Master thesis : Efficient and precise stereoscopic vision for humanoid robots

Auteur : Ewbank, Tom

Promoteur(s) : Boigelot, Bernard

Faculté : Faculté des Sciences appliquées

Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"

Année académique : 2016-2017

URI/URL : http://hdl.handle.net/2268.2/3144

UNIVERSITY OF LIÈGE
Faculty of Applied Sciences

Efficient and precise stereoscopic vision for humanoid robots

Final year thesis submitted for the
Master's degree in Computer Science Engineering

by Tom EWBANK

Supervised by
Prof. Bernard Boigelot

Academic year 2016-2017

Abstract

This thesis is realized in the context of the RoboCup contest: a competition where two teams of robots play against each other in a soccer game. The purpose of this work is to determine if a stereo vision system could be implemented on a constrained robot platform, and provide, in real-time, useful 3D information about the playing area and the game elements.

This paper starts by giving a theoretical explanation of the principles of stereo vision systems, followed by a quick review of the state of the art.

As the computation power of the considered robot platform is limited, this paper then proposes an adaptation of the algorithm developed by Sudeep Pillai in [1], claiming to achieve a good semi-dense approximation of the 3D environment at a frame rate that can reach 120Hz on a single CPU thread.

The testing of this algorithm with a stereo setup of two wide-lens cameras separated by a small distance shows that the depth of a soccer ball can be estimated with a mean absolute error of 5cm/m, by directly looking at the depth of generated 3D points supposed to belong to the ball. Another analysis also reveals that the inclination at which the floor is observed by the cameras can be estimated with a precision of less than 1°. It is thus likely that the accuracy of the ball localization could be further improved taking advantage of this precise floor plane estimation instead of assuming that the 3D surface of a ball would always be correctly rendered by the algorithm.

Acknowledgements

First of all, I would like to thank particularly my supervisor, Prof. Bernard Boigelot, who gave me the opportunity to work on a subject in my favorite field of study, robotics, but most importantly, who always took the time to answer my questions, and to guide me with precious advice throughout this master thesis.

I am also grateful to Prof. Marc Van Droogenbroeck, who helped me with some specific questions about computer vision and camera calibration.

Likewise, I can't fail to thank Lionel Van Laeken and Valentin Rapallo, from the team responsible of the structure of the robot, who designed and 3D-printed some useful parts for the testing of my system, who lent me some tools, and who even gave me a hand in the lab when I needed.

Finally, I would like to thank my family and friends who gave me all the support that I needed to complete this final project of my cursus inside the faculty of applied sciences of the ULg.

Contents

1. Introduction and problem statement

This master thesis was realized in the context of the development of a humanoid robot team for participating to the *RoboCup Soccer Contest.* This contest comprises different leagues in which the robots can play, each league having its own specificities and its own rules. The league targeted in this work is the *Humanoid/Kid Size* league. In this league, autonomous robots of a height ranging from 40 to 90cm, with a human-like body plan and human-like senses, play soccer against each other. The particularity of this category is that the task of perception and world modeling can not be simplified by using non-human like range sensors, and this is where the work performed in this thesis intervenes.

Indeed, the goal of this thesis is to provide the robot with the capability of depth perception. This can be accomplished using the concept of stereovision, based on images provided by two cameras side by side, acting like the human eyes. The depth perception gives thus the robot a 3D representation of the scene he observes, that should allow him for example to avoid obstacles, locate the soccer ball, and estimate the inclination of the floor in order to maintain his balance.

Stereovision is a well studied problem and there exists solutions that can give very accurate results. However, these solutions can require a lot of computation power, or specialized hardware, in order to be executed at a fast rate. An underlying purpose of this thesis is thus to build a stereovision software able to run in real time on a constrained robot plateform, without excessive latency caused by the computations, while still delivering a sufficient accuracy. The plateform considered here is an embedded motherboard, equipped with a medium speed quad core x86 processor, from which one cpu core can be dedicated to the stereovision software.

Another constraint of the problem is the fact that the robot will most of the time be moving while taking pictures with his cameras. A short exposure time taking the pictures is thus required in order to avoid getting blurred images, and the issue is thus that the pictures obtained will tend to be noisy. The stereovision system developed in this thesis should thus also be robust to noise.

To present the work accomplished for this thesis, this paper will start by explaining all the theory behind stereoscopic vision systems. Once this base is installed, it will briefly present the state of the art, the choices that were made accordingly, but most importantly, it will describe the principles of the stereovision software that was implemented for this project. Finally, after some notes about the implementation of the system, the different tests and results obtained with the developed solution will be presented.

2. Theory of stereoscopic vision systems

This chapter explains the theory of stereoscopic vision systems, starting with the mathematic model of a single camera, then expanding it to the concept of a stereo cameras system and the way it can infer the depth from multiple pictures of the same scene.

Several steps are needed before obtaining the depth information. These steps are first described separately, and lastly they are put together to form the general caneva of a stereovision algorithm.

2.1 Camera model

2.1.1 Intrinsic parameters

The simplest and most commonly used camera model is called the pinhole camera model. This model gives the mathematical relationship between 3D points and their projection, through a pinhole, into a 2D image plane, as illustrated in Figure 2.1. In this model, all the light rays converge to a unique point, the camera optical center, before arriving on the image plane, also called the sensor plane in an actual camera.

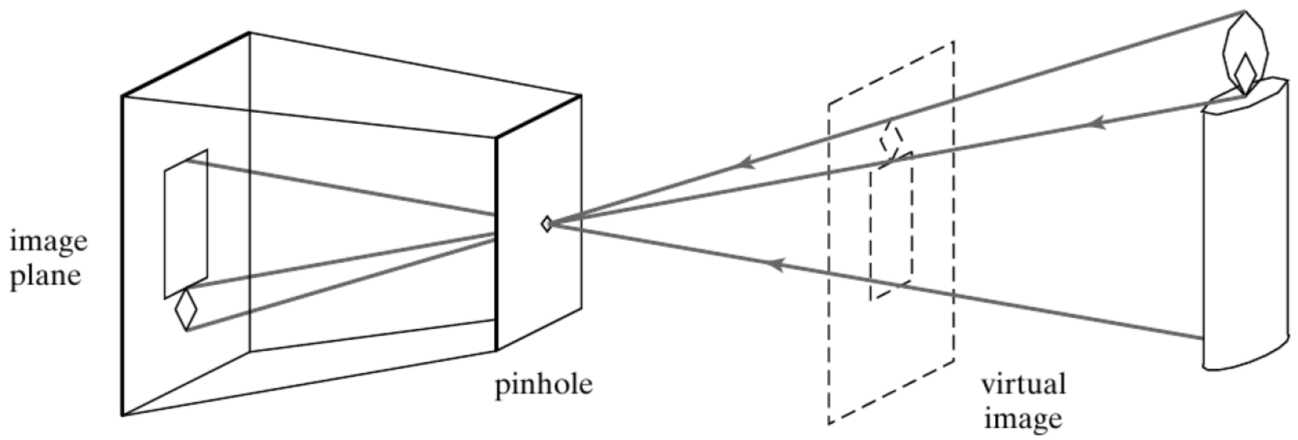


Figure 2.1 – Pinhole camera example (source: [2])

To derive the mathematical camera model, consider the schema in Figure 2.2. The non inverted image plane is represented in yellow. Considering a 3D point $P = (X, Y, Z)$, and using the similar triangles technique, the projection of P on the image plane is the point $p = (x, y)$, where x and y are given by

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z} \tag{2.1}$$

It is important to note that the inverse of this mapping is not unique. Indeed, a particular 3D point has a unique correspondence in the image plane, but all the 3D points belonging the the red line in Figure 2.2 are potential correspondences for a particular pixel (x, y) of the image plane. Without additional information, it is thus not possible to retrieve depth information from a single picture taken by a camera.

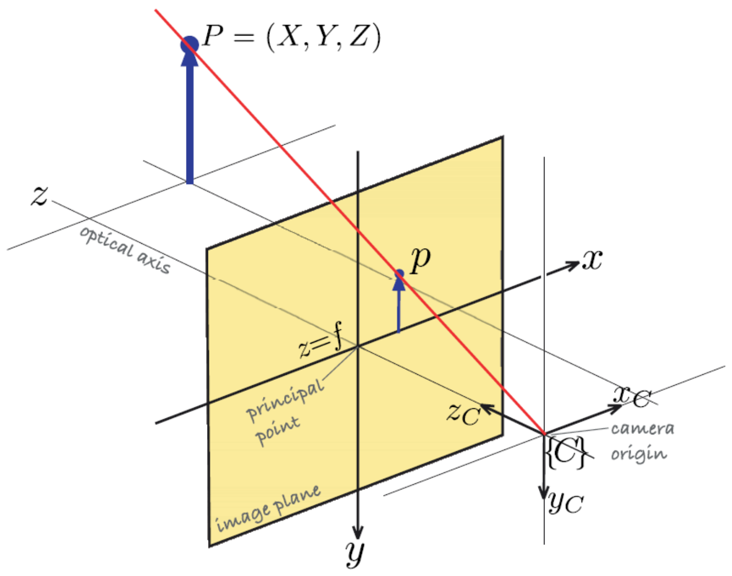


Figure 2.2 – The central-projection model. The image plane is at a distance f in front of the camera's origin and a non-inverted image is formed on it. The camera's coordinate frame is right-handed with the z-axis defining the centre of the field of view (source: [3])

Based on Equations (2.1), and using homogeneous coordinates, the relationships of a pinhole camera model can be expressed in a convenient matrix form:

$$\begin{aligned}\lambda\begin{bmatrix}x\\y\\1\end{bmatrix} &= \begin{bmatrix}f&0&0&0\\0&f&0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}\\ &= \begin{bmatrix}f&0&0\\0&f&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}\\ &= \boldsymbol{K}\left[\boldsymbol{I}_{3\times3}|\mathbf{0}_{3\times1}\right]\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}\end{aligned} \tag{2.2}$$

The matrix $\boldsymbol{K}$ is the matrix of the intrinsic parameters of the camera. The form presented above contains only one intrinsic parameter, the focal length f, but it is actually an approximation of the reality. In its real form, $\boldsymbol{K}$ contains 5 intrinsic parameters, as follow

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

The parameters c_x and c_y are the offsets of the principal point[1] in the reference frame of the image, as this latter is usually located at the corner of the image and not at the principal point.

The remaining parameters are introduced by the fact that camera sensors are not flawless. Some elements of the sensor may not be perfectly squared, leading thus to a slightly different aspect ratio along the x and y axis. This is why the focal length is separated into two parameters f_x and f_y. In practice, their values are practically equal to each other.

Finally, the parameter s is called the axis skew and allows to represent the distortion that can be caused by pixels that don't have a perfectly rectangular shape, as illustrated in Figure 2.3.

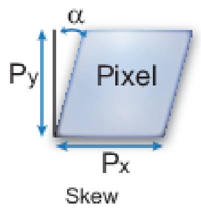


Figure 2.3 – Skew effect (source: [4])

Before going further, be aware that the sensor imperfections just mentioned are a minor source of distortion compared to the one that will be discussed in Section 2.1.3.

2.1.2 Extrinsic parameters

If 3D points in the world have to be expressed in another coordinate system than the one of the camera, as in Figure 2.4, the extrinsic parameters are needed. These parameters represent the transformation between the desired coordinate system and the coordinate system of the camera, and they are thus composed of a rotation and a translation. This transformation can be expressed as a matrix of the following form

$$\left[\boldsymbol{R}_{3\times3} | \boldsymbol{t}_{3\times1}\right] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.4}$$

[1]Intersection between the principal axis and the image plane. See Figure 2.2.

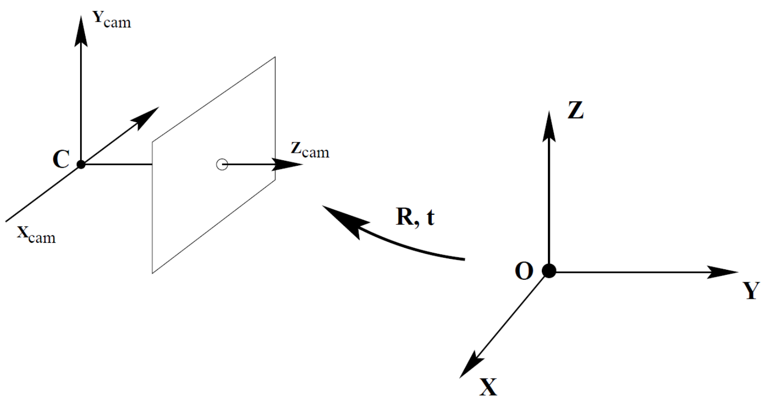


Figure 2.4 – Camera coordinate system VS arbitrary coordinate system (source: [2])

While it may seem that there are thus 12 extrinsic parameters, the rotation matrix R is actually build from 3 rotation parameters only, as there are 3 degrees of freedom in a rotation in the 3D world. There is thus 6 extrinsic parameters, 3 for describing the rotation of the coordinate system, and 3 for describing its translation.

In conclusion, in the case of a point (X, Y, Z) expressed in an arbitrary coordinate system, using the extrinsic parameters matrix, the Equation (2.2) can be reformulated as

$$\begin{aligned}\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} &= \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \\ &= K \left[\boldsymbol{R}_{3\times3} | \boldsymbol{t}_{3\times1} \right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}\end{aligned} \tag{2.5}$$

2.1.3 Distortion

Real cameras do not employ the pinhole system to take pictures. They use instead a lens to focus the light, which can introduce some distortion in the image, mostly radial distortion, but also slight tangential distortion.

To take into account these distortions, the model previously described can be extended as presented in [5]. The coordinates (x, y) of a 3D point (X, Y, Z) projected in the image plane are now given by the following formulas:

$$\begin{cases} x = f_x \, x''' + c_x \\ y = f_y \, y''' + c_y \end{cases} \tag{2.6}$$

Where x''' and y''' are obtained as follow:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{2.7}$$

$$\begin{cases} x'' = \dfrac{x'}{z'} \\ y'' = \dfrac{y'}{z'} \\ r^2 = x''^2 + y''^2 \end{cases} \tag{2.8}$$

$$\begin{cases} x''' = x'' \dfrac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x'' y'' + p_2 (r^2 + 2x''^2) \\ y''' = y'' \dfrac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1 (r^2 + 2y''^2) + 2p_2 x'' y'' \end{cases} \tag{2.9}$$

With k_1, k_2, k_3, k_4, k_5, k_6 the radial distortion coefficients, and p_1, p_2 the tangential ones.

Figure 2.5 illustrates two common types of radial distortion induced by lenses, the barrel distortion (typically $k_1 > 0$) and the pincushion distortion (typically $k_1 < 0$).

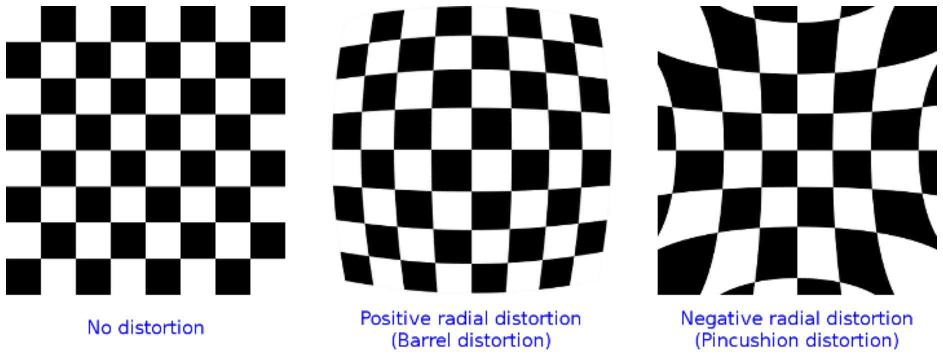


Figure 2.5 – Effect of common radial distortions (source: [5])

2.2 Stereo cameras system

This section explains how it possible to retrieve the depth information from two pictures of the same scene, which is the base concept of stereo vision.

To understand how stereo vision systems works, the concept of disparity needs to be introduced. Consider the setting presented in Figure 2.6, with two identical cameras, side by side, perfectly aligned, such that their sensors are coplanar, and their corresponding rows are located on the same lines. The disparity is the shift that can be observed between a point in the image of the left camera, and its counterpart in the image of the right camera. The closer an object is from

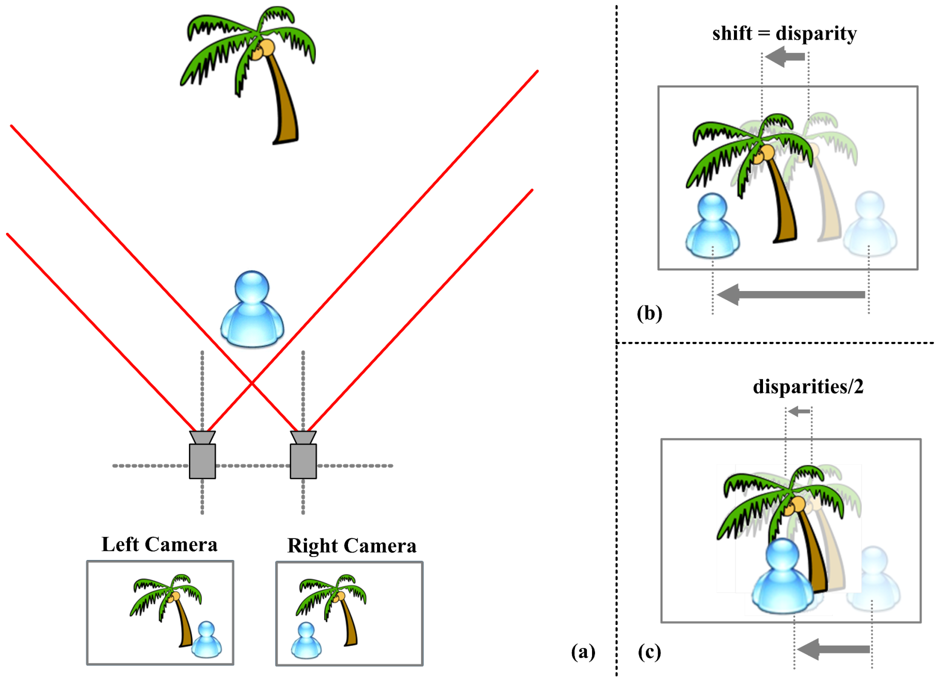


Figure 2.6 – Concept of disparity (source: [6])

the cameras, the bigger the disparity will be. In other words, the depth of an object in the scene increases as the disparity decreases.

Getting back to the simple model of the pinhole camera, the depth of a 3D point P can be obtained as follow. Consider Figure 2.7, which represents a view from above two cameras of center O_R and O_T, separated from each other by a distance B, called the baseline. The indices R and T stand for "Reference camera" and "Target camera". Both cameras have the same focal length f and the pink lines represent the image planes. The points p and p' are respectively the projections of P on the reference and the target cameras. Using the similar triangles technique with PO_RO_T and Ppp', the following relation is obtained:

$$\frac{B}{Z} = \frac{(B + x_T) - x_R}{Z - f} \quad \Rightarrow \quad Z = \frac{Bf}{x_R - x_T} \tag{2.10}$$

Where $x_R - x_T$ is the disparity.

Note also that the depth measured by a stereo vision system is discretized into parallel planes, as the image is discretized into pixels. Figure 2.8 illustrates this fact, showing also that the accuracy will obviously decrease as the depth increases. Some systems can however enhance this accuracy by using sub-pixel techniques.

Knowing all that, the main problem of stereo vision systems is thus to be able to retrieve the pixels that correspond to each other inside the two images. This problem is known as the stereo matching problem and will be further discussed in Section 2.5. What can already be said though, is the fact that, in the case of two identical, distortion free, and perfectly aligned cameras, corresponding pixels are located on the same horizontal line on the two images, as illustrated in

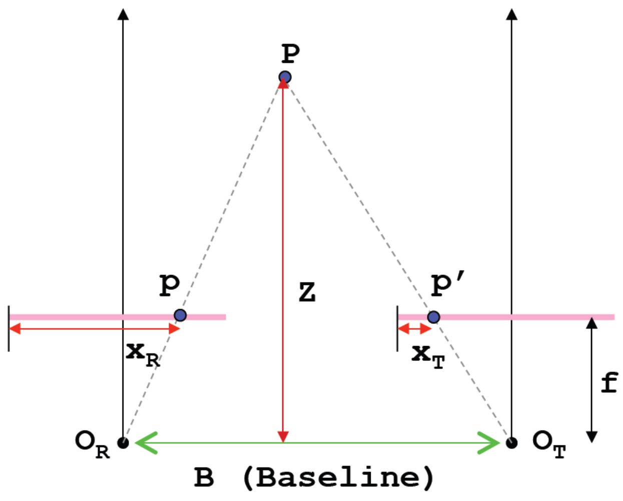


Figure 2.7 – Depth triangulation using two perfectly aligned pinhole cameras (source: [7])

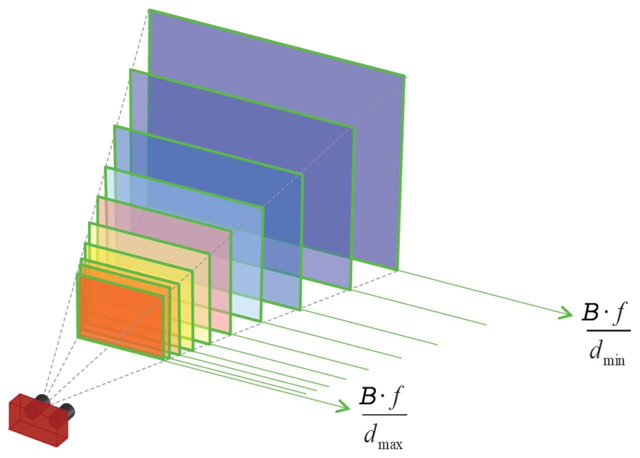


Figure 2.8 – Depth discretization of a stereo vision system (source: [7])

Figure 2.10. However, in real life, it is impossible to align perfectly the cameras, and it usually ends up in a configuration similar as the one presented in Figure 2.9, where O_L and O_R are the optical centers[2] of two unaligned cameras. To understand the consequences of such a misalignment, some concepts of epipolar geometry have to be introduced.

The **epipole** is the point of intersection of the line joining the two optical centers, the so called baseline, with the image plane. Thus, the epipole is the image, in one camera, of the optical centre of the other camera. The points e_L and e_R are the epipoles of the left and right images in Figure 2.9.

The **epipolar plane** is the plane defined by a 3D point X and the optical centers. The plane represented in green in Figure 2.9 is an example of an epipolar plane.

[2]See Section 2.1.1 §1

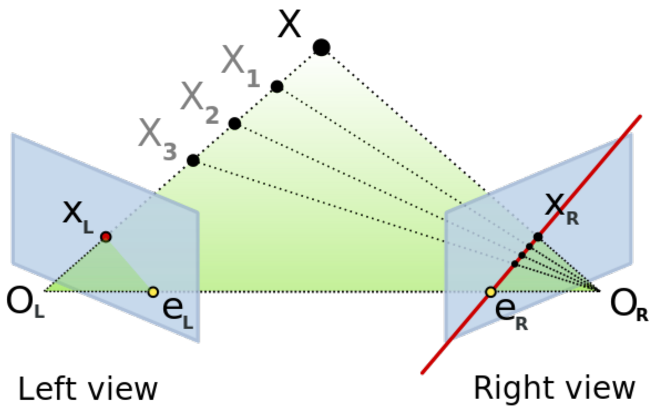


Figure 2.9 – Epipolar geometry (source: [8])

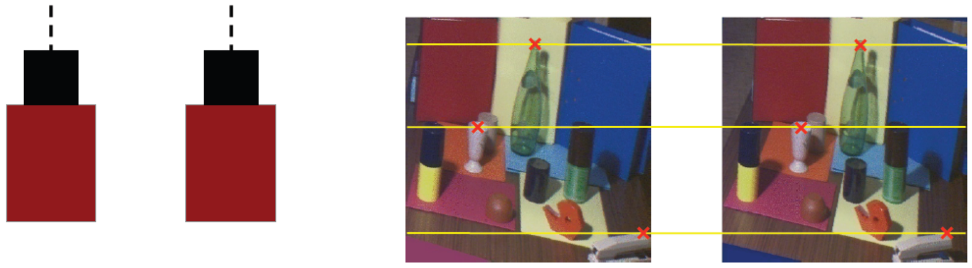


Figure 2.10 – Epipolar lines in stereo pairs taken with perfectly aligned cameras (source: [7])

The **epipolar line** is the straight line of intersection of the epipolar plane with the image plane. It is the image in one camera of a ray through the optical center and image point in the other camera. All epipolar lines intersect at the epipole. In Figure 2.9, the epipolar line of the green plane in the right image is the line $e_R X_R$, represented in red. The corresponding epipolar line in the left image is the line $e_L X_L$.

In such a configuration, corresponding pixels are thus not located on the same horizontal line anymore, but still on an epipolar line, as illustrated in Figure 2.11. This is one of the reasons a calibration of the stereo camera system has to be performed. The calibration will then allow to rectify the images so that they will appear as if the cameras were perfectly aligned, which greatly simplifies the search for matching pixels. These calibration and rectification steps are presented in the two following sections.

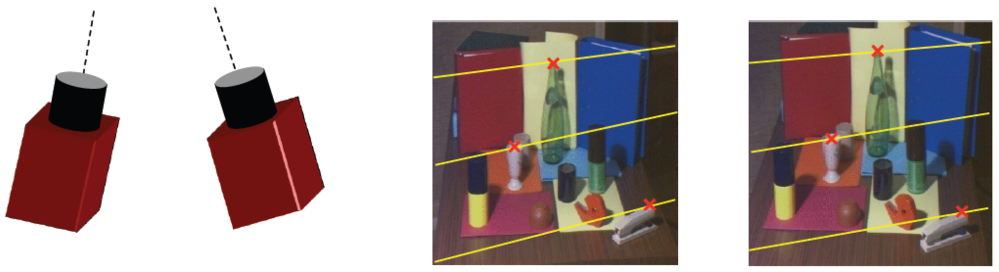


Figure 2.11 – Epipolar lines in stereo pairs taken with unaligned cameras (source: [7])

2.3 Stereo system calibration

The stereo system calibration is responsible for establishing the intrinsic parameters of each cameras, their distortion parameters, as well as the transformation between their reference frame. Indeed, all these parameters are needed in order to rectify a stereo pair before trying to solve the stereo matching problem. Moreover, these parameters are also required to be able to reproject a point in the 3D world on the basis of the disparity, as explained in the previous section.

There exist several calibration techniques, but the focus will be put here on the most commonly used, developed by Zhengyou Zhang [9]. This technique is based on the observation of a 2D checkerboard pattern, and allows to retrieve the intrinsic and distortion parameters of a camera, as well as the position in the 3D world of the checkerboard points, relative to the camera frame. Applying this technique for each of the two cameras, observing the same checkerboard pattern at the same time, it is then possible to deduce the transformation between them from the relative 3D position of the checkerboard in their respective reference frames.

The first step of the calibration is to take a series of stereo image pair of the checkerboard pattern, usually about 20 for better results, with various orientations of the checkerboard. An example of such stereo pairs is presented in Figure 2.12. Each of these pictures have then to be submitted to an algorithm that will detect the location of the checkerboard corners.

Once the coordinates of the checkerboard corners are known in the image plane, the coordinates of these corners in the 3D world have to be known to be able to establish the parameters of the camera as well as its position and orientation relative to the checkerboard. As the size and structure of the checkerboard is known, the trick is to arbitrarily set the 3D world coordinate frame at a corner of the checkerboard, so that all the points of the checkerboard lie on the XY plane of the world coordinate frame. All the corner points detected previously have thus a Z coordinate equal to zero, and their X and Y coordinates can easily be established as the structure of the checkerboard, and the size of its squares are known. An example of this setting is presented in Figure 2.13.

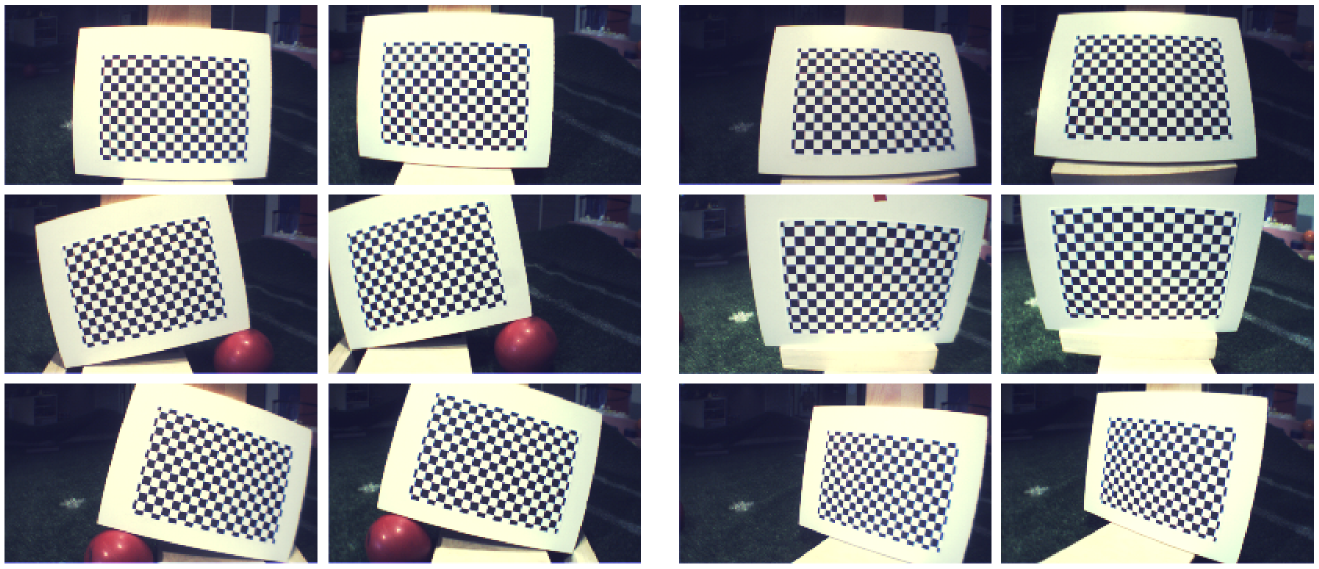


Figure 2.12 – Example of stereo image pair of a checkerboard pattern, parts of a series of pairs taken for the calibration of the stereo camera system developed in this thesis

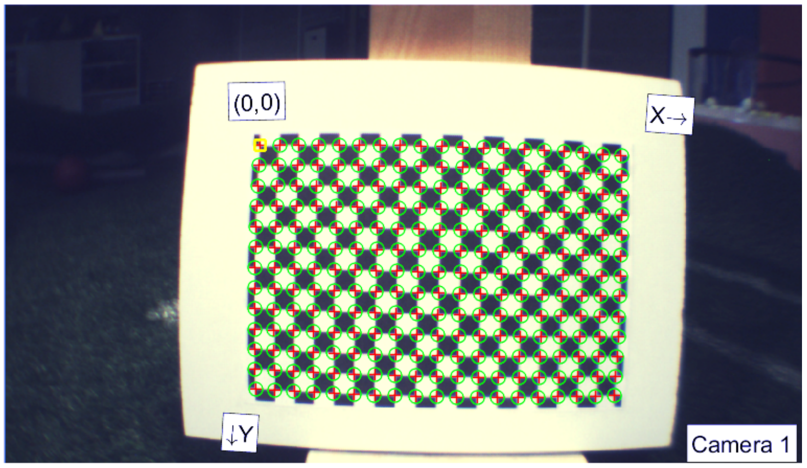


Figure 2.13 – Checkerboard corner detection and setting of the 3D world coordinate frame at the upper left corner of the checkerboard

If the distortion is not considered, for a checkerboard corner point i, we have then the following relationship:

$$\begin{aligned}\lambda \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} &= \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \\ &= \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix} \end{aligned} \tag{2.11}$$

Which can be rewritten as

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \boldsymbol{H}_{3\times 3} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \tag{2.12}$$

Where $\boldsymbol{H}$ is an homography matrix, which has 8 degrees of freedom. Indeed, although $\boldsymbol{H}$ has 9 elements, which would give 9 degrees of freedom, this matrix can be multiplied by any constant without modifying the system of equations. To be able to find the values of the matrix $\boldsymbol{H}$, at least 8 constraints are thus needed. As one point of the checkerboard is giving two constraints (one for x_i, one for y_i), at least 4 points of the checkerboard are needed to solve the system.

Once the matrix $\boldsymbol{H}$ is known, the next step is to be able to extract the intrinsic and extrinsic parameters of the camera from it.

From the following equation

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{h}_1 & \boldsymbol{h}_2 & \boldsymbol{h}_3 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{t} \end{bmatrix} \tag{2.13}$$

We can derive

$$\boldsymbol{r}_1 = \boldsymbol{K}^{-1}\boldsymbol{h}_1 \qquad \boldsymbol{r}_2 = \boldsymbol{K}^{-1}\boldsymbol{h}_2 \tag{2.14}$$

As $\boldsymbol{r}_1$, $\boldsymbol{r}_2$ and $\boldsymbol{r}_3$ come from a rotation matrix, they form an orthonormal basis, and therefore, we have the two following constraints

$$\boldsymbol{r}_1^T \boldsymbol{r}_2 = 0 \tag{2.15}$$

$$\|\boldsymbol{r}_1\| = \|\boldsymbol{r}_2\| = 1 \tag{2.16}$$

Injecting (2.14) into the constraint (2.15), we get

$$\boldsymbol{h}_1^T \boldsymbol{K}^{-T} \boldsymbol{K}^{-1} \boldsymbol{h}_2 = 0 \tag{2.17}$$

Doing the same for the constraint (2.16), we get

$$\begin{aligned} & \boldsymbol{h}_1^T \boldsymbol{K}^{-T} \boldsymbol{K}^{-1} \boldsymbol{h}_1 = \boldsymbol{h}_2^T \boldsymbol{K}^{-T} \boldsymbol{K}^{-1} \boldsymbol{h}_2 \\ \Leftrightarrow \quad & \boldsymbol{h}_1^T \boldsymbol{K}^{-T} \boldsymbol{K}^{-1} \boldsymbol{h}_1 - \boldsymbol{h}_2^T \boldsymbol{K}^{-T} \boldsymbol{K}^{-1} \boldsymbol{h}_2 = 0 \end{aligned} \tag{2.18}$$

Defining the symmetric and positive definite matrix $\boldsymbol{B} = \boldsymbol{K}^{-T}\boldsymbol{K}^{-1}$, (2.17) and (2.18) can then be simplified as

$$\boldsymbol{h}_1^T \boldsymbol{B} \boldsymbol{h}_2 = 0 \tag{2.19}$$

$$\boldsymbol{h}_1^T \boldsymbol{B} \boldsymbol{h}_1 - \boldsymbol{h}_2^T \boldsymbol{B} \boldsymbol{h}_2 = 0 \tag{2.20}$$

This simplification can easily be performed as the matrix $\boldsymbol{K}$ can be recovered from the Cholesky decomposition of $\boldsymbol{B}$. Indeed

$$\begin{aligned} chol(\boldsymbol{B}) &= \boldsymbol{A}\boldsymbol{A}^T \\ \Rightarrow \quad \boldsymbol{A} &= \boldsymbol{K}^{-T} \end{aligned} \tag{2.21}$$

To find $\boldsymbol{K}$, the elements of $\boldsymbol{B}$ have thus to be calculated. As $\boldsymbol{B}$ is symmetric, only 6 elements are unknown, which can be rewritten as a vector $b = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{22} & b_{23} & b_{33} \end{bmatrix}^T$. The expression $\boldsymbol{h}_i^T \boldsymbol{B} \boldsymbol{h}_j$ that appears in Equations (2.19) and (2.20) can be rewritten as $v_{ij}^T b$, so that these two equations become

$$v_{12}^T \boldsymbol{b} = 0 \tag{2.22}$$

$$v_{11}^T \boldsymbol{b} - v_{11}^T \boldsymbol{b} = 0 \tag{2.23}$$

Two such constraints are obtained for each image of the checkerboard. As $\boldsymbol{b}$ have 6 unknowns, at least 3 images are needed to solve the system. In practice, more images are taken to build a system of $2 \times n$ equations, with n the number of images. We end up thus with a system of the form $\boldsymbol{V}\boldsymbol{b} = 0$, which has a trivial solution $\boldsymbol{b} = 0$, not leading to a valid matrix $\boldsymbol{B}$. To find a correct solution, the additional constraint $\|\boldsymbol{b}\| = 1$ can be imposed.

In reality, measurements are noisy, which is why we have to increase the size of the system by taking more than 3 pictures and then try to find the solution that minimizes the least-squares error

$$\boldsymbol{b}^* = arg \min_{\boldsymbol{b}} \|\boldsymbol{V}\boldsymbol{b}\| \qquad with \quad \|\boldsymbol{b}\| = 1 \tag{2.24}$$

From $\boldsymbol{b}$, we can then obtain $\boldsymbol{K}$ using (2.21), and once $\boldsymbol{K}$ is known, the extrinsic parameters for each images can be readily computed. Indeed, from (2.13), we have

$$\begin{aligned} \boldsymbol{r}_1 &= \lambda \boldsymbol{K}^{-1} \boldsymbol{h}_1 \\ \boldsymbol{r}_2 &= \lambda \boldsymbol{K}^{-1} \boldsymbol{h}_2 \\ \boldsymbol{r}_3 &= \boldsymbol{r}_1 \times \boldsymbol{r}_2 \\ \boldsymbol{t} &= \lambda \boldsymbol{K}^{-1} \boldsymbol{h}_3 \end{aligned} \tag{2.25}$$

$$\text{with } \lambda = \frac{1}{\|\boldsymbol{K}^{-1}\boldsymbol{h}_1\|} = \frac{1}{\|\boldsymbol{K}^{-1}\boldsymbol{h}_2\|}$$

Up to now, the distortion introduced by the lens of the camera has not been considered. To estimate the distortion parameters, all the other parameters are first approximated using the technique presented above. After that, these parameters can be used as initial guess to solve the following minimization problem

$$\min_{(\boldsymbol{K},\boldsymbol{q},\boldsymbol{R}_i,\boldsymbol{t}_i)} \sum_i \sum_j \|\boldsymbol{p}_{ij} - \hat{\boldsymbol{p}}(\boldsymbol{K}, \boldsymbol{q}, \boldsymbol{R}_i, \boldsymbol{t}_i, \boldsymbol{P}_{ij})\|^2 \tag{2.26}$$

where $\boldsymbol{q}$ is the vector of the distortion parameters, $\boldsymbol{p}_{ij}$ is the actual projection of the j^{st} point of the checkerboard in image i, $\boldsymbol{P}_{ij}$ is the coordinates of this point in the 3D world, and $\hat{\boldsymbol{p}}$ is the estimated projection of $\boldsymbol{P}_{ij}$ according to all the camera parameters, and following the distortion model presented in Section 2.1.3. Using the Levenberg-Marquardt Algorithm [10] to solve this non-linear optimization problem, an estimation of the distortion parameters can be obtained, as well as a refined estimation of all the other parameters.

2.4 Stereo images rectification

The goal of the rectification of a stereo pair of images is to virtually align the cameras of the stereo system, and to remove the distortion, so that corresponding pixels will be located on the same row in the two images, and so that the depth can be correctly recovered from the disparity between those pixels. This section presents the overall procedure of a stereo pair rectification, knowing the intrinsic, extrinsic, and distortion parameters of the two cameras, as well as the transformation between their respective reference frames.

First, the distortion in the images has to be removed, so that the pinhole camera model can then be used when trying to virtually align the cameras. According to [11], this can be performed by building a distortion mapping, relating each point of the rectified image to the location where that point is mapped on the distorted image. Then, the pixel values of the undistorted image are filled using the distortion map, as follow

$$undistorted\_image(x, y) = original\_image(map_x(x, y), map_y(x, y)) \tag{2.27}$$

where map_x and map_y constitute the distortion mapping, obtained through the following process

$$\begin{aligned}
&x \leftarrow (u - c_x)/f_x \\
&y \leftarrow (v - c_y)/f_y \\
&r^2 \leftarrow x^2 + y^2 \\
&x' \leftarrow x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2) \\
&x' \leftarrow y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy \\
&map_x(u, v) \leftarrow x' f_x + c_x \\
&map_y(u, v) \leftarrow y' f_y + c_y
\end{aligned} \tag{2.28}$$

using the notations of Section 2.1.

Obviously, $map_x(x, y)$ and $map_y(x, y)$ will not necessarily have integer values, and will thus not correspond to a precise pixel of the original image. When this is the case, interpolation techniques can be used. The most commonly used is the bilinear interpolation, since [12] has shown that it performs similarly as other methods, while being faster.

When the distortion has been removed from the left and right images, these images can be rectified so that they look as if they were taken by perfectly aligned cameras. Getting back to the notions of epipolar geometry introduced in Section 2.2, the rectification of an undistorted stereo

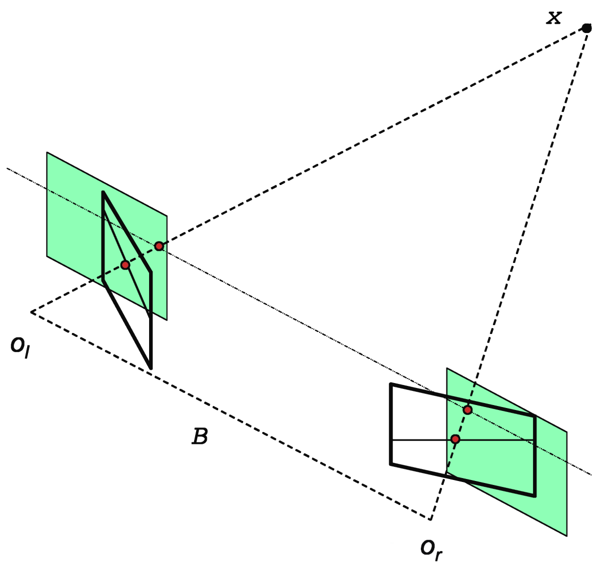


Figure 2.14 – Unaligned VS aligned image planes

pair is the process of remapping the points from the image planes of unaligned cameras, on new image planes which have been rotated such that the epipoles go to infinity and the epipolar lines of these planes are aligned. In other words, the epipolar lines of the new rotated planes should be horizontal and corresponding lines should be located on the same row in both images. This is illustrated in Figure 2.14, where the green image planes are the planes in which the initial images have to be remapped. The following explains the way the points of the undistorted images can be remapped on the new rectified image planes.

The first step is to build the rotation matrix that aligns the left, or reference image plane with the baseline. This matrix, R_{rect}, is built from a unit vector e_1 along the baseline, and the vector e_2, a normalized crossproduct of e_1 with the optical axis that is on the image plane, as follows

$$e_1 = \frac{T}{\|T\|} \tag{2.29}$$

$$e_2 = \frac{1}{\sqrt{T_x^2 + T_y^2}} \begin{bmatrix} -T_y \\ T_x \\ 0 \end{bmatrix} \tag{2.30}$$

$$e_3 = e_1 \times e_2 \tag{2.31}$$

$$R_{rect} = \begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix} \tag{2.32}$$

with T the translation part of the transformation between the two camera reference frames, obtained after calibration.

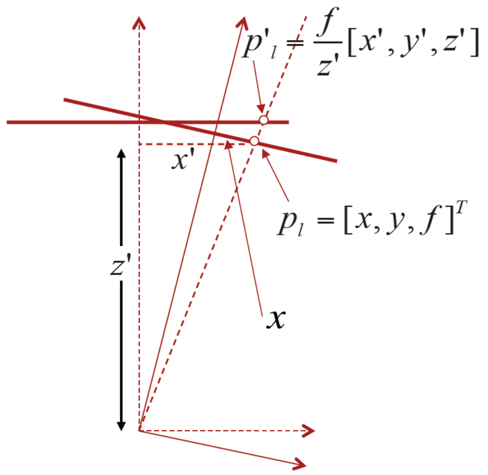


Figure 2.15 – Projection of a point from an image plane to the image plane aligned with the baseline (source: [13])

Consider now the schema of Figure 2.15. Using the matrix R_{rect}, a point p_l in the initial image plane, can be expressed in the reference frame of the new virtual camera aligned with the baseline:

$$p_l = [x, y, f]^T \tag{2.33}$$

$$R_{rect} p_l = [x', y', z']^T \tag{2.34}$$

Using then the similar triangle technique, the projection p'_l of this point in the new image plane can be obtained:

$$p'_l = \frac{f}{z'}[x', y', z']^T \quad = [x'', y'', z'']^T \tag{2.35}$$

And using the intrinsic parameters of the camera, the pixel coordinates (u, v) can be retrieved:

$$u = f_x \frac{x''}{z''} + c_x = f_x \frac{f x'}{z'} \frac{1}{f} + c_x = \frac{f_x x'}{z'} + c_x \tag{2.36}$$

$$y = f_y \frac{y''}{z''} + c_x = f_x \frac{f y'}{z'} \frac{1}{f} + c_y = \frac{f_y y'}{z'} + c_y \tag{2.37}$$

Of course, not only the points of the left image have to be remapped. Applying the rotation R_{rect}, followed by the rotation R between the unaligned cameras (obtained from calibration), the right image plane can be aligned with the new left image plane. The mapping between the points in the initial and new image planes is obtained exactly the same way as with the left image.

However, the mapping that was just described gives the location of a pixel in the new image plane, from a pixel in the initial image plane, while the inverse is actually needed. Indeed, as in the case of distortion removal, the rectified image is build by filling all the pixel values of the new image plane. A mapping that gives the location of a pixel in the initial image, from the coordinates of a pixel in the rectified image, is thus needed. Such a mapping can be obtained by performing backwards the previously described operations. The detailed mathematics can be found in [13].

Usually, once the rectified images have been obtained, they are cropped before being fed to a stereo matching algorithm, so that they only contain known pixel values. A rectangular region of interest can be defined for each image, which covers as much of the image as possible, without including unknown pixels values. The intersection between the region of interest of both images is then the cropped region. This is illustrated in Figure 2.16.

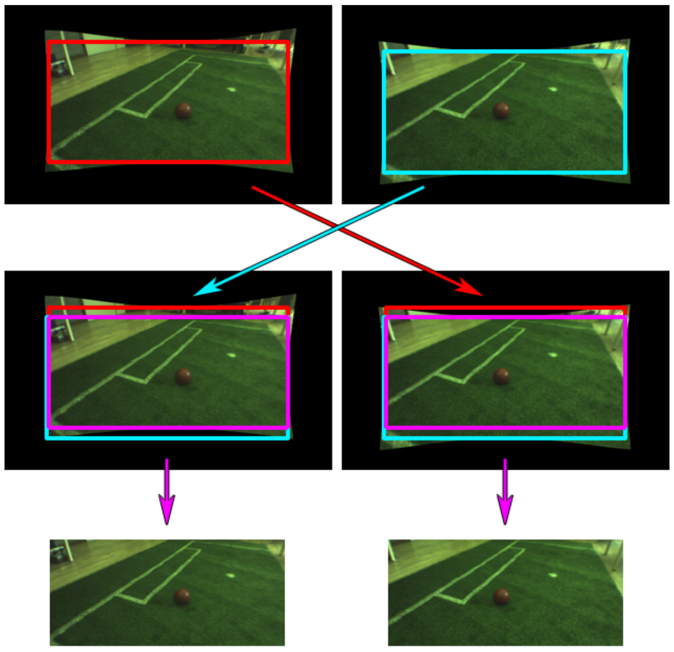


Figure 2.16 – Cropping of a rectified stereo image pair

To conclude this section, a simplified summary of the rectification main steps is presented in Figure 2.17.

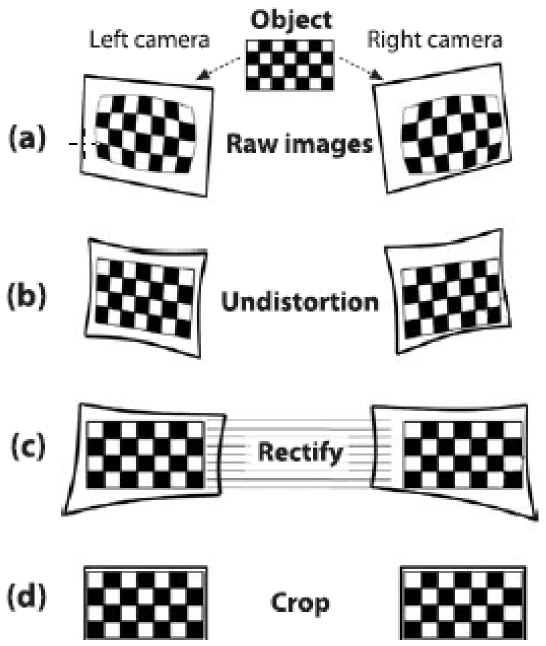


Figure 2.17 – Main steps of the rectification of a stereo image pair (source: [13])

2.5 The stereo matching problem

The stereo matching problem is the problem of finding corresponding pixels in a stereo pair of images. The goal of stereo matching is to obtain what is called a disparity map, containing a disparity value for each pixel of a reference image. Those disparities can then be transformed into depth as explained in Section 2.2.

This section will first describe the general approaches that are used to solve the stereo matching problem. Then, it will enumerate the main issues that make this problem challenging.

2.5.1 Solving the matching problem

Using a rectified stereo pair of images, the search of a corresponding pixel is 1-Dimensional, as a pixel from the reference image and its equivalent in the target image are located on the same row. Moreover, as the disparity can only be greater or equal to 0, the search can start on the same column as the reference pixel, and end at a maximal disparity value that can be defined in the system. This is illustrated in Figure 2.18, where the candidate pixels for correspondence are highlighted in the target image, for one particular pixel of the reference image.

To be able to select which of these candidates is the correct one, a similarity measure, also called matching cost, has to be defined. Selecting then, in the target image, the most similar pixel as the corresponding one, a disparity value can be directly computed from the positions of these two pixels in their respective frames. Doing so for every pixel of the reference image, the disparity map can be build.

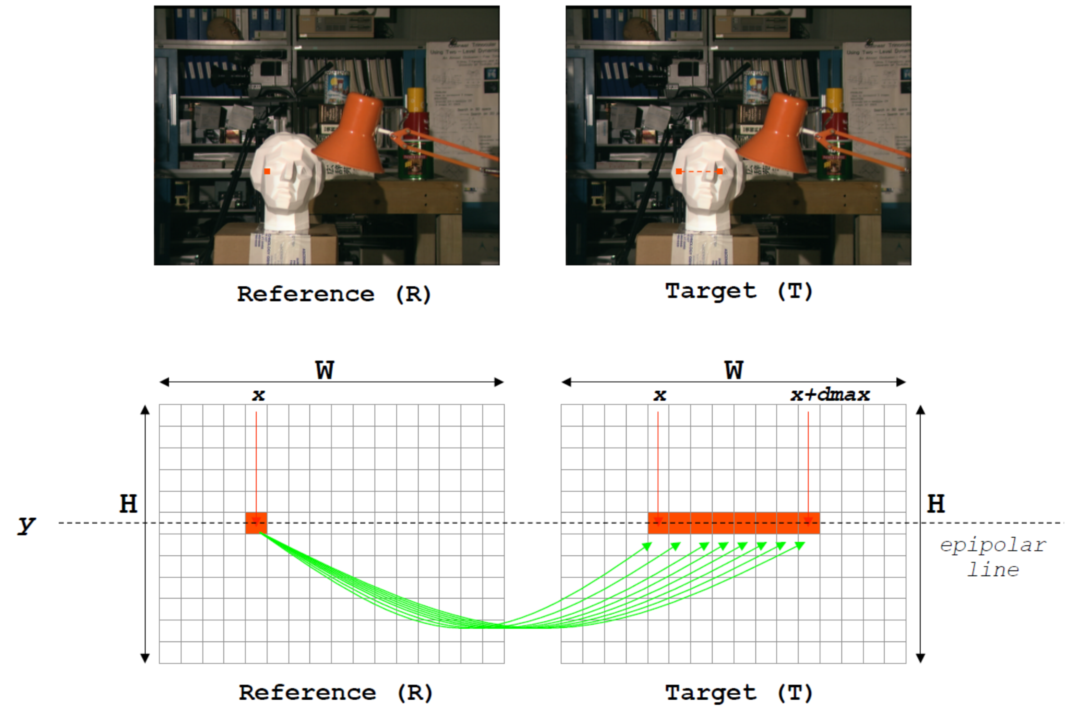


Figure 2.18 – Epipolar search of a correspondent pixel in a rectified stereo pair (source: [7])

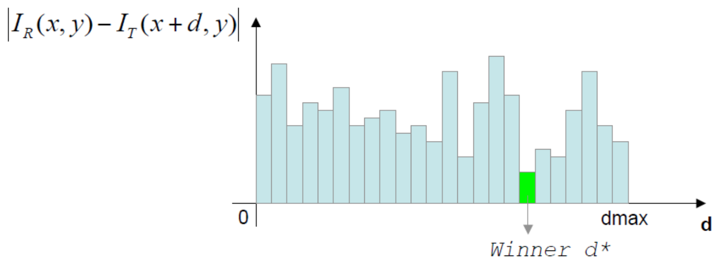


Figure 2.19 – "Winner Takes All" example using the difference of the pixel intensities as similarity measure (source: [7])

The naive method would be to use the difference in pixel intensities as similarity measure, and then to opt for what is called the "Winner Takes All" (WTA) selection. In other words, the pixel with the best similarity measure is selected, or in this particular case, the pixel for which the difference in intensities is minimum (see Figure 2.19). However, this technique leads to a very disappointing result, as shown in Figure 2.20. Indeed, looking at the intensities of single pixels over the full disparity range might lead to many similar candidates, and as the measurements of these intensities might not be perfectly identical in the the reference and target images, the most similar might not always be the right one. The same matching cost could even be obtained for multiple pixels, making thus the choice completely random with such a naive approach. In practice, more complex and robust similarity measures can be used, usually taking into account the neighborhood of the considered pixel.

Moreover, some techniques do not necessarily select the most similar pixel according to the chosen measure. Indeed, the ensemble of stereo matching techniques can be divided into two

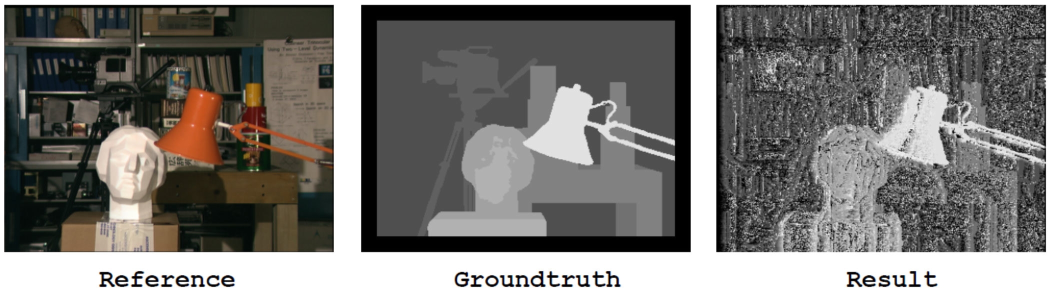


Figure 2.20 – Comparison between the disparity groundtruth and a disparity map obtained from a naïve stereo matching method (source: [7]). In those disparity map, each gray level represents a particular disparity value. The darker the gray is, the smaller is the disparity.

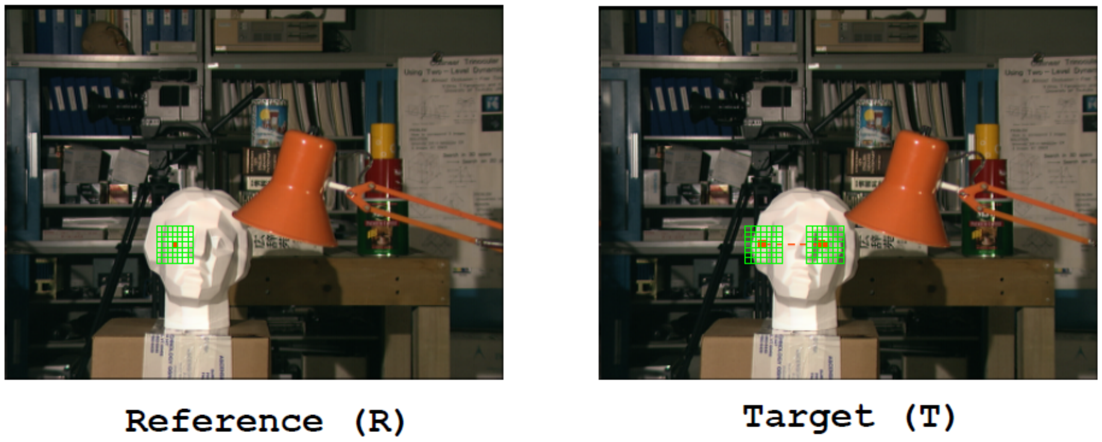


Figure 2.21 – Aggregation of the matching costs over a support window (source: [7])

categories: the local methods and the global methods.

Local algorithms are the ones using the simple WTA disparity selection strategy. However, to try to reduce the ambiguity between the match candidates, they do not consider the matching cost directly to find the most similar pixel, they aggregate the matching costs over a support window, as illustrated in Figure 2.21, and compare these aggregations to establish which pixel is the winner.

The global algorithms do not use the WTA strategy but rather search for disparity assignments that minimize an energy function over the whole stereo pair.

Lastly, disparity refinement methods can be applied to improve the accuracy of raw disparity maps computed by any stereo matching algorithm. Such method can try for example to identify and correct potential outliers, or to compute disparities at a sub-pixel level.

2.5.2 Technical difficulties

This section lists the main problematic situations that are encountered when tackling the stereo matching problem:

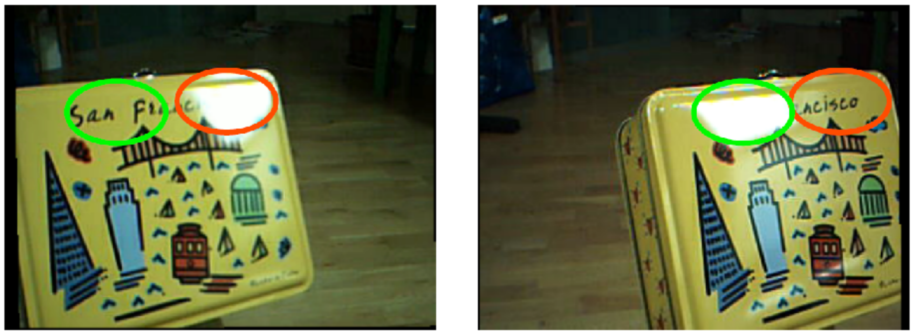


Figure 2.22 – Stereo pair of a specular surface (source: [7])

- **Specular surfaces:** the reflection of the light on such surfaces is perceived differently from different points of view, making thus difficult to identify the disparity in the areas where the light is reflected, as these areas occlude some details in the images, and are not located in the same place in the two images. An example is shown in Figure 2.22.

- **Foreshortening:** perspective changes from different points of view, which can lead to foreshortened objects details in one of the stereo pictures, making it more difficult to identify the similar points. This situation is illustrated in Figure 2.23.

- **Uniform regions:** Uniform, textureless regions are problematic because it not possible to use a similarity measure to identify correspondent pixels inside those areas, as these pixels are all similar. Figure 2.24 shows an example of this problem.

- **Repetitive patterns:** similar patterns can occur in the image. It is thus more difficult to select which of the occurrence of such a pattern is the correct one. A clear example is presented in Figure 2.25, but similar patterns can also be present inside images in a more subtle way.

- **Transparent objects:** transparent objects let the background visible through them, which results in some dissimilarity between correspondent pixels belonging to such objects. This situation is illustrated in Figure 2.26.

- **Occlusions and discontinuities:** Parts of some objects can only be visible in one of the stereo pictures, because these parts are occluded in the second image by other objects in the foreground. This problematic is highlighted in Figure 2.27.

- **Noise and all kinds of distortions:** Measurement noise and other kind of distortion (photometric distortion, radiometric distortion, ...) can also mislead the selection of correspondent pixels.

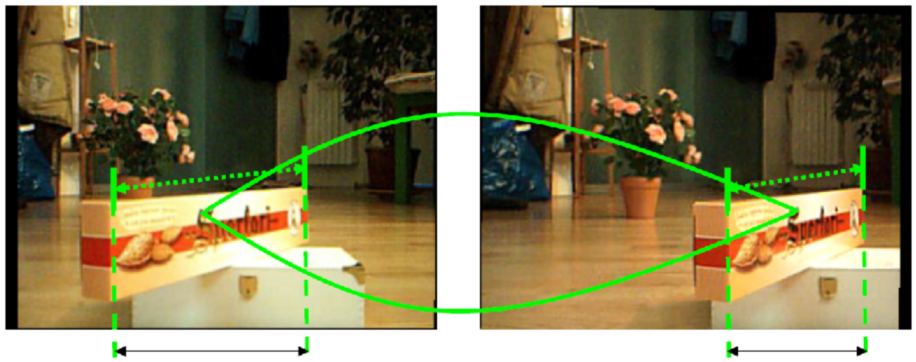


Figure 2.23 – Foreshortening effect in a stereo pair (source: [7])

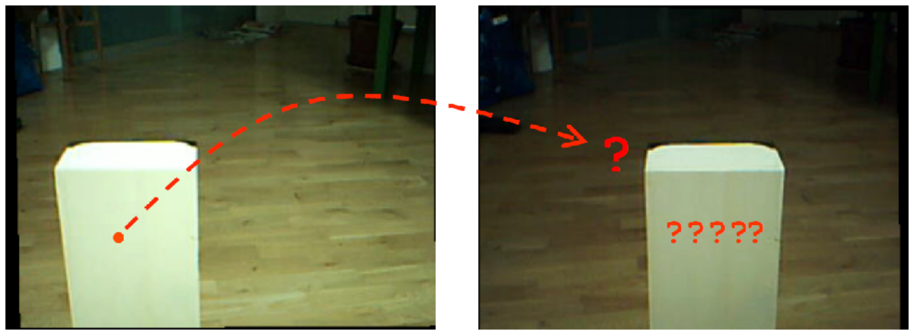


Figure 2.24 – Stereo pair of textureless surface (source: [7])

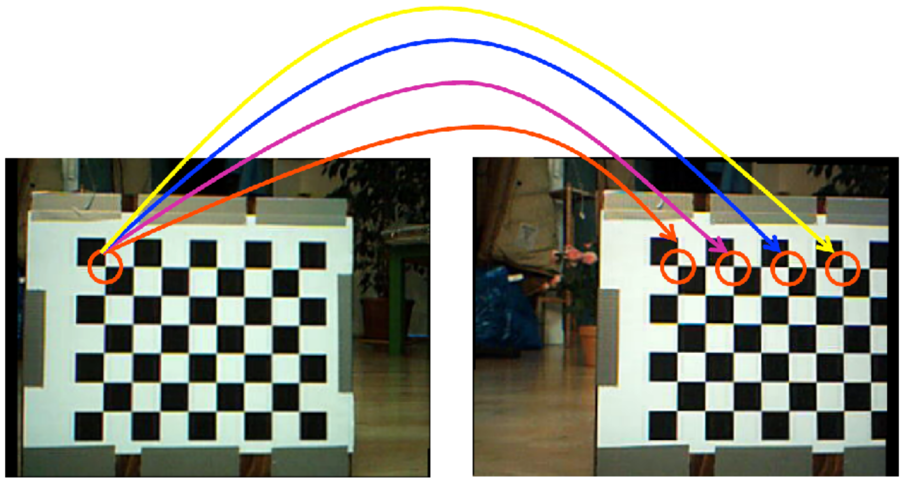


Figure 2.25 – Stereo pair of checkerboard presenting a repetitive pattern (source: [7])

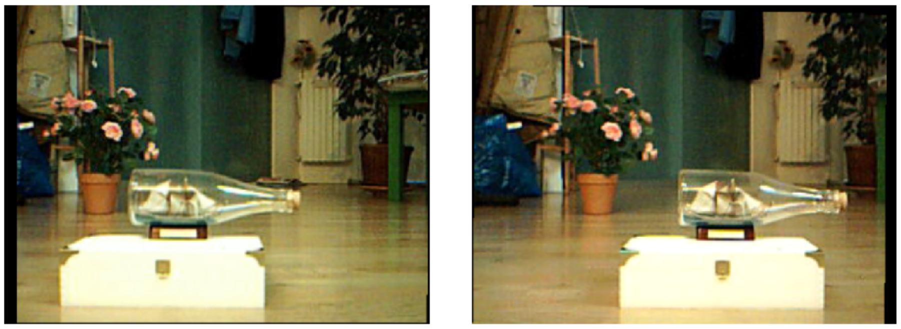


Figure 2.26 – Stereo pair of a transparent object (source: [7])

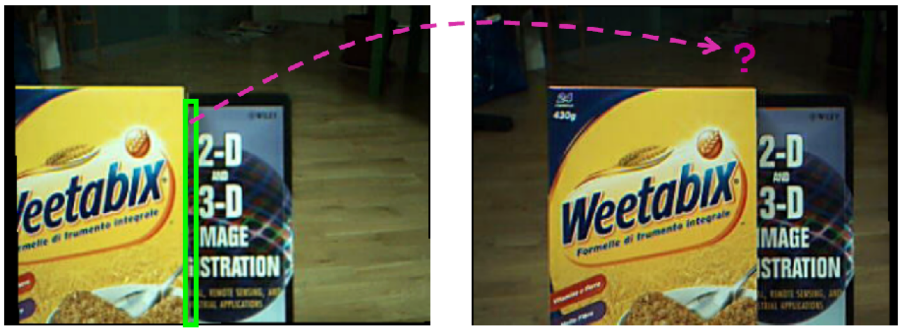


Figure 2.27 – Occlusion in a stereo pair (source: [7])

2.6 General algorithm for stereovision

A summary of the main steps of a stereo vision algorithm is presented in Figure 2.28.

The first step is the **calibration** of the stereo system. It is an offline operation supposed to be performed only once before being able to use the stereo system. As explained in Section 2.3, it is responsible for determining the intrinsic, extrinsic, and distortion parameters, from a series of stereo pair images of a checkerboard pattern.

The general algorithm for stereo vision is then composed of three main steps. The first of this step is the **rectification**, explained in Section 2.4. Given a stereo pair of images taken by the stereo system, and knowing the parameters of the cameras and their relative position, it produces a new stereo pair usable by the next step, the stereo matching.

The **stereo matching** was presented in Section 2.5. Given a stereo pair of rectified images, it is responsible for building the disparity map.

Finally, the last step is the **reprojection** of the reference image points in the 3D world using the disparity map and the parameters obtained during calibration. This part was explained in Section 2.2.

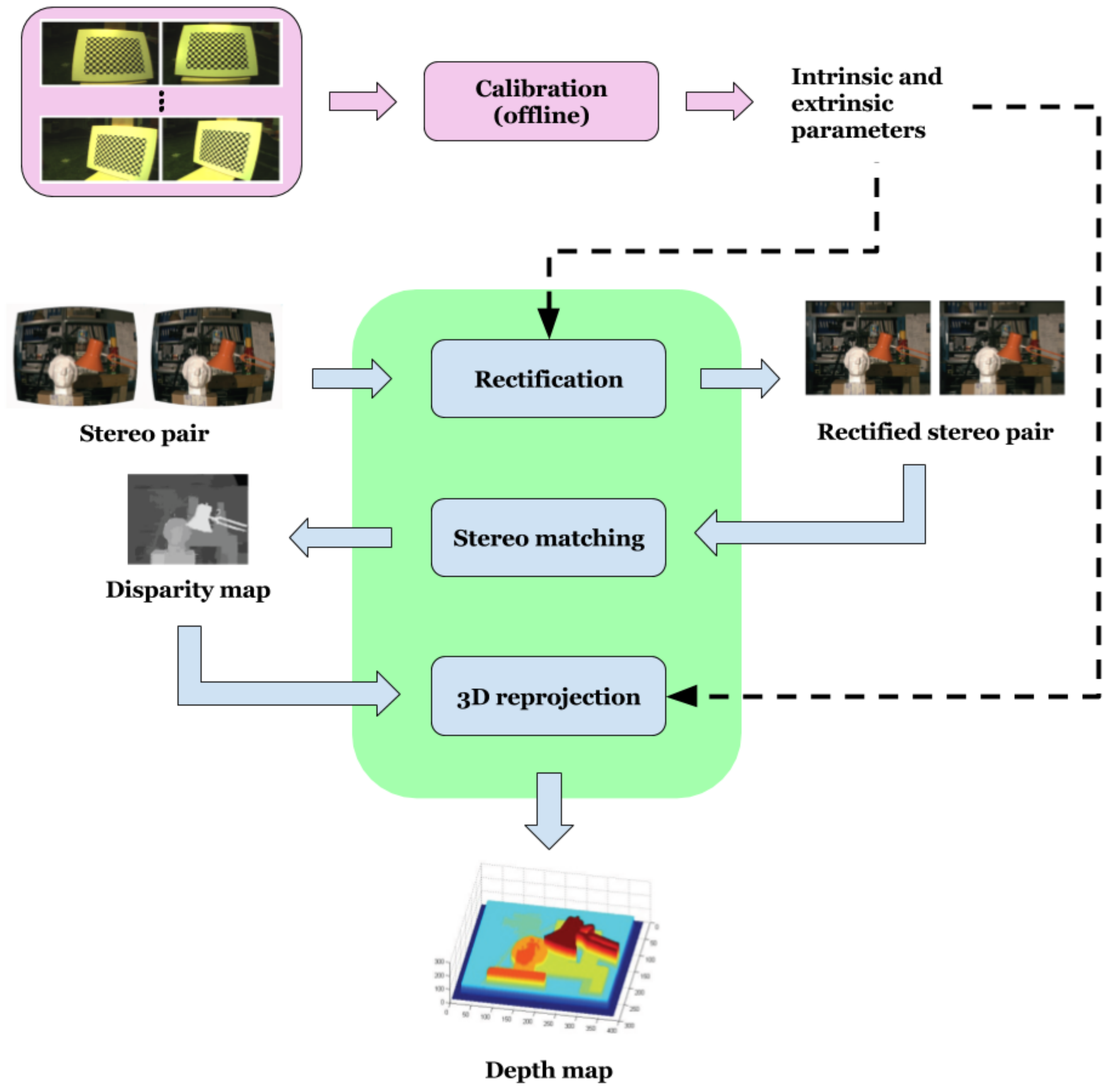


Figure 2.28 – Main steps of a stereo vision algorithm

3. State of the art and choices

Throughout the years, numerous algorithms have been developed trying to generate disparity maps as accurately as possible. However, disparity map generation is a computationally expensive process. Indeed, matching costs, sometimes complex, need to be computed for each pixel and each level of disparity. Moreover, these matching costs often needs to be aggregated over support windows or submitted to optimization processes, which increases again the number of computations required. Therefore, the algorithms that are able to generate an accurate result can not usually run in real time without the help of specialized hardware. A lot of efforts have thus been put by the scientific community on trying to take advantage of the computation power of FPGA or GPU systems to accelerate those algorithms.

FPGA stands for "Field Programmable Gate Array". A FPGA is an integrated circuit whose logic blocks can be freely configured to efficiently perform custom operations. FPGA can thus be programmed to handle the operations of a stereo vision algorithm faster than with a classic processor, whose set of instructions is predefined. An example of the time performance of a state-of-the-art FPGA stereo vision system is presented in Table 3.1, and some generated disparity maps can be observed in Figure 3.1. The stereo matching algorithm implemented in that particular case is called "Semi-Global Matching" [14], or SGBM, and is also one the few stereo matching algorithms implemented in the famous computer vision library *OpenCV*.

Graphical processing units (GPU) can also be used to accelerate stereo vision algorithms as they are electronic circuits specifically designed to manipulate images efficiently. The big difference between GPUs and FPGAs is that a GPU runs software, and software implementations are supposed to be slower than hardware implementations as time is needed, among others, to fetch instructions, performs math operations and send data to memory. However, the massively parallel design of GPUs allows them to run software much faster than a simple processor could.

For the platforms that do not dispose of such hardware, researchers also explored the possibilities of optimization using SIMD instructions which are available in most processors. SIMD stands for

	Image resolution			
Disp. range	640×480	800×592	1248×960	1440×1072
64 pixels	40 fps	25 fps	9 fps	7 fps
128 pixels	30 fps	20 fps	8 fps	6 fps
256 pixels	15 fps	10 fps	4 fps	n/a

Table 3.1 – Example of time performance achieved by the "Semi-Global Matching" algorithm implemented on a FPGA (Source: [15])

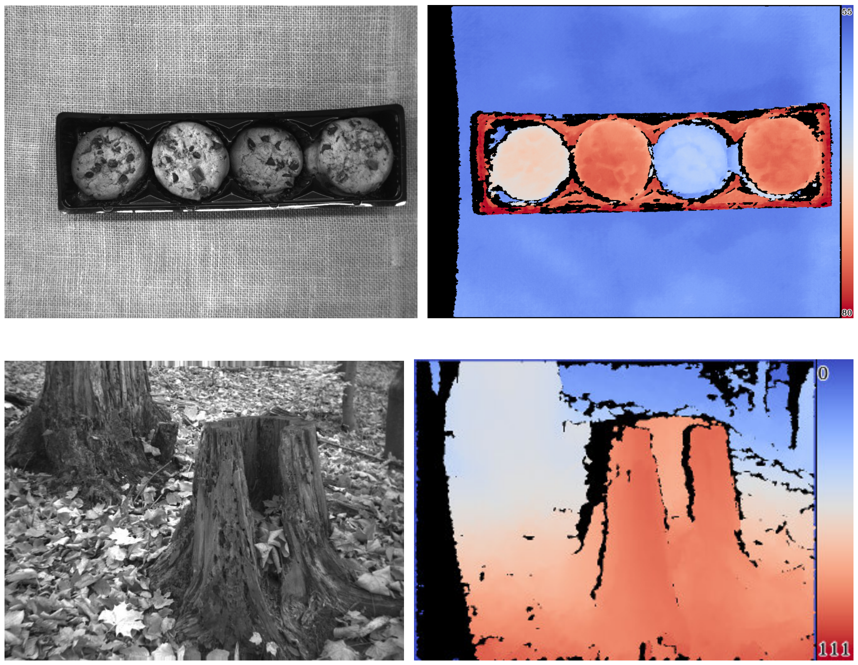


Figure 3.1 – Example of disparity maps generated using the "Semi-Global Matching" algorithm implemented on a FPGA (Source: [15])

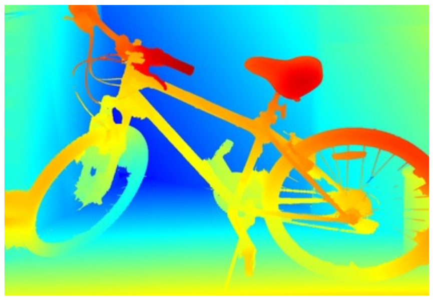
"Single Instruction/Multiple Data" and allows, as its name suggests, to process multiple data with a single instruction. Such techniques are thus particularly useful in the context of this thesis, as the developed technique is intended to be run on single CPU core.

Searching for the right compromise between speed and accuracy, the ranking of stereo matching algorithms proposed on the *Middlebury Stereo Evaluation* platform [16] was studied. This platform allows anyone to submit the results of his algorithm. The results can then be compared based on various error measures, or based on their execution time. The *localExp* [17] algorithm seems to be the most accurate, with a mean absolute disparity error of 2.24pixels, and only 13.9% of pixels for which the error is greater than 1. This algorithm needs however 11 minutes to process 1MP[1]. Most of the top algorithms in this ranking are unfortunately not suited for real-time on a simple CPU. When looking at the ranking in terms of execution speed, the fastest algorithm, *IDR* [18], is able to process 2.78MP per second, implemented on a GPU. The second fastest algorithm, called *ELAS* [19], can process 2Mp per second on a single i7 CPU core, which first seemed interesting in the context of this thesis. A visual comparison of the disparity maps that those algorithms can generate is presented in Figure 3.2, and allows to get a nice intuition about their performance in terms of accuracy.

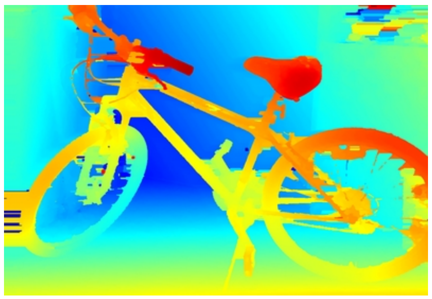
However, a processing time of 2MP per second might be too slow for the targeted application. Indeed, considering a VGA resolution[2], about 6 disparity maps could be output per second, not even considering the fact that the processor of the robot will be less powerful than an i7 core. Further researches were thus carried out. While they confirmed that the trend has been towards the improvement of the accuracy of the disparities, at the expense of speed, a paper that seemed

[1]MegaPixel
[2]640x480

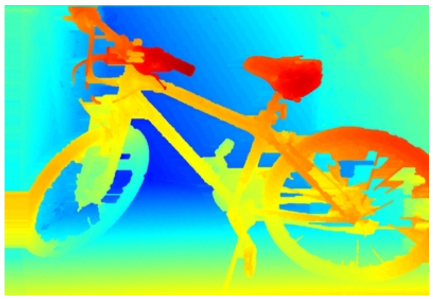
(a) Reference image



(b) localExp [17]
mean error: 2.24px



(c) IDR [18]
mean error: 6.35px



(d) ELAS [19]
mean error: 9.52px

Figure 3.2 – Comparison of disparity maps generated by some of the algorithms evaluated on the Middleburry platform (Source: [16])

to satisfy the requirements imposed in this thesis was still found.

The paper in question [1] proposes an algorithm that should be able to process more than 100 stereo pairs per seconds in VGA resolution. What differentiate this algorithm from the ones mentioned until now, is the fact that the disparity map that it outputs is semi-dense, meaning that the disparities are not computed for every pixel of the image. This amount of information could nevertheless be sufficient to help the robot perform its tasks. The semi-dense nature of the output disparity maps, and the optimization of some part of the algorithm using SIMD instructions, are the main reasons it can reach such a speed of execution. Moreover, the accuracy of the algorithm compares well with other popular methods, according to Table 3.2. This solution seemed thus to be an interesting compromise between speed and accuracy, and was thus chosen to be implemented and tested in this thesis.

Method	**Accuracy (%)**			
	< 2px	< 3px	< 4px	< 5px
SGBM [14]	89.0	93.9	95.6	96.5
ELAS [19]	92.7	96.1	97.3	97.9
proposed [1]	83.1	89.9	92.9	94.7

Table 3.2 – Comparison of stereo matching accuracies. The accuracies are expressed in percentage of disparities estimated with an error inferior to the specified value (Source: [1])

4. The implemented stereo matching algorithm

4.1 Principles

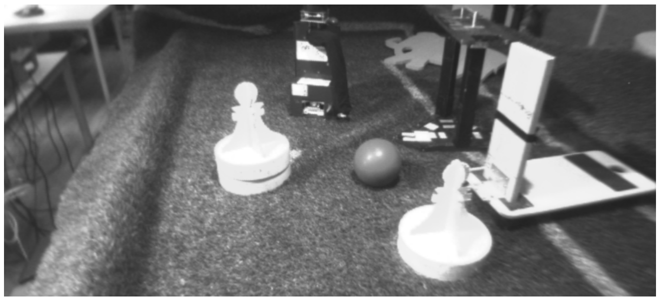
As explained in the previous section, the stereo matching algorithm that was chosen to be implemented and tested is the one from [1].

Given a stereo pair of rectified gray-scale images as input, this algorithm outputs a semi-dense disparity map, meaning that it does not necessarily contain a disparity value for every pixel in the image. The disparity is calculated only for the pixels having their gradient above a certain threshold, focusing thus more on the contour of the objects in the images, and the textured regions. Not having to calculate a disparity value for each pixel of the input reference image allows thus to gain some execution time.

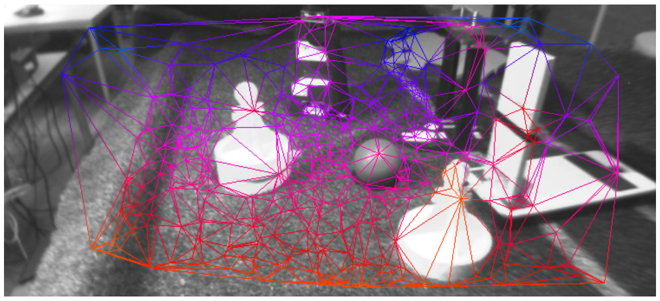
To calculate the disparities of all the high gradient pixels, the algorithm first uses the method developed in [20] to obtain a sparse set of points with a confident disparity value. These points are then used as support points to build a triangular mesh, or piece-wise planar surface, approximating the disparity in the image. Disparity values can thus be interpolated from this surface, anywhere in the image, and a system of matching costs also ensures that only the sufficiently confident disparities can be output.

Moreover, the triangular mesh can be refined using an iterative process, with a number of iterations that can be freely tuned to fit the needs in accuracy and speed. An example of different levels of refinement that can be obtained for the triangular mesh is shown in Figure 4.1. This refinement is performed by dividing the reference image into a grid, and for each cell of this grid, by selecting the pixel having the most confident interpolated disparity value, and the pixel having the less confident one. The first pixel is added as a support point to the triangular mesh. An epipolar search for the correspondent of the second pixel is then performed in the target image. If a correspondence is established with a sufficient certainty, the pixel is also added as support point to the mesh, with a new disparity computed from this new match. At each iteration, the number of cells in which the image is divided increases, allowing thus to add more and more points as supports of the mesh.

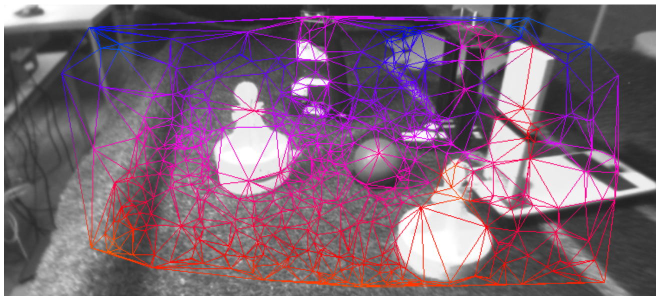
All the steps of this stereo matching technique are presented in Algorithm 1. Each of these steps are then described in details in separate sections. The meaning of the symbols appearing in the algorithm is detailed in Table 4.1. Note also that a few adaptations have been introduced



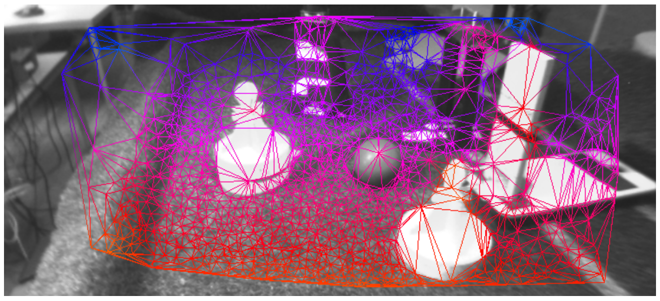
(a) Reference image



(b) Mesh at first iteration



(c) Mesh at iteration 2



(d) Mesh at iteration 4

Figure 4.1 – Refinement of the triangular mesh. Colors illustrate the scene depths, with orange indicating near-field and blue indicating far-field regions.

in comparison with the original algorithm from [1]. Preprocessing options have been added to enhance the detection of support points, contrast enhancement and image smoothing have been used to improve the overall results, and because the original paper was not providing any details about this matter, a custom epipolar search has been proposed, including costs aggregation and bidirectional matching.

Algorithm 1: Semi-dense stereo matching

```
Input  : $(I_l, I_r)$: Input gray-scale stereo images
Output: $D_f$: Disparities at high-gradient regions (Semi-Dense)
   // Initialize final disparity and associated cost
1  $D_f \leftarrow [0]_{[H \times W]}, \quad C_f \leftarrow [t_{hi}]_{[H \times W]}, \quad sz_{occ} \leftarrow 32$
   // Enhance the contrast of the input images
2  $I_l, I_r \leftarrow \textsc{HistogramEqualization}(I_l, I_r)$
   // Get high gradient pixels
3  $\Omega \leftarrow \textsc{GradientThresholding}(I_l)$
   // Apply some transformation to the input images
4  $I'_l, I'_r \leftarrow \textsc{ImageFiltering}(I_l, I_r)$
   // Census transform of the input images
5  $J_l \leftarrow \textsc{CensusTransform}(I'_l)$
6  $J_r \leftarrow \textsc{CensusTransform}(I'_r)$
   // S: Set of N support points
7  $S_1 \leftarrow \textsc{SparseStereo}(I_l, I_r, J_l, J_r)$
   // Tessellated mesh with estimated disparities
8  $G(S_1) \leftarrow \textsc{DelaunayTriangulation}(S_1)$
9  for $it = 1 \rightarrow n_{iters}$ do
       // Dense piece-wise planar disparity
10     $D_{it} \leftarrow \textsc{DipsarityInterpolation}(G(S_{it}), \Omega)$
       // Cost evaluation given interpolated disparity
11     $C_{it} \leftarrow \textsc{CostEvaluation}(J_l, J_r, D_{it}, \Omega)$
       // Refine disparities
12     $C_g, C_b \leftarrow \textsc{DisparityRefinement}(D_{it}, C_{it}, D_f, C_f, \Omega, sz_{occ})$
       // Prepare for next iteration, if not the last one
13     if $it \neq n_{iters}$ then
           // Re-sample regions with high matching cost
14         $S_{it+1} \leftarrow \textsc{SupportResampling}(C_b, C_g, S_{it}, J_l, J_r)$
           // Tessellated mesh with estimated disparities
15         $G(S_{it+1}) \leftarrow \textsc{DelaunayTriangulation}(S_{it+1})$
           // Decrease grid cell size by factor 2
16         $sz_{occ} = max(1, sz_{occ}/2)$
17     end
18 end
```

Name	Description
I_l, I_r	Input gray-scale stereo images
H, W	Dimensions of input image I_l
Ω	Set of high-gradient pixels
I'_l, I'_r	Filtered input images
J_l, J_r	Census transforms of the input images
S	Sparse support pixels with valid depths
$G(S)$	Graph resulting from Delaunay Triangulation over S
D_f	Final disparity image
C_f	Cost matrix associated to D_f
D_{it}	Intermediate disparity (interpolated)
C_{it}	Costs associated to D_{it}
C_g	Costs associated with pixels of high confidence matches
C_b	Costs associated with pixels of invalid disparities
sz_{occ}	grid cell size used for re-sampling
t_{lo}, t_{hi}	Lower and upper cost threshold for validating disparities
n_{iters}	Number of iterations the algorithm is allowed to run

Table 4.1 – Description of symbols used in the proposed stereo matching algorithm

The following sections will now describe the internal operations of the proposed stereo matching algorithm.

4.2 Histogram equalization

Histogram equalization is used to enhance the contrast in the input stereo gray-scale images.

According to [21], the histogram of an image is the graphical representation of its intensity distribution, and quantifies the number of pixels for each intensity value considered. In a gray-scale image, these intensities are usually encoded in 8bits and range thus from 0 to 255[1].

Histogram equalization is the process of stretching out the range of intensities that are observed in an image. Consider Figure 4.2, in which the original image contains pixels that are concentrated in a range of intensities going from 25 to 150. In the equalized image, all the intensities are spread over the full range, 0 to 255, which enhance the contrast.

Let I be a gray-scale image of dimensions $N \times M$. The normalized histogram of this image is given by

$$h_i = \frac{\text{number of pixels with intensity } i}{N\,M} \qquad i = 0, ..., 255 \tag{4.1}$$

[1]The cameras used in this work actually measure the pixel intensities using 12bits of information. As the image acquisition code, realized prior to this work, stores this information on 8bits, and as the libraries used do not support the 12bits format, the focus was not put on studying the benefit of such additional information.

The pixel intensities of the new image I' whose histogram is equalized can then be obtained using

$$I'(u,v) = \lfloor 255 \sum_{i=0}^{I(u,v)} h_i + 0.5 \rfloor \tag{4.2}$$

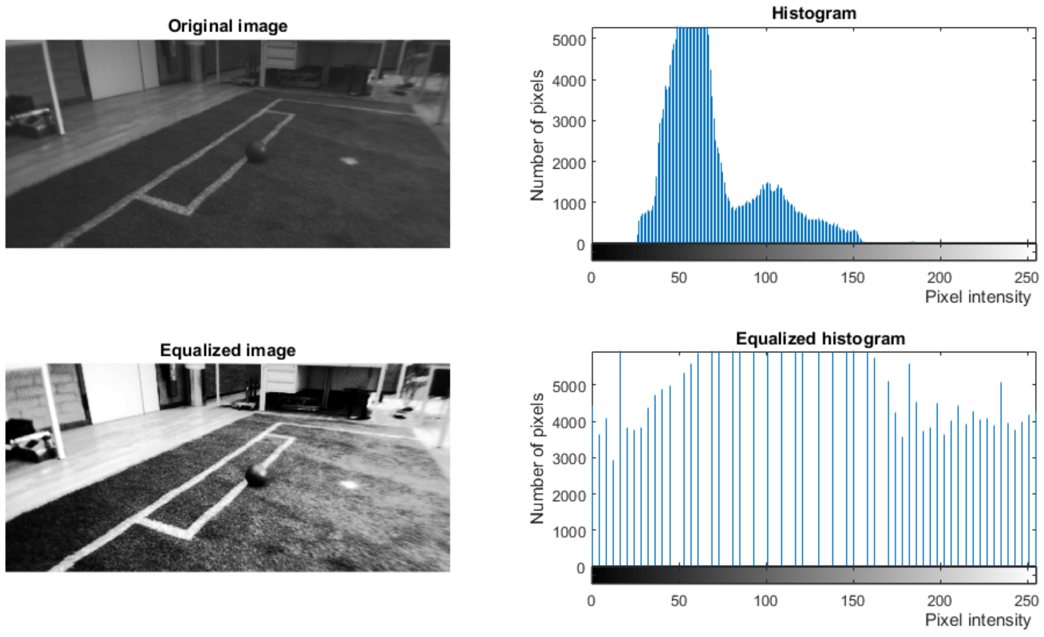


Figure 4.2 – Histogram equalization example

The reason the contrast of the images needs to be enhance in such a way, is that it will help the detection of features that is performed by the SparseStereo step of the algorithm (see Section 4.6). Moreover, it allows to obtain a more uniform basis for all the images that will go through the algorithm, which should therefore be more robust to variations in lighting conditions. Indeed, images under very different lighting conditions should become more similar after histogram equalization, as their intensities will have been spread in the same range of values. Such result is especially useful for the choice of the gradient threshold explained in the following section.

4.3 Gradient thresholding

This step is responsible for obtaining the set of high gradient pixels for which disparities will be interpolated and output.

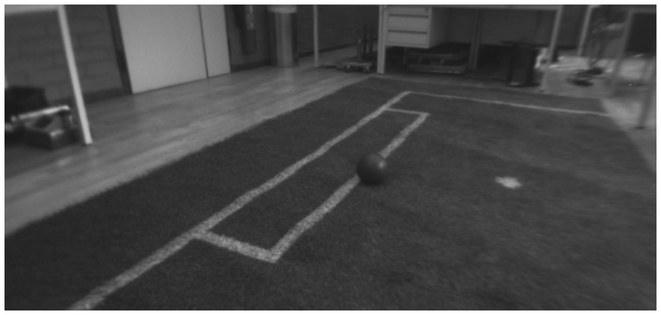
To obtain this set of points, the Laplacian of the image has first to be calculated. The Laplacian is given by the following formula

$$\begin{aligned} Laplacian(I) &= \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \\ &\simeq [I(x+1,y) + I(x-1,y) - 2I(x,y)] \\ &\quad + [I(x,y+1) + I(x,y-1) - 2I(x,y)] \end{aligned} \tag{4.3}$$

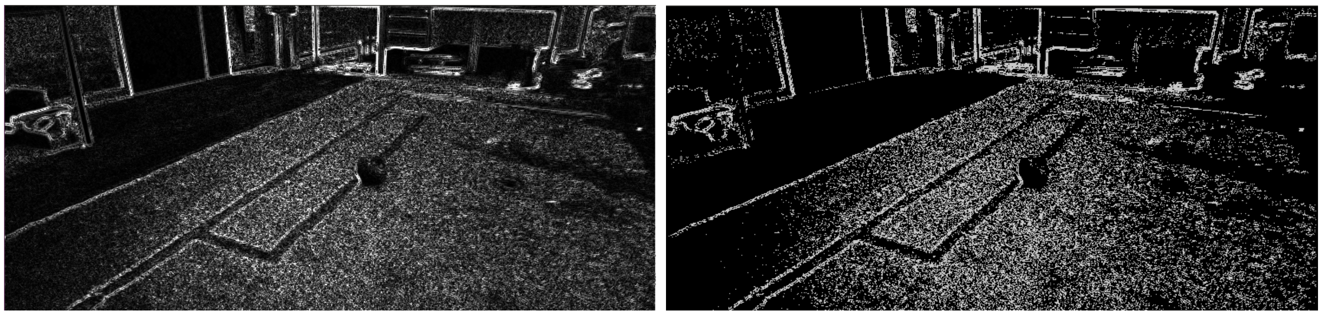
Which translates into the following kernel to apply to the image:

$$kernel = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.4}$$

The set of high gradient pixels is then simply obtained by keeping the ones having their Laplacian above a fixed threshold. An example of such selection is illustrated in Figure 4.3.



(a) Reference image



(b) Gradient of the reference image (c) Set of high gradient pixels

Figure 4.3 – Example of high gradient pixels selection

Note that the gradient maxima will be greatly influenced by the lighting of the scene. Choosing a too high gradient threshold could lead to a very small, or even empty, set of points in low light conditions, but choosing a low threshold could lead to an oversized set of points in strong light conditions. As the selected gradient points will provide the pixels constituting the semi-dense disparity map, varying light conditions can thus lead to very different densities for a fixed threshold value. This is where the histogram equalization comes really useful. As images under different lighting conditions will look more similar after that process, the size and distribution of the set of hight gradient points should be less affected by the lighting anymore.

4.4 Image filtering

This step is responsible for smoothing the image, which can enhance the matching of pixels, especially in the presence of noise. The smoothing technique employed in this algorithm is to apply a Gaussian blur on the images, which act as a low-pass filter, attenuating high frequency signals.

The choice of Gaussian blurring was made because testing revealed that it was giving better results than other smoothing techniques like simple averaging, median blurring, or bilateral filtering.

4.5 Census transform

The census transform [22] is a non-parametric transform that can be used in this context to evaluate the similarity between pixels. This transform maps the local neighborhood surrounding a pixel P to a bit string representing the set of neighboring pixels whose intensity is less than the one of P. An example is shown in Figure 4.4. The pixels inside a window centered on the considered pixel P are examined one after another. A 0 is output if the intensity is lower than the intensity of P, and a 1 is output otherwise.

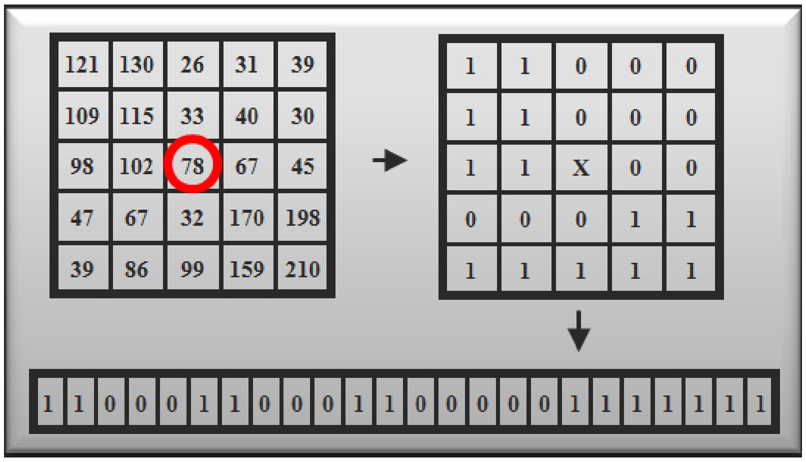


Figure 4.4 – Example of the census transform applied to one pixel (source: [23])

The similarity measure, or matching cost, that can then be used on the basis of the census transform is the Hamming distance, i.e, the number of bits that differ between two bit strings resulting from the census transform. Obviously, the less bits differs between the census transforms of two pixels, the more similar are these pixels. Figure 4.5 shows an example of Hamming distance computation.

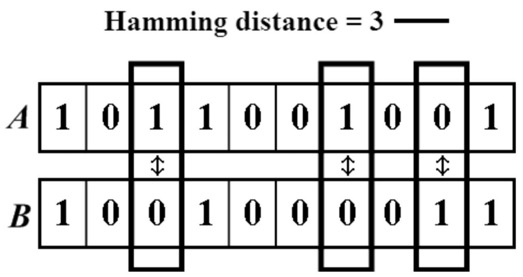


Figure 4.5 – Example of the hamming distance between two bits words A and B (source: [24])

The use of such a similarity measure has been proven to significantly improve the stereo matching robustness [25], and this is why the census transform is applied on the input stereo images in this algorithm.

4.6 Sparse stereo matching

This part of the algorithm is responsible for generating the first set of support points for the triangular mesh that will approximate the disparity in the reference image. The method employed to generate this set of support point is the one from [20]. It is divided in two main steps. The first step will detect feature points in the two input stereo images. Then, a matching algorithm will be applied on these sets of features to establish those that corresponds to each other in the reference and target images.

4.6.1 Feature detection

The feature detection is based on the FAST algorithm [26], due to its high speed of execution. This algorithm compares the image intensity at a given pixel to the intensities of the pixels on a corresponding circumcircle. A feature is detected if a contiguous arc is found that is significantly brighter or darker than the center pixel, according to a threshold value t_c. To minimize the number of comparison operations, this arc is detected with the help of a decision tree, generated by a machine learning algorithm. Additionally, a non-maximum suppression step is performed, which consists in removing the feature points that have an adjacent one with a better score.

However, FAST tends to detect many features in high contrast areas, but only a few in areas with less contrast. This can thus lead to situations where most of the features are clustered in a relatively small area, as on the example presented in Figure 4.6a. This behavior is undesirable in our application as it can cause objects of the scene to be missed if they do not provide sufficient contrast. This is why [26] proposed a modified version of the FAST detector, called *extended FAST*, or *exFAST*, that makes this effect less severe.

This extended method first runs the original FAST detector with a low constant threshold t_c and without non-maximum suppression. This leads to the detection of many features, as shown in Figure 4.6b.

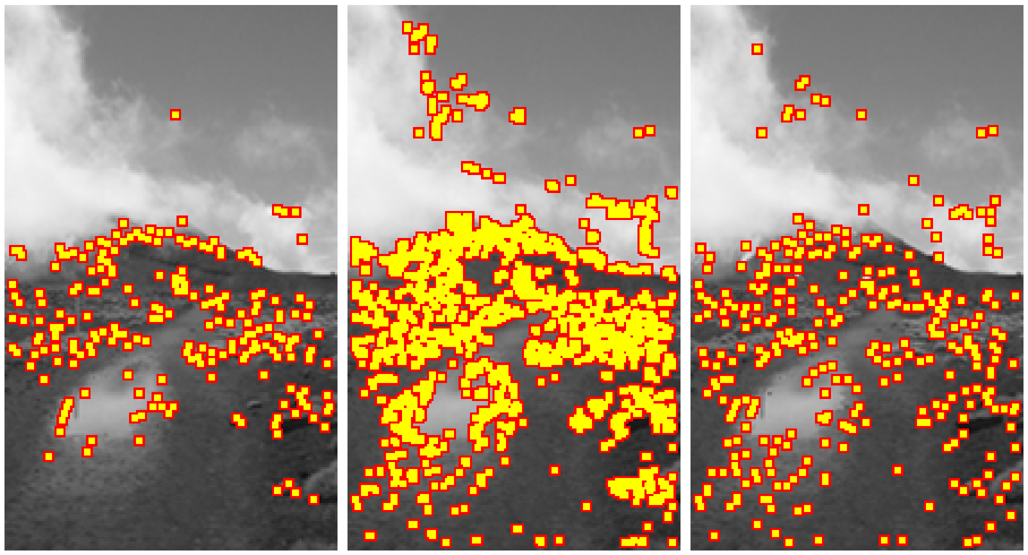

(a) FAST (b) First stage (c) exFAST

Figure 4.6 – Examples of feature detection results (source: [20])

For each of these detected features, a new adaptive threshold t_a is calculated. This adaptive threshold is defined as the product of the image contrast and adaptivity factor $a > 0$. Rather than using the common root-mean-square contrast as measure, a simplified version based on absolute differences is used, avoiding the computation of a square root. The formula for the threshold computation is given in Equation (4.5), where p is a pixel from the local neighborhood N_i of feature point i, I_p the intensity at p, and $\bar{I}$ the average intensity of all pixels in N_i. The local neighborhood is composed of the 16 pixels on the circle of radius 3 used by FAST.

$$t_a = \frac{a}{|N_i|} \sum_{p \in N_i} |I_p - \bar{I}| \tag{4.5}$$

The FAST detection is then rerun a second time using these new adaptive thresholds, with non-maximum suppression for the reference image only, and the feature points that pass both detection steps are returned by the algorithm. Moreover, during the second pass in the FAST detector, not only the central pixel is considered, but rather the average intensities of the four central pixels, as illustrated in Figure 4.7. This helps prevent noisy pixels to impede the detection of obvious features or cause the detection of false or insignificant features. An example of the results that can be obtained with exFAST is presented in Figure 4.6c.

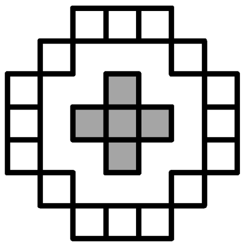
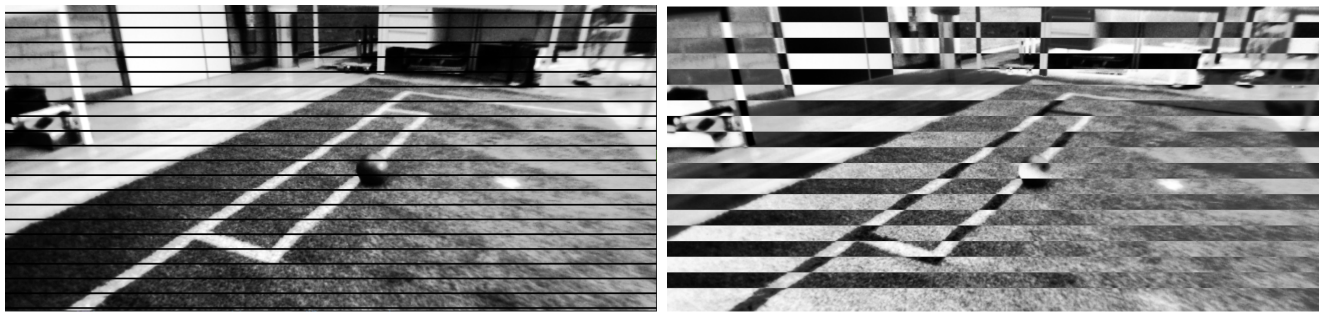


Figure 4.7 – Pixels used for feature detection. Grey pixels in the middle are averaged and compared to the circumcircle. (source: [20])

Finally, notice in Algorithm 1 that the images provided as input to the SparseStereo step have not been filtered, in order to keep all the details and to use them for the detection of features. Moreover, two optional steps that are not part of the original paper are proposed in order to try to increase the number of generated features, or improve their distribution. These steps are mutually exclusive. One of them consist in tracing horizontal black lines on the images before running *exFAST*. This is illustrated in Figure 4.8a. Since *exFAST* act as a corner detector, such lines can increase the number of features detected along vertical lines or contours in the images. The second option is to invert the intensities of the pixels on alternate rows of a certain width, as illustrated in Figure 4.8b. This can also increase the number of corners in the image, and does not hide potential corners behind black lines. However, black lines have the advantage of being more contrasted with the rest of the image, in the contrary of inverted rows in areas where the gray levels are in mid range, and such areas are more likely to be present in the images than black areas. The effect of these preprocessing options is further discussed in Section 6.



(a) Horizontal black lines drawing

(b) Rows inversion

Figure 4.8 – Preprocessing options for feature detection

4.6.2 Stereo matching

As mentioned in the feature detection step, non-maximum suppression is only applied to the reference image, leading thus to a bigger number of detected features in the target image. This way, more candidate points are available for the matching. This part of the algorithm will thus try to match each feature of the reference image with its equivalent in the target image, by looking at the features on the same epipolar line, and within a valid disparity range, comparing their census matching costs, as explained in Section 4.5. The pair of features with the lowest cost is retained as the most likely match.

For filtering-out matches with a high uncertainty, a uniqueness constraint is imposed. The matching cost for a selected feature pair has to be smaller than the cost for the next best match times a uniqueness factor u, between 0 and 1. This relation is expressed in Equation (4.6), where c_{min} is the cost of the best match, and C is the set of matching costs for all feature pairs.

$$c_{min} < u \min\left(C \setminus \{c_{min}\}\right) \tag{4.6}$$

In other words, this uniqueness constraint allows to reject the features that have several good candidates for matching, with very little difference between their costs. Indeed, as the scores are tight, the real match might not be the one with the smallest cost.

Finally, a dense left-right consistency check is also performed to increase again the confidence in the returned matches. To do so, the matched feature in the target image is considered. The matching costs between this feature and all the pixels (not the features) on the epipolar line of the reference image, in the valid disparity range, are computed. If any of these computed costs are below uc_{min}, the match is rejected. Optionally, to gain some execution time, a step value x can be defined. Instead of going through every pixel in the valid disparity range on the epipolar line, only one pixel out of x is thus considered.

4.7 Delaunay triangulation

A Delaunay triangulation for a given set S of discrete points in a plane is a triangulation $G(S)$ such that no point in S is inside the circumscribed circle of any triangle in $G(S)$ (see Fig. 4.9). Delaunay triangulations maximize the minimum angle of all the angles of the triangles. They thus tend to avoid sliver triangles. This triangulation was named after Boris Delaunay for his work on this topic from 1934 [27],[28].

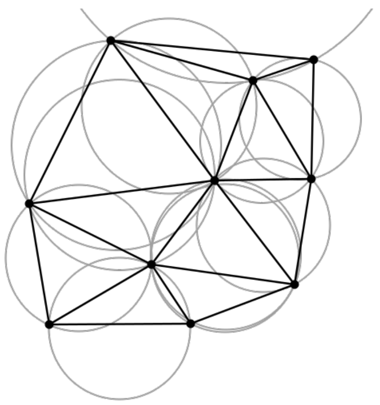


Figure 4.9 – A Delaunay triangulation in the plane with apparent circumscribed circles (source: [27])

Such a triangulation is often used to build meshes for space-discretized solvers such as the finite element method. Similarly, in this work, it is used to model a piece-wise planar surface representing the disparity in the input stereo images. The mesh is build from the set of support pixels returned by the SparseStereo step, or the SupportResampling step, and will then be used to interpolate disparities at pixels that are inside the triangles of the mesh. An example of such meshes was shown in Figure 4.1.

The cost of the fastest algorithm for computing a Delaunay triangulation is $O(n\,log\,n)$, with n the number of points. There exist thus many implementations that can generate a Delaunay Triangulation efficiently. However, they were not studied in this thesis. Section 5 will explain that a particular library was chosen to compute these triangulations, for which the source code is not accessible, and the implemented algorithms not provided. Nevertheless, open source implementations exists if needed.

4.8 Disparity interpolation

Each triangle of the mesh previously obtained represents a plane in a 3D space. The disparity of pixels inside a triangle can thus be interpolated using this plane. A 3D plane can be described by the following equation

$$a\,x + b\,y + c\,z + d = 0 \tag{4.7}$$

In the 3D space considered here, x and y are the pixel coordinates u and v, and the z coordinate is the disparity, called *disp*. Using these notations, Equation (4.7) can be rewritten, to emphasize the disparity, as

$$a'u + b'v + c' = disp \tag{4.8}$$

Once the parameters (a', b', c') are known, the disparity of any pixel belonging to the considered triangle can be calculated using Equation (4.8). To obtain these three unknown parameters, the three vertices $(u, v, disp)$ of the triangle are inserted in Equation (4.8) to form a linear system of three equations, that has then to be solved.

What actually happens in this step of the algorithm is that all the high gradient pixels are considered one after another, the triangle in which they lie is identified, and their disparity is interpolated. If this is the first time that the triangle is encountered, new plane parameters are calculated and then stored in a lookup table. Otherwise, the interpolation uses the parameters from the lookup table. Algorithm 2 details this process.

Algorithm 2: DisparityInterpolation

```
Input  : $(G, \Omega)$: Delaunay triangulation of the support points and set of high gradient
         pixels
Output: $D_{it}$: Interpolated disparities
   // Lookup table for plane parameters
1  $map$<Triangle, Parameters> $\leftarrow \emptyset$
2  for $(u, v) \in \Omega$ do
       // Get the triangle in which $(u, v)$ is located
3      $t \leftarrow$ GetTriangle$(G, u, v)$
4      if $t \in map$ then
5          $a', b', c' \leftarrow map[t]$
6      else
7          $a', b', c' \leftarrow$ ComputePlaneParameters$(t)$
8          $map[t] \leftarrow (a', b', c')$
9      end
       // Interpolate disparity
10     $D_{it}(u, v) \leftarrow a'u + b'v + c'$
11 end
```

4.9 Cost evaluation

The cost evaluation step is there to make sure that the disparities interpolated in the previous step are sufficiently accurate. To do so, a cost matrix is build, which contains the matching costs for each high gradient pixel. These matching costs are obtained by looking at the corresponding pixels in the two stereo images, according to the interpolated disparity, and computing the hamming distance as explained in Section 4.5. Algorithm 3 details this process.

The computed matching cost matrix will then be used in the next step to decide which disparities can be stored in the final disparity map, which points could be added to the support points of the mesh, and which points are very bad matches.

Algorithm 3: CostEvaluation

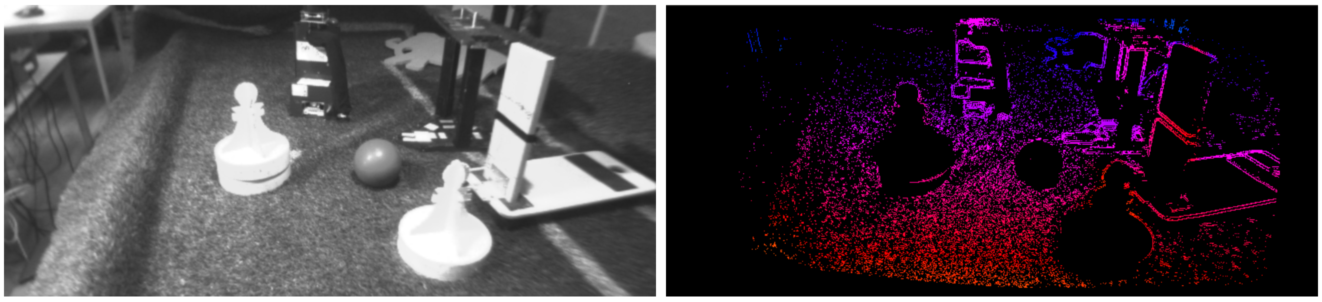
```
Input  : $(J_l, J_r, D_{it}, \Omega)$: Census transforms of the stereo images, interpolated disparities,
         and set of high gradient pixels
Output: $C_{it}$: Matching cost corresponding to $D_{it}$
1 for $(u, v) \in \Omega$ do
      // Interpolated disparity at $(u, v)$
2     $d \leftarrow D_{it}(u, v)$
      // Census-based matching cost
3     $C_{it}(u, v) \leftarrow$ HammingDistance$(J_l(u, v), J_r(u - d, v))$
4 end
```

4.10 Disparity refinement

This step of the algorithm is responsible for two tasks. The first one is to update the final disparity map with the interpolated disparities that have a better matching cost. The other is to identify the set of pixels that could be used as new support points in the triangular mesh.

The update of the final disparity map is done by checking if the matching costs of the interpolated disparities are below a certain threshold. When this is the case, the considered pixel is updated in the final disparity map. Moreover, this disparity refinement step is part of an iterative process, and can thus be repeated. For this reason, not only the interpolated disparity is recorded, but also the associated matching cost. That way, in the next iteration of the main algorithm, when new disparity values will have been interpolated, only the ones having a better matching cost than previously will be updated in the final disparity map. An example of final disparity map that can be output by the algorithm is presented in Figure 4.10b.

The second task of this step is performed by diving the image into cells, and for each of these cells, identifying the pixel with the best matching cost, and the one with the worst matching cost. Additionally, these pixels must satisfy another constraint. The matching cost of the best pixel also has to be under a certain threshold, and the matching cost of the worst pixel has to be above a certain threshold. These thresholds are there to make sure that the considered pixels have either a really a good, or a really bad match.



(a) Reference image (b) Disparity map

Figure 4.10 – Example of a semi-dense disparity map that can be output by the algorithm.

Algorithm 4 details the process of this disparity refinement step.

4.11 Support re-sampling

This step is responsible for adding new points to the set of support points of the triangular mesh. The pixels identified in the previous step, having the best matching costs, are simply added as new support points. The pixels that were identified to have the worst matching costs are strong indicators of occluding edges, and sharp discontinuities in depth, making them viable candidates for dense epipolar stereo matching. As [1] does not provide any details about the way the epipolar search was performed, it was thus assumed that the census transforms would be used, and the matching costs aggregated over a squared support window along the epipolar line, to be able to select the most similar pixel in the target image, since it is the common practice with local stereo matching methods (see Section 2.5). Moreover, to make sure that a valid match will be returned, a bidirectional matching is performed. This means that first the best match m_{tar} for the reference pixel p_{ref} is searched in the target image. Then, the best match m_{ref} for the target pixel m_{tar} is searched in the reference image. If m_{ref} and p_{ref} are the same, or at least very close to each other, the match is considered valid.

Algorithm 5 describe the process of the support re-sampling step.

Algorithm 4: DisparityRefinement

```
Input  : $(D_{it}, C_{it}, D_f, C_f, \Omega, sz_{occ})$: Interpolated Disparities, associated matching costs,
         final disparities, final costs, set of high gradient pixels and cell size of the
         refinement grid
Output: $C_g, C_b$: Costs associated with pixels of high confidence matches, and with invalid
         disparities
1  $H' \leftarrow \frac{H}{sz_{occ}}, \quad W' \leftarrow \frac{W}{sz_{occ}}$
   // $C_g$: cost matrix of confident supports: $(u, v, d, cost)$
2  $C_g \leftarrow [0, 0, 0, t_{lo}]_{[H' \times W']}$
   // $C_b$: cost matrix of invalid matches: $(u, v, cost)$
3  $C_b \leftarrow [0, 0, t_{hi}]_{[H' \times W']}$
4  for $(u, v) \in \Omega$ do
       // Establish the coordinates of the cell in which $(u, v)$ is located
5      $u' \leftarrow \frac{u}{sz_{occ}}, \quad v' \leftarrow \frac{v}{sz_{occ}}$
       // If matching cost is lower than previous best final cost
6      if $C_{it}(u, v) < C_f(u, v)$ then
7          $D_f(u, v) \leftarrow D_{it}$
8          $C_f(u, v) \leftarrow C_{it}$
9      end
       // If matching cost is lower than previous best valid cost
10     if $C_{it}(u, v) < t_{lo}$ and $C_{it}(u, v) < C_g(u', v', 4^*)$ then
11         $C_g(u', v') \leftarrow (u, v, D_{it}(u, v), C_{it}(u, v))$
12     end
       // If matching cost is higher than previous worst invalid cost
13     if $C_{it}(u, v) > t_{hi}$ and $C_{it}(u, v) > C_b(u', v', 3^*)$ then
14         $C_b(u', v') \leftarrow (u, v, C_{it}(u, v))$
15     end
16 end
```

*matrices are 1-indexed

Algorithm 5: SupportResampling

```
Input  : $(C_b, C_g, S_{it}, J_l, J_r)$: Matching costs for confident and invalid matches, current
         support points, and census transforms of the input images
Output: $S_{it+1}$: New support points for tessellation

1  $S_{it+1} \leftarrow S_{it}$
   // Add confident pixels to the support points
2  for $(u, v, d, cost) \in C_g$ do
3      if $cost < t_{lo}$ then
4          $S_{it+1} \leftarrow \{S_{it+1}, (u, v, d)\}$
5      end
6  end
   // Re-estimate disparities of invalid matches via dense epipolar search
7  for $(u, v, cost) \in C_b$ do
8      if $cost > t_{hi}$ then
9          $disparity \leftarrow$ DenseEpipolarStereoMatching$(J_l, J_r, (u, v))$
10         if valid disparity has been found then
11             $S_{it+1} \leftarrow \{S_{it+1}, (u, v, d)\}$
12         end
13     end
14 end
```

5. Implementation details

This section provide some information about the implementation of the stereo vision algorithm. It motivates the choice of the programming environment, enumerates the different libraries used, presents the structure of the source code, and explains how it can be used.

5.1 Programming environment

The main parts of the algorithm were implemented in C++. This choice was made for several reasons. First of all, an implementation of the SparseStereo step of the algorithm was available and already implemented in C++. Another reason is that the *OpenCV* library has a C++ implementation, and this library provides many useful built-in functions for image processing. This library also exists in Python, but as the speed of execution is one of the concerns of this thesis, a compiled language like C++ was preferable. Matlab® also provides a nice image processing toolbox, but it is not a compiled language either, and moreover, it is not free and not suited to be embedded in the robot platform.

Nonetheless, one part of the global stereo vision algorithm was still performed using Matlab®. The stereo calibration app of Matlab® was used to compute the parameters of the cameras. As the stereo rig calibration is an offline process, it does not affect the speed of the overall stereo vision algorithm. An implementation of this calibration process should obviously be provided at some point to run on the robot platform, but the focus of this thesis was put on the implementation of the stereo matching part, and it was therefore not an issue to use Matlab® as a tool. Now, it should be noted that the *OpenCV* library provides also some functions to perform the calibration of the cameras. However, the checkerboard corner detection implemented in this library was failing with almost all the pictures provided, while the technique implemented in Matlab® was almost always succeeding with the same pictures. This is why Matlab® was chosen to perform the calibration step of the stereo vision algorithm.

All the libraries that were used in the C++ implementation of the algorithm are listed below:

- **OpenCV:** an open source library for computer vision. It was used, among others, to rectify the input images, to equalize their histograms, to compute their gradient, and to filter them.
- **exFAST and Sparse Stereo Matcher [20]:** a library used to generate the first set of support points. It implements an optimized 5×5 census transform using SSE[1].

[1]Streaming SIMD Extensions

- **Fade2D [29]:** a library for efficient computing and handling of Delaunay triangulations. It is not open source but offers a student license free of charge for personal non-commercial scientific research.
- **Eigen:** a template library for linear algebra. It is used to solve the linear systems giving the plane parameters needed for the interpolation of the disparities.
- **MRPT:** the Mobile Robot Programming Toolkit. An open source library from which the RANSAC algorithm was used to test the results of the stereo matching algorithm (see Section 6.3).

5.2 Structure of the source code

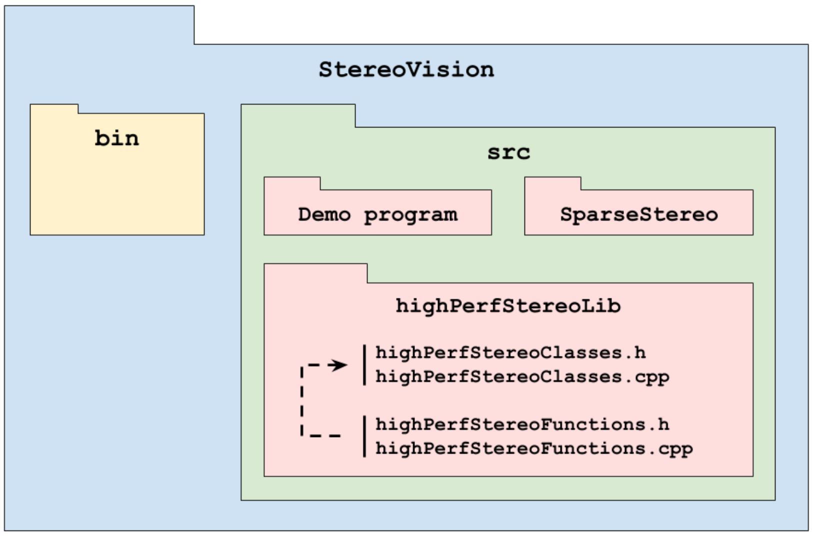


Figure 5.1 – Source code structure

A scheme of the file structure is presented in Figure 5.1. The main repository is named `StereoVision`. This repository contains two subdirectories, `bin` and `src`. The `bin` directory is the location where the executable will be generated. It also contains a folder with a stereo pair of images and the associated calibration file. The `src` directory contains the source code.

The `src` directory is itself divided into three folders containing the *exFAST and sparse stereo matcher* library, the developed stereo matching algorithm, and a demo program.

The implementation of the stereo matching algorithm is split into two modules. The `highPerfStereoFunctions` module implements all the steps of the proposed stereo matching algorithm. This module required the implementation of a series of classes, which are regrouped in a second module, `highPerfStereoClasses`.

The code can be compiled using *CMake*, as explained in the following section.

5.3 How to use

5.3.1 Running the example program

Prior to run the software provided with this thesis, the libraries mentioned in Section 5.1 need to be installed, except for the *exFAST and Sparse Stereo Matcher*, whose files are supplied in the source code, and except for the *MRPT* library, as it was only used for testing purposes.

The provided demo program can then be build with *CMake*, running the following commands from inside the `StereoVision` directory:

```
cmake -G "Unix Makefiles"
make
```

The executable will be generated in the `\bin` directory. This program rectifies the stereo pair provided in the `\bin\test_images` folder, compute its disparity map, and generate a text file containing the reprojected 3D points according to the disparities and the calibration parameters. The points are encoded with their RGB values in the original image, under the format `"x y z r g b"`. Such a file can be read using a 3D rendering software like MeshLab. Along its execution, the program also displays some images illustrating the process of the algorithm. When such an image is displayed, the program is actually paused and the image window needs to be closed before the program can resume.

5.3.2 Using the stereo matching library

To compute the semi-dense disparity map of a rectified gray-scale stereo pair, all the fields of a **StereoParameters** object have to be set, and the following function, from `highPerfStereoFunctions.h`, can simply be called:

```
highPerfStereo(leftImg, rightImg, stereoParameters,
               disparityMap, highGradPoints);
```

The meaning and influence of all the stereo matching parameters are presented in Table 5.1.

Stereo matching	
double uniqueness	The uniqueness factor for the sparse stereo matching (see Section 4.6.2). Range [0,1]. The smaller is its value, the bigger is the confidence in the generated matches, which has thus the side effect of decreasing their number.
int maxDisp	The maximal disparity value. It limits the size of the epipolar search.

int minDisp	The minimal disparity value. It limits the size of the epipolar search.
int leftRightStep	The step value, in pixel, for the left-right consistency check of the sparse stereo matching (see Section 4.6.2).
int costAggrWindowSize	The size of the squared support window used to aggregate the matching costs. Increasing this value decreases the chances of generating an erroneous match, but slows down the execution time. The value of this parameter has no effect if the number of iteration of the overall algorithm is not superior to 1.
uchar gradThreshold	The threshold value above which the pixels are considered as high gradient points. Decreasing this value can thus increase the number of pixels for which a disparity value will be output, but also increase the number of computations. Range [0,255].
char tLow	The upper bound of the matching cost values for which a pixel is considered as a confident support point.
char tHigh	The lower bound of the matching cost values for which a pixel is considered as a bad match, and for which the disparity can thus not be output on the final disparity map.
int nIters	The number of iterations of the global algorithm. Increasing this value allow to refine the triangular mesh but also increases the computation time. Note that the refinement will only happen in textured areas.
bool applyBlur	Filters the input images with Gaussian blur if set to *true.* This reduces the effect of the noise in the images.
bool applyHistEqualization	Equalizes the histogram of the input images if set to *true.* This enhances the contrast, helps the feature detection, ensures a similar behavior of the algorithm over different lighting conditions, and facilitates the choice of the gradient threshold.
int blurSize	Size of the Gaussian blur kernel. The right compromise need to be found for this parameter. Setting it too high will decrease the performance of the sparse stereo matching.

int rejectionMargin	The width of the frame in which the support points returned by the sparse stereo matching won't be considered to be added to the triangular mesh. It is useful in the case of an important radial distortion in the original images. In that situation, the rectification usually leads to a poor quality on the edges of the image. Consequently, invalid matches are often generated in those areas.
unsigned int occGridSize	Initial size of the grid cells in which the best and worst matches are selected during the refinement step of the algorithm (see Section 4.10). This value is divided by two at each new iteration of the algorithm.
Feature detection	
double adaptivity	The adaptivity factor of the *exFAST* feature detection (see Section 4.6.1. Increase this parameter reduces the number of detected features.
int minThreshold	The intensity threshold beyond which a pixel is considered brighter or darker in the FAST algorithm. Decreasing this value increases the number of detected features.
bool traceLines	If set to *true*, traces a series of black line in the images submitted to the feature detection step. This can increase the number of support points along vertical edges in the image.
int nbLines	The number of black lines to trace if *traceLines* is set to *true.*
int lineSize	The width of the black lines.
bool invertRows	If set to *true*, inverts the intensities of alternate rows in the images submitted to the feature detection step. This can increase the number of support points along vertical edges in the image.
int nbRows	The number of rows the image needs to be divided in.
Miscellaneous	
bool recordFullDisp	If set to *true*, records an image of the dense disparity map obtained by interpolating the disparity for every pixel.
bool showImages	If set to *true*, display images illustrating the process of the algorithm during its execution.

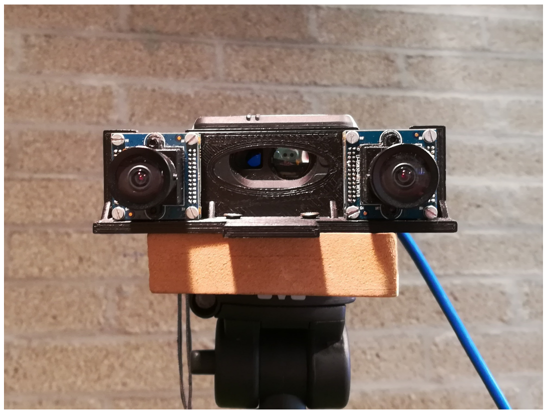
int colorMapSliding	An offset added to the hue of the HSL color space that is used to display the levels of depth in the disparity maps. This offset thus control at which color starts the lowest depth level.

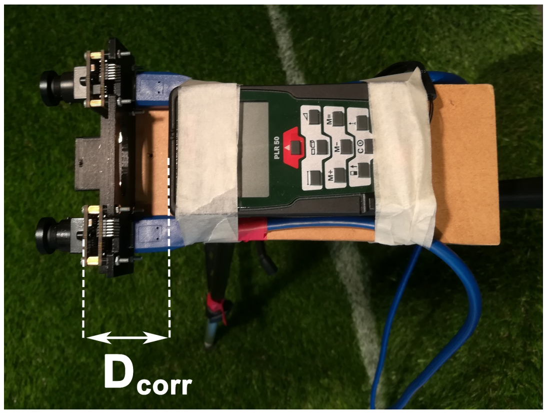
Table 5.1 – Stereo matching parameters and their meaning

6. Evaluation - Algorithm results

The setup that was build for testing the stereo vision algorithm developed in this work is presented in Figure 6.1. Two cameras *LI-USB30-M021X* from *Leopard Imaging* are mounted on a 3D-printed support in ABS, and equipped with wide-angle lenses, giving a focal length of $2.1mm$. The resolution of the taken pictures was of 800×460 pixels. The baseline between the cameras was fixed at $7cm$, which is the biggest distance that the design of the head of the robot can accommodate. This choice was made because the bigger the baseline is, the better the accuracy will be for the depth measurements of farther objects in the scene.

(a) Adjustable tripod


(b) Frontal view of the stereo cameras



(c) Top view of the stereo cameras

Figure 6.1 – Stereo cameras setup

The support of the cameras was then fixed at the edge of a wooden board, itself attached on top of an adjustable tripod. Behind the support of the cameras, a laser telemeter *PLR-50* from *Bosch* was positioned. A hole in the ABS support allows the laser to go through and measure one depth point. The depth measurements can be expressed relatively to the camera sensor plane of the reference camera by subtracting the distance D_{corr} (see Fig. 6.1c) from the initial measurements. Note however that it was not easy to accurately measure D_{corr}, and that the telemeter might also not be perfectly aligned with the camera optical axis, leading thus to a static error of a few millimeters, plus the error introduced by the telemeter itself, which is of $\pm 2mm + 0.05mm/m$.

As no system allowing to compute the ground truth of the disparity map could be used, the setup just described was used to carry out two kinds of tests evaluating the performance of the implemented stereo vision algorithm in some practical applications. The first test was specifically designed with the context of the *Robocup soccer contest* in mind, but its results are still relevant for other kind of applications. This test consists in trying to locate the depth at which a ball is located from the camera. As this test only focus on one particular object of the scene, a second test was performed to have a better idea of the 3D rendering over a bigger part of the image. To do so, this second test tries to estimate the inclination at which the camera is observing the floor, which is a valuable data that can also be used in the context of the *Robocup soccer contest*. Both of these tests allow to get a qualitative as well as a quantitative idea of the capabilities of the proposed stereo vision algorithm.

Before going into the details of these two tests, some comments about the results of the calibration and rectification steps will be provided, and lastly, the speed performance of the implemented stereo vision algorithm will be evaluated.

6.1 Calibration and rectification evaluation

To obtain good calibration results, the checkerboard pattern should contain as much corners as possible. However, the more corners, the smaller the squares would be, and they could thus be located less accurately on the images. A compromise had thus to be made on the size of the squares of the checkerboard pattern. In this case, it was decided that the pattern would be printed on an A4 sheet, for simplicity of manipulation, containing 13×19 corners, with a square size of $15.495mm$. The pattern was then glued on a wooden plane surface. If the surface is not perfectly plane, or the pattern waves due to humidity or the glue, the accuracy of the obtained camera parameters will be impacted.

To evaluate the accuracy of the calibration results, the reprojection errors can be inspected. A reprojection error is the distance between a pattern cornerpoint detected in a calibration image, and the corresponding world point projected into the same image using the estimated camera parameters. The best result that was achieved is a mean reprojection error of 0.18 pixels, over 24 image pairs of the checkerboard under different orientations, trying to cover most of the image plane. These errors are presented in Figure 6.2. The rectified images presented in Figure 6.3 also give a qualitative insight of the calibration accuracy, showing that the distortion seems well corrected, and the epipolar lines well aligned.

It is nevertheless difficult to establish precisely the exact procedure to repeat this accuracy. What can be said thought is that it was achieved indoor, under a strong spotlight, and that taking

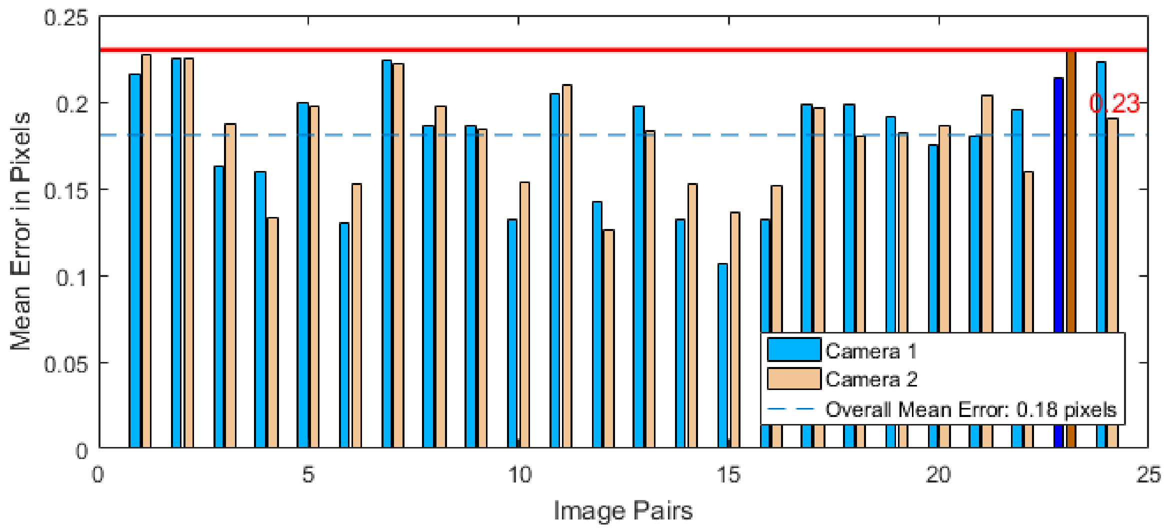
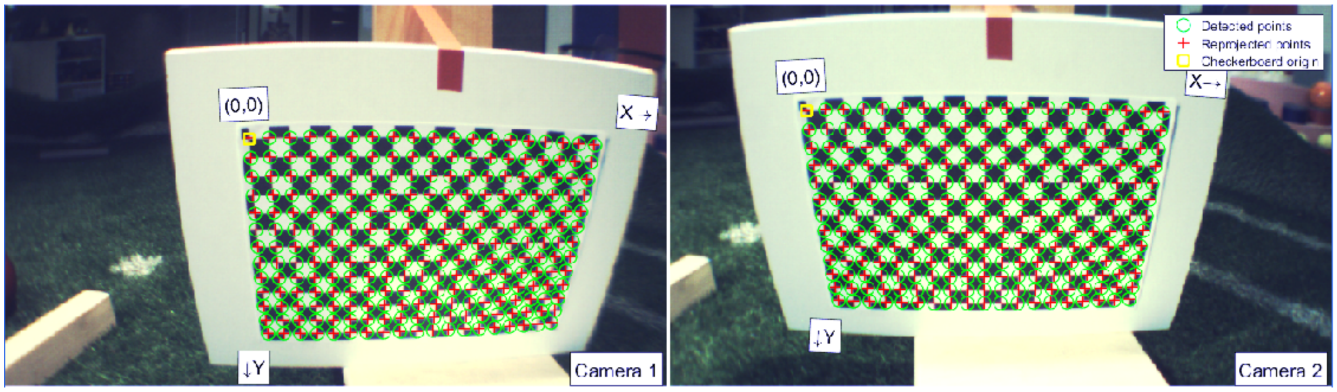
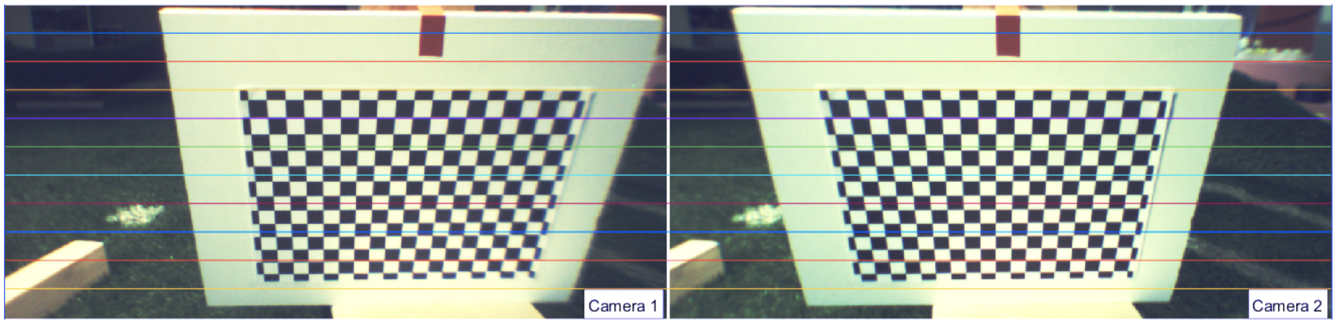


Figure 6.2 – Reprojection errors for the calibration results obtained with a set of 24 stereo image pairs of a checkerboard pattern



(a) Stereo pair before rectification



(b) Stereo pair after rectification

Figure 6.3 – Rectification of a stereo pair using the calibration results with a mean reprojection error of 0.18 pixels

pictures of similar orientations of the checkerboard should give similar results. The complete set of pictures giving the best calibration results is provided in Appendix A.

As seen in Figure 2.8 in Section 2.2, discrete disparities lead to discrete depths. Every point between those depths could thus be estimated with a maximum absolute error equal to the distance between two consecutive depths, assuming that the disparity is correctly estimated. Using the calibration parameters, the theoretical maximum error as a function of the depth can be computed. An instance of this curve is shown in Figure 6.4. This curve shows that, using this stereo setup, accurate results on the estimated depth should not be expected beyond a few meters.

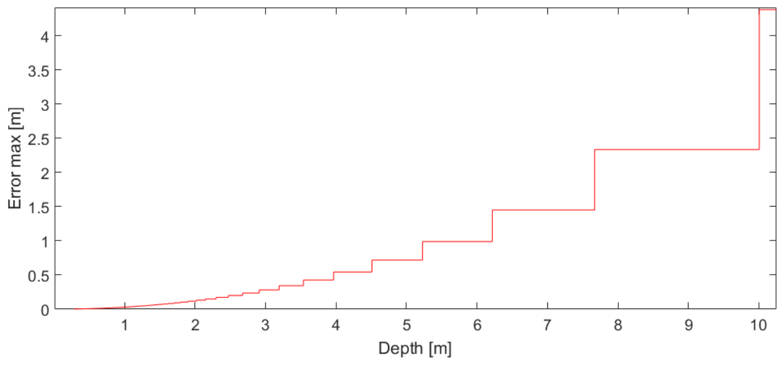


Figure 6.4 – Theoretical maximal error on the depth estimation calculated using the calibration results with a mean reprojection error of 0.18 pixels

Although the accuracy obtained was satisfactory, it might be possible to obtain better results for the transformation between the two cameras if they were better aligned from the start. Indeed, 3D-printed parts in ABS are susceptible to curl a little bit during the printing process, and that is what happened here with the support of the cameras. Another reason why the cameras were badly aligned could be that very small 3D-printed spacers were used between the cameras and the support, and these spacers might not have exactly the same length, or have been compressed inequitably during the mounting of the cameras.

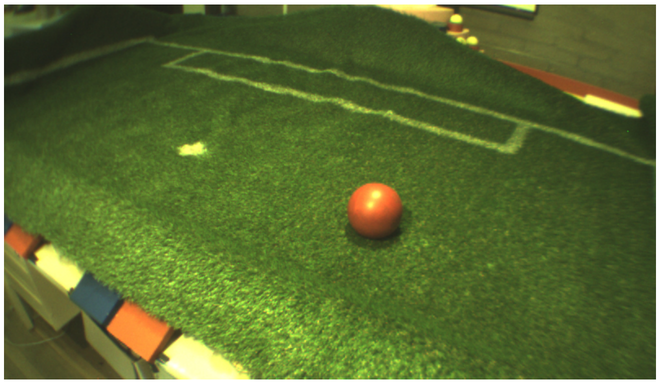
Moreover, the biggest issue that was encountered is the fact that there was some play between the cameras and the support in ABS. Even though this play was hardly noticeable for the human eyes, even a careful manipulation of the stereo rig could completely invalidate the calibration. Indeed, to give an example, a rotation of only $0.008 rads$ of one of the cameras around the baseline axis was leading to a vertical displacement of the image of about 6 pixels, which is enough to make the stereo matching impossible. Despite this problem, usable test sets of images were still obtained, but it is clear that the mounting system of the cameras inside the robot's head should be carefully designed to be as rigid as possible and to prevent the cameras from rotating in any direction. If it was not the case, it is likely that the robot would need a recalibration of its stereo system after every physical move. Regarding this issue, it might thus also be interesting to look at online calibration techniques that could be run at regular intervals to adjust the calibration over time without the need of the checkerboard pattern.

6.2 Ball localization

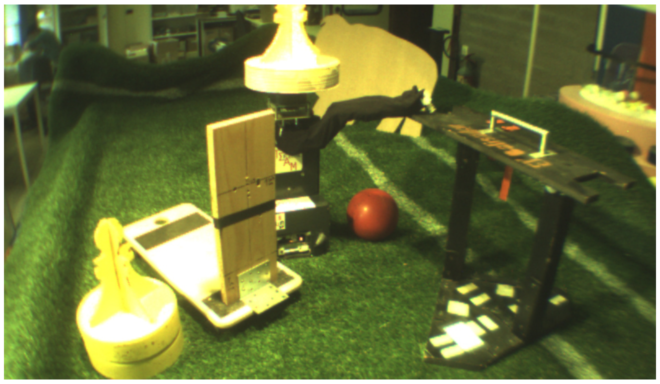
This test checks the accuracy at which the depth of a ball in the scene can be estimated using the proposed stereo vision algorithm.

(a) grass field under fluorescent light series

(b) wooden floor under fluorescent light series



(c) grass field under spotlight series

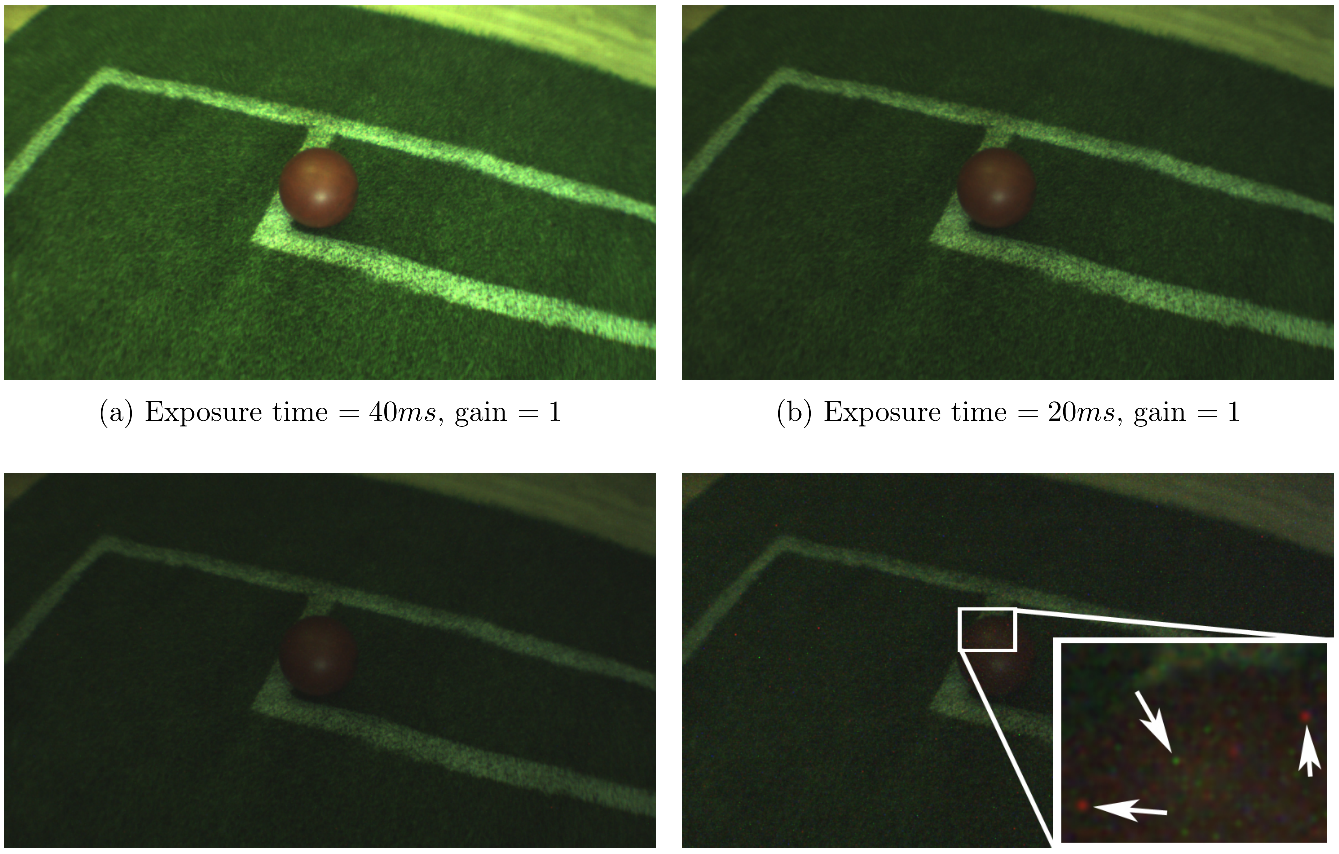


(d) complexe scene under spotlight series

Figure 6.5 – Examples of pictures used for the ball localization test. Each of these four pictures was taken with an exposure time of 20ms and a unit gain.

To do so, four series of stereo image pairs containing a small plastic ball at various distances were studied. Two of these series of pictures were taken under a strong spotlight, and the other two were taken under the usual fluorescent lighting of the laboratory where the test pictures were generated. These different cases were studied because the amount of light that is received by the sensor of the cameras will affect the quality of the image, and fluorescent lamps are less powerful. Moreover, although it is not visible for the human eye, the fluorescent lighting flickers if it is observed at high frequency. Short exposure times could thus lead to very different intensities between pairs of images, and thus give very dark and low contrast images. Aside from the difference in lighting , one of the series under fluorescent light was taken on a grass soccer field, and the other on the wooden floor of the lab, in order to study the results of the stereo vision algorithm facing different floor textures. The two series of pictures exposed to the strong spotlight were also taken on the grass soccer field, the difference being that, in one of them, many obstacles were disposed around the ball. This allows to study the robustness of the algorithm to scenes overloaded with more complex objects. Examples of pictures extracted from these four test sets are provided in Figure 6.5.

Moreover, as mentioned in the introduction, the robot will most of the time be moving while taking pictures, requiring thus a short exposure time to avoid getting blurred images. However, a short exposure time implies that the images will be more noisy, which could affect the efficiency of


(a) Exposure time = $40ms$, gain = 1 (b) Exposure time = $20ms$, gain = 1

(c) Exposure time = $5ms$, gain = 20 (d) Exposure time = $1ms$, gain = 50

Figure 6.6 – Effect of the variation of the exposure time and the gain on test pictures taken under fluorescent lighting . A zoom in (d) allows to highlight the noise appearing due to a low exposure time. Some very noisy pixels are pointed out by an arrow.

the stereo matching algorithm. This is why each of the stereo pairs was also taken using various combinations of exposure time and gain, in order to see what could be achieved in these different cases. Figure 6.6 illustrates the effect of a variation of these parameters.

For each of the stereo image pairs, the telemeter was used to record the depth of the center of the ball. After the images were rectified, a squared ROI[1] surrounding the image of the ball was encoded manually, using a self-made software. For each of these ROI, a small algorithm then selected the set of pixels belonging to the disk inscribed in the ROI, i.e. the pixels belonging to the ball. Using the proposed stereo vision algorithm, a 3D point cloud was generated from this set of pixels. Note that not every one of these pixels were necessarily given a corresponding 3D point, as the output disparity map is semi-dense. The depth of the point closest to the camera was then selected as an approximation of the measured depth of the ball.

To begin with, the images taken with a 20ms exposure time and a unit gain were considered, as they appeared to have a good image quality under the different lighting conditions. The parameters of the stereo matching algorithm that seemed to give the best results over all the image sets were established by manual tuning. It was not possible to implement an automated procedure to

[1]Region of interest

calculate the optimal parameters because the mean error returned by this test is not the only thing to be taken into account. Indeed, this test focus on the depth error of a single point of the image, but the overall quality of the triangular meshes and the disparity maps also needed to be checked visually. Moreover, the mean error of the different test sets needed to be checked separately, to make sure that a right compromise was chosen, so that similar results would be obtained for each test set, avoiding to select parameters values that would be better suited for only some of the test cases.

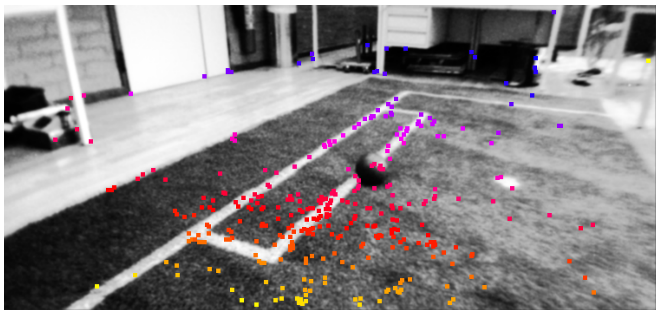
Manual tuning revealed several things. First of all, the number of iteration of the stereo matching algorithm was irrelevant to establish the disparity of the ball in this test. Indeed, the refinement of the mesh only happens in high textured areas, while the ball used in this test has a uniform color. A single iteration usually suffice to obtain a few support points belonging to the contour of the ball, or even to its surface in some cases. This is a positive point as it allow to save some computation time if we are only interested in the depth of simple object, but it does not mean that the iterative refinement process would not be useful in other applications.

It was also revealed that both the preprocessing options for feature detection (line tracing and row inverting) was not improving the accuracy at which the depth of the ball can be estimated. It was however clearly noticed that, using these options, more support points were generated on thin vertical objects like table legs, which could be interesting to better locate the poles of a soccer goal for example.

The parameter values that were chosen by manual tuning are stated in Table 6.1. These parameters were then used to run the stereo matching algorithm on the test images taken with different combinations of exposure time and gain. Results are presented in Figures 6.9 to 6.16. On these figures, each colored point indicates the error obtained on the depth of the ball for a particular stereo pair. Additionally, red curves were drawn, delimiting the area corresponding to the maximum possible error, assuming that the disparities are discrete and correctly estimated.

Figures 6.9 to 6.12 compare the results of the test for exposure times bigger than 20ms. These results show that there is no clear pattern implying that an increase in the exposure time above 20ms would really improve the accuracy. For some points it does, for others it doesn't. Now looking at Figures 6.13 to 6.16, the same kind of conclusion could be drawn, decreasing the exposure time to 5ms does not seem to affect so much the overall accuracy. However, what is not reflected by these graphs is the fact that a lot less confident support points can be generated when the exposure time decreases, and some regions of the disparity maps can also become highly inaccurate because of the appearance of invalid support points. Examples of these situations are presented in Figures 6.7 and 6.8.

As an underlying goal of this thesis is to obtain a method working with the lowest exposure time possible, the previously chosen parameters were tuned to address these issues. Using the second set of parameters indicated in Table 6.1, better looking disparity maps could be obtained, without significantly affecting the accuracy of the ball depth estimation. This accuracy could even be considered improved, as 60.8% of the results have a smaller error than before and the mean absolute error was lowered to 8.13cm against a previous value of 9.15cm. A graphical comparison of the results is provided in Figures 6.17 to 6.20.


(a) Exposure time $= 20ms$, gain $= 1$

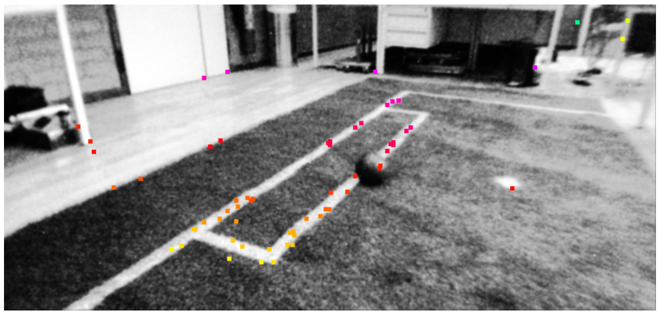
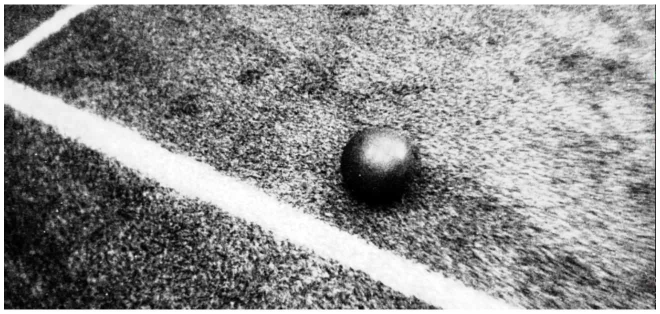

(b) Exposure time $= 5ms$, gain $= 20$

Figure 6.7 – Using a fixed set of parameters, the matching algorithm struggles more to find confident support points in images taken with a lower exposure time


(a) Reference image

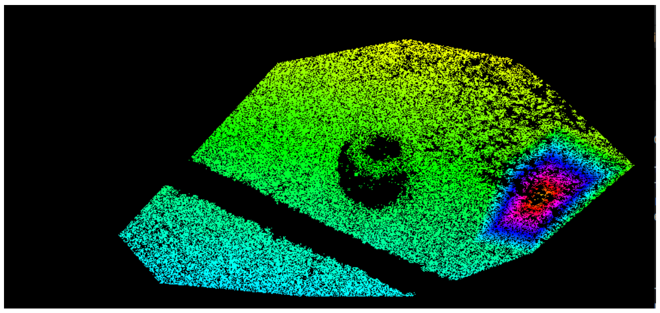

(b) Disparity map

Figure 6.8 – If the parameters are not correctly adapted, the matching algorithm can sometimes generate very bad support points in images taken with a low exposure time. An example of this behavior can be seen in (b), with the red part of the disparity map wrongly implying that an object is nearer to the camera than the floor.

To push the test even further, a new series of trials and errors was performed in order to determine if other parameters of the stereo matching algorithm could be found that would lead to similar results for images taken with an exposure time of 1ms. Unfortunately, no satisfying results could be obtained in this setting. The amount of noise in the images was so important that no combination of parameters was able to prevent the algorithm from generating invalid support points while still delivering a sufficient number of valid ones.

Now that the best accuracy seems to be reached given a low exposure time, it is especially important to discuss the quality of this accuracy. Making the approximation that it follows a linear evolution, the mean absolute error as a function of the depth was estimated at $5cm/m$. However, it can be seen on Figures 6.17 to 6.20 that the error is highly variable, and as a matter of fact, the standard deviation was estimated at $4.25cm/m$. Nonetheless, lots of errors are still located below the maximal theoretical error threshold, which is a good point. Another positive point is that the algorithm seems to work similarly under the different lighting s, thanks to the histogram equalization, and doesn't seem to be disturbed when the ball is surrounded by many other objects at different depth. The separate means and standard deviations for each test set are stated in Table 6.2.

Parameters	Tuned values for exposure = $20ms$ gain = 1	Tuned values for exposure = $5ms$ gain = 20
minimum disparity	36	36
maximum disparity	190	100
uniqueness factor	0.4	0.35
consistency check step	1	4
cost aggr. window size	11	11
gradient threshold	100	150
n_{iter}	1	1
t_{low}	2	2
t_{high}	6	6
histogram equalization	true	true
Gaussian blur	true	true
Gauss. blur kernel size	3	3
adaptivity factor	0.1	0.4
FAST threshold	4	2
black lines tracing	false	false
row inverting	false	false

Table 6.1 – Manually tuned parameters of the stereo matching algorithm in order to obtain the best results as possible

Another observation is that the errors seem to follow the negative curve of maximal error, meaning that the algorithm has a tendency to estimate the 3D points at a nearest depth than they really are. The errors that are positive, and therefore contradict this observation, might be explained by the fact that the 3D surface of the ball was not often rendered. Indeed, the ball has almost no texture, which thus doesn't always lead to support points, or even high gradient points, located in the center of the ball. This means that the depth that is most likely to be estimated is the depth of its contour, i.e. the distance measured by the telemeter, minus the radius of the ball, which is of 6.9cm. Sometimes also, the contour of the ball could not even count a single support point. In that case, even if the ball contains high gradient points, the depth returned would be the depth of the approximated surface around the ball, most likely the floor, which would give a positive error even bigger than the radius of the ball.

In conclusion, the simple way used in this test to estimate the depth of the ball might not be the most recommended, but the results provided seems however promising. Some kind of average over the results for consecutive images taken by the robot might smooth the error, and more complicated techniques could probably solve the issues mentioned above and lead to a better accuracy. As the floor seems well approximated by the triangular mesh, due to its texture, an idea could be to isolate the pixels belonging to it, estimate the 3D representation of its plane, and then find the intersection with the bottom of the ball.

	grass floor fluorescent light set	wooden floor fluorescent light set	grass floor spot light set	complex scene spot light set	overall set
mean	$5.5cm/m$	$4.4cm/m$	$4.4cm/m$	$5.5cm/m$	$5cm/m$
standard deviation	$5cm/m$	$3.1cm/m$	$4.3cm/m$	$3.1cm/m$	$4.25cm/m$

Table 6.2 – Mean absolute error on the depth estimation of the ball for the different test sets

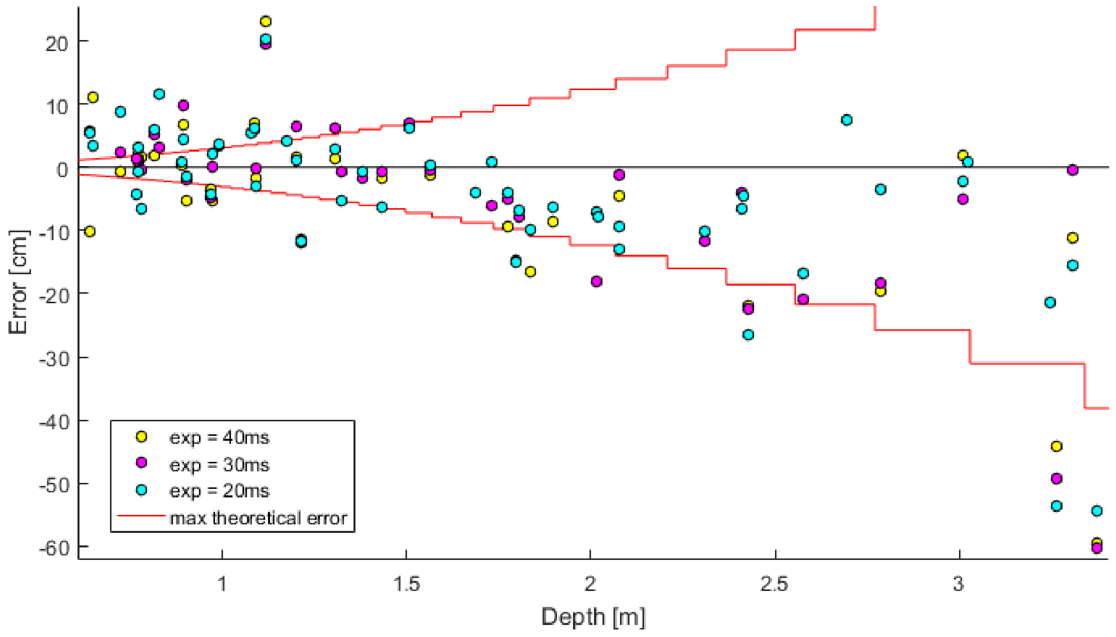


Figure 6.9 – Error on the estimated depth of the ball for the set of pictures with grass floor and fluorescent lighting . Comparison of the results for exposure times bigger than 20ms.

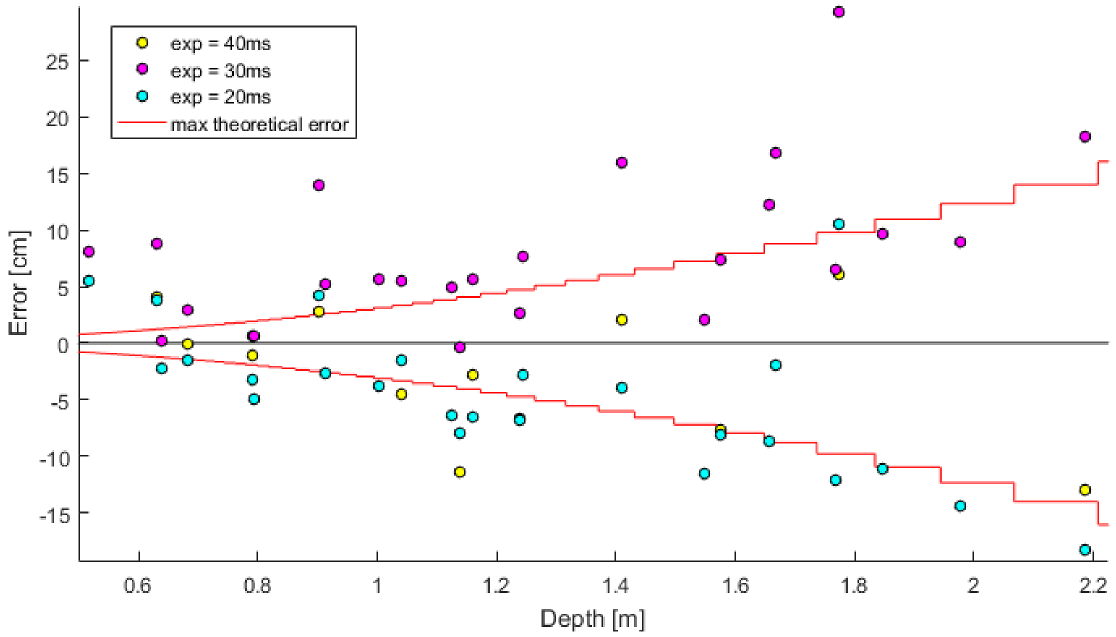


Figure 6.10 – Error on the estimated depth of the ball for the set of pictures with wooden floor and fluorescent lighting . Comparison of the results for exposure times bigger than 20ms.

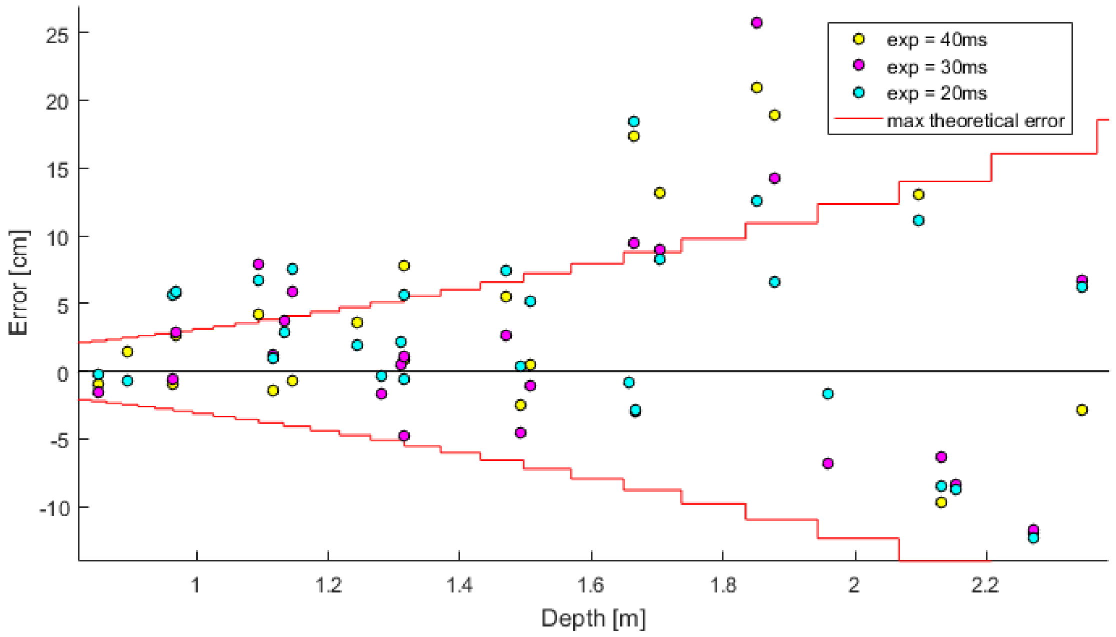


Figure 6.11 – Error on the estimated depth of the ball for the set of pictures with grass floor under strong spot light. Comparison of the results for exposure times bigger than 20ms.

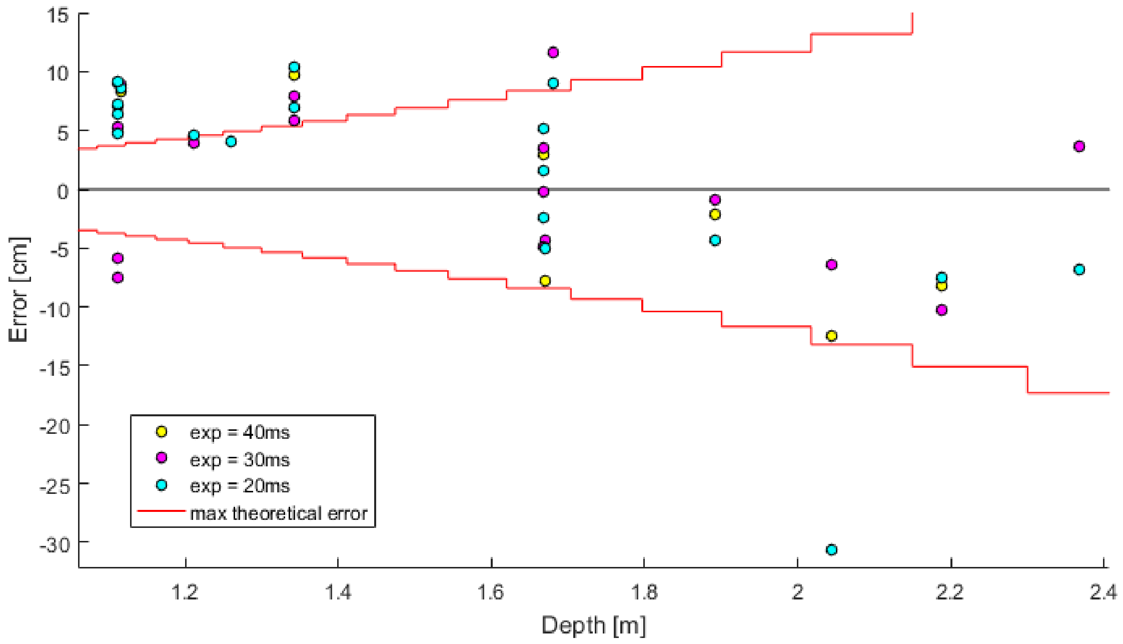


Figure 6.12 – Error on the estimated depth of the ball for the set of pictures of complex scenes under strong spot light. Comparison of the results for exposure times bigger than 20ms.

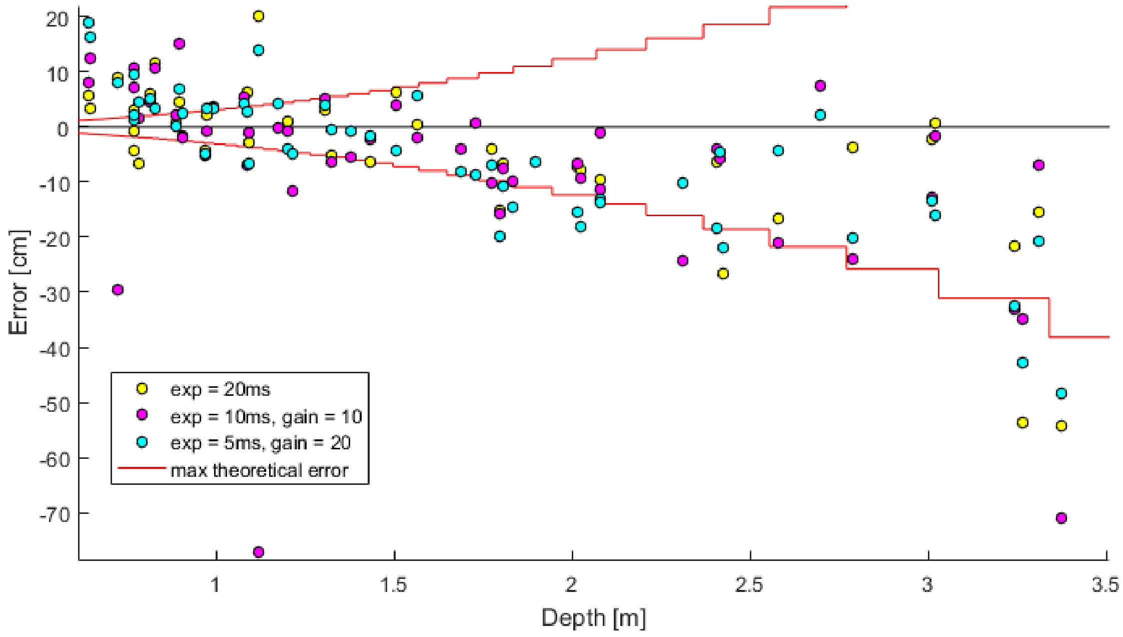


Figure 6.13 – Error on the estimated depth of the ball for the set of pictures with grass floor and fluorescent lighting . Comparison of the results for exposure times smaller than 20ms.

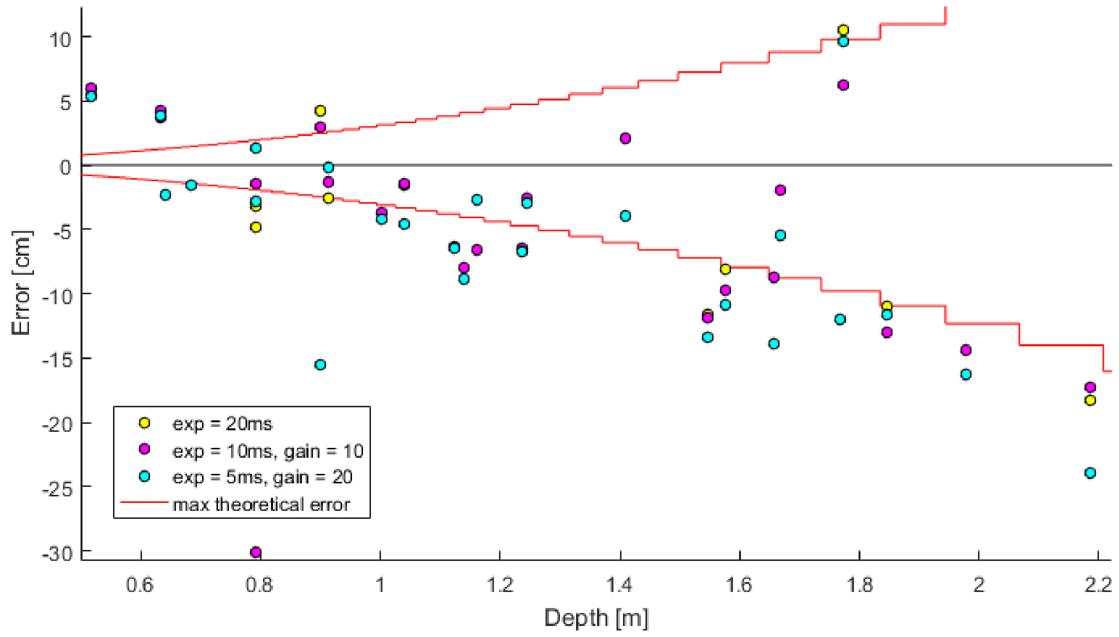


Figure 6.14 – Error on the estimated depth of the ball for the set of pictures with wooden floor and fluorescent lighting . Comparison of the results for exposure times smaller than 20ms.

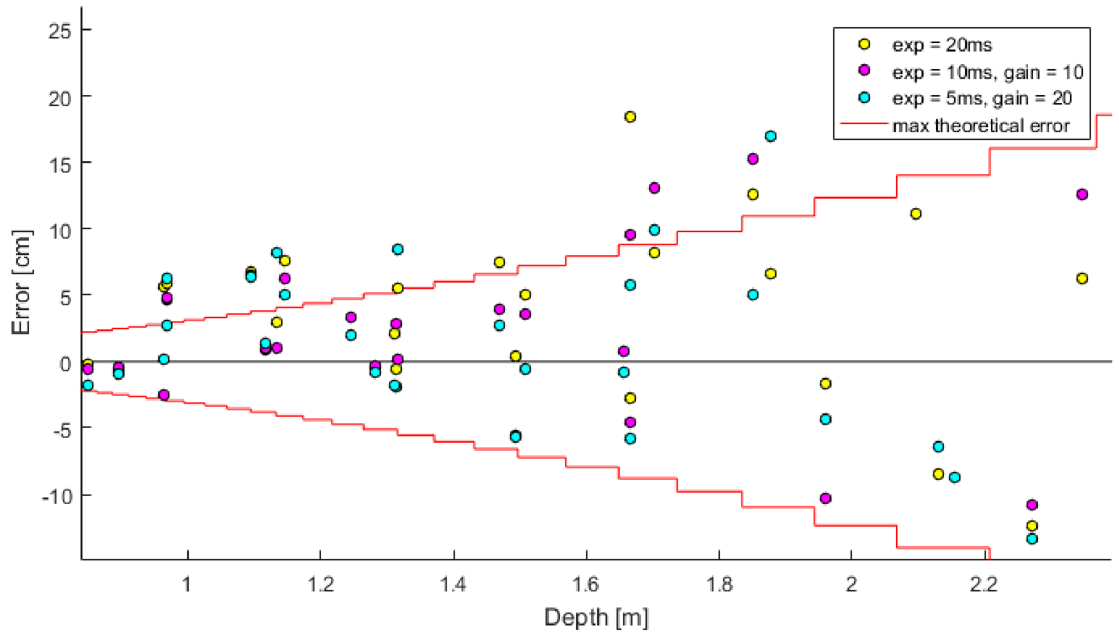


Figure 6.15 – Error on the estimated depth of the ball for the set of pictures with grass floor under strong spot light. Comparison of the results for exposure times smaller than 20ms.

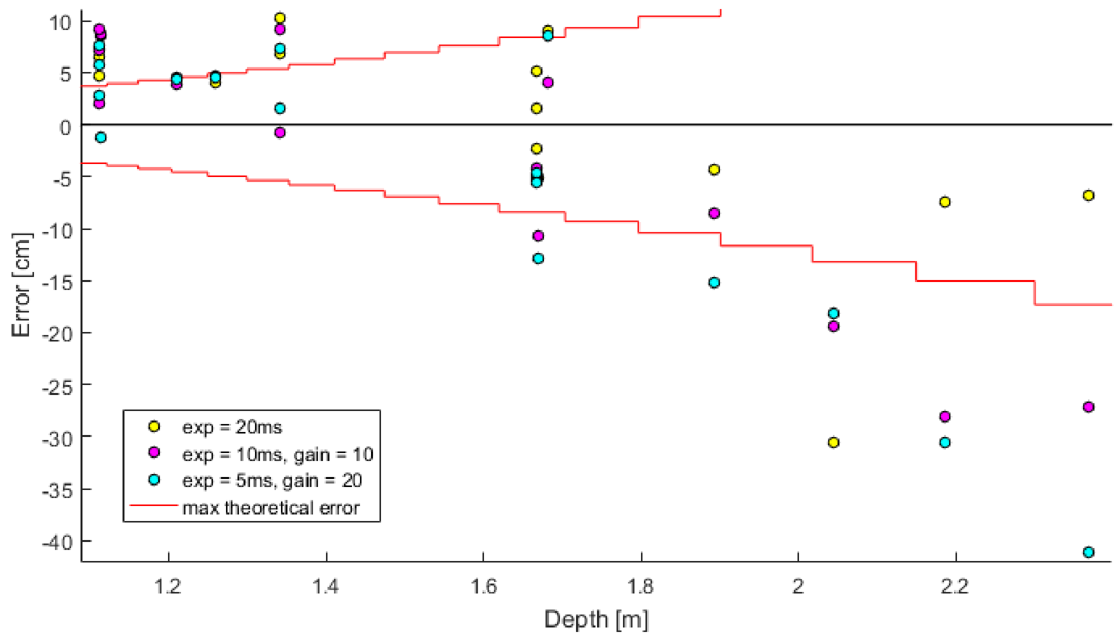


Figure 6.16 – Error on the estimated depth of the ball for the set of pictures of complex scenes under strong spot light. Comparison of the results for exposure times smaller than 20ms.

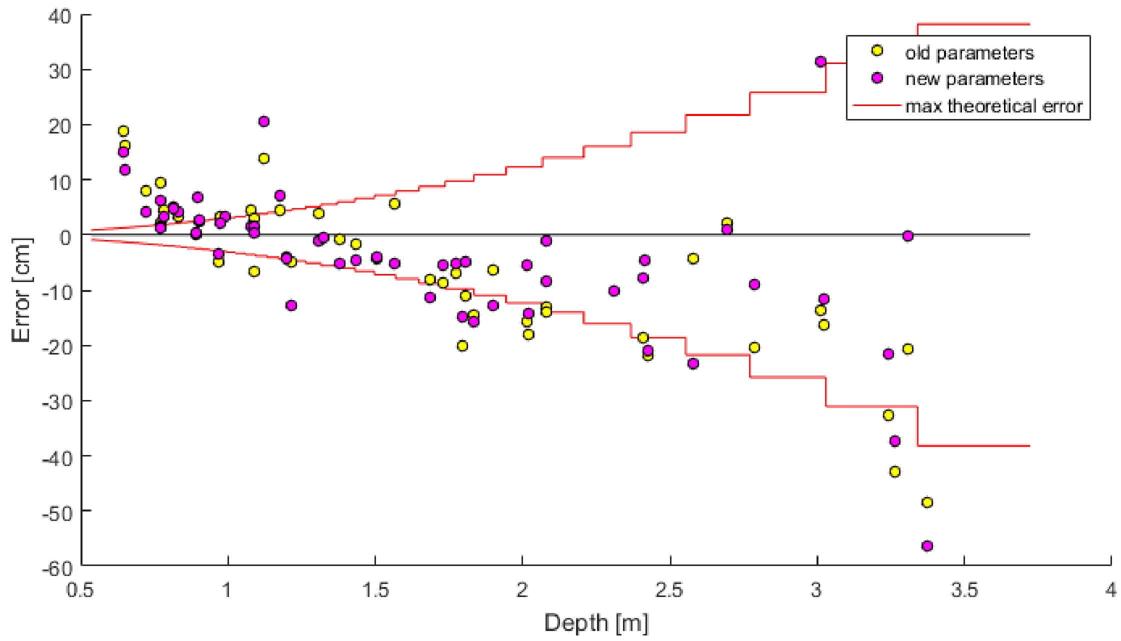


Figure 6.17 – Error on the estimated depth of the ball for the set of pictures with grass floor and fluorescent lighting . Comparison between the results of two set of stereo matching parameters, on pictures taken with an exposure time of 5ms.

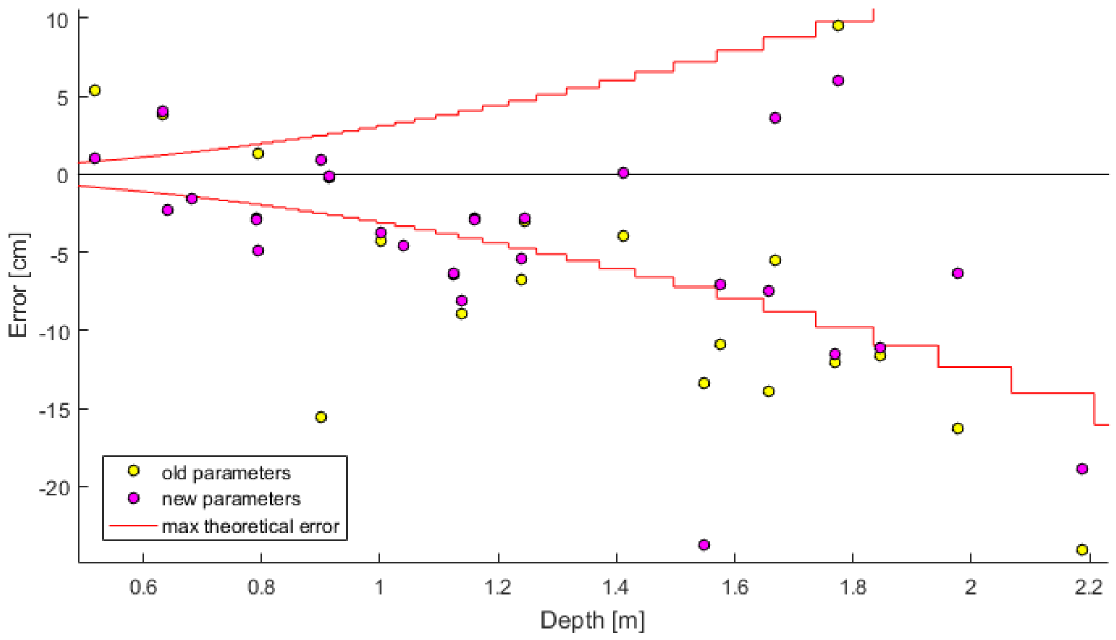


Figure 6.18 – Error on the estimated depth of the ball for the set of pictures with wooden floor and fluorescent lighting . Comparison between the results of two set of stereo matching parameters, on pictures taken with an exposure time of 5ms.

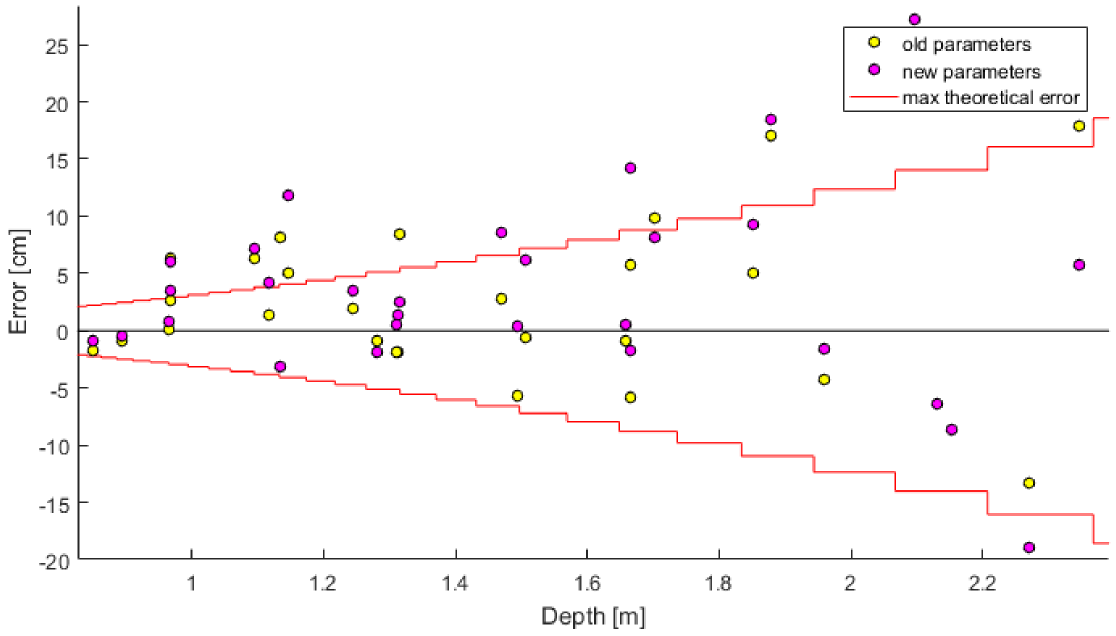


Figure 6.19 – Error on the estimated depth of the ball for the set of pictures with grass floor under strong spot light. Comparison between the results of two set of stereo matching parameters, on pictures taken with an exposure time of 5ms.

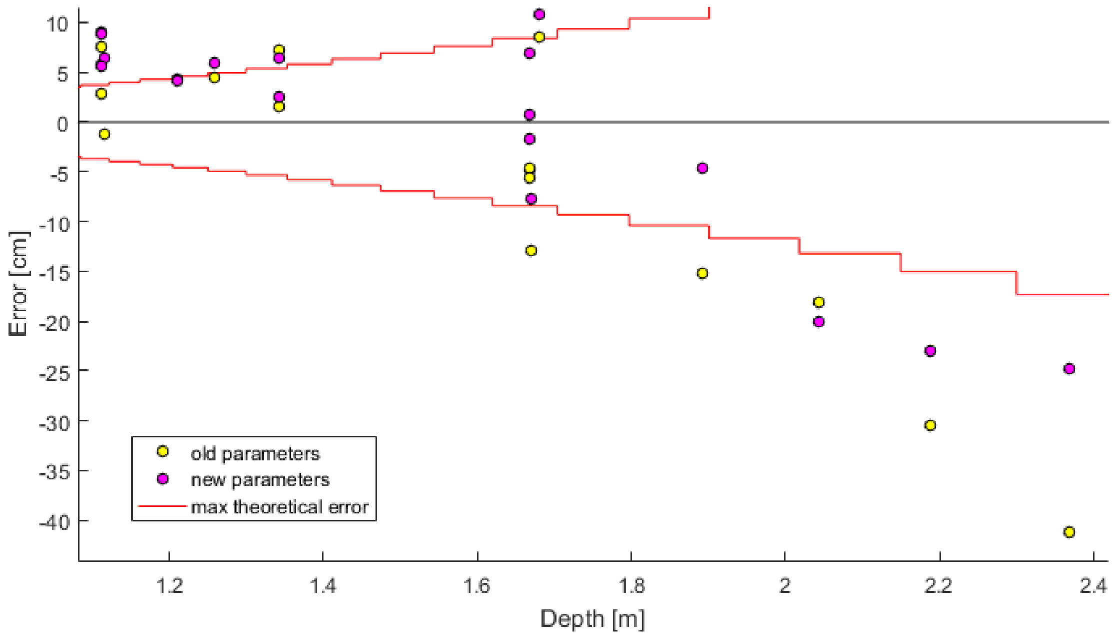


Figure 6.20 – Error on the estimated depth of the ball for the set of pictures of complex scenes under strong spot light. Comparison between the results of two set of stereo matching parameters, on pictures taken with an exposure time of 5ms.

6.3 Floor inclination computation

The test trying to estimate the depth of the ball was only carried out focusing on a single point of the image. Moreover, it was highlighted that the results of this test was not optimal due to the lack of texture of the ball and the naive approach used to estimate its depth from the semi-dense disparities. Therefore, as textured floors seemed however well approximated by the triangular mesh in this previous test, the evaluation presented in this section will try to estimate the accuracy at which the plane of a grass soccer field could be rendered in 3D. The technique employed to verify this accuracy was to measure the inclination at which the cameras were observing the floor, and compare this measurement with the inclination obtained thanks to the proposed stereo vision algorithm.

To perform this test, 10 stereo image pairs of the floor were taken with various orientations (see Figure 6.21 for some examples). These orientations were measured by replacing the telemeter of the setup presented in Figure 6.1 with a smartphone running a clinometer application. The clinometer application can measure two angles, as shown in Figure 6.22. One of these angles is the inclination of the floor with the optical axis, or z axis, of the reference camera. The other angle is the inclination of the floor with the baseline of the stereo system, or x axis of the reference camera.

The stereo vision algorithm was then run on the floor images, and for each stereo pair, a 3D point cloud representing the floor was obtained. To build these point clouds, only the pixels in ROI surrounding a large part of the floor image were considered. The RANSAC algorithm [30] was then used to fit a plane on each of these 3D point clouds.

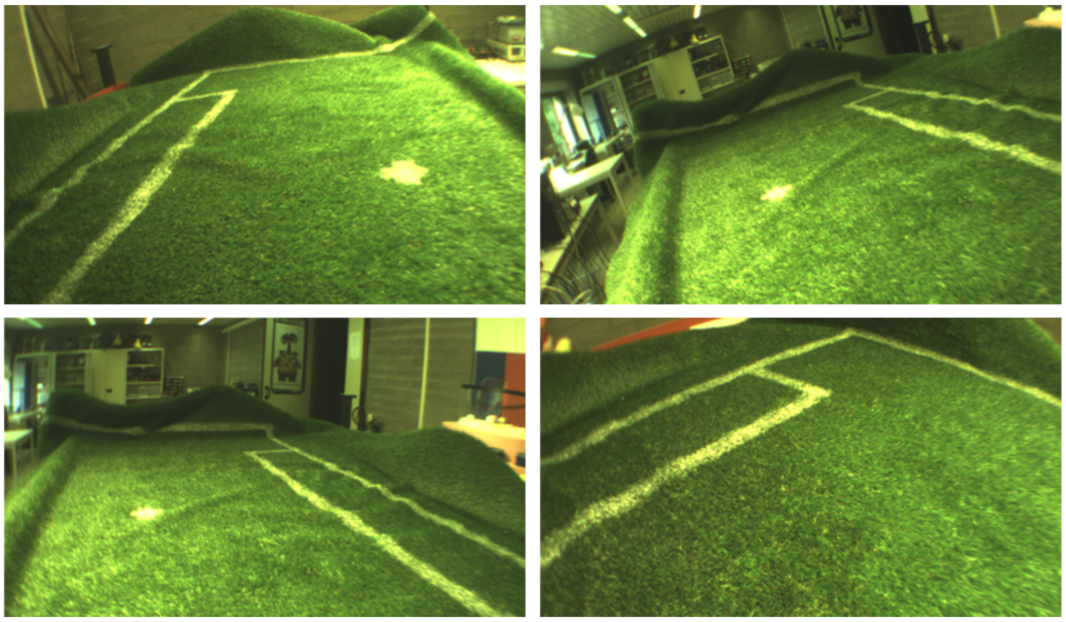


Figure 6.21 – Examples of floor images taken for the inclination test

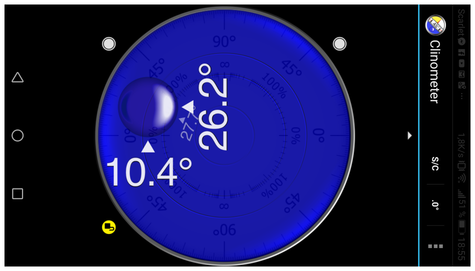


Figure 6.22 – Clinometer application

From the equation of a plane returned by RANSAC, the angles of this plane with the z and x axis of the reference camera can be computed as follow.

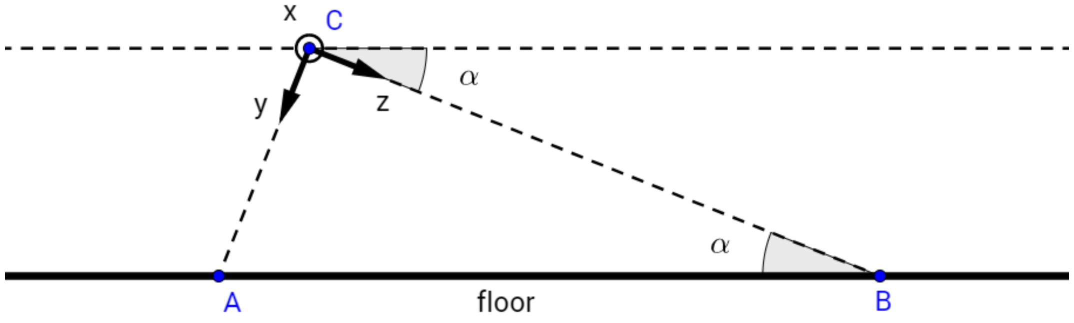


Figure 6.23 – Inclination of the camera optical axis with the floor

Consider Figure 6.23, with C the reference frame of the camera. The angle α is one of the angles measured by the clinometer application. The equation of the floor plane is of the form

$$a * x + b * y + c * z + d = 0 \tag{6.1}$$

The angle α can be obtained with the following simple trigonometric relation

$$\alpha = \tan^{-1}\left(\frac{|AC|}{|BC|}\right) \tag{6.2}$$

where $|AC|$ (resp. $|BC|$) can be easily obtained from (6.1) by setting z (resp. y) and x to zero. Equation (6.2) then becomes

$$\alpha = \tan^{-1}\left(\frac{c}{b}\right) \tag{6.3}$$

Following the same reasoning, the second angle β measured by the clinometer application can be obtained with

$$\beta = \tan^{-1}\left(\frac{a}{b}\right) \tag{6.4}$$

The test was run for two different exposure times, 20ms and 5ms. The parameters of the stereo matching algorithm that were used are the ones established in the ball localization test for each of these exposure times (see Table 6.1). The results are presented in Table 6.3. These results shows that the estimation of the inclination of the floor can be very accurately obtained from the proposed stereo vision algorithm, as the mean absolute error is below $1°$. This reinforces thus the belief that the depth of a ball could be better estimated than in the previous test if more clever techniques were implemented.

Image pair	Truth		exposure = $20ms$ gain = 1				exposure = $5ms$ gain = 20			
	α	β	α	Err_α	β	Err_β	α	Err_α	β	Err_β
1	16.00	-2.50	16.10	0.10	-2.95	-0.45	16.04	0.04	-3.08	-0.58
2	26.30	-2.40	25.06	-1.24	-1.79	0.61	25.48	-0.82	-3.48	-1.08
3	21.70	17.50	22.60	0.90	17.05	-0.45	22.83	1.13	16.50	-1.00
4	26.50	39.00	29.28	2.78	39.73	0.73	29.18	2.68	39.58	0.58
5	27.10	-2.20	26.87	-0.23	-2.35	-0.15	27.44	0.34	-2.85	-0.65
6	38.50	-2.60	37.70	-0.80	-2.53	0.07	39.69	1.19	-1.94	0.66
7	29.80	-1.00	28.46	-1.34	-1.11	-0.11	30.77	0.97	-2.67	-1.67
8	24.90	-1.90	25.49	0.59	-1.95	-0.05	25.87	0.97	-1.58	0.32
9	26.20	10.40	25.50	-0.70	9.22	-1.18	24.99	-1.21	10.18	-0.22
10	30.80	11.10	31.75	0.95	8.40	-2.70	30.43	-0.37	8.81	-2.29
Mean				0.10		-0.36		0.49		-0.59
Standard deviation				1.26		0.98		1.14		0.96
Absolute mean				0.96		0.64		0.97		0.91
Abs. stand. dev.				0.75		0.81		0.72		0.64

Table 6.3 – Results of the floor inclination estimation. The values are indicated in degrees.

6.4 Speed test

The speed of the algorithm was measured in a Linux virtual machine running on a 4 years old laptop with an Intel® Core™ i7-4700HQ. A single iteration of the algorithm was run 50 times while recording the execution times of each step of the algorithm. The results of this test are presented in Table 6.4. These numbers show that the current implementation of the stereo matching algorithm can reach a speed of 20Hz in the best case, and 13hz on average. Adding the time needed for the rectification of the images using *OpenCV*, 7 semi-dense disparity maps can be generated per second. Given that the slowest parts of the matching algorithm are the ones that were identically used in [1], it seems very likely that the testing environment is the reason the performance does not reach the expected 100Hz.

Step of the algorithm	**Average speed**	**Best timing reached**
Rectification	75.70	40.51
Histogram equalization	0.89	0.49
Gradient thresholding	3.03	1.63
Gaussian filtering	0.68	0.43
Left feature detection	6.95	1.61
Right feature detection	14.82	5.42
Census transform	3.95	1.53
Sparse stereo matching	41.02	21.17
Delaunay triangulation	3.17	1.81
Disparity interpolation	7.71	5.69
Costs evaluation	0.94	0.55
Disparity refinement	0.52	0.28
Complete stereo matching	77.97	49.34
Rectification + stereo matching	153.67	89.85
3D projection	0.52	0.23

Table 6.4 – Measures of the execution time of the algorithm, running in a Linux virtual machine with an i7 core. These measures are expressed in milliseconds.

7. Conclusion

The aim of this thesis was to determine if a stereo vision system could be implemented on a constrained robot platform, in order to help the robot play in a soccer game. The proposed stereo vision system should thus provide, in real-time, useful 3D information about the field of play and the game elements.

A review of the state of the art of stereoscopic vision systems revealed that the most accurate techniques needed specialized hardware like FPGA or GPU to be able to generate depth maps at a sufficiently high speed. Given that the considered robot platform is not supposed to include such technologies, researches were focused on finding the fastest algorithm possible that could still give usable results in terms of accuracy. One method that seemed to meet this criteria was chosen to be implemented, adapted, and tested with the stereo setup that should be installed in the robot.

The calibration of the considered stereo setup first showed that no accurate results should be expected beyond a few meters of the cameras. Moreover, it brought to light some issues with the mounting system of the cameras. From these problems, it was deduced that a particular attention should be paid to the design of the support of the cameras. This design should concentrate on blocking every possible rotations of the cameras. The rigidity of the support is thus an important parameter regarding that matter, but fixing both the front and the back of the cameras should also help. Moreover, it was assumed that using a straighter support might improve the calibration results and therefore the accuracy of the depth estimations.

As a first stage of development, the focus was put on the study of the accuracy that could be obtained with the proposed stereo vision algorithm. Two tests were performed to evaluate the quality of the output generated by the proposed stereo vision algorithm. A first test evaluated the accuracy at which the depth of a ball could be estimated, using a simple naive approach. This led to a mean absolute error of 5cm/m. While the error was also proven to be highly variable, it seemed reasonable to assume that better and more stable results could be obtained with a more clever approach than the one implemented here. Indeed, the semi-dense disparity maps output by the proposed stereo matching algorithm might not always render the 3D surface of a ball in the scene, but this doesn't mean that the location of this ball could not be deduced more accurately using the information provided by the disparity map. Besides, the second test revealed that highly textured plane surfaces could be accurately rendered, as the inclination at which the cameras are observing the floor was able to be estimated with an error of less than 1°. Aside from that, another important conclusion of these tests is the fact that the exposure time used to take the pictures should not be below 5ms because the resulting noise would then affect too much the stereo matching.

Testing the implemented algorithm in a Linux virtual machine running with an Intel® Core™ i7-4700HQ, 7 semi-dense disparity maps could be generated per second. As [1] claims that they could reach a hundred disparity maps per second, the testing on another platform, and further code optimizations, could very likely lead to a better speed.

In conclusion, the usability of the proposed stereo vision algorithm will depend on the four following parameters:

- The artificial intelligence that will be developed to take advantage of the depth information provided by the proposed algorithm
- The accuracy needed by the robot to perform its actions, which is not exactly known at this stage
- The sharpness of the images that can be taken with an exposure time of 5ms by the moving robot
- The execution time that will be achievable on the robot platform

Appendix A

Stereo calibration set example

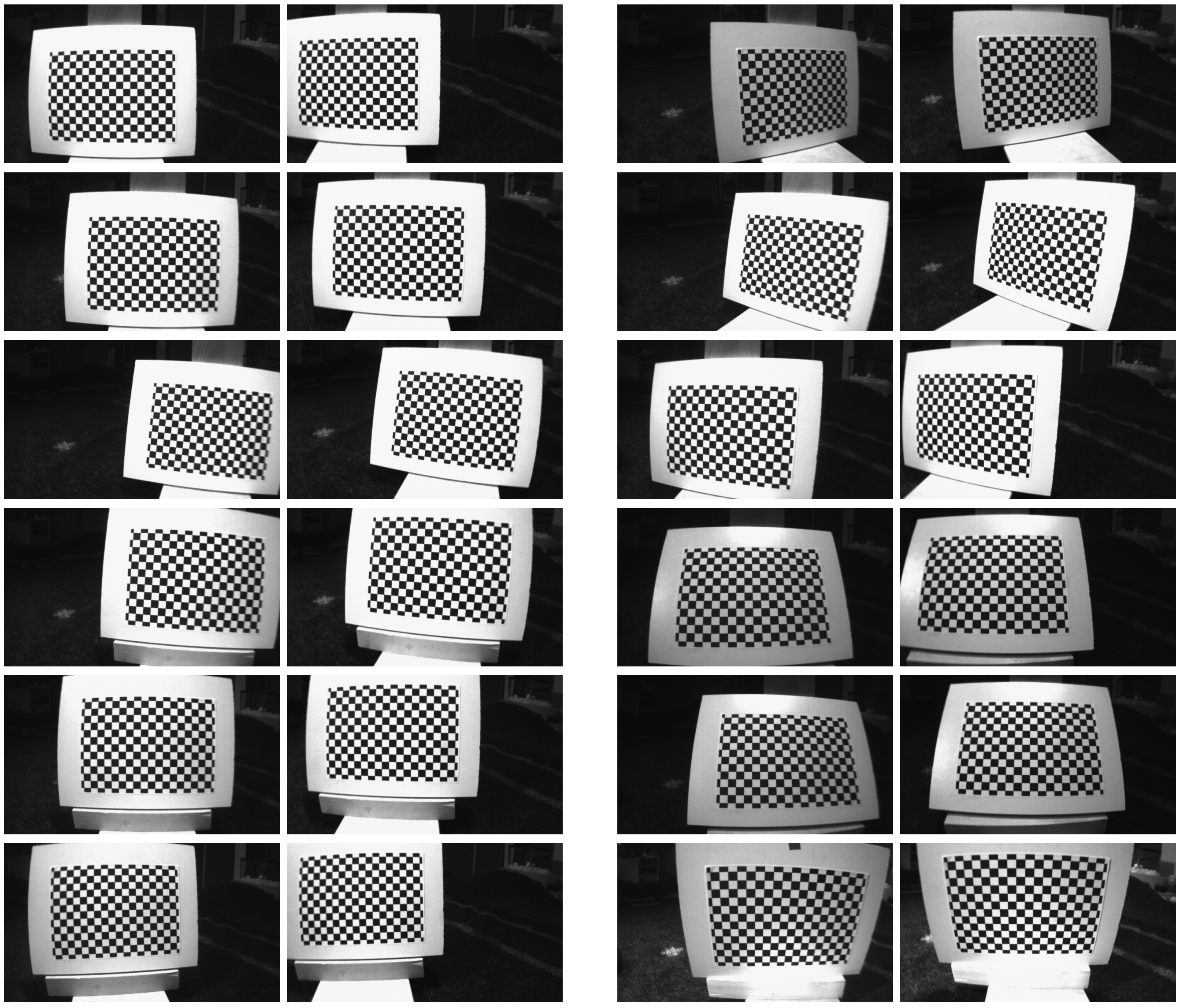


Figure A.1 – Stereo pairs used for calibration and leading to a mean reprojection error of 0.18pixels (Part 1)

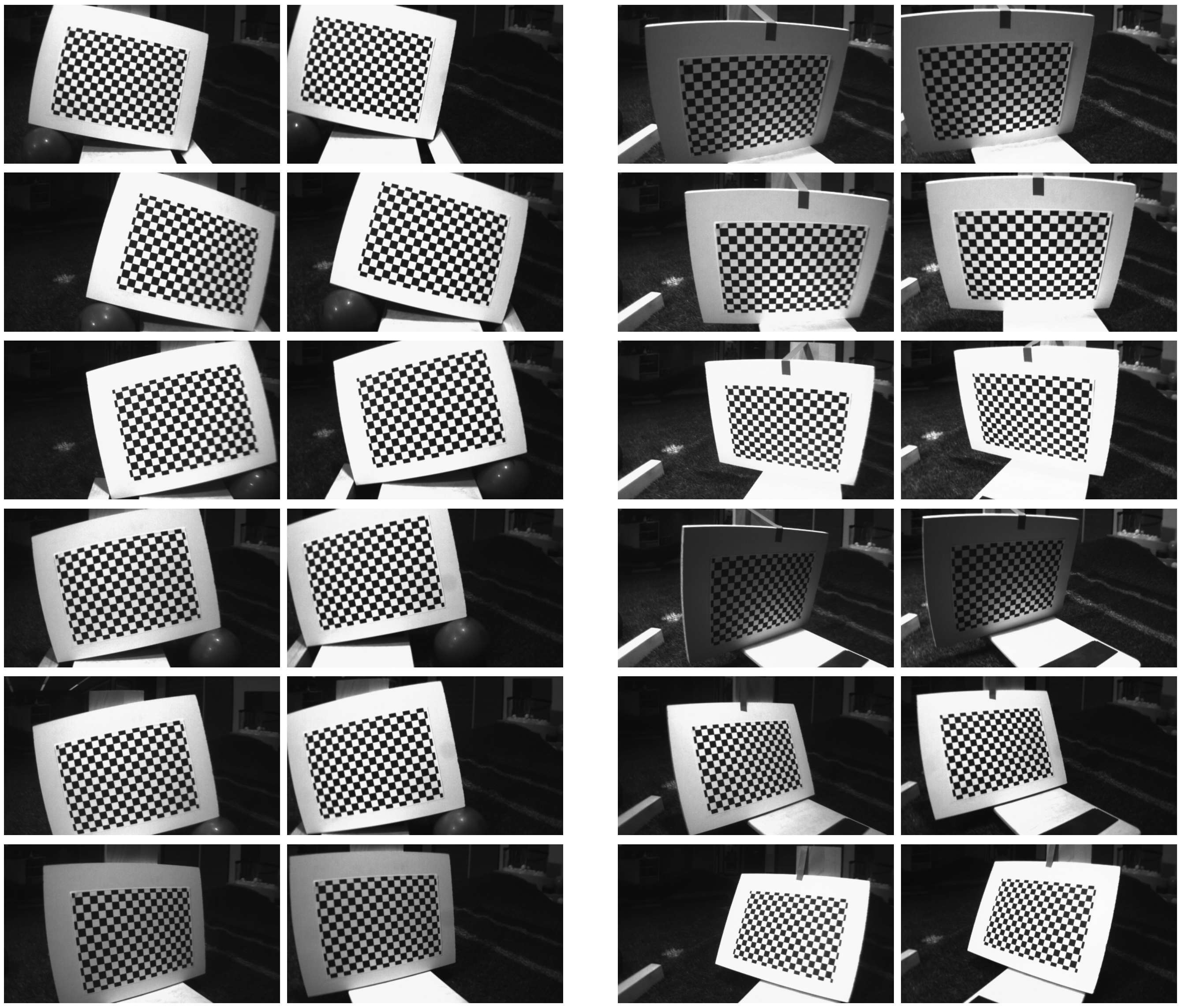


Figure A.2 – Stereo pairs used for calibration and leading to a mean reprojection error of 0.18pixels (Part 2)

Bibliography

[1] S. Pillai, S. Ramalingam, and J. J. Leonard, "High-performance and tunable stereo reconstruction," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on.* IEEE, 2016, pp. 3188–3195.

[2] M. V. Droogenbroeck. (2015) Computer vision (lecture notes). [Online]. Available: http://orbi.ulg.ac.be/handle/2268/184667

[3] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in Matlab.* Springer, 2011, iSBN 978-3-642-20143-1.

[4] MathWorks. What is camera calibration? [Online]. Available: https://nl.mathworks.com/help/vision/ug/camera-calibration.html?searchHighlight=skew&s_tid=doc_srchtitle

[5] OpenCV. Camera calibration and 3d reconstruction. [Online]. Available: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[6] M. Dumont. Stereo matching. [Online]. Available: http://www.dwerftje.net/phd/sm/

[7] S. Mattoccia, "Stereo vision: algorithms and applications," 2011.

[8] Wikipedia. Epipolar geometry. [Online]. Available: https://en.wikipedia.org/wiki/Epipolar_geometry

[9] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[10] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis.* Springer, 1978, pp. 105–116.

[11] OpenCV. Geometric image transformations. [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html

[12] C. Vancea and S. Nedevschi, "Analysis of different image rectification approaches for binocular stereovision systems," in *Proceedings of IEEE 2nd International Conference on Intelligent Computer Communication and Processing (ICCP 2006)*, vol. 1, 2006, pp. 135–142.

[13] D. D. J. Lee. (2012) Stereo calibration and rectification. [Online]. Available: http://ece631web.groups.et.byu.net/Lectures/ECEn631%2014%20-%20Calibration%20and%20Rectification.pdf

[14] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2. IEEE, 2005, pp. 807–814.

[15] Nerian. Sp1: Real-time 3d stereo vision through fpga-technology. [Online]. Available: https://nerian.com/products/sp1-stereo-vision/

[16] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[17] T. Taniai, Y. Matsushita, Y. Sato, and T. Naemura, "Continuous stereo matching using local expansion moves," *arXiv preprint arXiv:1603.08328*, 2016.

[18] J. Kowalczuk, E. T. Psota, and L. C. Perez, "Real-time stereo matching on cuda using an iterative refinement method for adaptive support-weight correspondences," *IEEE transactions on circuits and systems for video technology*, vol. 23, no. 1, pp. 94–104, 2013.

[19] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Asian Conference on Computer Vision (ACCV)*, 2010.

[20] K. Schauwecker, R. Klette, and A. Zell, "A new feature detector and stereo matching method for accurate high-performance sparse stereo matching," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on.* IEEE, 2012, pp. 5171–5176.

[21] OpenCV. Histogram equalization. [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html

[22] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *European conference on computer vision.* Springer, 1994, pp. 151–158.

[23] Y. K. Baik. Fast census transform-based stereo algorithm using sse2. [Online]. Available: http://cv.snu.ac.kr/hyunxx/research/stereo/fastCTSV.html

[24] R. Bayer. Hamming distance. [Online]. Available: https://people.rit.edu/rmb5229/320/project3/hamming.html

[25] H. Hirschmuller and D. Scharstein, "Evaluation of stereo matching costs on images with radiometric differences," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 9, pp. 1582–1599, 2009.

[26] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision–ECCV 2006*, pp. 430–443, 2006.

[27] Wikipedia. Delaunay triangulation. [Online]. Available: https://en.wikipedia.org/wiki/Delaunay_triangulation

[28] B. Delaunay, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[29] D. D. B. Kornberger. C++ constrained delaunay triangulation fade2d. [Online]. Available: http://www.geom.at/fade2d/html/

[30] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.