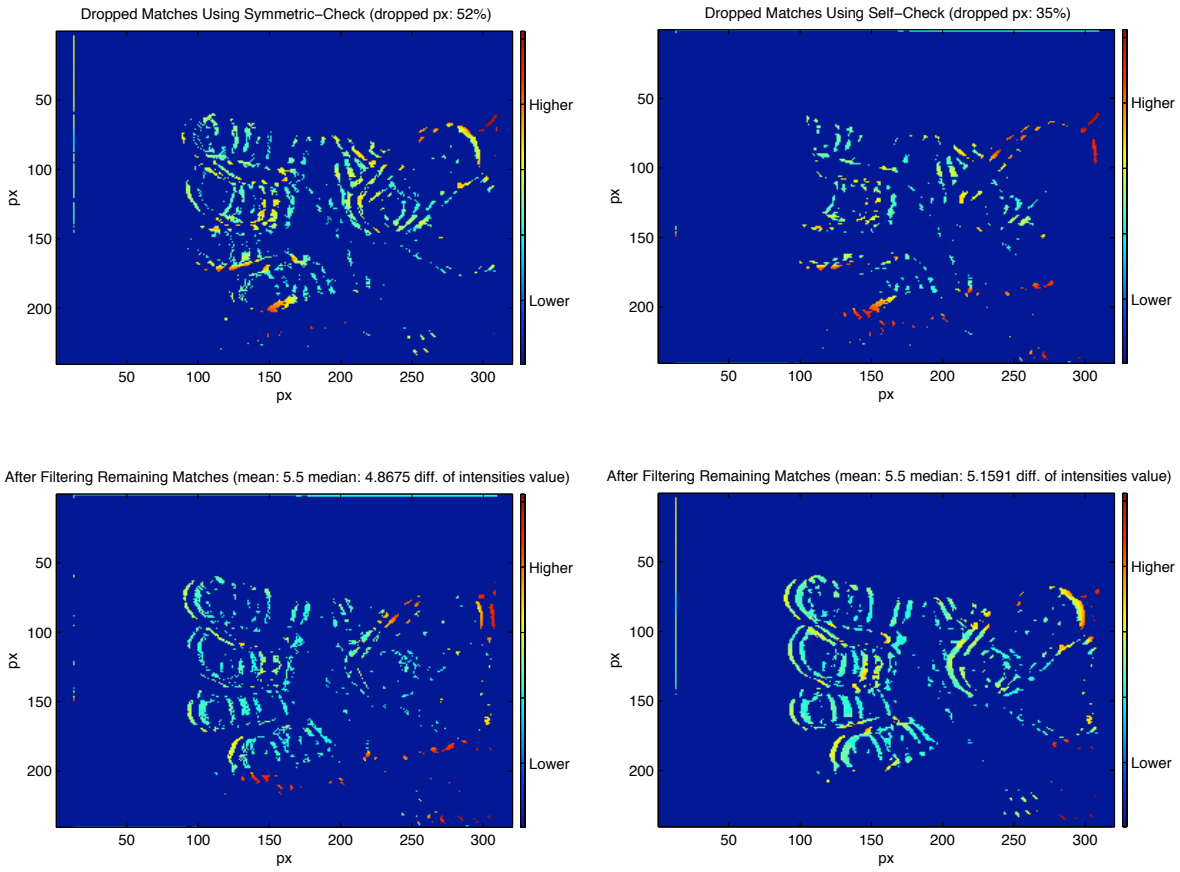
(d) The raw depth output. The red squares indicate the proximity.

Figure 5.8: This is the experience configuration we used to compare the self check and the symmetric consistency validity test. The comparison is made on figure 5.9.

map. Figure 5.9 illustrate the comparison between the symmetric consistency check and the self check validity test. The symmetric consistency check dropped the 55% of total correlated pixels, our method dropped 35% of total correlations and both methods has produced a comparable mean and median SSDPI value.

Back checking

Do to the difficulty to perfectly align our stereo vision system cameras view vectors we have introduced a validity test that usually is not required. The view vectors of our cameras horizontally cross they self after a certain distance (> 2.5m). Whit an hypothetic perfect hardware and vision system resolution the disparity zero correspond to the infinite distance. The image rectification is not able to completely correct a miss alignment, this subject is discussed in 6.1. In our case the disparity zero is already achieved after about two meters, limiting the maximal measurable distance to that value. To overcome this problem our algorithm as an initial disparity that cold be negative. If during a post processing phase a negative disparity is found,



(a) (b)

Figure 5.9: Comparison between the symmetric consistency validity test and the self check validity test. (The color indicate the SSDPI value for each correlated pixel).

it means that that image region is behind the cameras view vectors crossing distance. Such a situation is corrected by assigning the maximal disparity to all matches having a negative disparity. Please note that miss cameras alignment was solve with our second prototype of cameras support, the back checking test becomes useless and has been disabled.

Optimization

Code optimization is inspired by [17] section 2.2.6, when computing the correspondence between to windows, all correlation values for each different disparity are stored in a table. Since the next window to compare is shifted by one pixel on the left the already computed correlation values can be taken and slightly adjusted. The square difference of the pixel before the actual window for all disparities has to be added, and the square difference of the last pixel of the window for all disparities has to be subtracted. The explanation is tricky to understand, interested reader is addressed to [17] or picture 5.10 could provides a further example. This optimization technique is only partially applicable to our algorithm, since the texture quality filter drops pixels. We

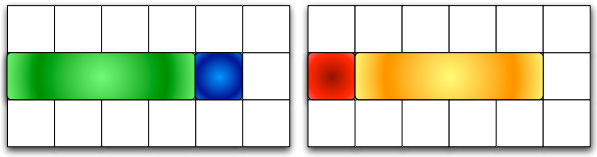


Figure 5.10: In this picture we represent a region of the left image. For one disparity the green rectangle is the SSD for a window, at the same disparity the yellow rectangle is the SSD of the window shifted by one pixel. The SSD value of the yellow window has not to be fully recomputed, it is the SSD value of the green window minus the red square SSD value plus the blue ones.

can adopt it only when a group of consecutive windows pass trough the texture quality filter.

5.3.4 Stereoscopic Machine Vision Algo v.4

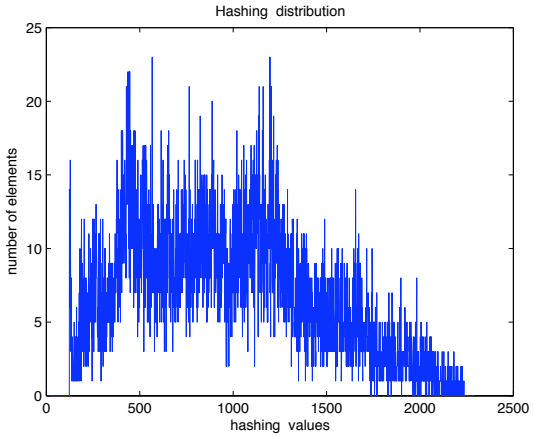


Figure 5.11: The graph shows the number of elements per hash value of a hash table where all windows of an image has been stored.

Our fourth version is just a feasibility test. We completely dropped the scanning method and we tried a new approach. In stand of comparing a window rising from the left image with all right windows with the same Y coordinate, we put all left windows in a hash table. Once all left image windows are in the hash table the algorithm scan the right image and step by step it compute the hash value of the right windows to search in the hash table all possible correspondences. All windows with a similar hash values are compared and only that having a close coordinate are taken in account to compute the SSD. Our hash function is simply the sum of the pixels intensities which is not an optimal distribution function since its depend of the input image. In a optimal situation (not too dark, not too clear, a full textured image) the windows distribution over the hash indexes is pretty good balance as shown in graph 5.11. Two windows with the same sum of pixels intensities are not necessarily equal but two window with a too different sum will hardly represent the same thing. If we want to have a tolerance of 5% on each pixel where the window size is 1x9 and the intensities values goes from 0 to 255 the sum vary between $\pm$ 150. With a tolerance of 5%, 300 indexes of the hash tables has to be visited to search a possible matching. Each index of the has table has in mean 6 elements, which make

1800 window to compare, this number further decrease since statistically only the 20% of that windows are in the proximity of the window to compare. Finally about 360 SSD comparison are made. This algorithm makes four time more correspondences computation in comparison to its predecessors, but it is highly more robust since it does not need rectified images or perfectly aligned cameras view vectors. We do not have further investigate on this algorithm, we only expose our idea on this report as a possible solution to handle bad quality input images.

Chapter 6

Stereo Head (Hardware)

As stereo optic for our robots Amphibot II [1, 2] and the Salamandra Robotica [3] we could not use any existing solution as they do not fit with the robots size. A research to find the most adapted stereo vision hardware was made, results are exposed in section 3.2.1. We chose to couple two Firely MV PointGrey cameras and to build our own physic support to bind them together. The stability and the precision of the physic support plays a crucial role, since without a correct images alignment the correspondence algorithm we employ would not work. In chapter 5 we present multiple correspondence algorithms, our last version is a robust method which do not need a perfectly aligned optic support, unfortunately its drawbacks make of it not the optimal solution. Managing bad aligned images increase the stereo matching incertitude and its computational cost, therefore we invested more time to conceive a reliable cameras support allowing us to adopt a simpler correspondence algorithm.

6.1 Cameras and they Support

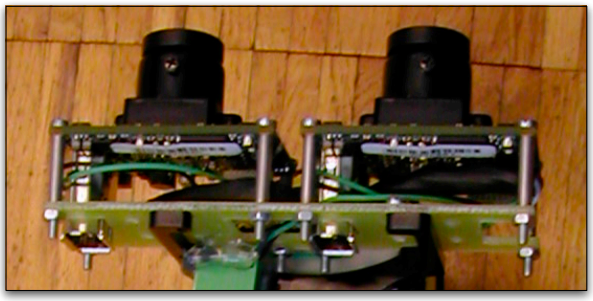
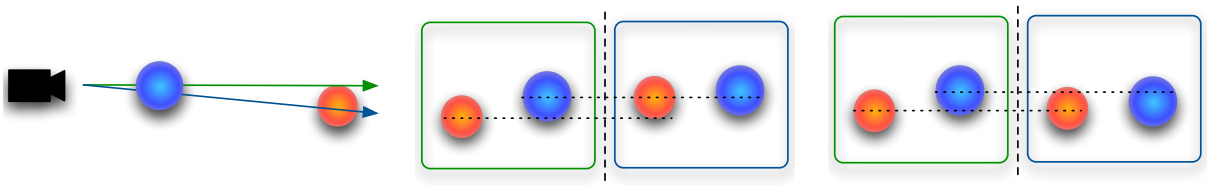


Figure 6.1: Cameras supports build with a RF-4 plate

Alessandro Crespi helps us conceiving our first cameras support and lending us his development card to generate the trigger signal. To reach a satisfying reliability we passed trough more prototypes. The first try was made of a RF-4[1] plate, we cuted and we perforated by hand, cameras was fixed trough 8 crews, figure 6.1. When we conceived our first support we underestimate the importance of a correct cameras alignment, and unfortunately our hand made cameras support was too imprecise. If the view vectors of the cameras are not parallel on the y axe the correspondence is not feasible, since the images are not vertically aligned and the

[1] FR-4 is a resin system material used for making printed circuit board.



(a) Cameras view vectors side-view (b) Cameras projections front-view (c) Red sphere alignment

Figure 6.2: This images illustrate a stereo system with two miss aligned cameras and they view vectors, in figure b the relative projections and in figure c an attempt to align the two images.

corresponding points amid the two images are not on the same height. Even by calibrating and rectifying the image this problem can not be solved. Taking a picture is equivalent to operate a projection from a three dimensional space to a flat two dimensional image and consequentially losing the depth information. We can not correct the miss alignment between two three-dimensional vectors without the third coordinate (the depth). As example look at figure 6.2, on the left we represent two miss aligned view vectors and two objects at a different depth. In the middle the projection of the two cameras, note the wrong vertical alignment. On the right picture we vertically shift one image to try to align they red sphere, but as you can see the alignment works only for a precise depth the blue sphere is now worst aligned. We emphasize the cameras alignment aspect as it is very important for the quality of the resulting depth map. We suggest to solve this problematic by conceiving an appropriate physical support, any other software adjustment solution (e.g. a recursive depth-computation and calibration adjustment method) is pointless and will result computationally expensive.
The second cameras support prototype was built by André Guignard it consist of a precise aluminum support where cameras are well fixed on, see figure 6.3. Unfortunately also this effort was not straight enough to align cameras view vectors. Do to time constraints we do not have the possibility to develop a third version. As further work we propose a new cameras support where bout cameras lenses are mounted on a single support pice, and therefore they should be necessarily aligned (e.g. look at picture 3.1(c)).

Figure 6.3: Aluminum cameras supports

6.2 Lenses

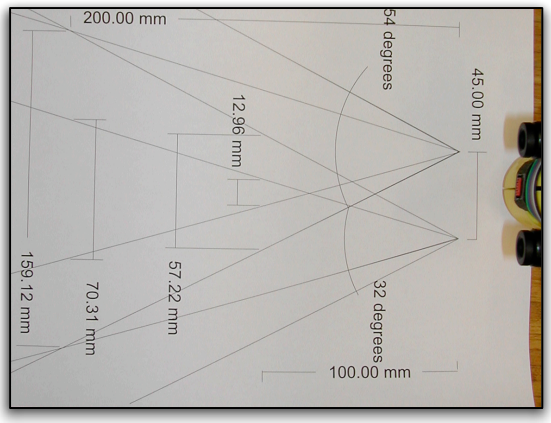


Figure 6.4: View angle with 32° lenses and 54°

FireFly MV cameras are delivered with a 32° angle of view lenses, which is wide not enough for our robots to move fluently in their environment. We have replaced the original lenses with two Boowon BW38B-1000 lenses (kindly offered by Peter Kim, Boowon sales division). This new lenses have a 52° angle of view, which at a distance of 20 cm approximatively increase the field view of about 57%, see figure 6.4.

6.3 External Trigger

To generate the trigger pulse we employed a PIC18F2580 micro-controller coupled whit a RS232 transceiver, they are mounted on printed circuit developed by Alessandro Crespi. We programmed the micro-controller to catch the incoming-data interrupt generated by the integrated UART[2]. Each time we send some data from the PC to the micro-controller trough the RS232 interface an interrupt is triggered. The interrupt is catch by the micro-controller which fires a trigger pulse to the cameras trough one of its I/O ports. The C code of the micro-controller is available on annex F. The printed circuit was not expressly developed for our purpose, it is overgrowth and need an external alimentation. A more comfortable USB device will be soon conceived by Alessandro, (thanks in advance!).

[2]Universal asynchronous receiver/transmitter

Chapter 7

Tests

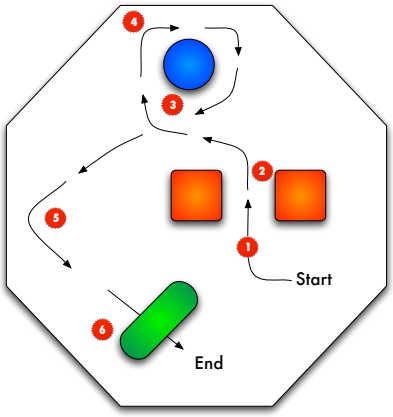


Figure 7.1: Test arena set up.

Aims of our test is to measure the practice usability of the whole stereo system and compare our correspondence algorithm with an existing one. Measuring the depth error produced by processing some known stereo images (stereo images where the depth is already known) is useless, since they only represent some particular cases and usually they are conceived to produce good results. What we want to measure is the global behavior when running the stereo algorithm in a real application case. Therefore in our lab (BIRG Lab) we have conceived a small arena where we could run tests. The arena is an area of 6 m^2 delimited by a 20 cm hight wall filled with textured images, (some forest pictures). Cables that connects the robot to the computer are supported by a rotating arms placed over the arena at 1m from the ground, see picture 7.2. The arena was constructed in collaboration with Matteo De Giacomi, to test our respective projects. Matteo want to test his obstacle avoidance state machine and we our depth map algorithm. We have placed some obstacles in the arena and measured if each obstruction (walls or obstacle) is correctly detected during a robot promenade. Figure 7.1 illustrate the test setup, there is 6 check points:

1. the vision system have to detect the opening between two obstacles.
2. the algorithm should detect the border of a close obstacle.
3. a small round object have to be detected.
4. the wall corner have to be correctly measured.

5. the slanted wall have to be correctly measured.

6. the algorithm have to detect an aperture under the bridge.

The test works as fallow, the stereo vision system is mounted on the Amphibot II robot, see figure 7.2. Each captured frames are treat twice, from our v.3 algorithm and from the SVS one. So both algorithm as the same identical input, and they outputs are then stored on a video. Each frame of the video shows the original image, the output of our algorithm and that of the SVS one. The Amphibot II was manually guided by Matteo De Giacomi, the robot path was formerly defined, see figure 7.1. We repeated the experience three times (obtained videos are available on http://www.elia.ch/svs), then we have examined the videos and for each check point of our test track we have identified a corresponding frame on the videos. We have than verified that the extracted frames from the three videos for the same check point was more or less corresponding to the same situations (distance form the obstacle, cameras orientation, etc.). Finally for each check point we had to choose the worst and the best result for the v.3 and for the SVS algorithm. Four the v.3 algorithm there were no parameters to set, the correspondence window size is fixed to 9 pixels, we have set the same size for the SVS algorithm and all other parameters was leaved to they default values. Result are exposed in figure 7.3. The depth information is given for a 5x3 grid, each grid cell contains a red rectangle. The rectangle size is proportional to the measured depth, if the rectangle has the size of the cell there is a very close object. If there is a red dot no depth information is available for that cell, undefined depth. Please note that the first depth column produced by the SVS algorithm is always undefined since it reduce the input image size. The two algorithms produce near results, thanks to our texture quality filter the v.3 algorithm dose not produce strange results when a portion of the image is not enough textured (point 3 image k and l) the undefined depth (red dot) is not considered a wrong estimation. Therefore the v.3 algorithm only in rarely occasion (point 2) produces erroneous depth estimation which is not the case for the SVS algorithm. On the other hand the SVS algorithm better handles situations with close objects (Point 6 and 2).
Computation performances in term of computed frames pro second are not comparable, since the SVS library employ MMX interactions to increase the frame rate. However we have disabled the MMX optimization on the SVS library to bring both algorithm on the same level and be able to compare their performances. It turns out that the SVS algorithm is slightly faster. We have executed both algorithms on the same machine [1] and treated the same set of 320x240 pixel images, we obtained a frame rate of 8 to 9 fps for the SVS implementation and 6 to 8 fps for the Stereoscopic Machine Vision Algo v.3

[1] A 3GHz personal computer

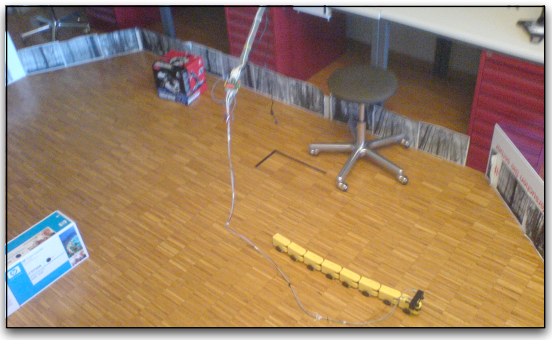
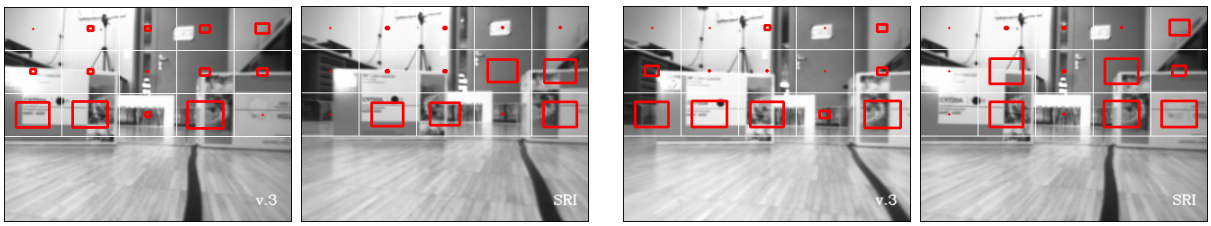
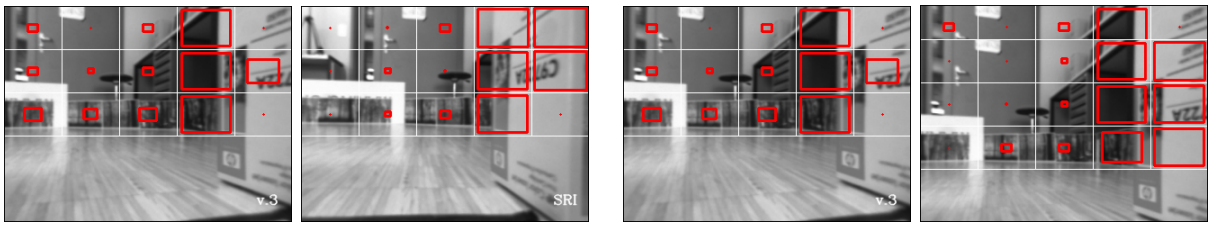


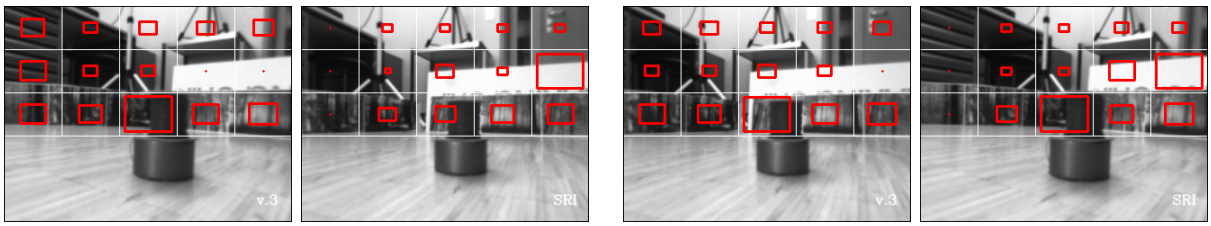
Figure 7.2: A picture of the test arena.



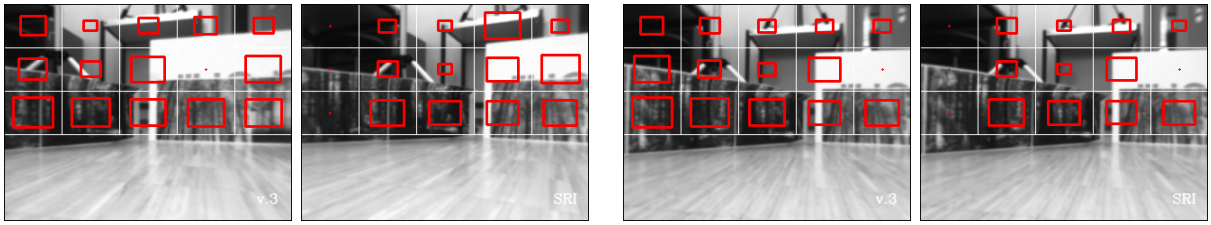
(a) Point 1: Algo v.3 worst (b) Point 1: SRI algo worst (c) Point 1: Algo v.3 best (d) Point 1: SRI algo best



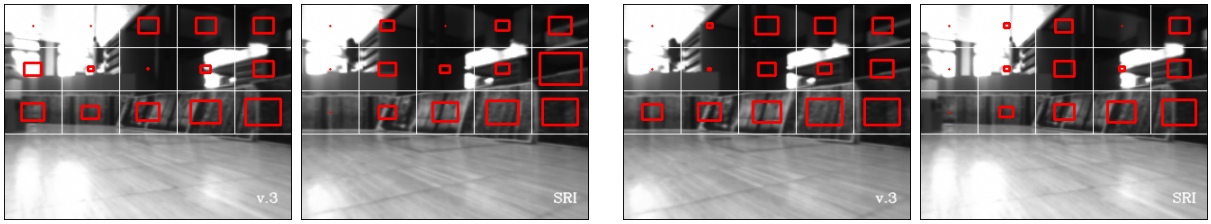
(e) Point 2: Algo v.3 worst (f) Point 2: SRI algo worst (g) Point 2: Algo v.3 best (h) Point 2: SRI algo best



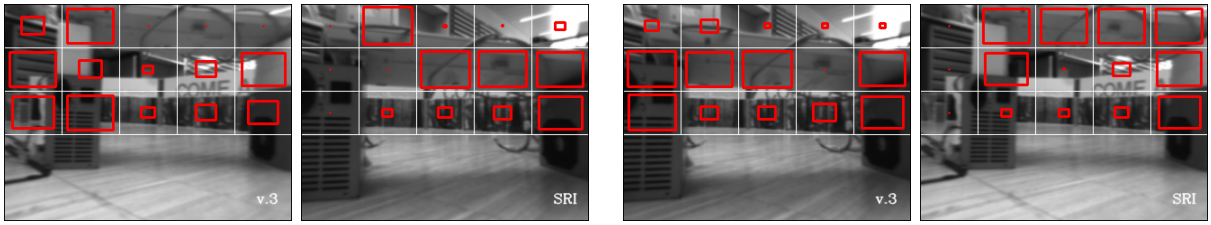
(i) Point 3: Algo v.3 worst (j) Point 3: SRI algo worst (k) Point 3: Algo v.3 best (l) Point 3: SRI algo best



(m) Point 4: Algo v.3 worst (n) Point 4: SRI algo worst (o) Point 4: Algo v.3 best (p) Point 4: SRI algo best



(q) Point 5: Algo v.3 worst (r) Point 5: SRI algo worst (s) Point 5: Algo v.3 best (t) Point 5: SRI algo best



(u) Point 6: Algo v.3 worst (v) Point 6: SRI algo worst (w) Point 6: Algo v.3 best (x) Point 6: SRI algo best

Figure 7.3: Comparison between the SVS algorithm and the Algo v.3.

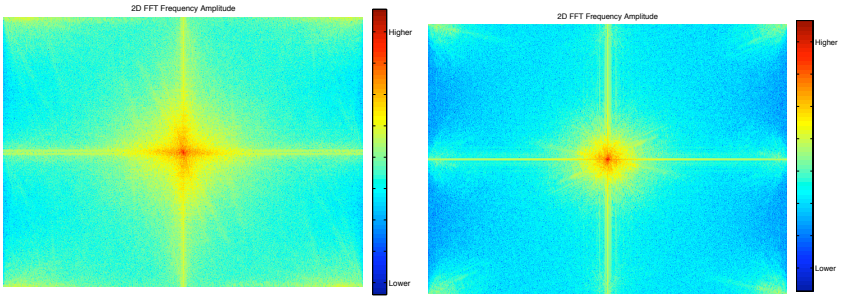
Chapter 8

Neural Networks

During our test we noted that some particular situations produce annoying results which are correct but could interfere with the navigation algorithm. For example if the ground is enough textured or has a repetitive pattern, the stereo algorithm will detect it and report close distance on the ground level. An other annoying situations is caused by an object that is so close to the cameras that no stereo information cold be deducted. To filter or detect this specific behavior we have trained an add-hoc neural network for each situation.

8.1 Too Close Object Detection

(a) A normal view (b) A view of a too close object



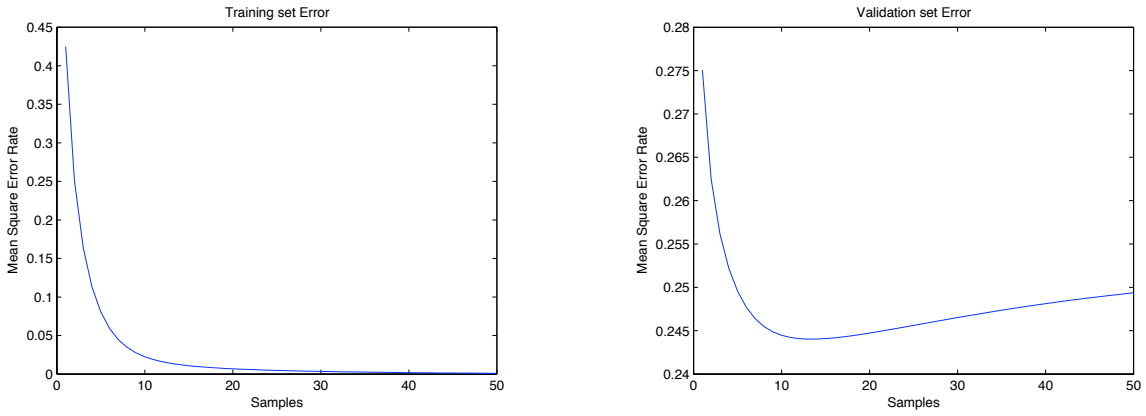
(c) The 2D FFT of the normal view (d) The 2D FFT of the close object

Figure 8.1: The difference of the frequency amplitude in a normal image compared with an image representing a close object.

Some times it hapens that an object is so close to the stereo vision system that only one

camera can see it. In such a situations the returned depth values are totally arbitrary, we want to detect such a behavior and inform the obstacle avoidance algorithm. A common characteristic of too close objects is that they will result blurred since they are out of focus. Also their textures will result more homogeneous since we are looking at a small detail of a close object. This two characteristics leads us to believe that an image representing a close object should have less important frequency than a normal one. To prouve or hipotesis we have operate a two dimensional FFT[1] on two image samples, result are exposed in image 8.1. The neuronal network should easily find a space separation for the two category using the FFT as image feature. An other advantage is that the FFT is computationally optimized and will not deteriorate the global stereo computation frame rate.
The neural architecture is composed by a single hidden layer and only allows forward connections. There is 8193 input neurons equivalent to the FFT frequency domain of 128x128 pixel image. As activation function we chose a hyperbolic tangent. The back-propagation method is used to train the neuronal network.

8.1.1 Training



(a) Training error

(b) Validation error

Figure 8.2: Training and Validation Mean Square Error Rate

The Neural networks implementations are based on the Torch library [27] which provides classes to model, and operate with the networks. To train our neural networks we have conceived an application which grab a video flow and ask the user to categorize each frame. Frames can be skipped or assigned to category A or B. For categorized frames the FFT is computed and stored on a buffer. Once the user chose to end the acquisition phase the buffers are empty and data are routed in to two large data-set files. A trainer application will later use this data-sets files to create and train a neural network (the trainer application was inspired by a source code provided us by Matteo De Giacomi)
We prepared a training and a test set, 100 samples for the training set, 50 sample of each category. The test data set was smaller and contains a total of 50 samples. We generated multiples networks varying the number of hidden neurons from a minimum of 25 neurons to 3500. The optimal solution for our data set is located around 100, figure 8.2 illustrate its training and validation function. The method to establish the most adapted number of hidden

[1] Fast Fourier Transform

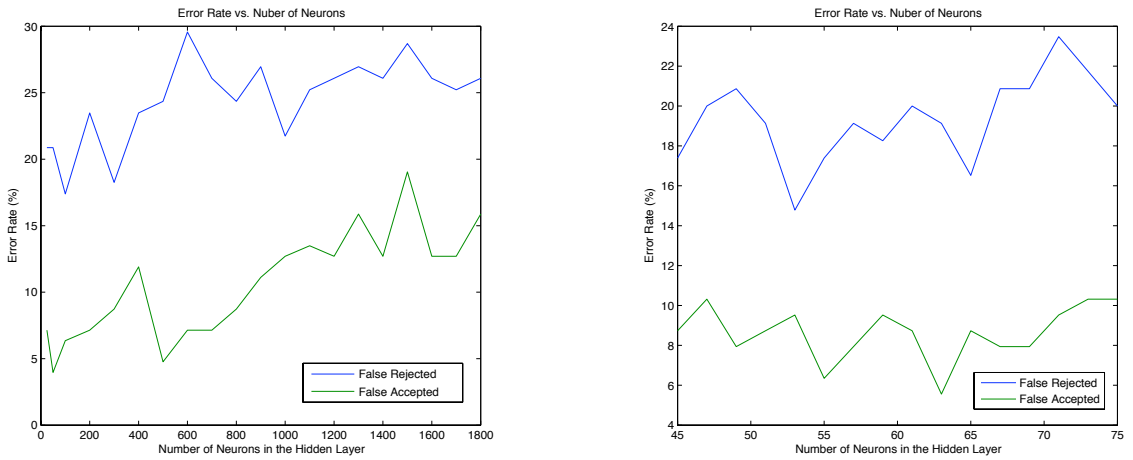
neurons is the same used for the floor detection and is explained in section 8.2.

8.2 Floor Detection

To helping the navigation algorithm we want to filter any depth information which is relative to the ground. We proceed by segmenting the image in smaller portions, we extracted their features and we ask the neural network to identify them. Like the close objects detection we chose the FFT as image feature since floors usually have a repetitive patterns which produce high amplitudes at some specific frequencies. We have reused the same frame categorizer and neural networks trainer we used for the too close object detection.

8.2.1 Training

Since the robot is moving on the floor, its view angle in relation with the floor varies, therefore also the floor pattern can change. To train our floor detection neural network we take an huge number of samples from different point of view. 500 samples for the training data set and 250 for the test data set. The neural network architecture is practically the same as we used for the close objects detection. We changed the number of input neurons which are 2049, since we want to detect the ground inside smaller windows. The window size is 64x64 pixel. Also the number of the hidden neurons changes. To establish the most adapted number of hidden neurons for our data set we tried multiple configurations. The first measure was made at a big granularity, we chosen than a smaller range where we made a more precise measure. Figure 8.3 illustrates



(a) Small granularity

(b) Large granularity

Figure 8.3: False Acceptance and Rejection rate depending of the number of hidden neurons

the false acceptance and false rejection error rate respect to the number of hidden neurons. We chose to use the network with 55 hidden neurons, since it minimize the false acceptance and the false rejection is still acceptable. Choosing 55 is like choosing the less noisy point, on the other hand a value has to be chosen, we already know that better results are given with a small number of neurons (graph 8.3(a)). Therefore we pick 55 since it was the best choice related to our data set. As further work we suggest to do multiple measures with multiple data sets, so that for both graphs (8.3(a) and 8.3(b)) the represented values would be the average of those

measures and as consequence the local noise (error rate for a specific number of neurons) will be reduced.

8.2.2 Application

We used the obtained neuronal networks to pre-filter the images before computing the depth map. Unfortunately on a real application by testing the filters with the robot we discovered that obtained results are less reliable as expected. This is probably caused by some unexpected interferences, like the motion blur or other hazards. We could not further test the networks and train it with a larger data set to improve their performance.

Chapter 9

Conclusion

We have proposed a new robust correspondence algorithm especially conceived for vision guided applications. We have further analyzed its output and identified two inconvenient situations, which we want to avoid, in order to simplify the obstacle avoidance algorithm tasks. To overcome this annoying behavior we propose a solution based on a neural network.
The new correspondence algorithm has a preprocessing phase which, thanks to a combination of two Difference-of-Gaussian functions, allows to filter un-textured image regions. We have shown how the filter increases the quality of the disparity map, by filtering that correspondence windows, which will probably yield to a miss match.
We shown how an one-dimensional vertical smoothing can help to minimize the effects of a bad image rectification. We found an alternative validity test to identify bad matches and we demonstrated that it is less selective than the symmetric consistency check. A less selective validity test is more adapted when handling close objects.
In order to provide good quality images to our stereo algorithm we created the Synchronized Frames Grabber, a driver, which allows to grab from a binocular firewire cameras system two perfectly synchronized frames. We shown how a standalone external trigger is not enough to guarantee the synchronization and we proposed a reliable solution.
We built together with Matteo De Giacomi an arena where we were able to test our solutions. We have compared our algorithm with the well know, and affirmed Small Vision System API developed by the SRI research institute and actually propriety of the Videre Design company. We are proud of our results since the project produced a stereo vision system, which has nothing to envy to his elder brother (the SVS of Videre Design).
Finally we have proposed a solution to detect strange situations such as too close objects, which are not detectable by the stereo engine. Also the floor can be detected and omitted from the depth map, since it could cause confusion. Both filters are based on neural networks, we showed that the FFT[1] is good image feature to distinguish this cases. As further work we propose to provide a bigger training data set, and make more massive tests to verify its usability for real applications.

[1] Fast Fourier Transform

Bibliography

[1] A. Crespi and A.J. Ijspeert. AmphiBot II: An amphibious snake robot that crawls and swims using a central pattern generator. In *Proceedings of the 9th International Conference on Climbing and Walking Robots (CLAWAR 2006)*, pages 19–27, 2006.

[2] A.J. Ijspeert and A. Crespi. Online trajectory generation in an amphibious snake robot using a lamprey-like central pattern generator model. In *Proceedings of the 2007 IEEE International Conference on Robotics and Automation (ICRA 2007)*, pages 262–268, 2007.

[3] A. Ijspeert, A. Crespi, D. Ryczko, and J.M. Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416–1420, 2007.

[4] O. Wulf and B. A. Wagner. Fast 3d scanning methods for laser measurement systems. July 2003.

[5] Matteo De Giacomi. Robotics Applications of Vision-Based Action Selection. Master's thesis, EPFL, August 2007. Very Frendly.

[6] V. Venkateswar and R. Chellappa. Hierarchical stereo and motion correspondence using feature groupings. *Int. J. Comput. Vision*, 15(3):245–269, 1995.

[7] Cordelia Schmid and Andrew Zisserman. The geometry and matching of curves in multiple views. In *ECCV (1)*, pages 394–409, 1998.

[8] U.R. Dhond and J.K. Aggarwal. Structure from stereo-a review. *Systems, Man and Cybernetics, IEEE*, 19:Page(s): 1489–1510, 1989.

[9] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[10] Myron Z. Brown, Darius Burschka, and Gregory D. Hager. Advances in computational stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(8):993–1008, 2003.

[11] Andrew Dankers, Nick Barne, and Alexander Zelinsky. Active vision – rectification and depth mapping. in Australian Conference on Robotics and Automation, 2004.

[12] C. Loop and Z. Zhang. Computing rectifying homographies for stereo vision. In *Technical Report MSR-TR-99-21*, 1999.

[13] Jean-Yves Bouguet. Camera calibration toolbox for matlab (http://www.vision.caltech.edu/bouguetj/calib_doc/), 2007.

[14] Zhengyou Zhang. Determining the epipolar geometry and its uncertainty: A review. *Int. J. Comput. Vision*, 27(2):161–195, 1998.

[15] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(11):1330–1334, 2000.

[16] Changming Sun. Fast stereo matching using rectangular subregioning and 3d maximum-surface techniques. *Int. J. Comput. Vision*, 47(1-3):99–117, 2002.

[17] Olivier Faugeras, B. Hotz, Hervé Mathieu, T. Viéville, Zhengyou Zhang, Pascal Fua, Eric Théron, Laurent Moll, Gérard Berry, Jean Vuillemin, Patrice Bertin, and Catherine Proy. Real time correlation based stereo: algorithm implementations and applications. Technical Report RR-2013, INRIA, 1993.

[18] D. W. Rosselot and E. L. Hall. Processing real-time stereo video for an autonomous robot using disparity maps and sensor fusion. In D. P. Casasent, E. L. Hall, and J. Roning, editors, *Intelligent Robots and Computer Vision XXII: Algorithms, Techniques, and Active Vision. Edited by Casasent, David P.; Hall, Ernest L.; Roning, Juha. Proceedings of the SPIE, Volume 5608, pp. 70-78 (2004).*, pages 70–78, October 2004.

[19] Heiko Hirschmüller, Peter R. Innocent, and Jon Garibaldi. Real-time correlation-based stereo vision with reduced border errors. *Int. J. Comput. Vision*, 47(1-3):229–246, 2002.

[20] Luigi Di Stefano and Stefano Mattoccia. Real-time stereo within the videt project. *Real-Time Imaging*, 8(5):439–453, 2002.

[21] Heiko Hirschmüller. Improvements in real-time correlation-based stereo vision, 2001.

[22] H.K. Nishihara and T. Poggio. Stereo vision for robotics. *First Int'l Symp. Robotics Research*, pages 489–505, 1984.

[23] Pascal Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. Springer Berlin / Heidelberg, 1993.

[24] Videre Design. Small vision system by videre design (http://www.videredesign.com/vision

[25] Pascal Fua. Combining stereo and monocular information to compute dense depth maps that preserve depth discontinuities. pages 1292–1298, 1991.

[26] David G. Lowe. Distinctive image features from scale-invariant keypoints. 60(2):91–110, November 2004.

[27] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torche (http://www.torch.ch/).

Appendix A

Dynamic Programming Case Study

Table A.1: Normal Situations

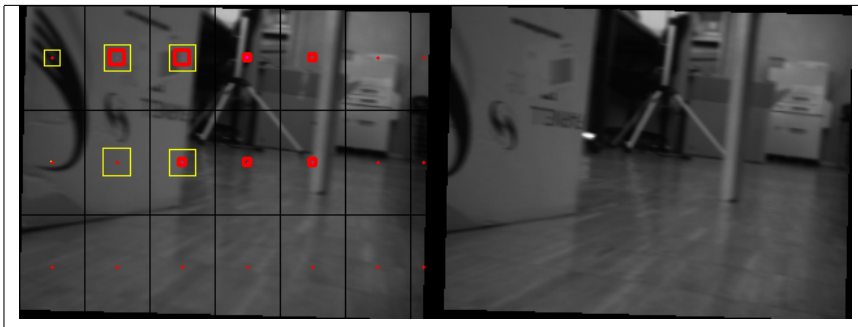
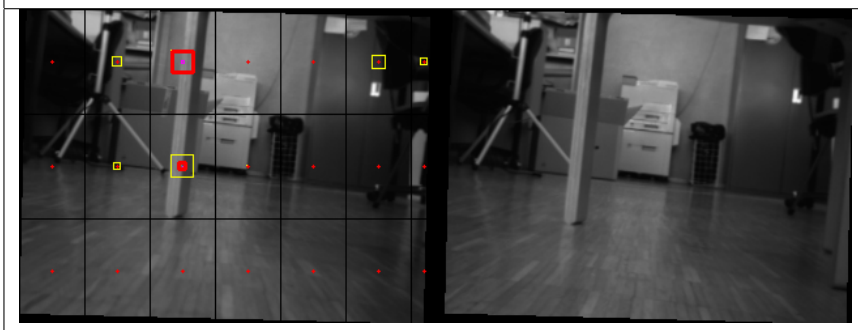
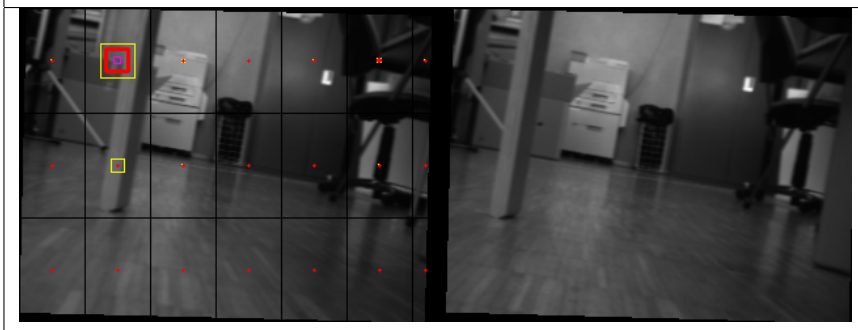
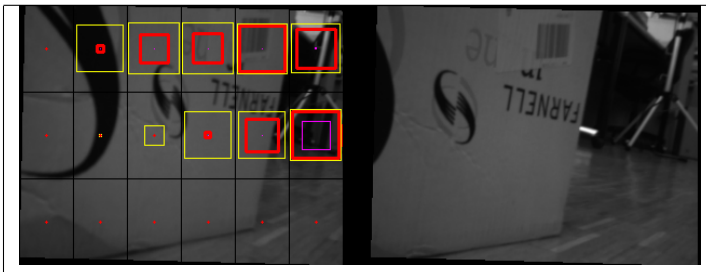
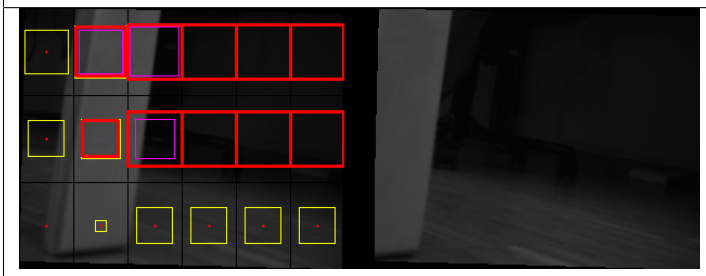
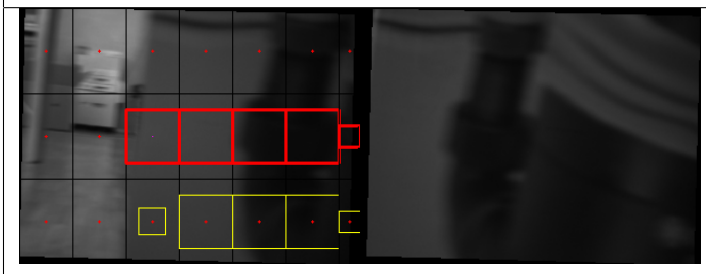
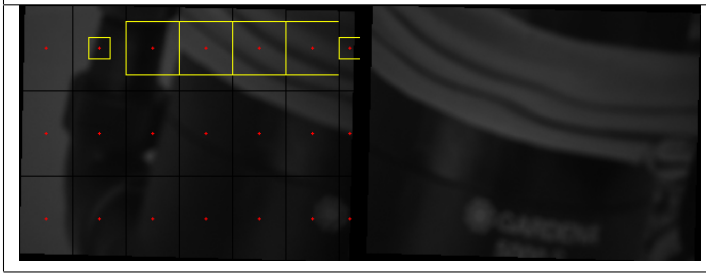
	Single object
	Infinite distance
	Tilt

Table A.2: Special Situations

	Small escape
	Dark
	Too far
	Too far 2

Appendix B

Distance at Disparity measure

Table B.1: Distance for Disparity measure

cm	disp
130	11
120	12
110	13
100	14
90	16
80	17
70	19
60	22
50	27
40	32
30	38
20	60
16	81

```
0001 % ========================================================
0002 % This matlab function compute the distance in cm relative
0003 % to the disparity. It take some partial measured data
0004 % and it extrapolate missing data to have a measure (cm) for
0005 % each disparity value.
0006 % x is the partial measure in cm
0007 % y is the corresponding disparity
0008 % Maxdisparity is the maximum disparity of the algorithm
0009 % ========================================================
0010 function [s] =  DepthEstimator(x,y,Maxdisparity)
0011
0012     file = fopen('Depth.h','w');
0013
0014 fprintf(file,'//============================================================/*\n');
0015 fprintf(file,'This h file was generated using the depthEstimator.m matLab file.\n');
0016 fprintf(file,'It contains the depth value for dispyrity.\n');
0017 fprintf(file,'For questions ask Elia Palme, info@elia.ch or elia.palme@unifr.ch\n');
0018 fprintf(file,'//============================================================*/\n');
0019
0020     p = polyfit(x,y,5);
0021
0022     s = zeros(Maxdisparity,1);
0023     r = zeros(Maxdisparity,1);
0024     count=0;
0025     for t= 1:1:Maxdisparity,
0026     count=count+1;
0027     s(count)=polyval(p,t);
0028     r(count)=t;
0029     end;
0030
0031     fprintf(file,'\t extern int depthMapping[%d]={0,',length(s)+1);
0032
0033     for j=1:length(s),
0034              %if(abs(g(j)) > filter)
0035                 fprintf(file,'%d, ',round(s(j)));
0036                 count = count+1;
0037              %end
0038      end
0039     fseek(file, -2, 'cof'); %to erease the last coma (,)
0040     fprintf(file,'};\n',count);
0041
0042
0043
0044     plot(x,y,'--mo','color','red');
0045     hold on
0046     plot(r,s);
0047     hold off
0048      xlabel('Disparity');
0049     ylabel('cm');
```

```
0050      title('Distance at Disparity');
0051      legend('Measured data','Extrapolate data');
0052
0053      fclose(file);
```

Appendix C

MatLab code to test the difference-of-Gausian filter

```
0001 %=========================================================================
0002 %ex values: computeWindowsTest('test.png',96,0.4,3.35,24)
0003 %=========================================================================
0004 function [f] = computeWindowsTest(filename,l,initSigma1,scale,kernleSize)
0005 input = extractLine(filename,l);
0006 line = cast(input, 'single');
0007 %trashold = 0.0004;
0008 [h,w,d]=size(line);
0009 X = 0:1:w-1;
0010 numScale=2;
0011 r = zeros(numScale,w);
0012 p = zeros(1,w);
0013 x = kernleSize/2*-1:1:kernleSize/2-1+1;
0014
0015
0016 %=========================================================================
0017 %Using the DOG operator (Difference of gausian) convolute tha image row
0018
0019     g1 = DOG (x,initSigma1,initSigma1+scale);
0020     g2 = DOG (x,initSigma1+scale,initSigma1+2*scale);
0021
0022
0023    %Apply the convolution using the computed DOG function
0024    for m=kernleSize/2:w-(kernleSize/2)-1+1,
0025        filter = zeros(1,kernleSize);
0026
0027        for d=-(kernleSize/2):(kernleSize/2)-1,
0028           % if(abs(g(j)) > trashold)
0029              filter(d+(kernleSize/2)+1) =(line(m+d+1) * g1(d+(kernleSize/2)+1));
0030            %end
0031        end
0032        r(1,m)=abs(sum(filter));
0033    end
0034
0035    for m=kernleSize/2:w-(kernleSize/2)-1+1,
0036        filter = zeros(1,kernleSize);
0037
0038        for d=-(kernleSize/2):(kernleSize/2)-1,
0039           % if(abs(g(j)) > trashold)
0040              filter(d+(kernleSize/2)+1) =(line(m+d+1) * g2(d+(kernleSize/2)+1));
0041            %end
0042        end
0043        r(2,m)=abs(sum(filter));
0044    end
0045
0046
0047 %=========================================================================
0048 %The windows scale correspond to the scale which hase the maxim difference
0049 for m=kernleSize/2:w-(kernleSize/2)-1,
```

```
0050
0051     max=-1;
0052     maxIndex=1;
0053     for j=1:numScale,
0054
0055          if(r(j,m) > max)
0056              max= r(j,m);
0057              maxIndex=j;
0058          end
0059     end
0060     p(m)= maxIndex;
0061 end
0062
0063
0064 figure('Name', 'Result');
0065 grapf = subplot(2,1,1);plot(X,line);
0066 axis(grapf,[0  w  0 260]);
0067 xlabel('x (coordonate)');
0068 ylabel('y (pixel intensity)');
0069 title('One pixels raw of an image in gray scale');
0070 %legend('pixel intensity');
0071 grapf = subplot(2,1,2);plot(X,p);
0072 axis(grapf,[0  w  0 numScale + (0.5*numScale)]);
0073 xlabel('x (coordonate)');
0074 ylabel('scale');
0075 title('Difference of Gaussian scale having the highest value');
0076 legend(['Initial sigma: ',
            num2str(initSigma1),
            ', scale factor: ',
            num2str(scale),
            ', kernel size ',
            num2str(kernleSize),
            'px']);
0077
0078 %=========================================================================
0079 %DOG operator compute the difference of gausian function
0080 function [g] = DOG (x,o1,o2)
0081 y = 1/sqrt(2*pi) * ( 1 / o1 * exp(-(x.^2)/(2*o1^2)) - 1 / o2 *
         exp(-(x.^2)/(2*o2^2)));
0082 g = cast(y, 'single');
0083
0084 function [p] = extractLine(filename,l)
0085 I = imread(filename);
0086 [h,w,d]=size(I);
0087 p = I(l:l,1:w);
0088
0089 function [g] = gaussian(x,o)
0090 g = (1/sqrt(2*pi*o^2) ) * exp ( - x.^2 / (2 * o^2));
```

Appendix D

MatLab code to generate the .h file containing the difference-of-Gaussian filter constants

```
0001 %=========================================================================
0002 %Find the size of the correlation window for each pixel postion of an image
0003 %row. Depending on the pixels intensity omogeneity arround that pixel
0004 %position.
0005 %PARAMETERS:
0006 %filename : the png file containing the image
0007 %l: the image row to analize.
0008 %scaleFactor: how mach increase the scale at ecah iteration (usualy 1)
0009 %numScale: how many scales
0010 %ex values: computeWindows(0.6,1.55,1,2,12)
0011 %better values: computeWindows(2,1.4,1.2,2,24),
0012 %computeWindows(1.7,1.8,1.2,2,24)
0013 %=========================================================================
0014 function [f] = computeScaleDOG(initSigma1, scale ,kernleSize)
0015
0016 x = kernleSize/2*-1:1:kernleSize/2-1+1;
0017 x2 = kernleSize/2*-1:0.1:kernleSize/2-1+1;
0018
0019 file = fopen('DOGfilter.h','w');
0020
0021 fprintf(file,'//============================================================/*\n');
0022 fprintf(file,'This h file was generated using the computeWindows.m matLab file.\n');
0023 fprintf(file,'It contains multiple DOG function in different scales.\n');
0024 fprintf(file,'For questions ask Elia Palme, info@elia.ch or elia.palme@unifr.ch\n');
0025 fprintf(file,'//============================================================*/\n');
0026
0027 fprintf(file,'struct dogFilter{\n');
0028 fprintf(file,'\tfloat values[%d];\n',kernleSize+1);
0029 fprintf(file,'\tint kernelSize;\n');
0030 fprintf(file,'};\n');
0031
0032 fprintf(file,'struct dogScales{\n');
0033 fprintf(file,'\tint numScales;\n');
0034 fprintf(file,'\tint maxKernelSize;\n');
0035 fprintf(file,'\tdogFilter  scales[%d];\n',2);
0036 fprintf(file,'};\n');
0037
0038
0039 %=========================================================================
0040 %Using the DOG operator (Difference of gausian) convolute tha image row
0041 figure('Name', 'DOG');
0042
0043     g1 = DOG (x,initSigma1,initSigma1+scale);
0044     g2 = DOG (x,initSigma1+scale,initSigma1+2*scale);
0045
0046     g1t = DOG (x2,initSigma1,initSigma1+scale);
0047     g2t = DOG (x2,initSigma1+scale,initSigma1+2*scale);
0048
0049     tempg1 = gaussian(x2,initSigma1);
```

```
0050     tempg2 = gaussian(x2,initSigma1+scale);
0051     tempg3 = gaussian(x2,initSigma1+2*scale);
0052
0053     printToFile(g1,1,file);
0054     printToFile(g2,2,file);
0055
0056     plot(x2, tempg1, '-', x2, tempg2, '--',x2,tempg3,'-.',
             'LineWidth',1,'MarkerSize',2),axis([-8 8 0 .3]), grid on;
0057     legend(['\sigma = ',num2str(initSigma1)],
                ['\sigma = ',num2str(initSigma1+scale)],
                ['\sigma = ',num2str(initSigma1+2*scale)])
0058     xlabel('x'), ylabel('Density Probabilities'),
0059     title('Gaussians')
0060
0061
0062     figure(2);
0063     plot(x2, g1t,'-',x2,g2t,'--'),axis([-8 8 -0.15 .20]), grid on;
0064     xlabel('x'), ylabel('y'),
0065     title('Difference of Gaussian');
0066
0067
0068 f=0;
0069
0070 fprintf(file,'dogScales dog =  { %d,%d,{',2,kernleSize+1);
0071
0072
0073        fprintf(file,'f%d, ',1);
0074        fprintf(file,'f%d, ',2);
0075
0076 fseek(file, -2, 'cof'); %to erease the last coma (,)
0077 fprintf(file,'} };\n');
0078
0079
0080 fclose(file);
0081
0082
0083
0084 %=========================================================================
0085 %save the DOG function
0086 function [x] = printToFile(g,s,file)
0087
0088 count =0;
0089 fprintf(file,'dogFilter f%d={{',s);
0090
0091 for j=1:size(g,2),
0092          %if(abs(g(j)) > filter)
0093             fprintf(file,'%ff, ',g(j));
0094             count = count+1;
0095          %end
```

```
0096  end
0097 fseek(file, -2, 'cof'); %to erease the last coma (,)
0098 fprintf(file,'},%d};\n',count);
0099
0100
0101 %=========================================================================
0102 %DOG operator compute the difference of gausian function
0103 function [g] = DOG (x,o1,o2)
0104 y = 1/sqrt(2*pi) * ( 1 / o1 * exp(-(x.^2)/(2*o1^2)) - 1 / o2 *
         exp(-(x.^2)/(2*o2^2)));
0105 g = cast(y, 'single');
0106
0107 function [p] = extractLine(filename,l)
0108 I = imread(filename);
0109 [h,w,d]=size(I);
0110 p = I(l:l,1:w);
0111
0112 function [g] = gaussian(x,o)
0113 g = (1/sqrt(2*pi*o^2) ) * exp ( - x.^2 / (2 * o^2));
```

Appendix E

XML calibration file example

```
<?xml version="1.0" encoding="utf-8"?>
<VSCalibrationData>
<Camera>
<Distortion dist0="-0.2031358778" dist1="0.3360900879"
            dist2="0.0026119531" dist3="-0.0034170868"/>
<ImageSize x="240.0000000000" y="320.0000000000"/>
<Intrinsic intr0="566.0656127930" intr1="0.0000000000" intr2="100.7078247070"
           intr3="0.0000000000" intr4="566.6765747070" intr5="170.1375122070"
           intr6="0.0000000000" intr7="0.0000000000" intr8="1.0000000000"/>
<Rotation rot0="0.0123223495" rot1="0.9976426363" rot2="-0.0675079376"
          rot3="0.9882473350" rot4="-0.0018631991" rot5="0.1528519839"
          rot6="0.1523658782" rot7="-0.0685980394" rot8="-0.9859406352"/>
<Transaction x="-2.9863090515" y="-5.3522324562" z="20.5555553436"/>
</Camera>
<Camera>
<Distortion dist0="-0.2383495867" dist1="0.5888255835"
            dist2="0.0002490598" dist3="-0.0024096163"/>
<ImageSize x="240.0000000000" y="320.0000000000"/>
<Intrinsic intr0="555.5454101563" intr1="0.0000000000" intr2="116.7631225586"
          intr3="0.0000000000" intr4="556.0297241211" intr5="152.7918090820"
          intr6="0.0000000000" intr7="0.0000000000" intr8="1.0000000000"/>
<Rotation rot0="-0.9984018803" rot1="0.0098599484" rot2="-0.0556461513"
          rot3="0.0011273620" rot4="0.9879410863" rot5="0.1548262835"
          rot6="0.0565016977" rot7="0.1545161158" rot8="-0.9863733053"/>
<Transaction x="3.1538500786" y="-5.2221846581" z="20.1681709290"/>
</Camera>
<Stereo>
<Quad x(0,0)="-13.1325979233" y(0,0)="11.2207002640" x(0,1)="244.5350952148"
    y(0,1)="10.5299797058" x(0,2)="239.9021606445" y(0,2)="326.6764221191"
    x(0,3)="-17.2835369110" y(0,3)="318.4469909668" x(1,0)="0.0034242452"
    y(1,0)="-0.2178648710" x(1,1)="252.3901824951" y(1,1)="0.0122055206"
    x(1,2)="247.9728851318" y(1,2)="309.5118713379" x(1,3)="-4.7434144020"
    y(1,3)="301.7959594727"/>
<Coeffs coeff(0,0,0)="1.0448307652" coeff(0,0,1)="-0.0130454206"
    coeff(0,0,2)="-13.1325979233" coeff(0,1,0)="-0.0041175033"
    coeff(0,1,1)="0.9614407391" coeff(0,1,2)="11.2207002639"
```

```
    coeff(0,2,0)="-0.0001177116" coeff(0,2,1)="0.0000042663"
    coeff(0,2,2)="1.0000000000" coeff(1,0,0)="1.0262211635"
    coeff(1,0,1)="-0.0148082858" coeff(1,0,2)="0.0034242452"
    coeff(1,1,0)="0.0009573988" coeff(1,1,1)="0.9421653755"
    coeff(1,1,2)="-0.2178648710" coeff(1,2,0)="-0.0001005995"
    coeff(1,2,1)="-0.0000053938" coeff(1,2,2)="1.0000000000"/>
</Stereo>
</VSCalibrationData>
```

Appendix F

Trigger micro controller C code

```
#include "triggerController.h"

volatile int1 b;
#int_RDA

//============================
// Interruption routine
//============================
void  RDA_isr(void) {
char c;
c = getc(); //empty the UART buffer
//printf("\x1b[1m%d\x1b[m  ", c); //DEBUG
b = !b;
}

void main()
{

   setup_adc_ports(NO_ANALOGS|VSS_VDD);
   setup_adc(ADC_OFF|ADC_TAD_MUL_0);
   setup_spi(SPI_SS_DISABLED);
   setup_wdt(WDT_OFF);
   setup_timer_0(RTCC_INTERNAL);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
   setup_vref(FALSE);
   enable_interrupts(INT_RDA);
   enable_interrupts(GLOBAL);


   output_high(PIN_B0); //Cameras work with inverted 1/0

//============================
// Main loop
```

```
//============================

while (1) {

  if (b) {
    output_high(PIN_C2);  //Basler cameras
    output_high(PIN_A0);  //led
    output_low(PIN_B0);   //Point Grey cameras
    delay_ms(20);
    output_low(PIN_C2);
    output_low(PIN_A0);
    output_high(PIN_B0);
    b = !b;
   }
}
}
```

Appendix G

BibTeX reference

```
@mastersthesis{EPalmeTechRep,
Author = {Elia Palme},
Date-Added = {2007-07-18 11:54:42 +0200},
Date-Modified = {2007-08-18 12:33:18 +0200},
Institution = {UNIFR and EPFL},
Keywords = {Stereo Vision System, Correspondence Algorithm, Robotics,
Obstacle Avoidance Applications, Real Time, Trigger, Synchronized Frames Grabber,
SSD Correlation, Difference of Gaussians, Neural Network},
Month = {8},
School = {University of Fribourg and EPFL, Switzerland},
Title = {Stero Vison Library for Obstacle Advoidance Applications},
Type = {Master Project Tecnical Report},
Url = {http://www.elia.ch},
Year = {2008}}
```