DESIGN AND IMPLEMENTATION OF A VISUAL SENSOR – BASED DEPTH ESTIMATION MODULE IN A SYSTEM-ON-CHIP ARCHITECTURE

JORGE IVÁN BOTERO GALLEGO

Advisor: HENRY DAVID CARRILLO LINDADO, Ph. D
Co-Advisor: CARLOS ALBERTO PARRA RODRÍGUEZ, Ph. D

DEPARTMENT OF ENGINEERING
MASTERS IN ELECTRONIC ENGINEERING
PONTIFICIA UNIVERSIDAD JAVERIANA
BOGOTÁ, COLOMBIA
JUNE 14TH 2017

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Nowadays, one of the main requirements for a robot to be autonomous in real world environments is that is must have the capacity to map its surroundings and locate itself within them. One approach to solving this problem are the SLAM (**S**imultaneous **L**ocalization **a**nd **M**apping) algorithms which allow a robot to know its location on a map as it builds it.

Usually, the input variables to be estimated by these algorithms are the robot's current pose, the record of the previous poses and the representation of the environment. Focusing exclusively on producing a reliable representation of the environment, the main goal of this work is to estimate the depth of a scene by using passive visual sensors. This mainly involves stereo vision techniques or dense multi-view estimation methods.

However, the complexity of SLAM algorithms forces them to be implemented on high-performance personal computers which limits their use on robotic platforms in terms of load support, size and power consumption. Hence the idea of taking advantage of a System-on-Chip (SoC) architecture that focuses the required computational resources (reconfigurable logic, memories, …) in the lowest number of chips to solve a particular problem. Due to their versatility in terms of applications, SoC architectures are easy to find nowadays in the market and are widely used in the Robotics field. The depth estimation module developed during this project is to be implemented on a specific SoC device. This thesis is derived from a postdoctoral research project financed by the Pontificia Universidad Javeriana that aims at implementing a state of the art SLAM algorithm on a SoC.

After defining the project's objectives, a concise theoretical framework is presented followed by the description of the entire development and implementation processes as well as the results obtained from testing. Finally, a set of conclusions is given on the overall work.

II. OBJECTIVES

The project's objectives are divided into one main objective and five specific objectives.

A. Main objective

- Design and implement a depth estimation system in static environments in a SoC architecture.

B. Specific objective

- Adapt a visual sensor-based depth estimation algorithm for it to be implemented in a commercially available system-on-chip (SoC) architecture.
- Implement the chosen algorithm in a commercially available system-on-chip (SoC) architecture.
- Assess the performance of the implemented depth estimation algorithm in terms of (i) the percentage of disparity error in non-occluded regions and (ii) the percentage of pixels with estimated depth provided by the algorithm. The performance will be assessed using the base line given by a state of the art dataset (e.g., "KITTI Vision Benchmark Suite", "Malaga Dataset", "Make3D", "Middlebury", "EISATS").
- Implement a calibration protocol of the visual sensors so they can be used by the selected algorithm.
- Assess, using real data obtained from the visual sensors, the depth estimation of the implemented system by comparing against a base line given by state of the art depth sensors (e.g., laser, RGBD camera, triangulation positioning system).

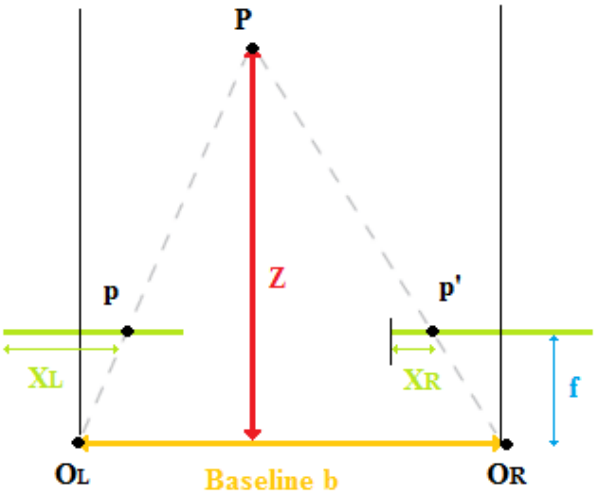
III. THEORETICAL FRAMEWORK

This section contains a synthesis of the theoretical basis exploited during the development of this project. It begins with an explanation of the canonical stereo geometry model that is inherently used by conventional calibration methods. This helps understand the possible sources of error in disparity measurements as well as the geometric foundation of depth estimation on a two-camera system. Afterwards, the analyzed disparity computation algorithms are summarized. A brief description of the filtering functions and state of the art is also included.

A. Canonical Stereo Geometry Model

The calibration process (which will be detailed in section IV) allows the assumption of having two perfectly aligned cameras where each one can be individually modeled as a pinhole camera. The canonical stereo geometry [1] of a two-camera system defines an exact copy of the left camera by translating a distance b along the X_S axis of the $X_SY_SZ_S$ camera coordinate system of the left camera. Therefore, the projection centers of the left and right cameras are (0,0,0) and (b,0,0) respectively. ***Figure 1*** corresponds to a graphical representation of the canonical model. $\boldsymbol{O_L}$ and $\boldsymbol{O_R}$ are the optical centers of the left and right cameras, $\boldsymbol{X_L}$ and $\boldsymbol{X_R}$ are the coordinates of their respective projected points ***p*** and ***p'***, ***P*** is the point in the real world, Z its corresponding depth (distance between the observed point and the camera) and f is the camera's focal distance.

Equation (1) establishes the relation between the depth and the disparity d by considering the triangles PO_LO_R and Ppp' as similar. By solving Z, equation (2) is derived. It is evident that a larger baseline and a larger focal length increases the number of depth levels supported but it also reduces the number of pixels with corresponding pixels on the second image. A higher image resolution may fix this issue by guaranteeing a higher accuracy in depth levels at the expense of a higher computational cost.



$$\frac{b}{Z} = \frac{(b + X_R) - X_L}{Z - f} \quad (1)$$

$$Z = \frac{bf}{X_L - X_R} = \frac{bf}{d} \quad (2)$$

Figure 1. Canonical stereo geometry model

These four conditions sum up the canonical stereo geometry model:

1. There must be two coplanar images of the same size $N_{rows} \times N_{cols}$
2. Optic axes must be parallel
3. Effective focal length f must be identical
4. Corresponding image rows must be collinear between images

The geometric limitations of the stereo camera setup can lead to an error related to the image resolution [2]. Such resolution error R is described in terms the pixel size r as:

$$R = \frac{rZ^2}{fb - rZ} \quad (3)$$

where Z corresponds to the estimated depth, f is the focal length and b is the camera baseline. To plot the resolution error, the minimum and maximum values of depth must be calculated. The theoretical maximum depth Z_{max} is obtained with equation (2) by using the lowest possible value of disparity which is 1. The focal length and the pixel size values are discussed in section IV. C. c).

$$Z_{max} = \frac{3\ mm \times 60\ mm}{1 \times 9.375\ um} = 19.2\ m \quad (4)$$

The expected minimum depth Z_{min} corresponds to the point where the overlap between the two stereo cameras begins and is related to the field of view Θ by:

$$Z_{min} = \frac{b}{\tan(\theta/2)} \quad (5)$$

Θ can be calculated with (6) and it is noteworthy to mention that the image width w equals to the number of horizontal pixels in the image multiplied by the pixel size. With a width of 320 pixels, the maximum field of view angle obtained is 59,03°. The resulting value of Z_{min} is 0.12 m, i.e., 12 cm.

$$\theta = 2\ arctan\left(\frac{w}{2f}\right) \quad (6)$$

By restricting $\boldsymbol{Z_{max}}$ to 1.5 m which is closer to the disparity computation algorithm's distance range, the resolution error R can be plotted in terms of the observed scene depth (***Figure 2***). At 1 meter, the error is 5.5 cm which corresponds to an expected maximum error of 5.5%.

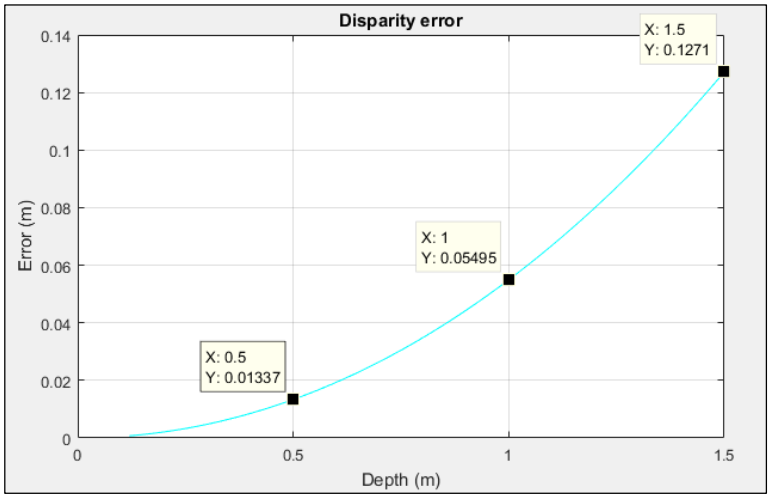


Figure 2. Resolution error R in terms of the scene depth

Disparity can be the source of another type of resolution error ϵ. Although both error functions seem almost identical, one relates to the error on a geometrical standpoint using pixel size and the latter deals with the disparity error ϵ_d [3] as shown in equation (7). In both cases, the errors proportionally increase as the depth does (see ***Figure 2***).

$$\epsilon = \frac{bf}{d} - \frac{bf}{d + \epsilon_d} = \frac{z^2\epsilon_d}{bf + z\epsilon_d} \quad (7)$$

B. Triangulation in Stereo Geometry

Based on the canonical stereo geometric model previously described, disparity is equal to the difference between the horizontal coordinates of the pixels on left and right images [4]. Since $X_L \geq X_R$, the disparity is generally referenced to the left image hence its value is $X_L - X_R$ as seen in (23). To recover the unknown tridimensional points $P = (X_S, Y_S, Z_S)$ in the camera coordinate system of the left camera, triangulation is used. The undistorted left and right image coordinates are named (X_{uL}, Y_{uL}) and (X_{uR}, Y_{uR}) respectively and satisfy $Y_{uL} = Y_{uR}$ since the pair is aligned and $X_{uL} \geq X_{uR}$ due to the left image being the reference image.

The Cartesian camera coordinate system is used to represent the points in the real world and allows the mapping of such points via a central projection defined with equations (8) and (9) where X_u and Y_u are the pixel locations on the undistorted image plane.

$$X_u = \frac{fX_S}{Z_S} \quad (8)$$

$$Y_u = \frac{fY_S}{Z_S} \quad (9)$$

Using this projection for both cameras the left camera's coordinate system is mapped into the undistorted image points p_{uL} and p_{uR} of the left and right image planes respectively. Throughout the mathematical process, it is assumed that calibration offers proper values of b and f.

$$p_{uL} = (x_{uL}, y_{uL}) = \left(\frac{f.X_S}{Z_S}, \frac{f.Y_S}{Z_S}\right) \quad (10)$$

$$p_{uR} = (x_{uR}, y_{uR}) = \left(\frac{f.(X_S - b)}{Z_S}, \frac{f.Y_S}{Z_S}\right) \quad (11)$$

By solving the equivalences above, the triangulated point coordinates correspond to those in equations (12), (13) and (14). Note that for Y_S, $Yu = Y_{uL} = Y_{uR}$.

$$X_s = \frac{bX_{uL}}{X_{uL} - X_{uR}} \quad (12)$$

$$Y_s = \frac{bY_u}{X_{uL} - X_{uR}} \quad (13)$$

$$Z_s = \frac{bf}{X_{uL} - X_{uR}} \quad (14)$$

In terms of disparity computation methods, window-based strategies did not account for the scene's topology leading to notorious errors in the output disparity maps. Therefore, probabilistic methods such as block matching are considered as an alternative to compute disparity. Sections C, D and E discuss three different disparity computation methods with underlying probabilistic basis.

C. Semi-Global Matching and Mutual Information (SGBM)

The semi-global matching method proposed by Hirschmuller et al. [5] uses pixelwise matching of Mutual Information (MI) and approximates a global 2D smoothness constraint which is the combination of various 1D constraints.

The first stage is the matching cost calculation for a base image pixel p from its intensity I_{bp} and the associated correspondence I_{mq} with q (equation 15) of the match image. $e_{bm}(p, d)$ represents the epipolar line in the match image for the base image pixel p with line parameter d. This is true for rectified images only.

$$q = e_{bm}(p,d) = [p_x - d, p_y]^T \quad (15)$$

One option for calculating pixelwise matching cost is Birchfield and Tomasi's method where the cost is determined as the absolute minimum difference of intensities at p and q within a half a pixel in each direction along the epipolar line. Although this method is insensitive to sampling, Mutual Information is unaffected by changes in illumination and recording and is derived from the entropy H of two images and their joint entropy as:

$$MI_{I1,I2} = H_{I1} + H_{I2} - H_{I1,I2} \quad (16)$$

Using the probability functions of the corresponding images, the individual entropy of an image is defined by equation (17) and the joint entropy is defined by equation (18).

$$H_I = -\int_0^1 P_I(i) log\,(P_I(i)) di \quad (17)$$

$$H_{I1,I2} = -\int_0^1 \int_0^1 P_{I1,I2}(i_1,i_2) log\,(P_{I1,I2}(i_1,i_2)) di_1 i_2 \quad (18)$$

After an extensive mathematical analysis, the resulting formulation of Mutual Information is:

$$MI_{I1,I2} = \sum_P mi_{I1,I2}(I_{1p}, I_{2p}) \quad (19)$$

$$\text{where } mi_{I1,I2}(i,k) = h_{I1}(i) + h_{I2}(k) - h_{I1,I2}(i,k) \quad (20)$$

The MI matching cost is hence:

$$C_{MI}(p,d) = -mi_{Ib,fD(I_m)}(I_{bp}, I_{mq}) \quad (21)$$

$$\text{where } h_{I1,I2}(i,k) = -\frac{1}{n}\log\left(P_{I1,I2}(i,k) \otimes g(i,k)\right) \otimes g(i,k) \quad (22)$$

$$\text{and } P_{I1,I2}(i,k) = \frac{1}{n}\sum_P T[(i,k) = I_{1p,}I_{2p}] \quad (23)$$

In the Parzen estimation in equation (22), $\otimes g$ (i, k) represents a convolution with a 2D Gaussian and, in equation (23), the T operator is equal to 1 if its argument is true and 0 if false.

The second stage of the algorithm consists in cost aggregation which pretends to correct the possible ambiguities of pixelwise cost calculation. The idea is to penalize changes in the neighboring disparities which gives support to smoothness. For $P_2 \geq P_1$, the pixelwise and smoothness constraints are expressed with the energy function $E(D)$ depending on the disparity D:

$$E(D) = \sum_P \left(C(p, D_P) + \sum_{q \in N_p} P_1 T\left[\left|D_p - D_q\right| = 1\right] + \sum_{q \in N_p} P_2 T\left[\left|D_p - D_q\right| > 1\right]\right) \quad (24)$$

While the first term represents the summation of all pixel matching costs for the disparities of D, the second term uses P_1 to penalize small disparity changes (1 pixel) in all pixels q in the neighborhood N_P of p and the third term uses P_2 to penalize greater disparity changes. At this point, stereo matching consists of finding the disparity D that minimizes the energy $E(D)$. Since it is a 2D global minimization which is NP-complete (i.e. there is no known efficient path to find a solution), the alternative requires aggregating matching costs in 1D from all directions equally. The aggregated cost $S(p, d)$ for a pixel p and disparity d is determined by adding all the costs of all 1D minimum cost paths that end in that same pixel p at the same disparity d (***Figure 3***).

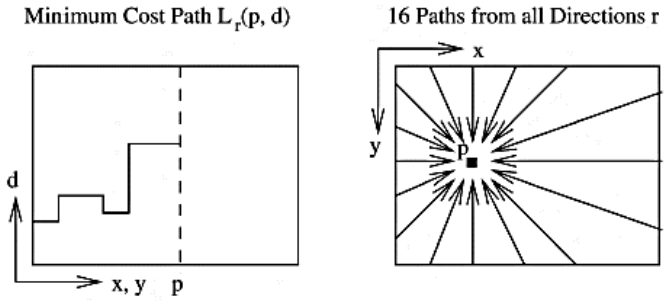


Figure 3. Aggregation of costs in disparity space through paths projected as straight lines

The cost *Lr (p, d)* along a path traversed in the direction *r* of the pixel *p* at disparity *d* is a recursive function described in equation (25). The third term corresponds to the minimum path cost of the previous pixel and it is subtracted to prevent that the values of *Lr (p, d)* increase permanently across a path.

$$\boldsymbol{L_r}(p,d) = \boldsymbol{C}(p,d) + \min(A) - min_k\ (\boldsymbol{L_r}(p-r,k) \qquad (25)$$

where $A = (\boldsymbol{L_r}(p-r,d), \boldsymbol{L_r}(p-r,d-1) + P_1, \boldsymbol{L_r}(p-r,d+1) + P_1, \min_i(\boldsymbol{L_r}(p-r,i) + P_2)$ (26)

The third stage of the method is the disparity computation. The disparity image D_b that corresponds to the image I_b is obtained by choosing the disparity d with the minimum cost for each pixel p. The disparity image D_m that corresponds to the image I_m is determined by using the same costs along the epipolar line of pixel q in the match image. Calculating D_b and D_m allows the detection of occlusions and false matches by comparing D_b and D_m. The disparity is defined as invalid is equation (27) is not satisfied:

$$D_p = D_{bp} \quad \text{if } |D_{bp} - D_{mq}| \leq 1 \qquad (27)$$

The fourth stage is multibaseline matching which is optional since it is an extension of the original method and is performed by calculating a combined pixelwise matching cost of all the correspondences between the base image and the match images.

The fifth and final stage is the disparity refinement which can be achieved by using three techniques: peak filtering to eliminate outliers, intensity consistent disparity selection when foreground objects are in front of an untextured background and interpolation for preserving discontinuities.

As it is detailed in section IV, this method is implemented in OpenCV's SGBM class with a few modifications:

- Since the algorithm is single-pass, only 5 directions are considered instead of 8. By setting the class mode to **StereoSGBM :: MODE_HH**, all directions can be included but with the cost of more memory consumption.
- The original matching is performed pixelwise but the algorithm matches blocks. Though, setting the block size to 1 would reduce it to a pixel.
- A Birchfield-Tomasi sub-pixel metric is implemented instead of the MI cost function.
- Some pre- and post- processing filters are included such as Sobel filter, quadratic interpolation and speckle filtering.

D. Real-Time Disparity Computation Using Variational Method

An optic flow is a 2D displacement field of corresponding pixels that originates from consecutive frames in an image sequence. Variational methods can be used to compute such displacement which is directly related to a 1D disparity field. Sergey Kosov's approach [6] is detailed here.

Given a stereoscopic pair of images I_1 and I_2 and the disparity *u (x, y)*, the grey value constancy assumption can be written as:

$$I_1(x,y) - I_2(x + u(x+y), y) = 0 \qquad (28)$$

Since real-time disparity may not necessarily be obtained as integer values, linear interpolation is applied to express disparity as a sum of an integer component A and a floating-point component $b < 1$.

$$u(x,y) = A(x,y) + b(x,y) \qquad (29)$$

By replacing equation (18) into (17), the latter's linearized form is described in equation (30).

$$I_1(x,y) - |b(x,y)| \,.\, I_2(x + A(x,y), y) - (1 - |b(x,y)|) \,.\, I_2(x + A(x,y) + sign(b(x,y), y) = 0 \qquad (30)$$

This leads to an energy function with two terms: a data term that assures grey value constancy and a smoothness term that normalizes the first term's local solution. The data term is derived from the grey value constancy assumption of equation (28) as the initial form of the energy functional:

$$E\big(u(x,y)\big) = \iint_\Omega \|I_1(x,y) - I_2(x + u(x+y), y)\|^2 \, dS \qquad (31)$$

The smoothness term (equation (32)) intends to eliminate local disparity outliers considering neighboring regions as a part of the same object and therefore having similar disparity values.

$$\boldsymbol{\psi}(|\nabla u(x,y)|^2) \qquad (32)$$

The energy functional can then be reformulated by adding the smoothness term:

$$E\big(u(x,y)\big) = \iint_\Omega \|I_1(x,y) - I_2(x + u(x+y), y)\|^2 + \varphi . \boldsymbol{\psi}(|\nabla u(x,y)|^2) \; dS \qquad (33)$$

Table 1 describes the three regularization functions included in OpenCV implementation **StereoVar**. While the Tikhonov method assumes overall smoothness, the other two methods assume piecewise smoothness and maintain discontinuities.

Regularization	$\psi(s^2)$
Tikhonov	s^2
Charbonnier	$2\lambda^2 \sqrt{1 + \frac{s^2}{\lambda^2}} - 2\lambda^2$
Perona-Malik	$\lambda^2 \, ln(\lambda^2 + s^2) - \lambda^2 ln(\lambda^2)$

Table n° 1. Regularization functions

E. Efficient Large-Scale Stereo Matching

Geiger et al. [7] propose a probabilistic model called ELAS (**E**fficient **LA**rge-**s**cale **S**tereo) which can perform dense matching by using small aggregation windows. The algorithm is split into three phases. The first phase involves defining a set of support points, i.e., pixels that can be robustly matched by taking their texture and uniqueness. The authors state that the most efficient way to match support points on a regular grid is achieved by using the l_1 distance between vectors formed by concatenating both the horizontal and vertical responses of a Sober filter in 9x9 pixel windows. To assure robustness, consistency is checked from left-to-right and right-to-left.

The second phase consists of the probabilistic model itself which uses a reference image and the previously mentioned support points to draw samples from the other image. ***Figure 1*** shows a graphical description of the method.

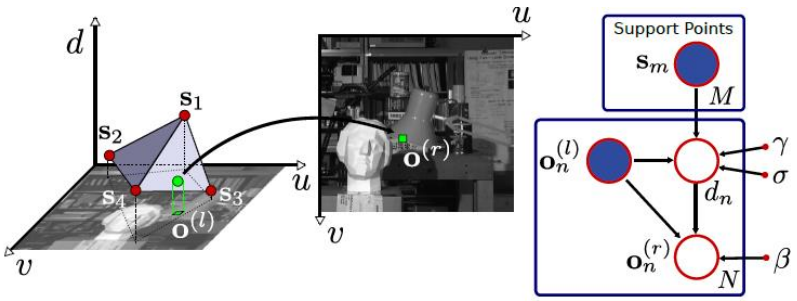


Figure 4. Sampling process and graphical model. *Given a set of support points S and an observation on the left image* $o_n^{(l)}$ *, a disparity d is computed. With the disparity and the observation on the left image, an observation on the right image* $o_n^{(r)}$ *is drawn.*

The set of support points $S = \{s_1, \dots, s_M\}$ is defined as having each support point $s_m = \{u_m, v_m, d_m\}^T$ described by its image coordinates (u_m, v_m) and its disparity d_m. A set of image observations $O = \{o_1, \dots, o_N\}$ is also defined with each observation $o_n = \{u_n, v_n, d_n\}$ described by its image coordinates (u_n, v_n) and a feature vector f_n (pixel's intensity or low-dimensional descriptor of a small neighborhood). Stating that the observations $\{o_n^{(l)}, o_n^{(r)}\}$ on the left and right images respectively and the support points S are conditionally independent, the joint distribution of the model is proposed in equation (34).

$$p(d_n, o_n^{(l)}, o_n^{(r)}, S) \propto p(d_n | S, o_n^{(l)})\, p(o_n^{(r)} | o_n^{(l)}, d_n) \qquad (34)$$

The joint distribution is factorized as the product of the prior (first term) and the image likelihood (second term). Basically, the prior is proportional to the combination of a uniform distribution and a sampled Gaussian. The image likelihood is a constrained Laplace distribution related to the feature vectors of the left and right images. As shown in Figure 1, the samples from the corresponding observation in the right image are generated in two steps:

1. By taking S and $o_n^{(l)}$, a disparity d_n is drawn from $p(d_n | S, o_n^{(l)})$
2. By taking $o_n^{(l)}$ and d_n, an observation $o_n^{(r)}$ is drawn from $p(o_n^{(r)} | o_n^{(l)}, d_n)$

The third and final phase of the approach lies on estimating the disparity map by using the maximum a-posteriori function of equation (35) where $o_1^{(r)}, \dots, o_N^{(r)}$ englobes all observations in the right image located in the epipolar line of $o_n^{(l)}$.

$$d_n^* = argmax\, p(d_n | o_n^{(l)}, o_1^{(r)}, \dots, o_N^{(r)}, S) \qquad (35)$$

As the joint distribution was factorized in the second phase, the distribution inside the *argmax* function is split into two distributions as seen in equation (3). By using the analysis in the previous phase, the first term leads to an energy function which can be minimized to obtain a dense disparity map. The second term is modelled as the sum of the distributions of all the observations along the epipolar line on the right image for each d_n.

$$p(d_n | o_n^{(l)}, o_1^{(r)}, \dots, o_N^{(r)}, S) \propto p(d_n | S, o_n^{(l)})\; p(o_1^{(r)}, \dots, o_N^{(r)} | o_n^{(l)}, d_n) \qquad (36)$$

F. Filtering Methods

a) Bilateral Filter

A bilateral filter [8] combines both domain and range filtering instead of just focusing on the image domain as traditional filters do. The pixel values are replaced by similar or close-by pixel values. In smooth areas, neighboring pixels are similar and bilateral filtering uses standard domain filtering to discard poorly correlated pixels which correspond to noise. Equation (37) defines the bilateral filtering method and equation (38) defines the normalization term *k*.

$$I_{filtered}(x) = \frac{1}{k(x)} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|) \quad (37)$$

$$k(x) = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|) \quad (38)$$

$I_{filtered}$ is the resulting filtered image from the original image I, x are the coordinates of the current pixel, Ω is the x-centered window, f_r is the range kernel for intensity smoothing and g_s is the spatial kernel for coordinate smoothing. The last two components represent the bilateral filter's main advantage since it can achieve good boundary filtering due to the domain component while preserving crisp edges due to the range component.

b) Gaussian Filter

The shift-invariant Gaussian filter [9] is a specific case of the bilateral filter where both kernels are Gaussian functions of the Euclidian distance between their arguments. Basically, it can be described as the product of the two 1D functions previously mentioned as equation (39) shows.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \quad (39)$$

c) Median Filter

In opposition to the Gaussian filter which performs a linear operation, the Median filter [10] is non-linear. As its name implies, this method replaces the pixel value with the median of neighboring pixel values. The median is determined as the middle value of a numerically ordered sequence. If the total of numbers in the sequence is even, the median is equal to the average of the two values closest to the middle. It is widely used to remove noise and preserving edges simultaneously.

G. State of the Art

a) Project Tango

Developed by Google, Project Tango [11] is an AR (augmented reality) platform that detects mobile devices without requiring GPS or other external signals. It takes advantage of computer vison techniques and allows developers to program indoor navigation, virtual reality, 3D mapping and environmental recognition applications. The Tango technology was first available on a commercial level on Lenovo´s Phab 2 Pro [12] last year. It has a Snapdragon 652 processor, an 8 Megapixels rear-view camera and internal storage of 64 GB. The Asus Zenfone is the second smartphone to feature Tango Technology and is set to be released around summer 2017.

b) Kinect for Windows

Microsoft's Kinect [13] for Windows is another platform where developers can build their interactive experiences with computer vision software. The main features of its SDK are body and face recognition and tracking, multi-app support on a single sensor and Unity pro compatibility. Users can even publish and sell their own creations on the Windows Store.

c) Real Sense Technology

Intel's Real Sense technology [14] integrates a variety of sensing techniques to enable 3D imaging, interior mapping and feature tracking. Regarding the field of Robotics, it can recognize people and objects and navigate the surrounding environment which can be implemented onto smart drones. It can also deliver 3D models through scanning and even offers a project creation tool for visual sensor and Robotics applications.

IV. DEVELOPMENT PROCESS

A. Resources

The three stereoscopic cameras considered as resources were Leopard Imaging (LI) INC.'s USB30-V024 model (***Figure 5***) [15], DUO 3D's MLX model (***Figure 6***) [16] and Minoru 3D (***Figure 7***) [17]. Although the DUO MLX is a more compact model, the Leopard Imaging camera was initially chosen due to its compliance with the UVC protocol and the fact that the DUO MLX's drivers were very expensive. However, the Leopard model was replaced with the Minoru 3D since there were some unsolvable exposure control issues with the former camera which will be explained later.

LI-USB30-VO24 STEREO

- MT9V024 Aptina global shutter WVGA sensor
- Dimensions: 80 mm x 15 mm x 17 mm
- Active pixels: 752H x 480V
- Sampling rate: 30 FPS
- Image format: 1/3"
- Focal length: 2.35 mm
- Pixel size: 6.0×6.0um
- Supports stereoscopic mode
- Raw data. USB 3.0 real-time transmission
- Power supply: USB 3.0 +5V DC
- Weight: 12 g

Figure 5. LI-USB30-V024 STEREO camera

DUO MLX

- DUO compact sensor unit
- Dimensions: 52,02 mm × 25,40 mm × 13,30 mm
- Sampling rate: 0,1 – 3000 FPS
- Focal length: 2,0 – 2,1 mm
- Pixel size: 6.0x6.0 µm
- Shutter speed: 0,3 µsec – 10 sec.
- Field of view: 170° wide angle lens and less than 3% distortion
- Filters: 850 – 870 nm narrow pass-band
- Power consumption: +/- 2,5 W @ +5 V_{DC} (USB).
- Interfaces: 480 Mbps – USB 2.0 (Micro USB)
- Weight: 12,5 g.

Figure 6. DUO MLX camera

Minoru 3D

- VGA CMOS Sensor
- Maximum frame rate: 30 FPS
- Maximum resolution: 640 × 480
- Baseline: 6 cm
- Field of view: 42°
- Manual focus: From 10 cm to infinity
- No synchronicity between L/R shutters with a maximum deviation of 16.5 ms
- Automatic USB detection as individual cameras
- Aperture always open (recommended for indoors)

Figure 7. Minoru 3D camera

The two options considered for the SoC architecture were the ODROID XU-4 [18] and the NVIDIA's Jetson TK1 [19] models. The latter was finally chosen due to its CUDA development platform which is widely used in computer vision, robotics, medicine and other fields. NVIDIA's products are becoming more and more popular to the point that is has been confirmed that Nintendo's new home console, the Nintendo Switch is powered by NVIDIA technology [20].

B. Leopard Imaging Camera

a) Controlled Capture Algorithm

Generally, computers have a basic application for capturing images with the built-in webcam which serves the same purpose for monocular cameras connected through the USB port. However, it is not the case for most stereoscopic cameras where programs only display right and left pictures interleaved in one frame. Therefore, to allow the appropriate processing of the images coming from each lens individually, a controlled capture algorithm was designed in C++ taking full advantage of the camera's compliance with the USB video device class protocol (also known as UVC) [21].

Basically, the sample file *video_proc.cpp* included in OpenCV was adapted to enable the controlled capture of a fixed number of images. A class named **Controlarcaptura** was created with the variables and functions described in ***Table 2***. Following the conventional class format of C, a header file (*.h*) and a definition file (*.cpp*) were created. The maximum number of photos captured can be modified by simply changing the threshold of the variable *contador*.

	Declaration	**Definition**
Variables	int contador	Number of times that the 'Space' key is pressed
	uvc_device_handle_t *devh	Device controller variable
	uvc_device_t *dev	Device controller variable
	uvc_context_t *ctx	Context variable
Functions	**void cuenta (void)**	Counts the number of times that the 'Space' key is pressed and goes to the **stop** function when the desired value is reached.
	void stop (void)	The UVC controller is stopped
	void cargarinfo (void)	Variables *dev*, *devh* y *ctx* are updated to correctly close the UVC.

Table n° 2. Variables and functions within Controlarcaptura

b) Camera calibration procedure in MATLAB

The initial method used to calibrate the LI camera was MALTAB R2016A's *Stereo Camera Calibrator* app [22]. The first step of the calibration protocol consists on taking at least 10 pairs of stereoscopic pictures. ***Figure 8*** shows the pattern used for calibration; placed on glass, it is a 7 x 9 chess-like grid with a square size of 27.5 mm. It is recommended for the selected pattern to be asymmetrical, i.e., it should at least have a different number of rows and columns. ***Figure 9*** shows an example of images (left and right) used for calibration. All images were stored in a gray-scale JPEG format since that's the camera's output format. 19 pairs of images were taken in total.

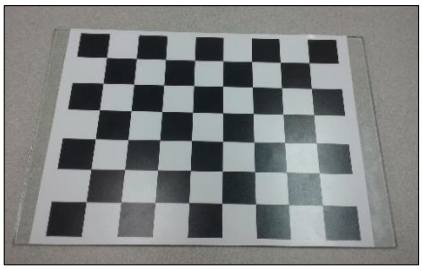


Figure 8. Calibration pattern

After storing the captured images, left and right images must be separated into different folders since the program receives them individually. The names of the pictures must be consecutive numbers which are the same for each pair so that the algorithm can match them correctly: an example would be *leopard_left000.jpg* and *leopard_right000.jpg*. Apart from the pictures, the only input needed by the app is the square size in integer world units (mm, cm or inches) which in this case is 27 mm.

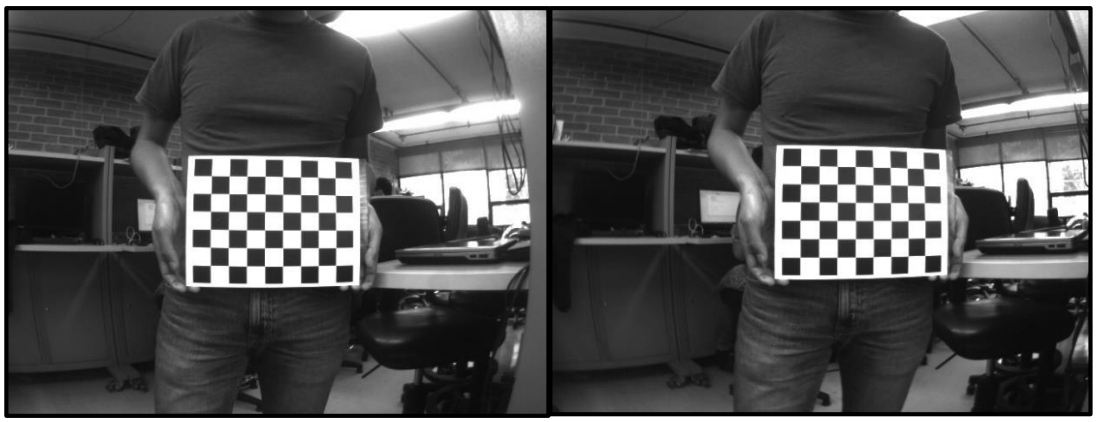


Figure 9. Example of stereoscopic images (left and right) taken for calibration

Figure 10 shows a pair of the stereoscopic images processed with the application. Although 19 pairs were processed, four of them were rejected still leaving 15 pairs for calibration. The upper row of images corresponds to the detection of the inner corners of the pattern and the definition of the coordinate system. The lower row corresponds to the rectification output for each image. It is noticed that the checkerboards' points are correctly detected and that the corresponding reprojected points match exactly with the original coordinates. Moreover, the rectification is validated graphically when the radial distortion is greatly reduced. In this figure, the rectified images were adjusted to coincide with the other images; the rectification process delivers bigger images since it is literally stretching pixels.

Figure 11 shows the results obtained from the calibration process. ***Figure 11.a.*** corresponds to a bar chart representing the mean reprojection error in pixels for each image. The overall mean error can be detailed as well, which in this case, is 0.1794 pixels. A value of the error lower than 1 pixel is considered adequate. ***Figure 11.b.*** shows the extrinsic parameter visualizations in the camera-centric mode. While MATLAB also offers a pattern-centric view which is useful when the pattern is stationary, the camera-centric view is the one needed in this case since the camera was motionless during the capture. These views are meant to confirm that there are no substantial errors during calibration, i.e., an error would be indicated by a pattern behind the camera.

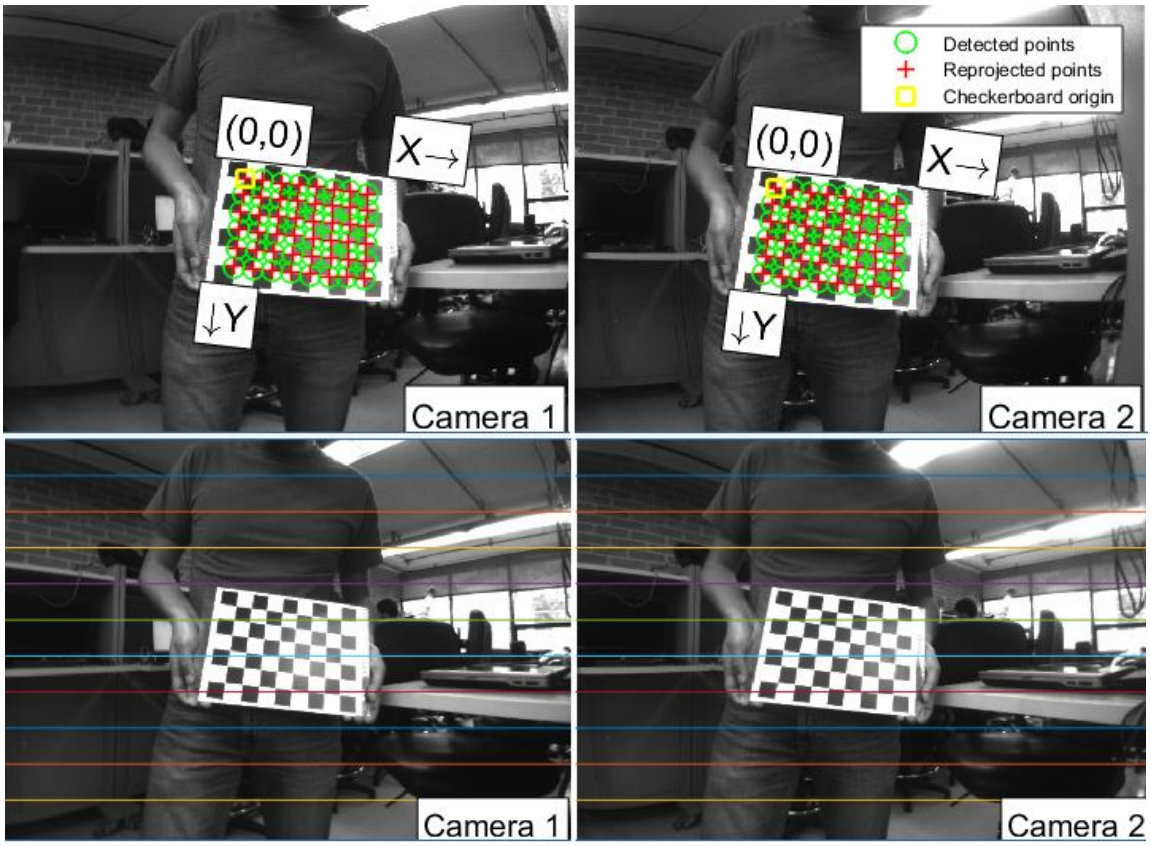


Figure 10. Stereoscopic camera calibration

- ***Upper row: Point detection in stereoscopic images (left and right)***
- ***Lower row: Stereoscopic image rectification (left and right)***.

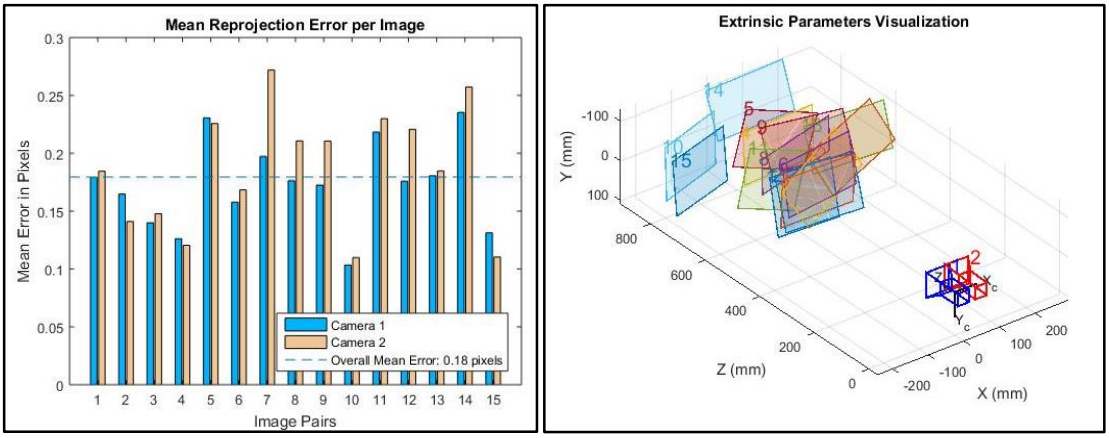


Figure 11. Stereoscopic camera calibration results
a. Mean reprojection error in pixels and overall mean error
b. Extrinsic parameter visualization (camera-centric view)

After validating that the calibration was performed correctly, the next step consists of extracting the camera parameters obtained so they can be used in the real-time implementation. The *Stereo Camera Calibrator App* delivers a structure called *stereoParams.mat* that includes different variables related to the stereoscopic cameras such as their radial and tangential distortion, the rotation and translation of the second camera in respect to the other one, the fundamental matrix and the essential matrix. The two last variables are matrices that define the mathematical correspondences between both cameras.

For the following stages, only the camera parameters are needed. When performing the calibration process, only two radial distortion coefficients were considered since the third coefficient applies for when the camera has a wide field of view. Since it is assumed that the X and Y axis are perpendicular the skew was not computed. Tangential distortion was not computed either because the lens' principal axis is supposed to be perpendicular to the camera sensor. ***Table 3*** includes all the parameters extracted for both cameras. They are all rounded to four decimals but they were exported to the algorithm with 13 decimals for higher precision.

Parameter		Camera 1	Camera 2
Radial Distortion	K_1	-0.4074	-0.4103
	K_2	0.1855	0.2092
Focal Length	F_x	442.7831	437.2752
	F_y	441.0206	435.6438
Principal Point	C_x	331.8848	330.2686
	C_y	230.4943	233.7124

Table n° 3. Stereo camera parameters obtained from the calibration process

These parameters are exported to the capture and rectification algorithm which uses OpenCV's function *undistort* which corrects lens distortion on the raw input images. The values of Table 2 must be inserted into the camera matrix *M* (equation 40) and the distortion coefficients vector *dist_coef* (equation 41) which serve as the function's inputs. p_1 and p_2 correspond to the tangential distortion coefficients which are both equal to 0 since they were not computed as was previously mentioned.

$$M = \begin{bmatrix} f_x & 0 & c_y \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad (40)$$

$$dist\_coef = (k_1, k_2, p_1, p_2) \qquad (41)$$

c) Camera calibration procedure in OpenCV

Since the disparity must be determined in real-time, MATLAB's calibration parameters were only useful to test the matching algorithms on previously rectified images. To calibrate the stereo camera in OpenCV [23], the sample file *stereo_calib.cpp* was used. ***Figure 12*** gives an overview of the main stages followed by the method.

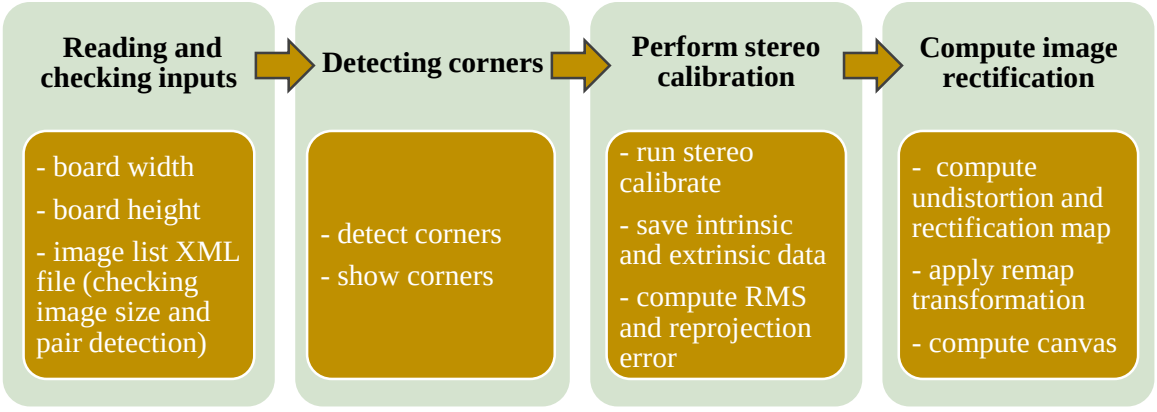


Figure 12. Main stages of the OpenCV stereo calibration method used for the Leopard camera

To build and compile the project, Cmake is used since it is straightforward specially in an Ubuntu environment. After creating a folder for the project and copying the *stereo_calib.cpp* file inside it, the next step is to create a file called *CMakeLists.txt* where the project name is defined and the needed OpenCV

structures, C/C++ files and libraries are referenced. Using the terminal window when located in the project folder, the following sequence of commands is introduced to create and compile the project:

- **mkdir build:** Create a directory
- **cd build:** Change current directory location to folder build
- **ccmake .. :** Compile the project and create *CMakeCache.txt* which is basically a configuration file
- **make:** Create an executable file with the name given in *CMakeLists.txt*

This command sequence works for any project compilation that has all required files in the same folder. The two main differences from one project to another are the files and libraries included in *CMakeLists.txt* and the command used to run the executable file. In this case, stereo calibration is run by using:

./stereo_calib -w 14 -h 12 ./stereo_calib.xml

The first field is the name of the executable file followed by the width (***w***) and height (***h***) of the calibration pattern given in terms of the inner corners. The pattern used during this calibration procedure differs from the previous one since the algorithms available in OpenCV require that the pattern occupies at least one third of the scene so it can be processed properly. Although this was the pattern initially built for calibration, the first pattern shown offered more accurate results in MATLAB while the second was selected for OpenCV due to its size. The only modifications made to the sample file were related to the size of the pattern squares in mm (60 mm) and the validation of the flags that shows the corner detection and image rectification outputs. *stereo_calib.xml* is an xml file containing an ordered list of the input images.

Figure 13 is an example of the inner corner detection routine. It is noticeable that all corners are correctly detected and that the detection is performed one row at a time and then heading to the beginning of the next row. ***Figure 14*** corresponds to the rectification canvas computed during the last stage of the algorithm. The red rectangles delimit the regions where all pixels are valid between the left and right images. The error values obtained were 0.6483 for the RMS error and 0.443872 for the average reprojection error which is still appropriate since it is still lower than 1.

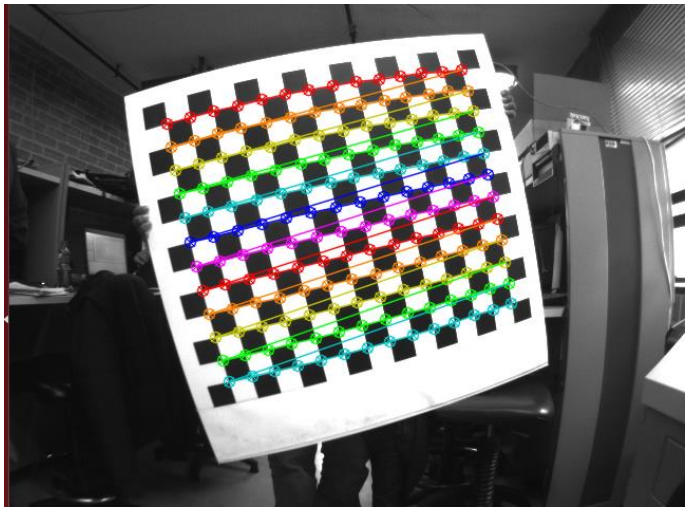


Figure 13. Corner detection example using stereo_calib

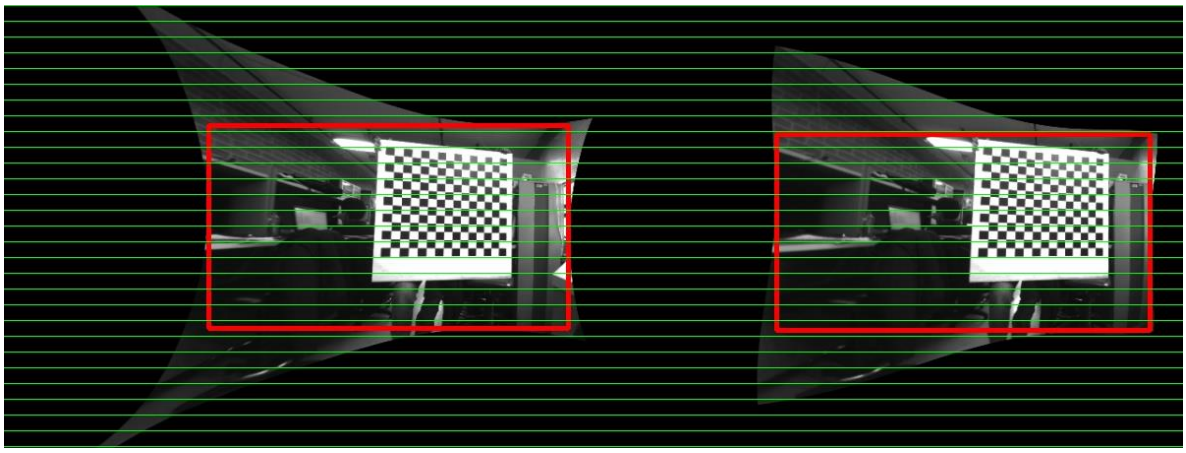


Figure 14. Rectification routine output – 'canvas' using stereo_calib

d) Camera calibration in ROS

As ***Figure 13*** shows, the rectification output from OpenCV is not accurate enough which can be evidenced by a clear difference in the distance of the pattern's upper left corner to the upper border of the rectangle. Moreover, the area outside the rectangle is dramatically stretched so that the valid pixel regions barely coincide. This lead to search for another calibration alternative.

ROS is an open-source operating system destined for robot implementations and includes a wide range of functions and tools necessary in computer vision. ROS' calibration tool [24] is used to extract new stereoscopic camera parameters that improve the rectification previously obtained with OpenCV. Although ROS is not a real-time framework, its main advantage lies in the capacity of grouping processes into packages and stacks. It currently runs only on Unix-based platforms which is convenient to this project.

ROS calibration is performed by launching a node called *camera_calibrator.py.*
The only requirements for the algorithm are the following:

- The pattern must be 8x6 or 7x6 checkerboard with known square dimensions.
- It must be hold horizontally, i.e., more checkers horizontally than vertically
- A well-lit area with no obstructions and no other patterns
- A stereo camera capturing images in ROS

The calibration node is run using a command (***Figure 15***) similar in structure to the one in OpenCV. This opens up a calibration window that allows to detect corners from live video instead of receiving the captured images like MATLAB and OpenCV. Another difference from the other methods used is that multiple square sizes can be introduced by adding more *–size and --square* values. The node should recognize the other sizes if they are noticeably different.

```
rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.027
right:=/my_stereo/right/image_raw left:=/my_stereo/left/image_raw right_
camera:=/my_stereo/right left_camera:=/my_stereo/left
```

Figure 15. Run calibration command for ROS

Figure 16 shows the calibration interface. When enough data is captured, the CALIBRATE button will light and it can be clicked. Afterwards, the calibration window is updated with the final results. The values for the epipolar error and the dimension of the square size can be seen in the right-most column. An epipolar error is considered acceptable if it is below 0.25 pixels and excellent if it is below 0.1 pixels. By using the upper bar to change the scale, the user can narrow the image view going from the valid pixel region (scale 0.0) to the entire image (scale 100.0).

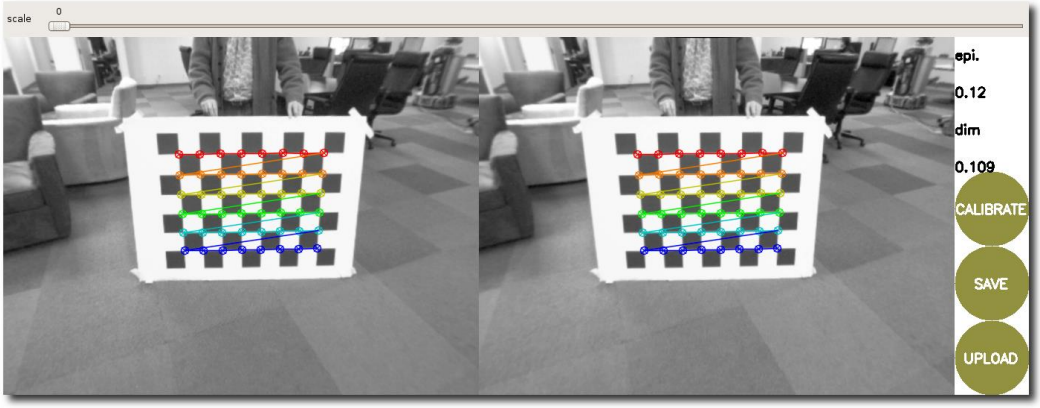


Figure 16. ROS stereo calibration interface

The camera parameters obtained with this method need to be inserted into the C++ rectification algorithm hence replacing the parameters obtained with the *stereo_calib.cpp* project. There is a difference between both sets of parameters regarding their type and the functions they need. ***Table 4*** lists the parameters given by OpenCV and ROS and the OpenCV function declarations they need to rectify the input camera images.

	OpenCV (*stereo_calib*)	**ROS**
Calibration Parameters	- Camera matrices - Distortion coefficients vectors - Rotation matrix between camera coordinate systems - Translation vector	- Camera matrices - Distortion coefficients vectors - Projection matrices - Rectification matrices
OpenCV functions required	- stereoRectify - initUndistortRectifymap - crop - remap	- initUndistortRectifymap - remap

Table 4. Camera parameters and functions involving OpenCV and ROS calibration methods.

The only common parameters between both approaches are the camera matrices and the distortion coefficients (*stereo_calib* gives eight distortion coefficients and ROS gives five). The two main reasons that explain why *stereo_calib* did not offer an optimal rectification outcome is that the valid regions of interest (ROI) computed by the *stereoRectify* function were not exactly of the same size which lead to an inaccurate cropping with the function *crop* hence an inaccurate remapping of the rectified images. In contrast, the ROS calibration node directly delivers the parameters needed by the *initUndistortRectifymap* function and bypasses the previous ROI manipulation. The ROIs obtained with the latter had the same size.

C. Minoru 3D camera

Although accurate rectification was achieved with Leopard Imaging's camera, the functions used to control basic variables such as exposure, gain or even hue never managed to influence the amount of light caught by the camera feed. This "blinded" the visual sensor which led to many errors in entire regions of the disparity map. Errors were more prominent in white objects or walls or even bright sections of the scene. Some of the *libuvc* [25] functions used were:

- **uvc_set_ae_mode:** enables auto-exposure
- **uvc_set_gain:** set gain value
- **uvc_set_hue:** set hue
- **uvc_set_backlight_compensation:** set backlight compensation value

The GTK UVC video viewer [26] was installed on both the computer and the SoC to try to manually adjust the exposure and gain. However, this only worked during GTK's live display and did not affect the OpenCV feed even without disconnecting the camera. Hence, the Minoru 3D webcam was purchased due to the extensive amount of documentation available regarding stereo applications with that specific device. Additionally, the Minoru includes a built-in exposure control which is extremely useful in disparity computation methods.

a) Camera calibration and rectification

To calibrate the Minoru model, the following steps are required:

1. Capture calibration pattern images (at least 10 pairs)
2. Calibrate both cameras separately (as monocular cameras)
3. Compute the "conjoint" stereo calibration parameters
4. Rectify the input images and compute the remapping grids

The overall process is very similar to the one performed with the Leopard Imaging sensor but the foremost difference is that both left and right cameras must be calibrated individually at the start. This assures that the translation matrix T is properly calculated in the stereo calibration which can be confirmed when the first element of vector $T^{3 \times 1}$ in fact corresponds to the camera baseline. R is also obtained which aligns the imaging planes from the cameras. The main reason for this is to compensate for any small manufacturing defects which can lead to such planes not being exactly vertically aligned nor parallel to each other.

Figure 17 shows the three main stages of the calibration process used for the Minoru camera. The calibration and rectification codes are completely detailed in [27]. The first stage involves the individual calibrations for left and right visual sensors. The algorithm (*code 9.1*) receives as inputs the camera images, the number of inner corners of the pattern and its square size in mm. It must be run once for each camera (left and right) and delivers as output an XML file which includes the camera matrix (3×3) and the distortion coefficients vector (1×5). The default name of the file is *cam_calib.xml*. Hence it must be renamed with the prefixes *left_* and *right_* so that they can be distinguished by the stereo rectification algorithm (*code 9.5*) in the second stage. It consists on the stereo calibration which uses the left and right XML files and images to calculate the translation and rotation matrices (*R* and *T*), the left and right camera matrices and distortion coefficients as well as the essential matrix *E* and the fundamental matrix *F*. The stereo calibration section as a stand-alone algorithm is available in *code 9.4* whereas both calibration and rectification are performed by *code 9.5*.

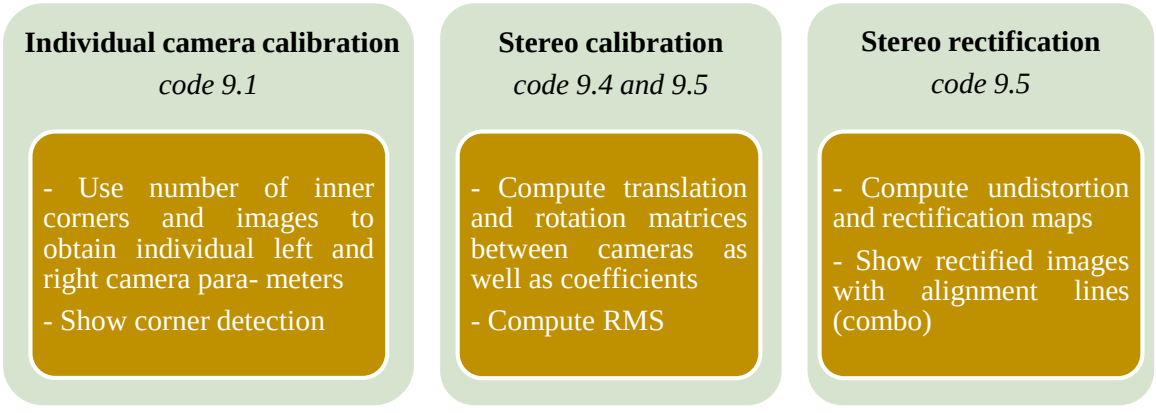


Figure 17. Main stages of the OpenCV stereo calibration method used for the Minoru camera

The third stage of rectification is continuously performed live so that the disparity computation could be performed live as well. ***Table 5*** describes the classes and functions defined during rectification. It is important to clarify that the concept of rectification can encompass the calibration process when working with stereoscopic cameras but they are set apart in this section to facilitate both their explanation and code organization. Therefore, rectification will strictly refer to actually rectifying left and right cameras. The disparity computation section was implemented within the *show_rectified* function.

	Classes	Functions
Calibration stage	- **Calibrator:** Contains the basic calibration parameters (camera matrices, rotation matrix, translation vector, essential matrix and fundamental matrix), image and object points, image dimensions (width and height) and folder paths.	- **Calibrator:** Reads left and right camera images. - **Calc_image_points:** Calculates the object points in the object co-ordinate system while drawing the chessboard corners for every pair. - **Calibrate:** Reads individual left and right camera calibration parameters and computes the basic calibration parameters. - **Save_info:** Stores the computed parameters.
Rectification stage	- **Rectifier:** Contains the pixel maps for rectification and the folder path of the stereo calibration XML file.	- **Rectifier:** Reads individual left and right camera calibration parameters, calculates transforms to rectify the images and calculates the pixel maps for rectification. It also stores them. - **Show_rectified:** Adjusts frame size, remaps images with the pixel maps and combines rectified images in one window display.

Table 5. Classes and functions used in the Minoru rectification method

b) Disparity computation

The chosen disparity computation method was OpenCV's SGBM class from [2]. The program runs at 15 FPS with an image size of 320 x 240 pixels which corresponds to the same size used for calibration. 15 FPS is the default frame rate of the Minoru device and is also the minimum rate needed to satisfy the project's implementation requirements. The algorithm is intended to be implemented over the AscTec Firefly [28] drone (***Figure 18***).

Figure 18. AscTec Firefly UAV model

Its maximum acceleration is 3.6 m/s^2 but only 20% is normally used to avoid damaging it. By setting a minimum measured distance of 20 cm, the minimum braking distance is defined as half of such distance, i.e., 10 cm. With the uniform lineal acceleration formula (42), the maximum initial speed can be estimated. Hence, a distance of 10 cm can be traversed by the drone in 69.44 ms which leads to a processing rate of at least 14.4 FPS. So, the frame rate is defined as 15 FPS for the disparity computation method.

$$V_i = X_f \times 2a = 10\,cm \times 2 \times 7.2\,\frac{m}{s^2} = 1.44\,m/s \quad (42)$$

Figure 19 exemplifies the corner detection process of an image pair. The calibration is performed each time the program is executed so that the user can update the rectification by simply changing the individual camera files and pictures. The user must press the *'Space'* key to go from one pair to the next one. After all stereoscopic pairs have been processed, the program will output the RMS calibration error and then display the live camera feed with the following data (***Figure 20***):

- **Combo:** The live stereo camera feed is rectified and both left and right camera views are juxtaposed in one window (***Figure 20.a***). Red horizontal lines are plotted to facilitate the visualization of the rectification's effect. It also helps the user to adjust the camera position; if one point in the left camera intersects with a red line, that same red line should intersect with the corresponding point in the right

camera. Hence, if the camera is horizontal to the plane (the camera eyes are not inclined), all points in one camera should be at the same "height" as their corresponding images on the other camera. This validates the rectification process as the figure shows.

- **Disparity:** The second display (***Figure 20.b*** **-** *left*) is the disparity map obtained with semi-global matching. The values used for the class variables were chosen after several trials where each parameter was modified one at a time. The only parameter that is calculated is the number of disparities with equation (43). In this case, image width is 320 pixels and the number of disparities is 64 which is big enough for proper precision and small enough to avoid unnecessary straining of the SoC device which may slow down the overall performance of the method. This is critical on programs running on a frame-by-frame basis.

$$num_{disparities} = \left(\frac{image\ width}{8}\right) - 15 \sim 16 \quad (43)$$

- **Cropped disparity:** A drawback of the SGBM algorithm is that a portion of the disparity map is not computed as can be seen in ***Figure 20.b*** **-** *right*. The reason for this is that the right camera cannot "see" the left-most part of the scene shown by the left camera. Therefore, the method has no information to compute the disparities on that section. The size of such leftmost area depends directly on the camera baseline (6.0 cm) and the total image width (320 pixels). Based on the images captured during testing, the section's width was defined as 45 pixels and was used to crop the SGBM disparity output into 275 × 240 pixel images.

During the initial tests with the Leopard model, bilateral, median and Gaussian filters were applied to the output disparity maps. The median filter proved to be the most convenient to smoothen the overall result so it was also implemented on the final testing phase with the Minoru hardware. The pattern used for this calibration is the smaller one previously used in the first MATLAB calibration with the Leopard camera. The RMS calibration error was 0.271744 which is an appropriate value and lower than the RMS from the previous OpenCV method.

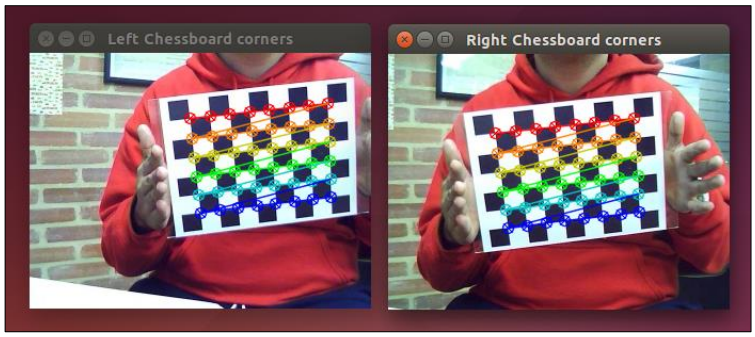


Figure 19. Pattern corner detection for stereo calibration

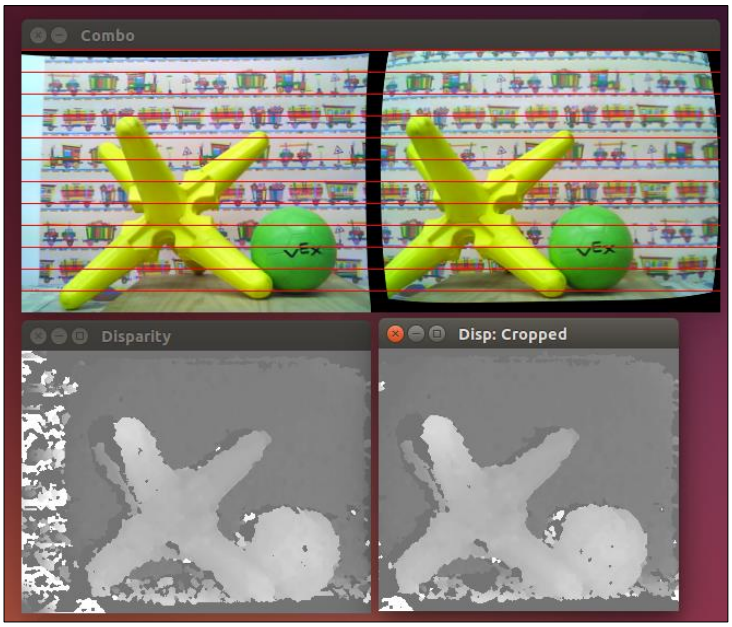


Figure 20. Live stereo camera feed displays
a. 'Combo': rectified view of both cameras
b. Disparity map (SGBM)
c. Disparity map (Cropped SGBM)

Figure 21 shows all the objects used to create the testing scenarios. They were chosen mainly based on the Middlebury [29] datasets and trying to have a variety of shapes, sizes, colors and textures. The background was formed by replicating an image with a highly-detailed colorful and consistent pattern to assure that the algorithm detects it as a flat surface. It was printed on matte paper to reduce the amount of light reflected on the Minoru sensor. An x-y coordinate grid was initially traced on the table surface to help locate the objects on the scene by defining 10 cm squares and setting the origin at the upper left corner. Since the Leopard camera was greatly affected by the exposure, covering the table with newspaper was intended to diminish reflected light on the sensors and to also avoid damaging the table with the grid traces.

Figure 21. Objects used on testing scenarios

Due to the Minoru camera's field of view (42°), the objects were separated into groups to define the four scenarios shown in ***Figure 22***. Going from left to right and top to bottom, the scenarios are numbered consecutively from 1 to 4 and will be referred to as ***Scenario 1***, ***Scenario 2***, etc. For each scenario, 30 pictures were taken to assure that enough disparity maps were analyzed and compared to the ground truth sensor data, i.e., 30 SGBM raw disparity maps and 30 SGBM disparity maps with median filter were captured on the SoC architecture.

Figure 22. Testing scenarios for experimental setup

c) Depth estimation

The resulting maps for each scene were processed in the MATLAB R2016A software. Using equation (23), the disparity values obtained from the rectified images of the scene were converted into real-world depth measurements. Although the camera baseline is already known (60 mm), the pixel size was estimated based on the image width and the focal length. The Minoru camera has a CMOS sensor [17] that measures across 3 mm which is the focal length and, if the image is focused over the entire sensor with a 320 × 240 resolution, the size for each pixel is 9.375 µm as can be seen in equation (44).

$$pixel\ size = \frac{focal\ length\ in\ mm}{focal\ length\ in\ pixels} = \frac{3\ mm}{320\ pixels} = 0.009375\frac{mm}{pixel} = 9.375\ \mu m/pixel \quad (44)$$

Figure 23 displays a couple of disparity maps obtained from ***Scenario 1*** with the raw image on the left and the map filtered with the median function on the right. At first glance, the effect of the filter is noticeable since it smoothens some of the areas with disparity errors. To extract depth values from the disparity maps, the definition of three specific target points on each scene was proposed. Although the centroid (mean position of all the points of an object in all the coordinate directions) property was considered as an alternative to extract points from the observed objects, it was not implemented since the centroids may be located in areas with abrupt disparity changes and, furthermore, they may have different coordinates from one image to another within the same scene. ***Figure 24*** is an example of the points chosen for ***Scenario 1***. For all scenarios, the points were named ***A***, ***B*** and ***C*** and at least one point per object was established while considering that it is not too close to a disparity "tipping point" as was just mentioned. The latter facilitates that the data is more uniform in the same scene. The objects' geometry was also taken into account when selecting the points' coordinates.

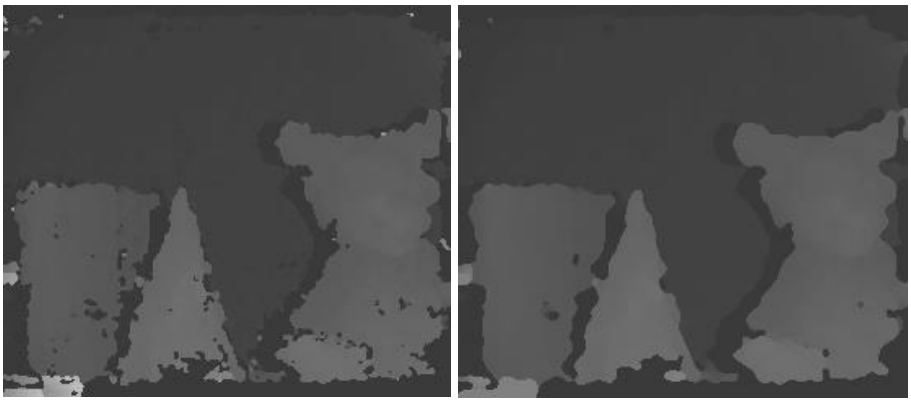


Figure 23. Disparity map from Scenario 1: raw (left) and with median filter (right)

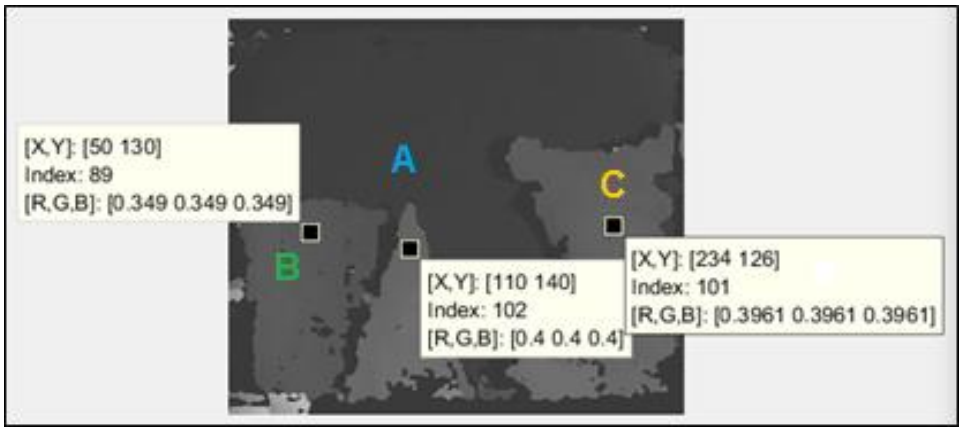


Figure 24. Disparity map with selected target points for Scenario 1

Before choosing the target points, all the images are read and converted into the RGB format with the function ***ind2rgb***. Since it is a grayscale image, the values for R, G and B are equal as ***Figure 24*** puts in evidence. This facilitates all the calculations for further processing. The depth is then computed for each target point with equation (45) where ***b*** is the baseline, ***f*** is the focal length in mm and ***p*** is the pixel size in mm/pixel.

$$Z = \frac{b \times f}{d_{RGB} \times p} = \frac{60\ mm \times 3\ mm}{d_{RGB} \times 9.375\ \mu m/pixel} \quad (45)$$

d) Depth comparison with ground-truth sensor

After converting the disparities into depth values, they can be compared to the data obtained from a state of the art depth-sensing sensor. The Kinect V2 sensor (***Figure 25***) was chosen to supply such task. Although the Intel Real Sense Camera [14] was considered, the Time-of-Flight (ToF) method used by the Kinect sensor [13] has proven to be more accurate than the structured-light approach used by the Real Sense sensor as detailed in [30]. On one hand, the structured-light strategy (which was installed of the first version of the Kinect) projects a set of previously known patterns on the object. The pattern is deformed by the object's shape and another camera observes it from a different direction. By using the distortion (disparity) of the pattern, the depth information is computed with equation (23). On the other hand, the ToF method measures the time it takes for an emitted light to go back and forth from the sensor array to the object. The underlying technique is called *Continuous Wave Intensity Modulation* which detects a time shift related to the distance from the object to the camera and the finite speed of light. Such distance is calculated with the registered time.

The Kinect V2 sensor only requires a USB 3.0 connection port and the installation of the Kinect for Windows SDK on the target computer. The sensor is compatible with MATLAB R2016A version and can be manipulated with its built-in image acquisition interface *imaqtool* (***Figure 26***).

Figure 25. Windows Kinect V2 Sensor

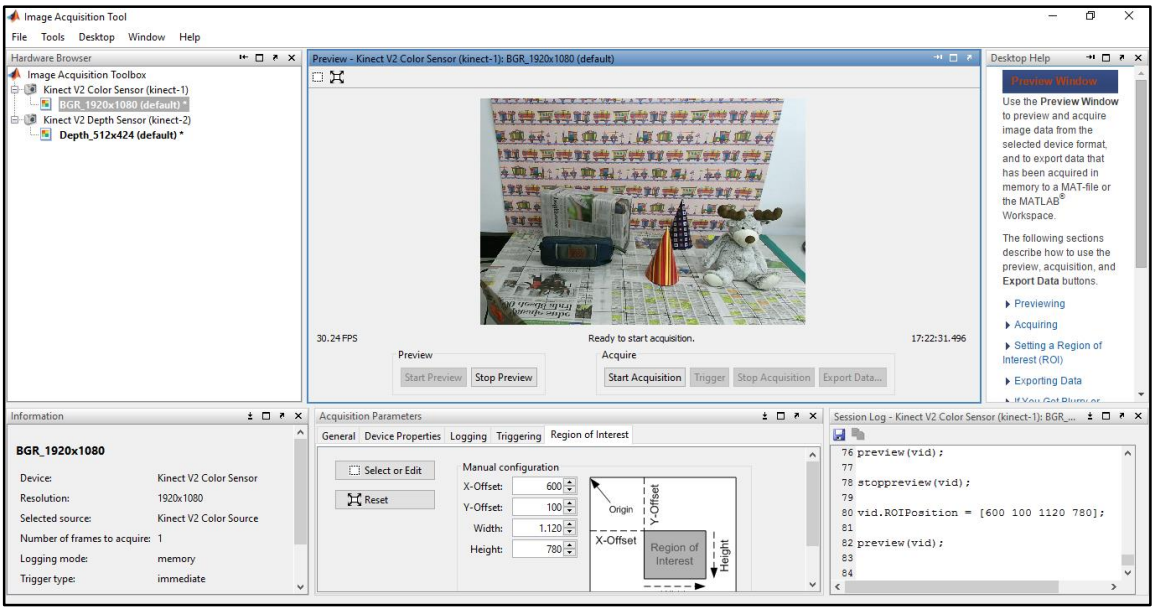


Figure 26. MATLAB image acquisition toolbox interface

The Kinect device has two sensors which are detected by the toolbox: the RGB camera which corresponds to the color (BGR) sensor and the depth sensor. Their default resolutions are 1920 x 1080 and 512 x 424 respectively. For both sensors, the options available include:

- ***General:*** The number of frames per trigger and the color space can be set here. Available color space options are grayscale, RGB, YCbCr and Bayer (50% green, 25% blue and 25% red).
- ***Device Properties:*** The user can reset default properties and enable or disable body tracking.
- ***Logging:*** Logs can be saved on disk, memory or both.
- ***Triggering:*** The number of triggers can be set here. The total number of frames captured corresponds to the product between the number of frames per trigger and the number of triggers chosen. The trigger can also be set as manual or immediate.
- ***Region of Interest:*** Depending on the scene, the region of interest (ROI) may need to be adjusted. This can be achieved by introducing an offset on axes *x* and *y* and modifying the image width and height. The '*Enter*' key must be pressed each time one of the parameters is modified to make it effective on the live feed

Since the Kinect's resolutions are much higher than the Minoru's resolution, the ROIs of the output from the depth sensor were manually adjusted to become as close as possible to the original Minoru disparity maps although reducing the regions does not modify in any way the distances measured by the sensor. The Kinect outputs (***Figure 27***) are horizontally flipped by default but it is not an predicament to manually define the target points ***A***, ***B*** and ***C*** based on the corresponding points defined on the SGBM maps. After the acquisition is finished, the image must be exported as a *mat* file that can be loaded onto the workspace as a ***uint16*** matrix. The matrix values (field *Index*) represent the depth values in mm. They must be divided by 10 so they can be compared to the computed distances from the SGBM disparity maps which are given in cm.

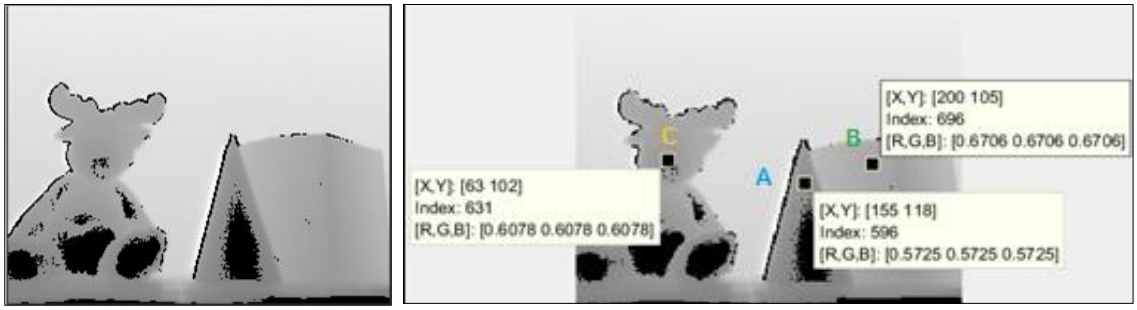


Figure 27. Kinect depth map for Scenario 1 (left) and same depth map with target points (right)

Table 6 collects all the SGBM distances, Kinect depth values, percent errors and variances for the target points established in all four scenarios. The variances were computed for the vector containing the depths of the same target point throughout the 30 disparity maps taken per scenario. All calculations were also performed for the filtered images as the table shows. The percent errors were calculated with equation (46) by defining the Kinect depth as the theoretical value and the SGBM depth as the experimental value.

$$percent\ error = 100 \times \left(\frac{depth_{KINECT} - depth_{SGBM}}{depth_{KINECT}}\right) \quad (46)$$

The percent errors range from 0,47 to 4,23% which is acceptable considering, for instance, that 4% of 60 cm corresponds to 2.4 cm and that would represent an estimation of the maximum possible error with the SGBM class. When comparing the results with and without the median filter, the percent errors are very similar except for two cases: point ***C*** in ***Scenario 2*** (0,8642% difference) and point ***B*** in ***Scenario 4*** (0,3491% difference). The other error differences do not surpass 0,3%.

The variances remain approximately in the same powers of 10 on both filtered and non-filtered sets: 10^{-6} and 10^{-5} in most cases and reaching both extreme values in ***Scenario 3*** with the lowest (equal to zero) and highest (power of 10^{-3}) variances. Thus, the median filter does not seem to have a major effect at least on the two measured variables up to this section.

SCENARIO 1							
Point	**SGBM (cm)**	**Kinect (cm)**	**Error (%)**	**Variance ^{SGBM}**	**SGBM with median (cm)**	**Error with median (%)**	**Variance with median (%)**
A	60,8592	59,6	2,1128	5.46 * 10-6	60,8592	2,1128	6.53 * 10-6
B	68,0525	69,6	2,2234	7.35 * 10-6	67,9906	2,3124	7.94 * 10-6
C	61,1890	63,1	3,0285	2.97 * 10-5	61,0786	3,2035	6.45 * 10-6
SCENARIO 2							
Point	**SGBM (cm)**	**Kinect (cm)**	**Error (%)**	**Variance ^{SGBM}**	**SGBM with median (cm)**	**Error with median (%)**	**Variance with median (%)**
A	67,4576	69,6	3,0782	8.15 * 10-6	67,4360	3,1092	4.67 * 10-6
B	57,0886	58,3	2,0779	2.49 * 10-6	57,1074	2,0456	9.89 * 10-7
C	66,1625	65,3	1,3208	6.31 * 10-4	65,1982	0,4567	2.32 * 10-4
SCENARIO 3							
Point	**SGBM (cm)**	**Kinect (cm)**	**Error (%)**	**Variance ^{SGBM}**	**SGBM with median (cm)**	**Error with median (%)**	**Variance with median (%)**
A	66,7393	67,4	0,9803	1.15 * 10-3	66,9000	0,7418	3.1877 * 10-3
B	70,7857	72,4	2,2297	0	62,7857	2,2297	0
C	70,8784	73,7	3,8285	1.84 * 10-6	62,7857	3,9543	0
SCENARIO 4							
Point	**SGBM (cm)**	**Kinect (cm)**	**Error (%)**	**Variance ^{SGBM}**	**SGBM with median (cm)**	**Error with median (%)**	**Variance with median (%)**
A	57,6106	57,2	0,7178	3.96 * 10-6	57,4724	0,4762	1.84 * 10-6
B	71,5117	74,4	3,8821	3.69 * 10-6	71,2520	4,2312	3.54 * 10-6
C	58,6887	59,9	2,0222	2.9 * 10-32	58,6887	2,0222	2.87 * 10-32

Table 6. Depth measurements from SGBM and Kinect, percent errors and variance ^{SGBM}

e) Disparity error on testing scenarios

To evaluate the disparity error (which will be detailed with the datasets in section IV. E), the number of pixels with erroneous disparity is determined. Those pixels are considered local outliers and represent the percentage of disparity error. The percentages of disparity error and estimated disparity for scenarios 1 to 4 are summarized in ***Table 7*** for filtered and non-filtered datasets. Such percentages were calculated by averaging the individual disparity percentages from all SGBM results in each scenario.

In contrast to the previous table, where the median filter did not have a noticeable effect on the results, it is evident that applying the median filter on the disparity maps increases the number of pixels with estimated disparity. When averaging the differences for all scenarios, the increase is equal to 1.03%. This makes sense since the median filter is a form of interpolation that replaces outliers with the "middle" value within a sequence of neighboring ordered numbers. The OpenCV function for median filtering (*medianBlur*) was implemented with a 5 × 5 square window throughout the project since the compiler did not allow sizes of 7 or higher to be used.

	SCENARIO 1	
	Non-disparity (%)	**Detected (%)**
SGBM	2,6733	97,3267
Median	1,472	98,5280
	SCENARIO 2	
	Non-disparity (%)	**Detected (%)**
SGBM	2,5944	97,4056
Median	1,3447	98,6553
	SCENARIO 3	
	Non-disparity (%)	**Detected (%)**
SGBM	1,5903	98,4097
Median	0,8791	99,1209
	SCENARIO 4	
	Non-disparity (%)	**Detected (%)**
SGBM	2,0406	97,9594
Median	1,1014	98,8986

Table 7. Percentage of pixels without disparity and percentage of estimated disparity

D. Evaluating Disparity Methods on the Middlebury dataset

The disparity methods approached were the Semi-Global Block Matching, Variational and Libelas (ELAS) algorithms described in section III. They were all tested on pictures taken from the Middlebury [29] dataset such as those in ***Figure 28***.

The Libelas algorithm can be freely downloaded from [31]. The only modification necessary to run the method is to add the two images' names in the *main.cpp* file and copy them into the *img* folder. Both SGBM and Variational methods are included in OpenCV as classes. The variational disparity computation's class is called *StereoVar* and includes the following variables: number of layers used, image scale, number of iterations, minimum and maximum disparity values, polynomial expansion's pixel neighborhood, smoothness parameter, threshold for edge-preserving smoothness, type of penalization and multigrid cycle. The semi-global block matching approach's class is called *SGBM* and includes the following variables: minimum and maximum disparity values, matched blocked size, P1 and P2 (disparity smoothness factors), maximum allowed difference in disparity check, truncation pre-filtering cap, uniqueness ratio, maximum size of smooth regions and maximum disparity variation.

Figure 28. Middlebury dataset examples
first row – 'Cones': left and right images
second row – 'Teddy': left and right images

Figure 29 shows the disparity results obtained for the 'Cones' stereoscopic pair. From left to right, the disparities correspond to the Libelas, SGBM and Variational methods respectively. Since the horizontal gap between the input images is smaller than the images captured with the Minoru, the leftmost section without disparity is also reduced. At first sight, the Libelas and SGBM disparities have sharper and more detailed outputs while the Variational OpenCV class renders a very high blurriness even when its smoothing parameters are lowered. Moreover, the leftmost side of the image is an important source of error since it is horizontally 'stretching' the neighboring disparity values instead of computing disparity from real data. The stretching is performed throughout the image because the smoothening is very strong in this strategy. Therefore, the variational method was discarded. Although the Libelas algorithm offered a sharper result, its architecture-specific SSE3 lead to incompatibility errors which prevented it from being implemented in the System-on-chip architecture. In contrast, the OpenCV libraries can easily be implemented in the SoC.

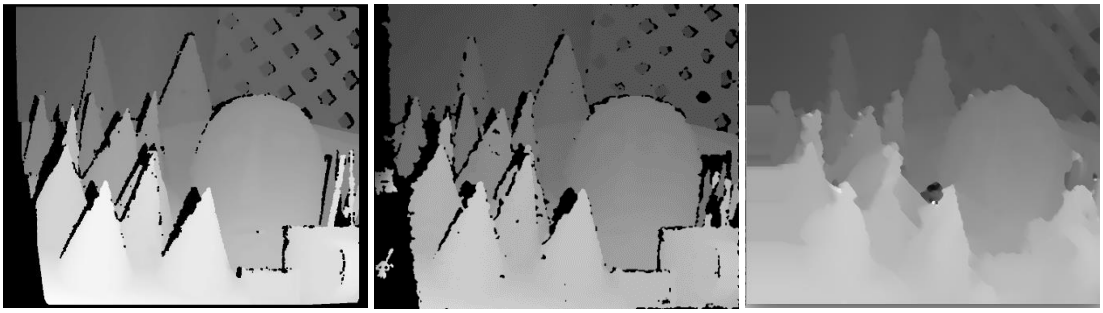


Figure 29. Disparity methods results for 'Cones': from left to right, Libelas, SGBM and Variational

To evaluate the performance of the chosen disparity computation strategy, two metrics were determined based on the KITTI Vision Benchmark Suite [32] ranking: the percentage of disparity error in non-occluded areas and the density, i.e., the percentage of disparity that has been provided by the method. ***Table 9*** shows both metrics obtained with the SGBM algorithm with and without the median blur. The percentage of given disparity by the algorithm was calculated as 1 minus the number of unknown disparity pixels ($d = 0$) divided by the total number of pixels. T is an operator that turns into 1 if the equation inside is true and 0 otherwise.

$$d_{given} = 1 - \frac{unknown\ disparity\ pixels}{total\ number\ of\ image\ pixels} = 1 - \frac{\sum_{ij} T[d(i,j)==0]}{width \times height} \quad (47)$$

The disparity error percentages were obtained as the number of pixels with unknown disparity:

$$d_{error} = \frac{unknown\ disparity\ pixels}{total\ number\ of\ image\ pixels} \quad (48)$$

	SGBM		SGBM with median filter	
	Estimated disparity (%)	**Error (%)**	**Estimated disparity (%)**	**Error (%)**
Aloe	94,7819	5,2181	94,9141	5,0859
Art	89,7657	10,2343	89,8786	10,1214
Baby1	94,8474	5,1526	94,9948	5,0052
Baby2	94,4476	5,5524	94,6451	5,3549
Baby3	92,238	7,762	92,4157	7,5843
Books	90,6536	9,3464	90,8698	9,1302
Bowling1	88,1347	11,8653	88,43	11,57
Bowling2	87,1376	12,8624	87,3367	12,6633
Cloth1	97,5702	2,4298	97,5972	2,4028
Cloth2	96,232	3,768	96,2922	3,7078
Cloth3	97,144	2,856	97,2036	2,7964
Cloth4	95,3853	4,6147	95,4317	4,5683
Computer	80,2062	19,7938	80,5227	19,4773
Cones	86,7366	13,2634	87,0661	12,9339
Dolls	96,0504	3,9496	96,1305	3,8695
Drumsticks	91,2773	8,7227	91,4032	8,5968
Dwarves	91,7249	8,2751	91,8763	8,1237
Flowerpots	75,0741	24,9259	75,3249	24,6751
Lampshade1	88,0123	11,9877	88,2527	11,7473
Lampshade2	87,2797	12,7203	87,5507	12,4493
Laundry	90,3528	9,6472	90,6359	9,3641
Moebius	92,1509	7,8491	92,3097	7,6903
Monopoly	93,7264	6,2736	93,8145	6,1855
Reindeer	94,3	5,7	94,3928	5,6072
Rocks1	95,258	4,742	95,3269	4,6731
Teddy	82,2335	17,7665	82,525	17,475
Wood1	93,4727	6,5273	93,5301	6,4699
Wood2	95,0518	4,9482	95,1075	4,8925
	91,1159	8,8841	91,2778	8,7222

Table 8. Percentages of given disparity and disparity error on Middlebury dataset

The Middlebury dataset includes a total of 38 different scenarios:

- 28 of them were tested as the table shows.
- 4 of them could not be read by the algorithm even after changing their format from ppm to jpg. This is possibly due to the nature of the scenes which mostly involved thin layers of paper or paper-like materials with patterns similar to those in the background.
- 6 scenes that could not be correctly copied to the SoC due to specific permits and/or restrictions. The *chmod* command was used from the terminal but it did not work.

The amount of processed stereoscopic pairs is sufficient to represent the entire dataset due to the variety of shapes, colors and textures from one pair to the other. It is noteworthy that most of the scenarios within the Middlebury dataset contained 6 to 8 different views of the same scene with a small horizontal gap between the consecutively-numbered images under different lighting and exposure parameters. To build a more consistent dataset, all pairs downloaded from the website corresponded to views 0 and 1 with illumination 2 (*Illum2 folder*) and medium exposure (*exp1*) except for the *'Cones'* and '*Teddy*' scenes which only include one pair of photos each.

The overall performance of the semi-global matching algorithm is very convincing with an average of 91.16% rate of disparity estimation for the stereoscopic pairs gaged. The median filtering process improves once again the original results with a 0.1619% average margin. The most evident outlier is the '*Computer*' scene in ***Figure 30*** with an 80.2% rate of disparity estimation. A statement can be made concerning the object arrangement in this case; the region with no disparity over the CPU may be caused by the color and texture similarity between the edge of the keyboard and the CPU. The two big regions with no disparity in the left side of the image can be explained by the light intensity arriving over the speaker and the other keyboard. This issue recalls the initial exposure problems encountered with the Leopard model over bright or white and poorly textured areas.

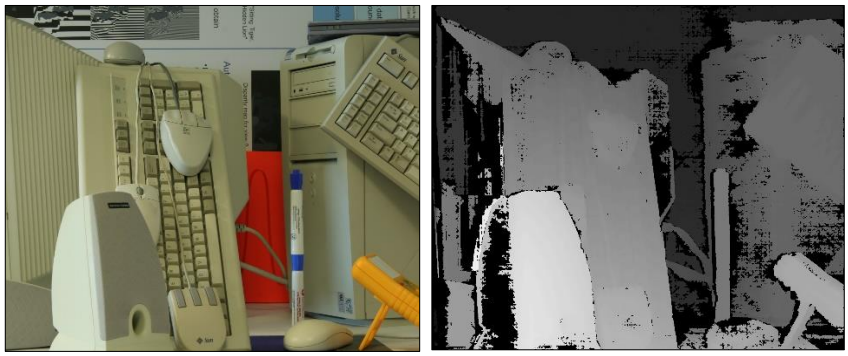


Figure 30. 'Computer' scenario and SGBM disparity map

E. Measuring Time and Power Consumption on Jetson TK1

The SGBM program can be executed at rates of 15 and 30 FPS in the Jetson TK1. Using the library *time.h*, the run-time performance of the algorithm can be evaluated over 100 values. By setting a clock before and after the disparity computation is performed, the execution time can be determined for both FPS rates. As the chart (***Figure 31***) indicates, the overall time is slightly higher for 30 FPS. This can be confirmed by calculating the averages of the 100-data sequences: 459,9 ms and 462,2 ms for 15 and 30 FPS respectively. The variances were also computed: $9{,}13 \times 10^{-6}$ and $1{,}25 \times 10^{-5}$ for 15 and 30 FPS respectively.

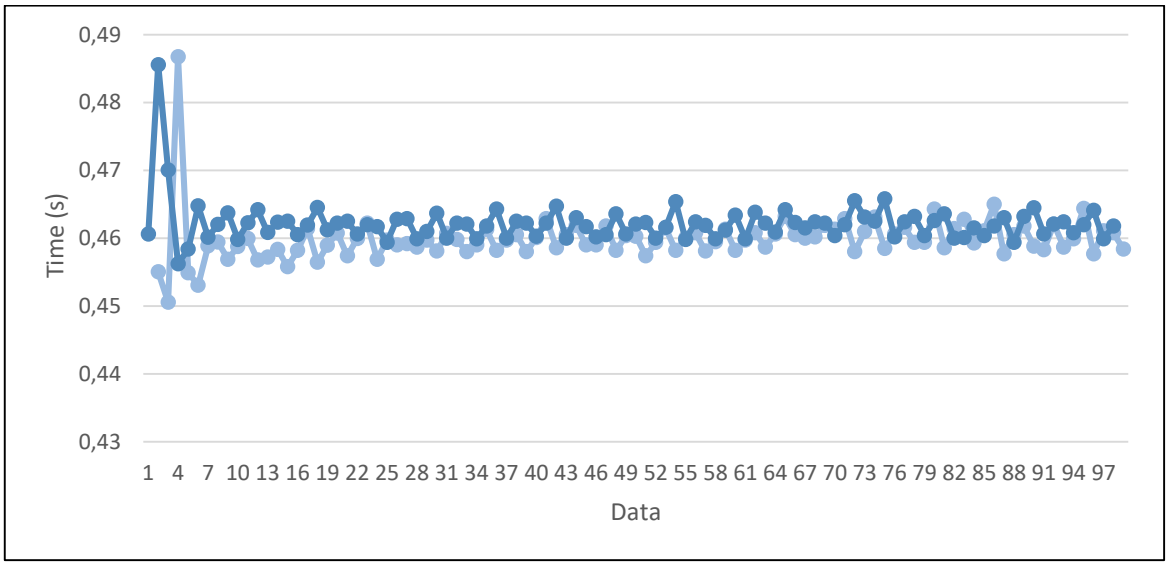


Figure 31. Processing times in Jetson TK1 for SGBM method at 15 FPS (light blue) and 30 FPS (blue)

The power consumption of the Jetson TK1 model can be measured by checking *the regulator.4* folder in the path */sys/devices/platform/tegra12-i2c.4/i2c-4/4-0040/as3722/-regulator.0/regulator/regulator.4*. This folder contains the current values of the consumed voltage (in microvolts) and current (in microamps). Without executing any program, the Jetson power consumption was measured and its values are compiled in ***Table 8.*** While the current stays constant, the voltage does offer a range from 0.65 V up to 1.3 V.

	V (mV)	I (A)	P (W)
Minimum value	650	3,5	2,275
Current value	950	3,5	3,325
Maximum value	1300	3,5	4,550

Table 9. Current, minimum and maximum power consumed by Jetson TK1 SoC

To measure the power consumption during the execution of the designed algorithm, a bash (*.sh*) file [33] was created. While one terminal executes the disparity computation algorithm, another terminal must run the bash file which will have the role of displaying the voltage and current consumed every 100 ms during 10 s of program execution. A simple sleep command inside a for loop to establish a 100 ms delay before each value from the previously voltage and current files are read.

Figure 32 shows the power consumption while the SGBM algorithm is run. The power values were obtained by multiplying the voltage and current data shown in terminal. As in the previous table, the current throughout the 100-value sequence remained the same at 3.5 amperes while the voltage varied slightly in the 0,9 – 1,01 V range. This translated into a very slight power variation in the 3,5 – 3,53 W range with three prominent outliers at 2 s, 5 s and 8,6 s. With an overall average of 3,508 W, running the program therefore increases the consumption by 0,183 W from the 3,325 W shown in ***Table 9***. The power's variance was calculated with a result of $9{,}1207 \times 10^{-3}$.

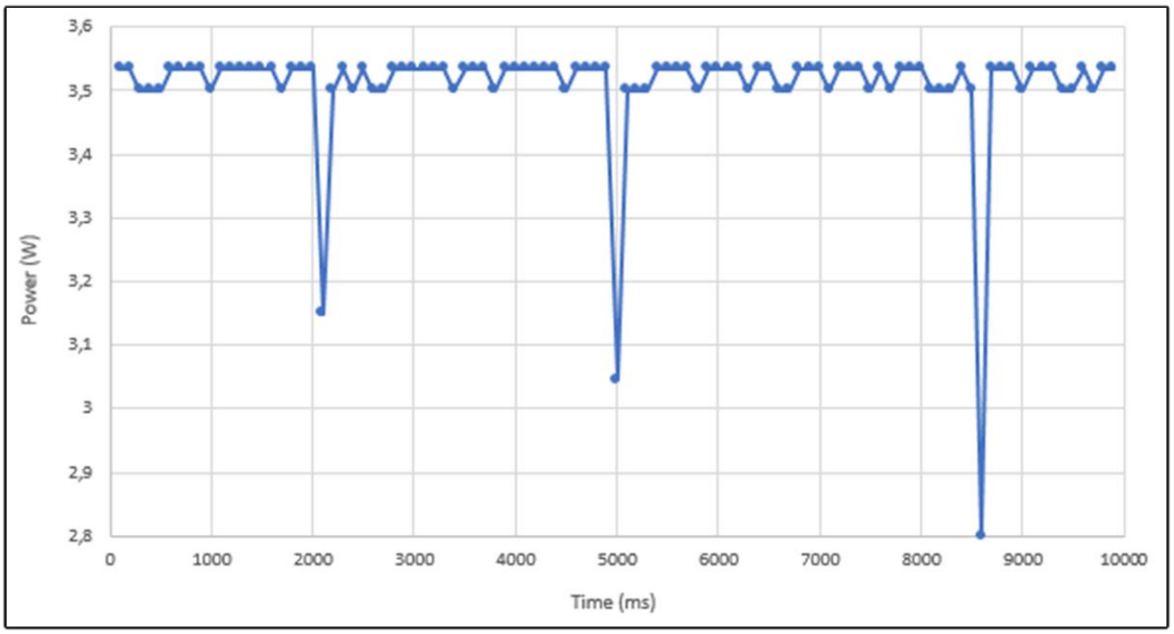


Figure 32. Power consumption in Jetson TK1 for SGBM method

V. PC TO SOC ALGORITHM ADAPTATION

This section presents some basic guidelines to adapt any algorithm from a personal computer to a system-on-chip architecture using the same Ubuntu distribution. In this case, version 14.04 was used on both systems. The modifications made on the disparity computation method are discussed afterwards.

A. Considerations for Adapting an Algorithm from a PC to a SoC Architecture

There are four major considerations to contemplate when adapting the algorithm from a computer to a specific system-on-chip architecture. It is suitable to attend to these suggestions:

1. Make sure that all the required programs, libraries and functions are previously installed in the target architecture and in the correct version. Bypassing this can lead to severe compilation errors that can even disable or block some of the SoC's basic modules such as the display port or even standard booting instructions. Also, check the computer and the SoC's OS versions in case they might differ.
2. Make sure that the all the used libraries and functions are compatible with the SoC's architecture. For instance, Libelas had a performance similar to SGBM during initial tests but its architecture-specific instructions prevented it from running the executable file. In some cases, this can be bypassed with similar libraries or packages.
3. When dealing with video streaming, check that the demanded frame rate does not force the device and that the number of windows (image matrices) displayed periodically is the minimum required.
4. Closely related to the last consideration, it is recommended to reduce the amount of operations within the algorithm (see subsection B).

When working with OpenCV, it is very important to check if the 3.2.0 version is really necessary for the desired implementations. The OpenCV 3.2.0 version has dependencies with the *contrib* library that are not easy to solve even after correctly installing the software on Ubuntu. OpenCV 2.4.8 was sufficient to compile all programs used during the development of this thesis. The *contrib* package is essential in computer vision including face recognition, stereo correspondence and the OpenFABMAP [34] operations. The latter is an underlying part of the remapping process and is at the core of the rectification code.

Since OpenCV has a high compatibility with Ubuntu distributions, it can avoid a lot of complications. NVIDIA's CUDA software was tested and even its cross-compilation from the computer to the SoC but OpenCV's standalone samples were user-friendly and simple enough to keep using them during the entire project. CUDA's matching sample code was even tested during early stages but was abandoned since it used a very specific image size. The Jetson TK1 model has its own OpenCV version installed which was used.

B. Modifications made to the algorithm between the PC and SoC Architecture versions

Apart from the previously mentioned suggestions, specific changes were performed on the disparity computation algorithm. Most of them involved the SGBM class fields and intend to reduce time consumption. The main modifications were the following:

1. The number of disparities (*numDisparities* field) was calculated with equation (43) using the image size for calibration and display which lead to a total of 64 disparities. During testing, higher disparity levels did not significantly improve the map's overall aspect so it was maintained as 64. This parameter can be inputted as an integer if the user wants to use less levels.
2. The minimum disparity value *minDisparities* was set to 30 since the method has a maximum measuring distance (≅ 80 cm) which renders useless the computation of very low disparity values. Keep in mind that depth and disparity are inversely proportional according to the canonical model.
3. The left-right disparity check option was disabled by setting a negative value of -1 for the *disp12MaxDiff* variable.
4. The SGBM class allows the user to run Hirschmuller's full-scale two-pass dynamic algorithm with the binary field *mode* setting it as **true**. It is by default set as **false**.
5. The three filters described in Section III were simultaneously implemented in the post-processing stage on the PC but only the median filter was left due to a slightly better performance.

VI. FURTHER TESTING WITH LASER METER

As it was suggested, two additional scenarios that can be seen in ***Figure 33*** were defined to verify the Kinect's performance against another state of the art sensor. The same background was used and some of the objects even have small details such as the logo on the ceramic cup (Scenario 5) and the different-shaped buttons on the Switch controller (Scenario 6). The Leica Disto 210 [36] laser (***Figure 34***) was chosen to compare distance measurements with both the SGBM implemented strategy and the Microsoft sensing device.

Figure 33. Additional scenarios 5 and 6 used for laser testing

Leica's distance measurement device was borrowed from the university's Civil Engineering Laboratory. Complying with the ISO Standard 16331-1 for laser distance meters, it includes a variety of features such as:

- High accuracy of +/- 1.0 mm
- Ergonomic and compact design to be easily manipulated
- End-piece detection to reduce systematic errors
- Simultaneous display of three values

Figure 34. Leica Disto210 distance measurement laser

The same metrics defined in the previous section are used to evaluate the performance of the algorithm and the state of the art sensors: the percent error in equation (46) and MATLAB's standard variance function [37]. Three target scenarios were established as well on both testing scenarios. The Disto210 was located over a flat surface facing the target points to avoid human errors caused by holding the laser meter with the hand. Its values are given in meters so they were converted into cm to facilitate the analysis with the data obtained from SGBM and Kinect. ***Table 10*** includes the depth data and percent errors obtained with all three measurement strategies: semi-global block matching, Kinect sensor and laser meter. The variance for the SGBM class was also calculated as in the previous section but in this case 40 disparity maps were processed instead of 30. The distance values shown in this table correspond to the average of 10 measurements taken for each target point to assure a higher reliability.

SCENARIO 5							
	SGBM (cm)	Kinect (cm)	Laser (cm)	Variance ^{SGBM}	Error (%) SGBM - Kinect	Error (%) SGBM - Laser	Error (%) Kinect - Laser
Point A	59,4237	60,5	61,49	2.8445 * 10-32	1,7790	3,3604	1,6100
Point B	57,9543	60	60,79	3.9433 * 10-6	3,4095	4,6647	1,2996
Point C	55,6144	55	55,56	3.9334 * 10-6	1,1171	0,0979	1,0079
SCENARIO 6							
	SGBM (cm)	Kinect (cm)	Laser (cm)	Variance ^{SGBM}	Error (%) SGBM - Kinect	Error (%) SGBM - Laser	Error (%) Kinect - Laser
Point A	58,6629	61,7	61,64	1.7252 * 10-6	4,9224	4,8298	0,0973
Point B	61,1709	62	61,3	7.4922 * 10-7	1,3373	0,2106	1,1419
Point C	75,5136	78	77,79	2.0111 * 10-6	3,1877	2,9263	0,2700

Table 10. Depth measurements from SGBM, Kinect and Disto210, percent errors and variance ^{SGBM}

The results show that the distances measured with the Kinect and Disto210 laser meter have a higher similarity between them in comparison to the SGBM distances which translates into a percent error of about 1% between both state of the art devices. The percent errors for the SGBM class with such devices are different in some cases, though they remain within the 5% error margin in both scenarios. These two assertions contribute to validate the experimental results from section IV since the performance of the Kinect can be deemed as relatively similar to the high-accuracy laser device (+/- 1.0 mm deviation as indicated in [36]). The overall variances computed for these new cases are lower than in the four initial experiments with orders of 10^{-6} and below.

It is noteworthy to mention that the Disto210 laser meter distances were established as the theoretical values since they come from a device specially dedicated to the task of measuring distances which guarantees a superior performance during testing than the Kinect sensor. Furthermore, the Kinect works with depth maps which are directly related to an image-building process so it has a higher chance of inducing error by estimating a series of pixel values. In contrast, the laser meter focuses on one specific depth value at a time computed using only the speed of light constant and the recorded time taken by the laser beam to reach the target and return to the sensor. All distances registered by the Disto210 meter start from the device's lower section not the upper one from where the laser beam is emitted.

VII. CONCLUSIONS

Concerning the described stereo calibration protocol, one of the most important stages is the individual camera calibration since it guarantees that the baseline (horizontal distance between both lenses) is accurately computed and included into the translation matrix. The final calibration results offered a 0.27 RMS error which is deemed acceptable under 0.5 pixels and a camera baseline of 57.2 mm which is pretty close to the physical value for the Minoru camera (60 cm). The rectification results were also validated graphically with the 'Combo' grid assuring that corresponding image points are the located same 'height' between the left and right feeds.

During the development of this thesis, a considerable amount of time was spent working with the Leopard Imaging camera. Based on the aforementioned experience, it is highly recommended that the selected resources' capabilities and limitations are carefully analyzed before implementing visual-sensor dependent software. One of the Minoru 3D camera's advantages was its built-in self-exposure control which greatly eased the following stages of code development and adaptation. Basically, the only advantages that the Leopard model had over the Minoru were its default image size and its pixel size. The Leopard camera worked with a 640 × 480 pixels display and a 6 μm pixel size while the Minoru worked with a 320 × 240 pixels display and an estimated pixel size of 9.375 μm. However, this may be compensated by increasing the levels of disparity of the SGBM without overly augmenting the algorithm runtime. Both cameras could efficiently display at a 30 FPS rate.

Using the Kinect sensor as a source of ground-truth measurements, the performance of the semi-global block matching approach was assessed. The overall error for all scenarios reached a maximum value of approximatively 4% which is acceptable for 60-70 cm distances. Furthermore, there are some unavoidable sources of error on both the disparity computation strategy (section III.D) and the Kinect sensor. As [30] indicates, the Kinect's ToF method has a systematic error due to the approximation of optical signals as sinusoidal shapes. Additional tests revealed that the SGBM algorithm's range is about 20-80 cm without the appearance of considerable patches on the surface of the objects at close range and on the background at the maximum distance. Even though the Kinect's algorithm only works at a minimum distance of about 45 cm (up to 4.5 m), it was preferred due to a higher precision over the Real Sense device. Additional tests with the laser meter validated the Kinect sensor's performance with a 1% error between both depth-measuring devices.

When testing the chosen disparity method on the Middlebury dataset, the results were convincing with an overall disparity estimation rate of 91.28% with a small improvement caused by the median filtering process. The number of images taken from the dataset (28 pairs from a total of 38) is sufficient to assess the performance of the algorithm since the website contains a variety of cases. When comparing the obtained disparity estimation error (8.72%) with the KITTI benchmark stereo rankings [33], the *SGBM + median* combination ranks slightly higher than the BRIEF algorithm on position n° 52 with 8.99% error. Interestingly enough, the LIBELAS approach is at position n° 54 with 9.72% disparity error.

The process of adapting an algorithm from a PC to a system-on-chip architecture has to consider two main principles: architecture-specific software compatibility as well as time and power consumption. The first factor can be mainly solved by finding similar alternatives that do not increase the second factor. The latter is especially vital in this project since the SoC depends on the battery duration while being ported on a UAV such as the Firefly.

Although it is out of the scope of this project, future work may include the implementation of a multi-view rendering algorithm to handle occlusions as [35] suggests depending on the targeted application. The main goal of such strategies is to combine multiple source images to cover all regions of the image which is not necessarily crucial for depth estimation but for accurate environment depiction.

VIII. REFERENCES

[1] OpenCV. *OpenCV 3.1.0. Depth Map from Stereo Images.* 2015. docs.opencv.org/3.1.0/dd/d53/tutorial_py_depthmap.html

[2] Cyganek B., Siebert J. P. *An Introduction to 3D Computer Vision Techniques and Algorithms.* John Wiley & Sons. 2009.

[3] Gallup, D., Frahm, J-M., Mordohai, P., Pollefeys M. *Variable Baseline/Resolution Stereo.* 2008.

[4] Klette R. *Concise Computer Vision*. *An Introduction into Theory and Algorithms.* Springer. 2014

[5] Hirschmuller H. *Accurate and Efficient Stereo Processing by Semi-Global Block Matching and Mutual Information*. Institute of Robotics and Mechatronics Oberpfaffenhofen. Wessling, Germany. 2005.

[6] Kosov S. Thormählen T, Seidel H-P. *Accurate Real-Time Disparity Estimation with Variational Methods.* Max-Planck Institute Informatik. Saarbrücken, Germany. 2009.

[7] Geiger A., Roser M., Urtasun R. *Efficient Large-Scale Stereo Matching*. Department of Measurement, Institute of Technology. Chicago. 2010.

[8] C. Tomasi and R. Manduchi. *Bilateral Filtering for Gray and Color Images.* Proceedings of the 1998 IEEE International Conference on Computer Vision. Bombay, India. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html

[9] *Gaussian Filtering.* Computer Science, University of Auckland. 2010. https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf

[10] R. Fisher. S. Perkins. A. Walker. E. Wolfart. *Median Filter*. School of Informatics, University of Edinburgh. 2003. http://homepages.inf.ed.ac.uk/rbf/HIPR2/copyrght.html

[11] Google. *Tango.* 2017. https://get.google.com/tango

[12] Lenovo. *Lenovo Phab 2 Pro.* 2016. shop.lenovo.com/co/es/smartphones/serie-phab/phab2-pro

[13] Microsoft. *Meet Kinect for Windows.* 2017. https://developer.microsoft.com/en-us/windows/kinect

[14] Intel. *Intel Real Sense™ SDK*. 2017. https://software.intel.com/en-us/intel-realsense-sdk

[15] Leopard Imaging Inc. *LI-USB30-V024STEREO.* 2016. https://www.leopardimaging.com/LI-USB30-V024STEREO.html

[16] Code Laboratories Inc. *DUO MLX*. 2016. https://duo3d.com/product/duo-minilx-lv1

[17] Minoru 3D. *Minoru 3D.* http://www.minoru3d.com/

[18] ODROID. *ODROID Platforms. ODROID XU-4.* 2013. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825

[19] NVIDIA. *JETSON TK1 – Unlock the power of the GPU for embedded systems applications.* 2017. http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

[20] NVIDIA. *NVIDIA Technology Powers New Home Gaming System, Nintendo Switch.* 2016. https://blogs.nvidia.com/blog/2016/10/20/nintendo-switch/

[21] Universal Serial Bus. *USB-IF Device Class Documents.* http://www.usb.org/developers/docs/devclass_docs/

[22] Mathworks. *Stereo Calibration App*. 2017. https://www.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html

[23] OpenCV. *Camera Calibration and 3D Reconstruction.* 2017. http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[24] ROS Wiki. *How to Calibrate a Stereo Camera*. 2017. http://wiki.ros.org/camera_calibration/Tutorials/StereoCalibration

[25] *Libuvc: Video capture and processing controls.* 2015. https://int80k.com/libuvc/doc/group__ctrl.html

[26] GUVCView. *GTK + UVC Viewer*. Paulo Assis. 2008. http://guvcview.sourceforge.net/

[27] Brahmbatt, S. *Practical OpenCV. Chapter 10. 3D Geometry and Vision.* 2013.

[28] Ascending Technologies (Intel). *AscTec Firefly.* 2017. http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-firefly/

[29] Scharstein D., Szeliski R. *Middlebury Stereo Vision Page.* 2015. http://vision.middlebury.edu/stereo/

[30] Sarbolandi H., Lefloch D., Kolb A. *Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect.* 2015.

[31] Geiger A. *LIBELAS: Library for Efficient Large-scale Stereo Matching.* 2017. http://www.cvlibs.net/software/libelas/

[32] Geiger A., Lenz P., Stiller C. *The KITTI Benchmark Suite. Stereo Evaluation 2015.* Chicago. 2017. http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo
[33] Mike G. *BASH Programming – Introduction HOW-TO*. 2000. tidp.org/HOWTO/Bash-Prog-Intro-HOWTO-7.html/
[34] OpenCV. *Contrib. Contributed/Experimental Stuff.* 2014. docs.opencv.org/2.4/modules/contrib/doc/contrib.html
[35] Morvan Y. *Multi-view coding. 4.4. Occlusion handling.* http://www.epixea.com/research/multi-view-coding-thesisse20.html
[36] Leica Disto ™ D210. *Leica Geosystems*. https://lasers.leica-geosystems.com/d210
[37] Var – variance. Mathworks. https://www.mathworks.com/help/matlab/ref/var.html