Università Politecnica Delle Marche

Dipartimento di Ingegneria dell'informazione

Laboratorio di meccatronica

Report Finale

Gruppo: Stereo Vision

Componenti: Cannata Giuseppe Pio

Isca Alessio

Moscoloni Elena

Academic year 2018 / 2019

Index

1. Introduction

What we intend to do with this document is to describe the characteristics of a stereo vision system by introducing both the methodologies necessary to determine information from the 3D world starting from a pair of cameras, and the performance of the various methods developed.
In particular, the preliminary procedures will be introduced to the caretization of a stereo system like the calibration and rectification techniques and the characteristics in terms of the ability to perceive distances by a stereo vision system.

The vision system to be referred to below is a prototype developed during the Mechatronics course at the Polytechnic University of the Marche.

2. State of art

Among the various computer vision techniques known in the literature and aimed at reconstructing the three-dimensional structure of a scene, the Stereoscopic vision is the one that has received the greatest attention since it does not impose any constraints on the characteristics of the objects present in the scene (eg presence or not) of moving objects).

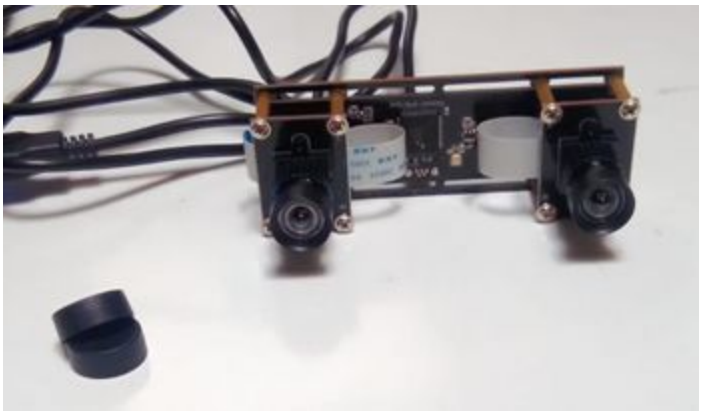
The principle underlying the stereoscopic vision, known since the Renaissance, consists of a *triangulation* aimed at relating the projection of a point of the scene on two (*binocular vision*) or multiple image planes of the cameras. These points are called *homologous* points.
The identification of the homologous points, a problem known in the literature as *matching stereo*, allows to obtain a size called *disparity*, through which, knowing appropriate parameters of the stereoscopic system, it is possible to trace the 3D position of the point considered.

A triangulation approach is used by Kinect3D for the study of depth and 3D reconstruction of the scene:

3. Hardware

This paragraph will list the different hardware components used and for each of them, if necessary, the different characteristics will be listed

3.1 StereoCamera



The stereoscopic camera is a particular type of camera equipped with two parallel lenses, placed at the same distance of the human eye (*6.35 cm*). This allows the camera to simulate binocular vision and then create three-dimensional images.

The characteristics of the camera used are as follows:

- *Resolution*: 3040x1080
- *Saturation*: 40
- *WhiteBalance*: 4600
- *Gamma:* 100
- *Exposure*: -6

Resolution	Specifies the video resolution (*width-by-height*) of the video stream. Cameras usually support different types of resolution.
Saturation	Indicates saturation level, which adjusts the amount of color in the image.
WhiteBalance	Indicates color temperature in degrees Kelvin.
Gamma	Indicates gamma measurement.
Exposure	Specify exposure, in log base 2.

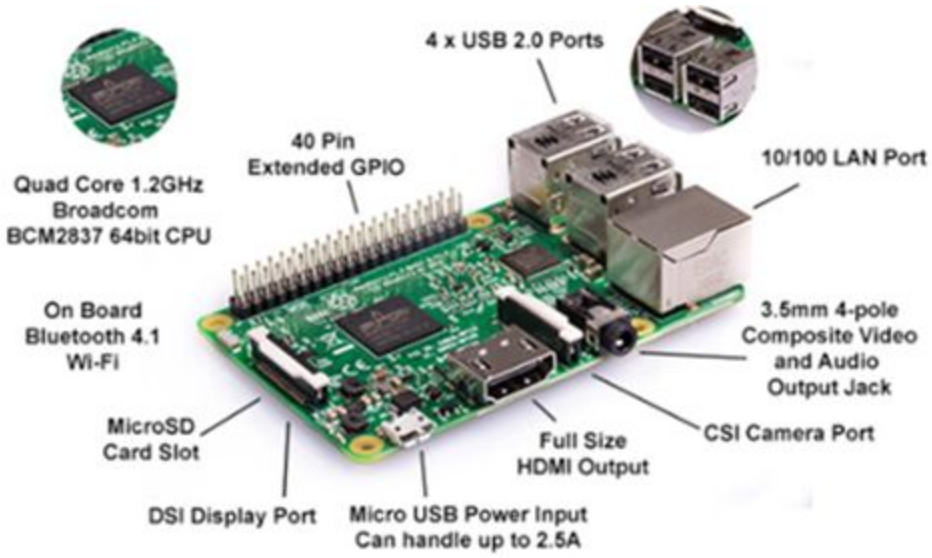
3.2 Spike compact red dot laser

The characteristics are presented below:

- *Colour*: Red
- *Weight*: 3 g
- *Power*: 2 mw
- *Wavelength*: 650 nm
- *Distance reached:* 100 m
- *Size*: 7,62 x 1,71 x 1,43 cm

The laser pointer can be adjusted on the x axis and on the y axis.Furthermore, the base can also be adjusted to allow it to be anchored.

3.3 Raspberry Pi 3 model B



Raspberry Pi is a microcomputer equipped with the *GNU / Linux* operating system of which the 3 is the latest version, released in February 2016 and equipped with an operating system, *Raspbian*, which allows a more profitable use.

Raspberry Pi 3 model B is one of the latest models born of the microcontroller family designed and built by the Raspberry Pi Foundation on the occasion of its fourth birthday. This new version updates the board with more modern components without affecting the principle.

Raspberry Pi 3 integrates in the space of the previous versions a quad-core ARM Cortex-A53 processor from 1.2 GHz to 64-bit and the wireless connectivity, Wi-Fi802.11n and Bluetooth 4.1, so far usable only with additional modules.

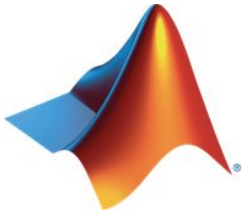
The rest of the data sheet remains in common with the previous version, compressed cable connectivity (*Ethernet, HDMI, USB ...*) and 1 GB of RAM. Compared to the first version of Raspberry Pi, the third promises computing power up to 10 times higher, while maintaining total hardware and software compatibility, including that with Windows 10 IoT Core (*but a 5V and 2.5A power supply is recommended*).

3.4 Monitor, USB mouse and keyboard

This hardware was used to access Raspbian, the operating system of the Raspberry and work directly with it.

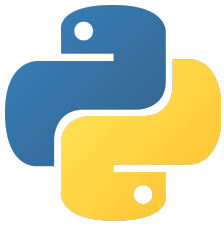
4. Development environment and programming language

4.1 MatLab



Matlab (abbreviation of matrix laboratory) is an environment for numerical calculation and statistical analysis written in C that facilitates the development of programs that perform complex calculations of numerical calculation thanks to an integrated development environment and a specific programming language equipped with a rich library of mathematical functions. Matlab allows you to manipulate matrices, view functions and data, implement algorithms, create user interfaces, and interface with other programs; it works on different operating systems, including Windows, Mac Os, Gnu / Linux and Unix.

4.2 Python



Python is a dynamic object-oriented programming language that can be used for many types of software development. It offers a strong support to integration with other languages and programs and it is equipped with an extensive standard library. Python is distributed under an Open Source license approved by the OSI: its use is free and free even for commercial products.

The following libraries have been integrated for image analysis:

The use of the document will be detailed below.

5. Laser and StereoCamera support design

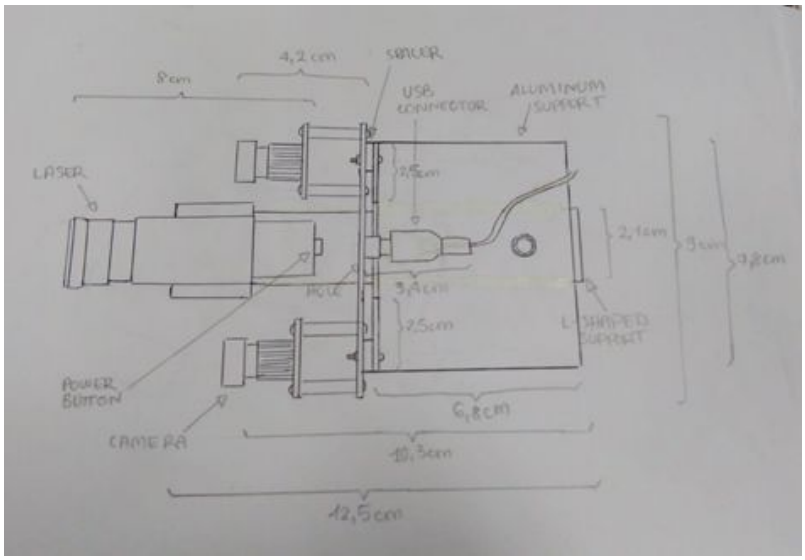
The construction of the support was preceded by a design phase.
The general idea consists of a metal support with an *L-shape* on which the laser can be fixed thanks to a hook already present in its structure.
The StereoCamera will also be set on a further aluminum piece L-shaped, built ad hoc in laboratory. The latter will have one side fixed in the first piece of metal and the other will be used to fix the StereoCamera.
Important consideration to make is that the StereoCamera must be fixed at such a height that its lenses are aligned with the laser pointer.

5.1 Mockup

The mockup produced during the design phase are presented below.
Four different points of view are given:

1. above
2. front
3. side
4. behind

1. **Above**

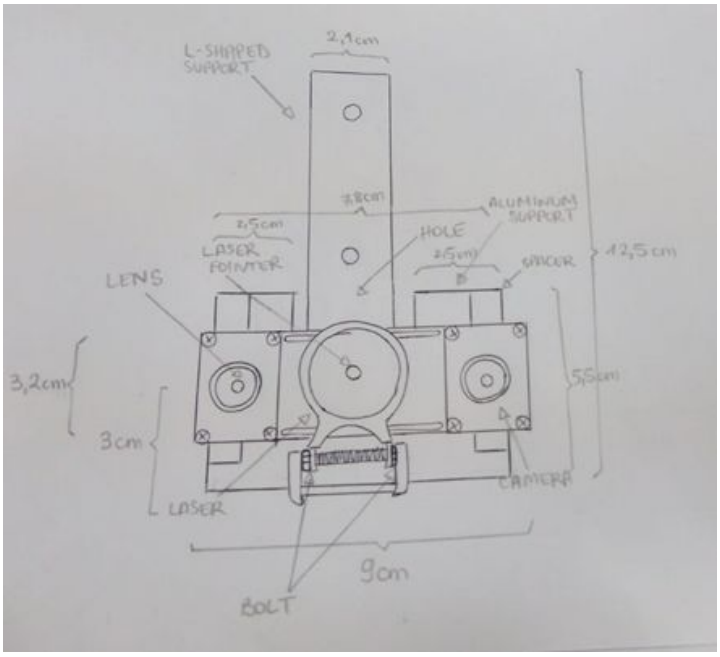


In this drawing we can notice that the StereoCamera support must be fixed at a distance that allows the laser to be switched on using a button located behind the laser itself. At the same time we must consider the presence of the USB connector, *3.4 cm* long. The *L-shaped* support must therefore be sufficiently long.
With a length of *12.5 cm* we are able to solve both problems.

Positioning the StereoCamera and the laser at this distance we can also guarantee that the structure of the laser will not cover the visual field of the video camera, therefore the images will not be disturbed by the presence of the laser itself.

Behind the camera we can see the aluminum support used in order to fix the camera at the height specified. The camera and the support are fixed through some screws screwed in the slots realized in the support of the same camera. Between the camera and the aluminum support we also inserted an other member that works as a spacer to consider the presence of the screws. The aluminum support has a height of *5.5 cm* and has a hole in the middle because of the USB connector.

2. Front

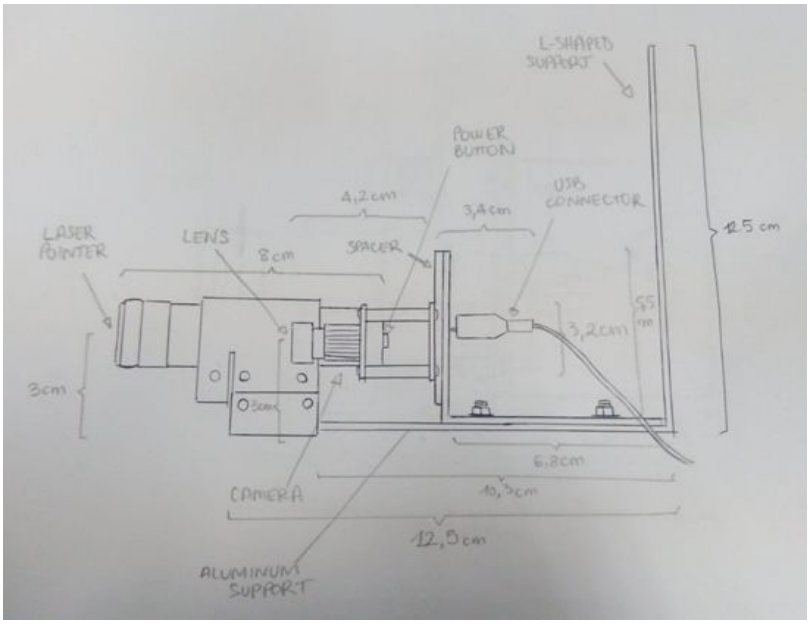


In this drawing it can be noticed that, by fixing the camera at such height, we can guarantee that the laser pointer and the lenses of the video camera will be aligned: they are at a height of *3 cm* from the ground.

A bolt was also inserted into the laser structure because otherwise, once the laser was attached to the *2.1 cm* wide piece of metal, this was not fixed but could move to the right and left.

Also from this point of view it is possible to notice the hole made in the aluminum support to take into account the presence of the USB connector.

3. Side

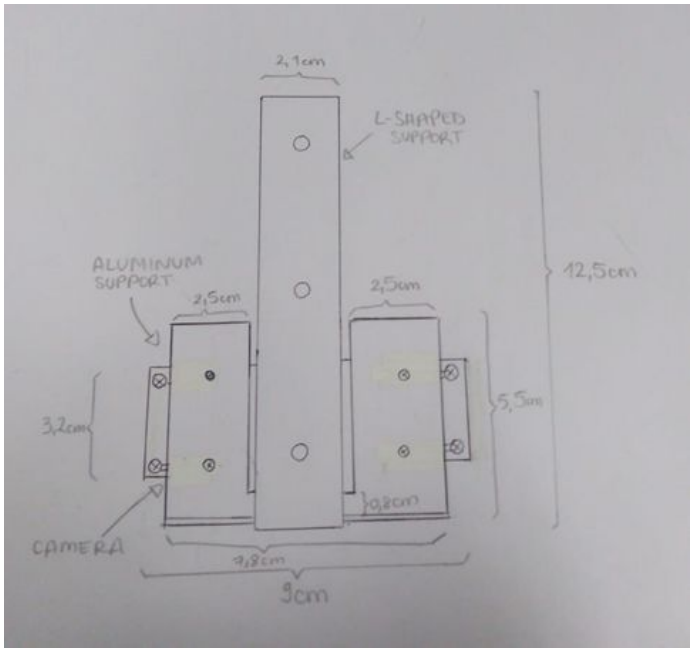


In this drawing we can notice that by fixing the aluminum support for the video camera at a distance of *6.8 cm* from the bottom of the piece of metal with *L-shape* we are able to solve the two problems presented in point 1: at this distance it is possible to turn on the laser thanks to its power button and it is possible to insert the USB connector.

Also from this drawing it can be noticed that the StereoCamera must be fixed at a height so that its lenses are aligned with the laser pointer which is *3 cm* from the ground.

It also can be seen the aluminum support with the spacer and the screws and bolts used to fix the support to the other *L-shaped* support.

4. Behind



In this drawing we have a view from behind; You can see the piece of metal in the shape of an L, the back of the support of the video camera in aluminum and the back of the video camera itself.

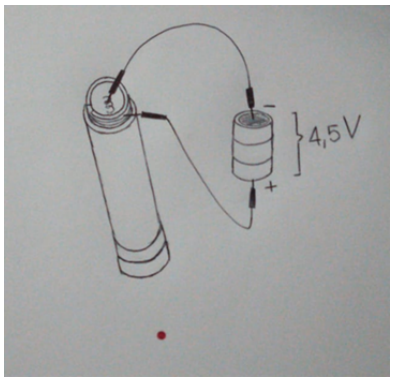
5.2 Laser power supply

In order to avoid the user turning the laser on and off using the special button, it was decided to build a small circuit which, powered by Raspberry, allows the software to control the laser.

First of all we deduced how the laser was powered by the batteries. We have:

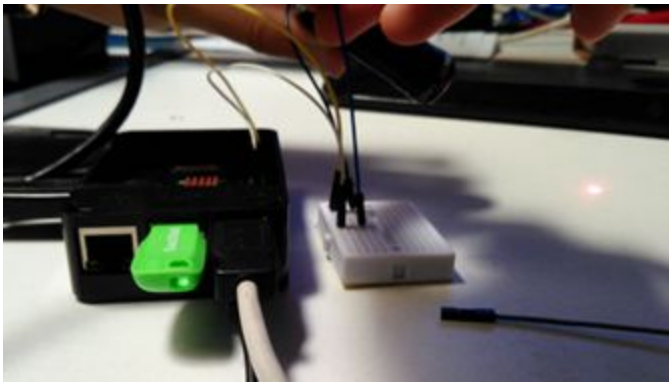
- unscrewed the cap
- extracted the three *1.5V* batteries and kept them in contact with each other
- created a connection between the negative pole of the batteries and a cable
- connected this cable to the internal spring to the laser
- created a connection between the positive pole of the batteries and a cable
- connected this second cable to a metal band around the laser

In this way the laser turns on. We can observe a practical demonstration in the following drawing:



Starting from this assumption, we have looked for a way to power the laser directly through the voltage supplied by the GPIO(*) of the raspberry and to control the switching on and off.

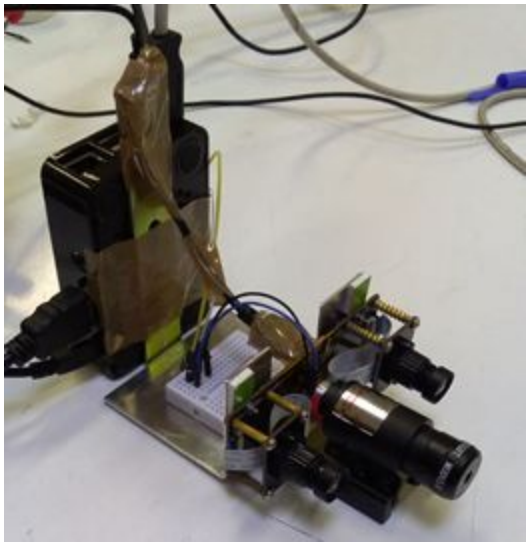
Considering the fact that the laser is powered with a voltage of *4.5V* we thought to exploit a programmable pin of the Raspberry that can provide us with a output voltage of 3.3V to evaluate if it was sufficient to turn on the laser. As we can see from the following picture we connected the metal band of the laser with the **GPIO 17** that was set as output pin and the spring with the GROUND pin of the Raspberry. Piloting the **GPIO 17** the laser has enough voltage to be able to turn on and consequently turn off.



So we inserted an insulating material (Fig. 1) inside the laser cavity. This insulating material wraps a screw that has been screwed into the inner spring. The screw extends beyond the laser and a wire has been fixed on its head with tape. Still with the tape, another wire was fixed in the metal bar that surrounds the laser. These two wires will have the other terminal inserted in the bread board fixed in the support. From the

breadboard two other wires connect it to the GPIO 17 and to the GROUND pin of the Raspberry according to the scheme described above. At the end the Raspberry was fixed to the L- shaped support through double-sided scoth (Fig. 2).

(Fig. 1)



(Fig. 2)

(*)The GPIO port of the Raspberry Pi

The Raspberry has 42 pins, some of which are programmable and others not. The diagram on the right shows us how these GPIO are arranged in the microcontroller.

A GPIO interface is a set of pins (*the Raspberry Pi has 40*) that communicates almost directly with the registers of the processor and that are easily programmable to connect peripherals.

The GPIO interface is a digital interface: it can send and receive only digital signals, that is, alternations of 0 volts and 3.3 volts.

The pins can follow two different numbers:

- the first, defined as the **BOARD**, associates to the pins to the number that grows to the physical position that they occupy in the microcontroller (*GPIO in the previous image*)
- the second, defined as **BCM**, follows a standardized numeration.

6. General steps to calculate distance

The steps that allow you to establish the distance of an object from the Stereo Cameras are:

1. Calibration of StereoCamera
2. Rectification pair images
3. Matching Stereo
4. Distance

We describe in detail the steps listed above.

1. Calibration of StereoCamera

With the calibration of the stereo camera it is possible to obtain the **intrinsic** and **extrinsic** parameters of the camera:

- The **intrinsic parameters** express the mapping between the geometric coordinates and the coordinates in pixels in the digital image produced. Among these parameters we find:
    - focal distance
    - vector translation center of the image.
    - distortion of the x and y axes of the image (skew)
- The **extrinsic parameters** allow to pass from the reference system of the world to the one of the camera. Among these we find:
    - rotation matrix
    - translation matrix

The study was carried out only for the intrinsic matrix. The theoretical results found are shown below.

Intrinsic parameters

The intrinsic parameters are traceable in the **intrinsic matrix** (or also called **camera matrix**).

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

We analyze the parameters shown in detail:

- **fx e fy**

The coefficients **fx** and **fy** represent projections of the focal distance.These two values are calculated respectively:

$$fx = F*k \qquad (1.1)$$

$$fy = F*l \qquad (1.2)$$

In which with **F** is the effective focal distance while **k** and **l** are respectively the length and height of the pixel. Ideally the fx and fy values coincide. This means that the values k and l are equal and therefore the pixel remains a square. In reality, the sensor of camera during the representation of the digital image introduces distortion. This means that k and l differ bringing the pixel to be a rectangle. The ideal pixel dimensions can be found in the datasheet of the camera sensor at the entry *pixel size*. If these features are not present, it is possible to calculate them in this way:

$$k = W / w$$

$$l = H / h$$

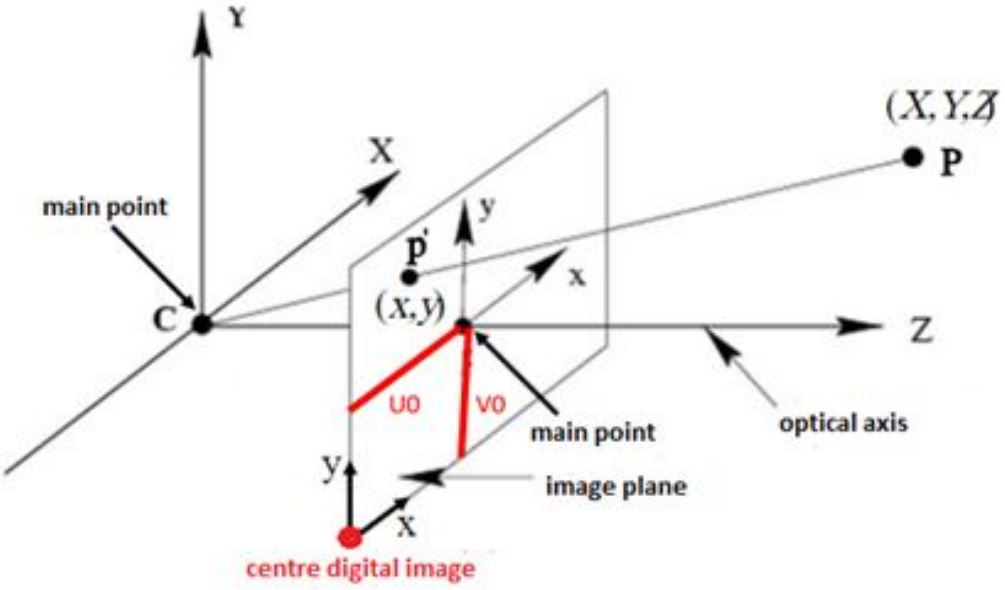
In which:

- W is sensor length in mm (*Datasheet sensor, item Image area*)
- w is maximum image length expressed in pixels (*Datasheet sensor, item image array*)
- H is sensor height in mm (*Datasheet sensor, item Image area*)
- h is s maximum image height expressed in pixels (*Datasheet sensor, item image array*)

- **cx e cy**

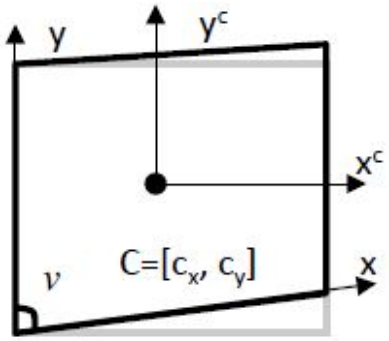
One thing to be considered is that the coordinates of the image originate at the center of the image *(main point)* where the *optical axis* intersecs the image plane* in a perpendicular way. On the other hand, digital images have their origin in the lower left corner of the image.The parameters cx and cy express the coordinates of the main point. It can be seen as the deviation from the center of the digital image.

image plane*: is the plane where the image is "built". Corresponds to the CMOS sensor.



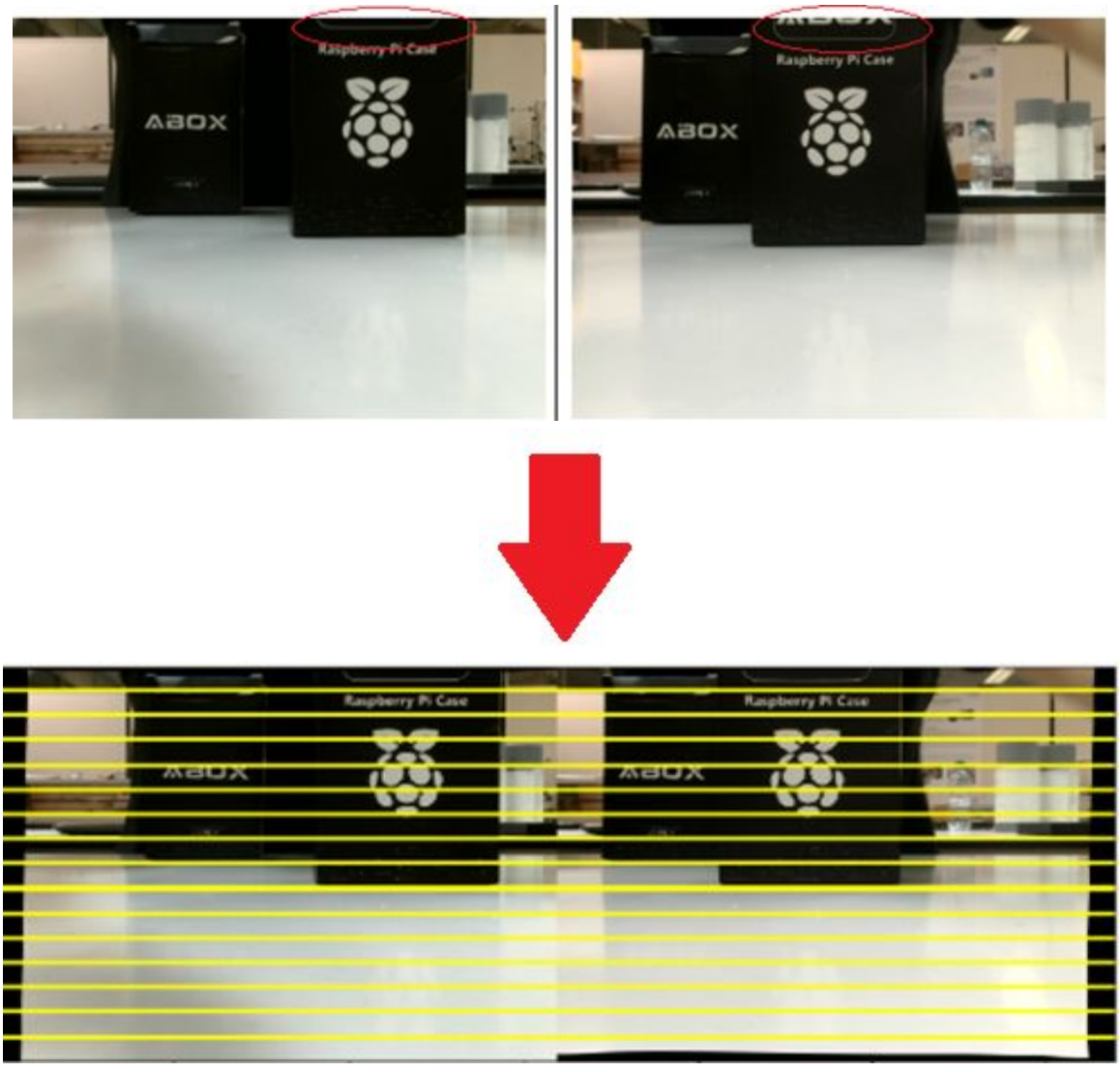
- **skew (s)**

We know that the image plane is a CMOS sensor. However this sensor has the axes that form an angle between them greater than or less than 90 °, thus causing distortion in the image. Well the *skew-factor* wants to model just that. Most cameras have zero-skew, but some degrees of asymmetry may occur due to sensor manufacturing errors.



2. Rectification pair images

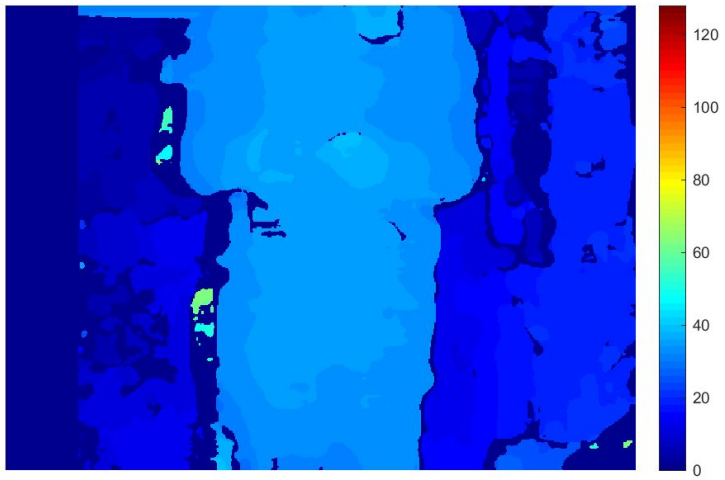
It is the operation that allows to obtain corresponding points in the same epipolar line and it eliminates any distortion of the lenses and misalignments caused by incorrect positioning of the cameras. This phase is repeated every time a photo is taken.
The following photos represent the same subject, before and after rectification.
It is possible to notice that the unrectified images show differences, that are eliminated in the rectified ones:



3. Matching stereo

The phase of matching stereo consist in the search for the right image and the image of the left homologous points. Thanks to this we can calculate the distance in pixels between homologous points. This magnitude is called *disparity*.

For objects that are close to the camera you will have higher disparity values. As objects move away from the camera, disparity values between homologous points decrease



Instead of calculating disparity for each homologous of the two pair images, we decided to consider the laser pointer as a point of interest. So the process considered will be the following:

1. Locate the laser in both the right and left image
2. This last represents the homologous point of our interest
3. Consider the laser pixel coordinates respect to the x axis
4. make a difference between xL and xR
5. disparity = | xL - xR |

In the following paragraphs, related to the calculation of the distance in MatLab and Python, we will describe in detail the techniques of locating the laser pointer

4. Distance

The relation linking the distance with the disparity is as follows:

d = *disparity*

z = distance

B = baseline

f = focal distance

$$d = \frac{B * f}{z}$$

Disparity is inversely proportional to distance.

Since the focal distance is not known, the product between B and f has been estimated, so that the equation becomes:

$$z = \frac{c}{d}$$

where should estimate the product $B * f.$

The steps that make the calculation of **C** are as follows:

1. place an object at a *known distance*
2. detect the laser
3. identify the x coordinate of the laser in the left image
4. identify the x coordinate of the laser in the right image
5. calculate disparity as | *xL - xR* |
6. calculate c as distance *know_distance*disparity*

Repeat the previous steps at different *known distances* and then calculate the estimate of c.

7. Calculate distance on MatLab

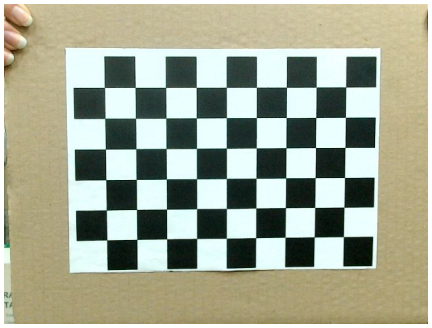
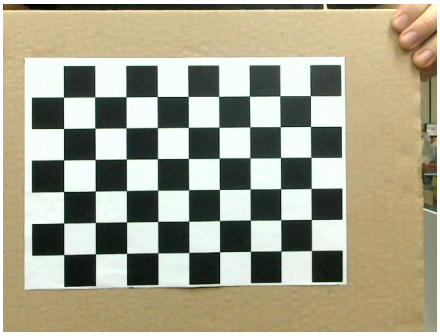
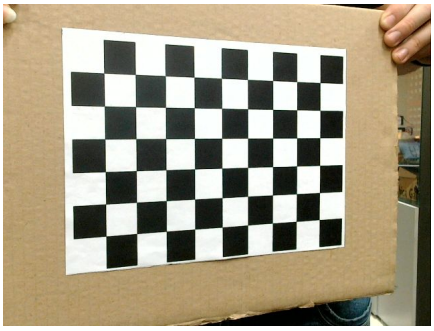
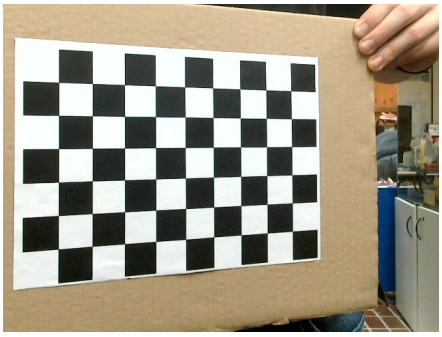
In this section the steps commented on in paragraph 6 will be taken up, having an eye for the development carried out in the MatLab environment.

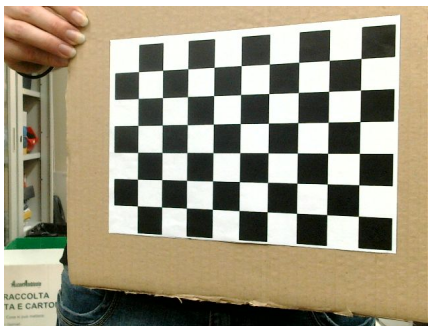
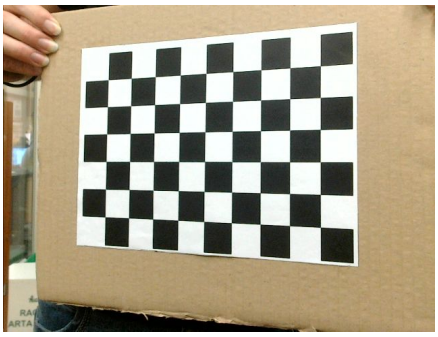
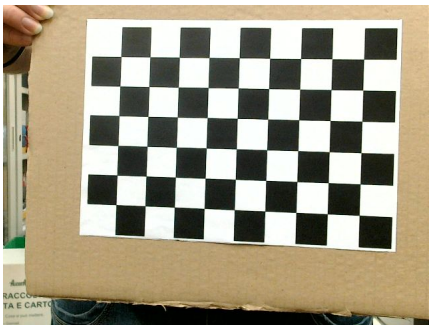
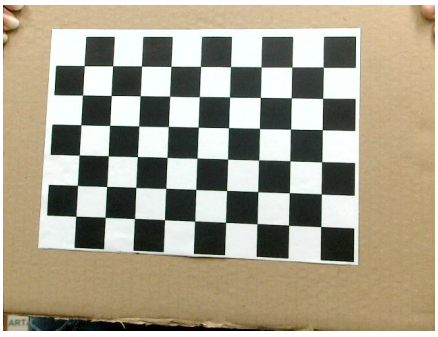
1. Calibration of StereoCamera

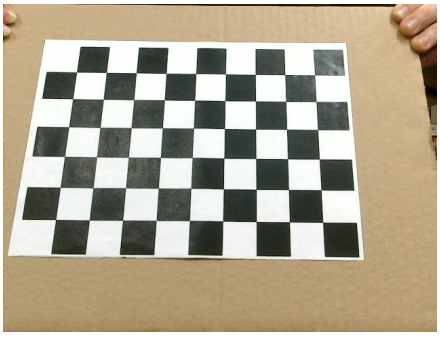
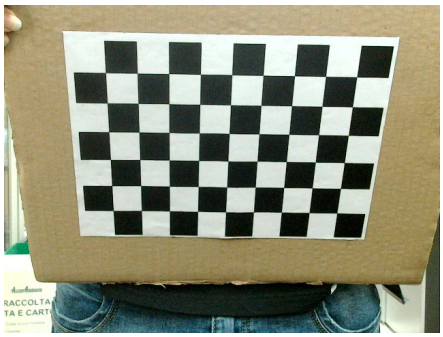
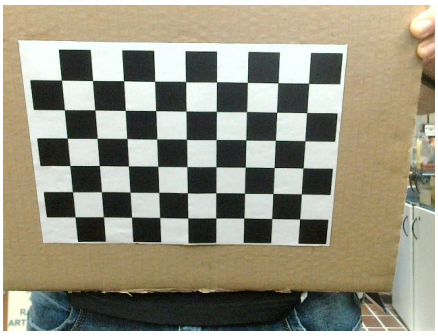
To calibrate the camera download a Matlab tool dedicated to this operation is required: Computer Vision System Toolbox.
This tool works by calibrating the camera through a set of images taken from it that represent a chessboard.

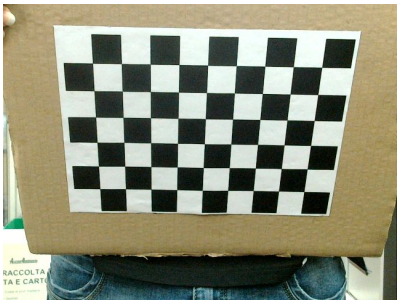
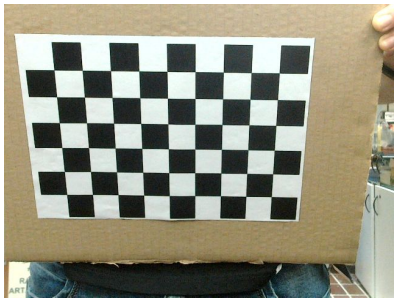
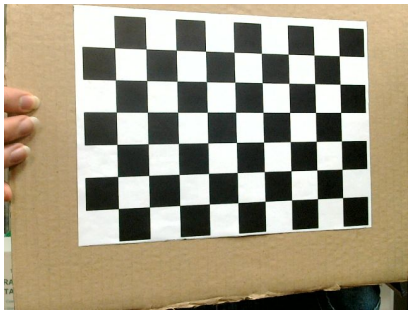
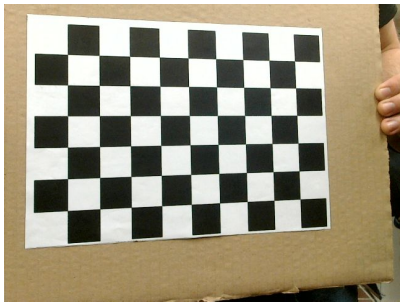
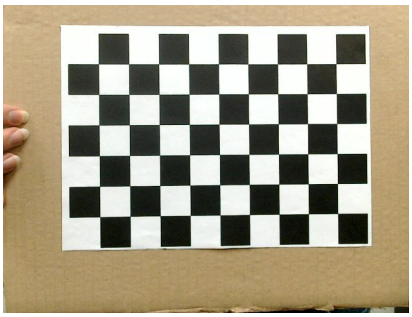
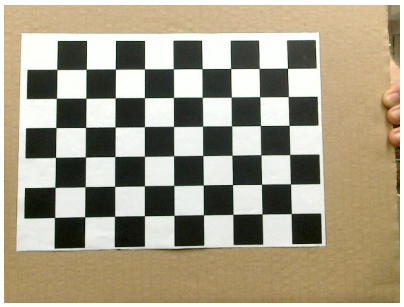
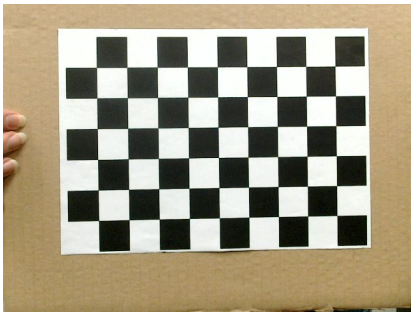
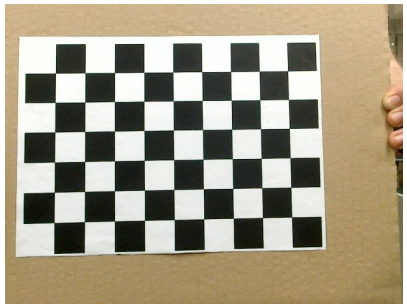
1. First of all, the dataset was formed. The latter consists of 20 images (10 right camera, 10 left camera), whose subject is a chessboard in different angles in 3D space and at different distances from the stereo camera.

DATASET:

Left camera	Right camera
	
	

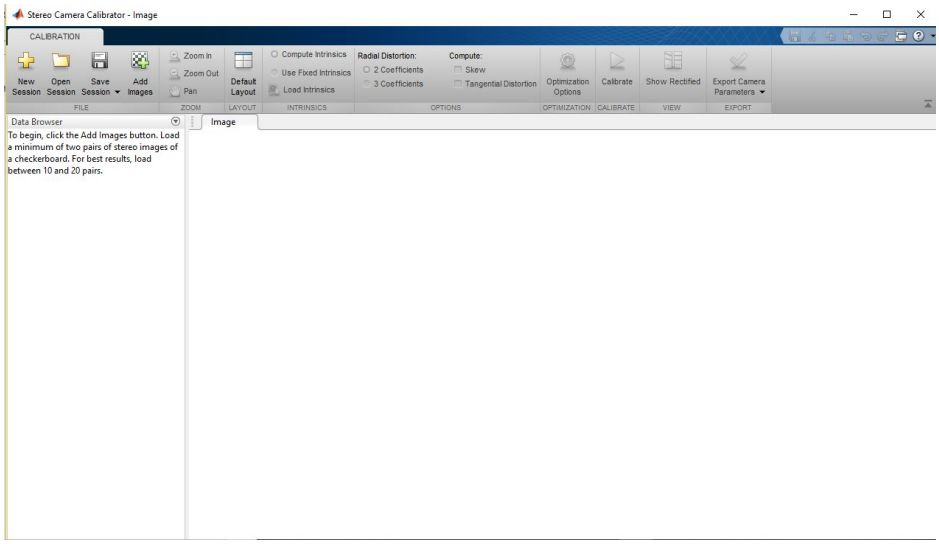
	
	
	
	

Following its characteristics of the chessboard used:

Num. columns	9
Num. rows	6
Dim of square	24 mm

2. Save these photos in two different folders (for left ones and right ones)

3. Launch a function that the tool just downloaded provides: *stereoCameraCalibrator.* The window obtained is the follow:



4. Click on add images button
5. Select folders just created and impose the size of checkboard square (24mm in this case)
6. Images will be loaded
7. Click on Calibrate button in order to boot the process of calibration
8. Matlab returns its results providing the estimation of the distance between the support with the chessboard and the stereo camera
9. Click on *Export Camera Parameters* and then on *Export Parameters to Workspace* button in order to save camera parameters.

The object that most interests us is the intrinsic matrix of the camera.

The result (*Camera left*):

[[842.48 0 304.70] [0 845.60 239.06] [0 0 1]]	**fx**	842.48
	fy	845.60
	cx	304.70
	cy	239.06

The result (*Camera right*):

[[847.69 0 398.95] [0 850.82 162.13] [0 0 1]]	**fx**	847.69
	fy	850.82
	cx	398.95
	cy	162.13

2. Rectification pair images

Thanks to the calibration we were able to get the camera parameters. In the code below, we consider, for ease of treatment, that the matrices obtained during the calibration phase have already been imported.
To do so, just import the "stereoParams" file created during the calibration phase into the workspace.

The MatLab function used to images rectification is called: rectifyStereoImages.
It returns undistorted and rectified versions of left and right input images using the stereo parameters stored in the stereoParams object.

```
% J1, J2 are the rectification about left,right
[J1,J2]=rectifyStereoImages(left,right,stereoParams);
```

Let's look at this line of code in detail:

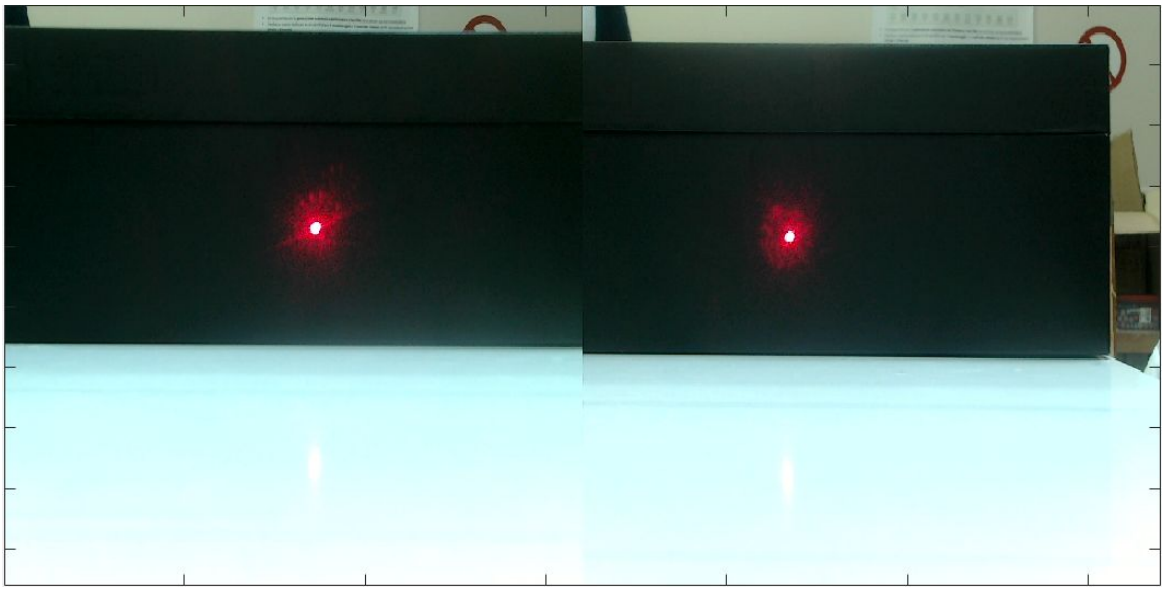
Input parameters:

left , right	they are the images that must be rectified
stereoParams	Matrix of intrinsic parameters of the left and right camera

Output parameters:

J1	it is the rectified left image
J2	it is the rectified right image

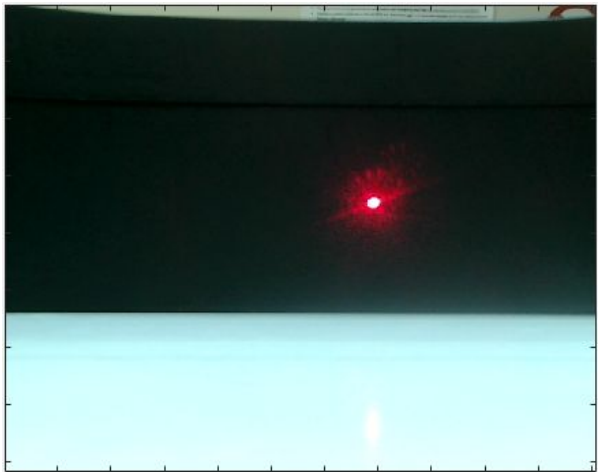
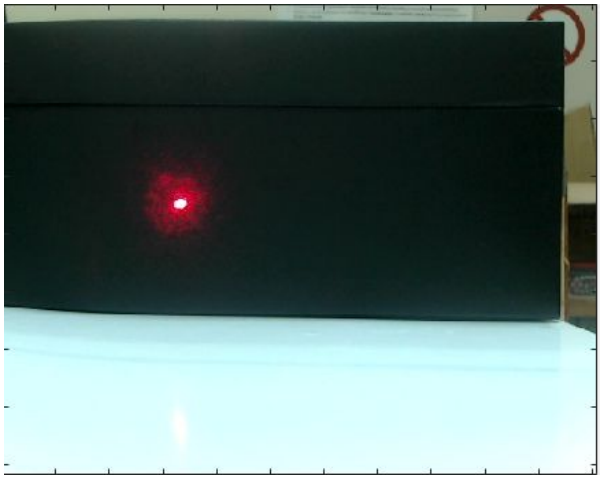
Let's see an example of the effectiveness of this function:



left image	right image

As you can see from the two pictures above, the stereo camera has a phase shift along the Y axis.

After the rectification this is the result:

	
Left image rectified	Right image rectified

As you can easily see now the laser is arranged exactly at the same y as we expected.

3. Matching stereo

According to the description in chapter 6 regarding the stereo matching part, let's see how the laser pointer was detected in MATLAB.
The steps of our algorithm are as follows:

1. Acquire an image with the laser;
2. Acquire the background;
3. Divide the two images into left and right and leftBackground and rightBackground respectively;
4. Subtract leftBackground to left (*both converted to grayscale*) and RightBackground to right (*both converted to grayscale*);
5. Make an average of the position of the white points obtained from the subtraction image deriving from left-leftBackground and make an average of the position of the white points obtained from the subtraction image deriving from right-rightBackground;

6. Make the difference between the two averages previously obtained and this will be the diparity that we will use later to calculate the distance;

The code is as follows:

```
%photo with laser

img = snapshot(cam);
left = img(1:480,1:640,1:3);
right = img(1:480,641:1280,1:3);
[J1,J2]=rectifyStereoImages(left,right,stereoParams);
frameLeftGray  = rgb2gray(J1);
frameRightGray = rgb2gray(J2);

%photo without laser

img1 = snapshot(cam);
left1 = img1(1:480,1:640,1:3);
right1 = img1(1:480,641:1280,1:3);
[J11,J21]=rectifyStereoImages(left1,right1,stereoParams);
frameLeftGray1  = rgb2gray(J11);
frameRightGray1 = rgb2gray(J21);

immagine=(frameLeftGray- frameLeftGray1);
n=1;
m=1;
for i=1:1:409
    for j= 1:1:554
        if immagine(i,j)> 100
            a(n)=i;
            b(m)=j;
            n=n+1;
            m=m+1;
        end
    end
end

%calculation of the laser position in the left photo
contatore_righe=0;
contatore_colonne=0;
```

```
for j= 1:1:m-1
    contatore_colonne=contatore_colonne+b(j);
end
for i=1:1:n-1
 contatore_righe= contatore_righe+a(i);
end


posy=floor(contatore_righe/(n-1));
posx=floor(contatore_colonne/(m-1));


z=1;
c=1;
immagine1=(frameRightGray- frameRightGray1);
for i=1:1:409
    for j= 1:1:554
        if immagine1(i,j)> 100 %let's see the section 12. The problem of range on
%MatLab environment
            s(z)=i;
            t(c)=j;
            z=z+1;
            c=c+1;
        end

    end
end

%calculation of the laser position in the right photo
contatore_righe1=0;
contatore_colonne1=0;

for j= 1:1:c-1
    contatore_colonne1=contatore_colonne1+t(j);
end
for i=1:1:z-1
    contatore_righe1= contatore_righe1+s(i);
end


posy1=floor(contatore_righe1/(z-1));
posx1=floor(contatore_colonne1/(c-1));
```

Once the laser's coordinates are identified you can calculate the disparity:

```
# Calculate disparity
differenza=posx-posx1;
```

4. Distance

Knowing at this point that the relationship we expect to exist between our distance and our disparity is as follows:

$$y = c/x$$

where **y** represents distance and **X** represents disparity.

So we thought to follow the following steps to give an estimate of the constant c to implement a code that uses this relationship to calculate the distance:

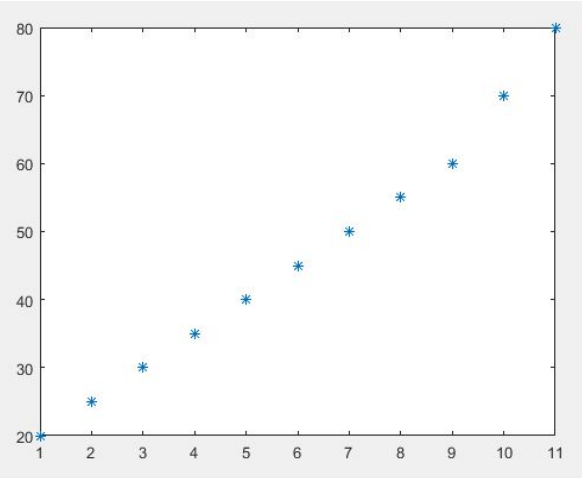
1. place the object at a known distance
2. calculate with the code previously reported the disparity
3. multiply the two values to obtain the c corresponding to this particular measure.

After repeating this procedure 10 times, we make the average of the obtained values so as to be able to have an estimate of the c in the whole interval.
The implemented code is as follows:

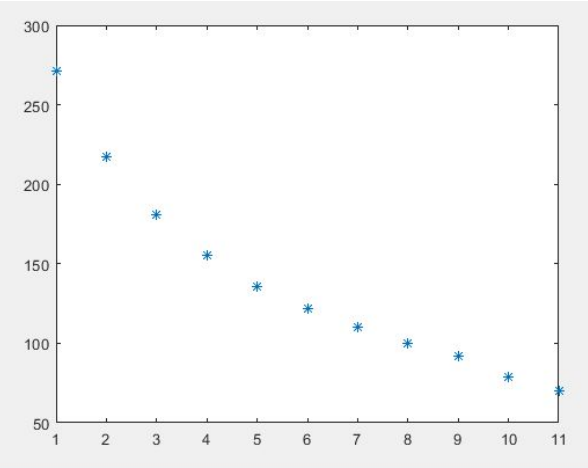
```
c(1)= 20*271;
c(2)= 25*217;
c(3)= 30*181;
c(4)= 35*155;
c(5)= 40*136;
c(6)= 45*122;
c(7)= 50*110;
c(8)= 55*100;
c(9)= 60*92;
c(10)= 70*79;
```

```
c(11)= 80*70;
cont=0;
for i=1:11
   cont=cont+c(i);
end
cdefinitiva=cont/11
```

Measurements	**Distance (± 0.2cm)**	**Disparity**
1	20cm	271
2	25cm	217
3	30cm	181
4	35cm	155
5	40cm	136
6	45cm	122
7	50cm	110
8	55cm	100
9	60cm	92
10	70cm	79
11	80cm	70



(distance at different moments of time)

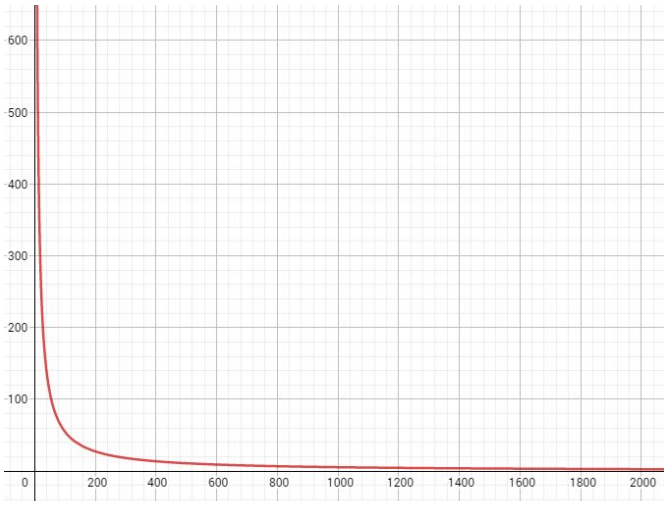


(disparity at different moment of time)

According to paragraph 6, for small distances (objects close to the camera) the disparity values are high, as we move away from the camera (objects away from the camera) the values of disparity increase.

The value of **C** obtained is: **5480.**

The formula becomes: z=5480/d



Once we have obtained the value of c we can implement the formula:

$$z = \frac{c}{d}$$

as follow:

```
# Calculate distance
distanza=5480/abs(differenza)
```

8. Calculate distance on Python

In this section the steps commented on in paragraph 6 will be taken up, having an eye for the development carried out in the Python language.

In python the manipulation of the images was performed by:

- **OpenCV** library released for image processing

- **Numpy** , Python extension for matrix and vector manipulation.

1. **Calibration of StereoCamera**

The tool developed for calibration of StereoCamera is traceable to the following link: https://github.com/GiuseppeCannata/StereoVision/blob/master/Calibrazione.py

The images used to measure the power are those in the paragraph related to MatLab.

Below is the code that allows the calibration of a single camera as the procedure can be repeated for the other

Code :

```
import numpy as np
import cv2

nCol = 6   # number of rows of chessboard -1
nRow = 9   # number of columns of chessboard -1
dimSquare = 24

# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, dimSquare,
0.001)

objp = np.zeros((nCol*nRow,3), np.float32)
objp[:,:2] = np.mgrid[0:nCol,0:nRow].T.reshape(-1,2)
```

```
# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

for fname in os.listdir('/home/giuseppe/Scrivania/left'):

    # acquire the images
    img = cv2.imread(os.path.join('/home/giuseppe/Scrivania/left',fname))

    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) # conversion in to gray scale

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (nCol,nRow))

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        imgpoints.append(corners2)

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1])
```

We comment the source code istruction:

- **ret, corners = cv2.findChessboardCorners(gray, (nCol,nRow))**

  Formal parameters:

gray	The image converted to grayscale
(nCol,nRow)	We had to specify (6,9) as patternSize parameter to specify the number of **inner** corners in the checkerboard. In practice, we are excluding incomplete rows and columns, as well as the first and last complete rows and columns in the checkerboard. This is done to obtain a more reliable detection

  Return parameters:

ret	It assumes *True* or F*alse* values.

	True the pattern (chessboard) was found. *False* the pattern was not found.
corners	The corners variable contains 54 vectors of dimension 1 x 2. Each vectors represents the coordinates (in pixel)of a detected corner.

- **ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1])**

Formal parameters:

Important input datas needed for camera calibration is a set of 3D real world points

(*objpoints*) and its corresponding 2D image points (*imgpoints*).

2D image points (X,Y) can easily find from the image(*These image points are locations*

where two black squares touch each other in chess boards).

As for the 3D points (X,Y,Z) of the real world We can say chess board was kept stationary at XY plane, (so Z = 0 always) and camera was moved accordingly. This consideration helps us to find only X, Y values. Now for X, Y values, we can simply pass the points as (0,0), (1,0), (2,0), ... which denotes the location of points.

Return parameters:

ret	-
mtx	Is the intrinsic matrix described in paragraph 6
dist	Is the vector of distorsion coefficient of the image
rvecs	Is the rotation matrix
tvecs	Is the traslaction vector

The object that most interests us is **mtx**, the latter expresses the intrinsic matrix of the camera.

The result (*Camera left*):

	fx	811.77

[[811.77 0 290.21] [0 813.27 243.84] [0 0 1]]	**fy**	813.27
	cx	290.21
	cy	243.84

The result (*Camera right*):

[[810.77 0 340.41] [0 813.31 189.46] [0 0 1]]	**fx**	810.77
	fy	813.31
	cx	340.41
	cy	189.46

2. **Rectification of pair images**

The tool developed for rectification of pair images is traceable to the following link: https://github.com/GiuseppeCannata/StereoVision/blob/master/Rettificazione.py

In the code below, we consider, for ease of treatment, that the matrices obtained during the calibration phase have already been imported.
So the **mtx_left , dist_left , mtx_right , dist_right , R** and **T** matricies these are the ones we get during calibration and which are assumed to have been saved at the end of it as file .npy. To do the rectification phase just import them

Code:

```
# Acquire image left and right to rectification
im_left = cv2.imread('/home/giuseppe/Scrivania/left.jpg',1)
im_right = cv2.imread('/home/giuseppe/Scrivania/right.jpg',1)

# creation of numpy matrix to obtained new camera matrix
R1 = np.zeros((3,3))
R2 = np.zeros((3,3))
P1 = np.zeros((3,4))
P2 = np.zeros((3,4))
Q  = np.zeros((4,4))
```

```
 # start rectification
cv2.stereoRectify( mtx_left , dist_left , mtx_right , dist_right , (640,480) , R , T , R1 , R2 ,
                        P1 , P2 , Q , flags = cv2.CALIB_ZERO_DISPARITY)


# build imageplane with new camera matrix calculate with StereoRectify
map1_x , map1_y = cv2.initUndistortRectifyMap(mtx_left , dist_left , R1 , P1, (640,480) ,
                                                    cv2.CV_32FC1)
map2_x , map2_y = cv2.initUndistortRectifyMap(mtx_right , dist_right , R2 , P2 , (640,480),
                                                    cv2.CV_32FC1)

# remap the images to have same imageplane
im_left_remapped = cv2.remap(im_left,map1_x,map1_y,cv2.INTER_CUBIC)
im_right_remapped = cv2.remap(im_right,map2_x,map2_y,cv2.INTER_CUBIC)
```

Let's focus on the **cv2.StereoRectify** method:

Input parameters:

mtx_left , mtx_right	Matrix of intrinsic parameters of the left and right camera (*Explained in the paragraphe 6*).
dist_left , dist_right	Matrix of optical distorsion of the left and the right camera.
(640,480)	Corresponds to a tuple in which the two arguments are respectively the length and height of the image.
R	Is the rotation matrix of the stereo camera. Obtained in Calbration phase.
T	Is the traslation matrix of the stereo camera. Obtained in Calbration phase.

The matrices above are obtained in the calibration phase. To be able to use them whenever necessary, they were saved in *.npy* format so that they can be recreated as numpy arrays in python environment

The most interesting topics are the matrices R1 , R2 , P1 , P2.
These are the result we get from the rectification process and that we use to remap (rectify) all the images obtained by the camera.

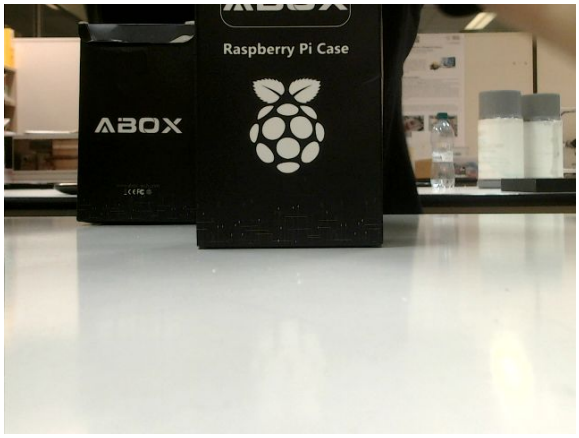
R1	3 x 3 matrix, rectification transform (*rotation matrix*) for the left camera
R2	3 x 3 matrix, rectification transform (*rotation matrix*) for the right camera
P1	3 x 4 matrix new camera left matrix
P2	3 x 4 matrix new camera right matrix

The images that we acquire present a shift along the Y axis, the correction allows to solve this problem through the calculation of the matrices P1, P2.
The latter are presented in this way:

$$P1 = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad P2 = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_2 & T_y * f \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

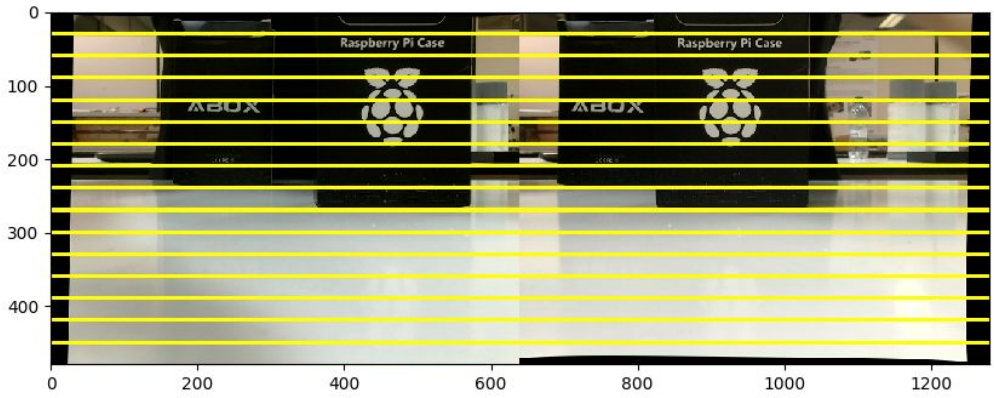
Ty represent the vertical shift between the cameras.
(*The parameters f cx cy are explained in the paragraphe 6*)

As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to **initUndistortRectifyMap()** to initialize the rectification map for each camera.
im_left_remapped and im_right_remapped are respectly the left and right rictyfy images.
The results obtained with the correction are below:

	
Left image not rectify	Right image not rectify

As you can see from the two pictures above, the stereo camera has a phase shift along the Y axis. In particular we note this detail:

After the rectification this is the result:



3. Matching stereo

According to the description in chapter 6 regarding the stereo matching part, let's see how the laser pointer was detected.

The laser has its own tone and brightness. Considering that the image returned by the camera is in GBR color space, the idea is to scan frame-by-frame a ROI (*Region Of Interested*) in which, according to the range of hue defined, it goes in search of the laser pointer.

The brightness ranges are defined in this way:

```
# Define range of red color in BRG
lower_red = np.array([0, 0, 255])  #BGR
upper_red = np.array([240, 240, 255])  #BGR
```

In which:

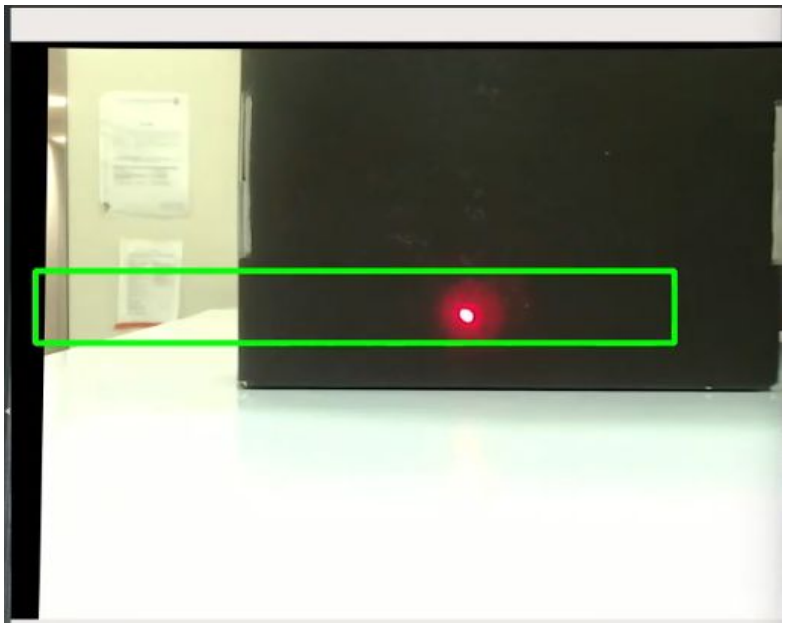
[0 , 0 , 255]	red color in GBR color space
[240 . 240 , 255]	white color in GBR color space

Once this range has been defined, it is possible to search for the laser in a ROI:

```
# We define the ground-thruth of the laser in a ROI
maskL = cv2.inRange( im_left_remapped[190:250 , 20:550] , lower_red , upper_red)
maskR = cv2.inRange( im_right_remapped[190:250 , 20:550] , lower_red , upper_red)

# We find the coordinates of the laser detected
(_, _, minLoc_L , maxLoc_L) = cv2.minMaxLoc( maskL )
(_, _, minLoc_R , maxLoc_R) = cv2.minMaxLoc( maskR )
```

Given an image of (640,480) the ROI in which the search for the laser pointer occurs is the rectangle consisting of the following coordinates: **[190:250,20:550]**.
The results are:



In particular:

The ROI was introduced essentially to limit the matching between the given range(*of laser*) and other "shimmering" objects that are in the scene.

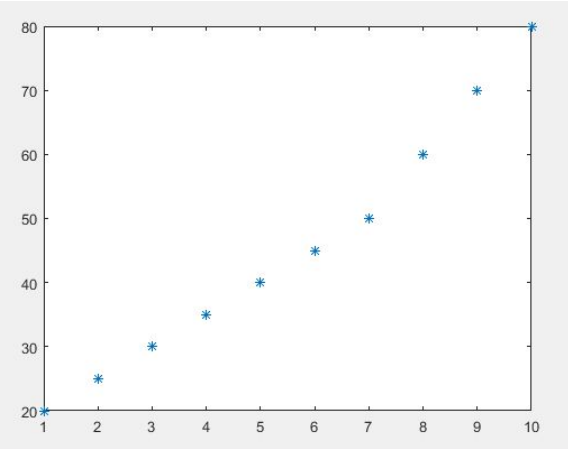
Once the laser has been identified, its coordinates are identified by the array *max_Loc_L* and *max_loc_R*. So you can calculate the disparity

```
# Calculate disparity
disp = abs( maxLoc_R[0] - maxLoc_L[0] )
```

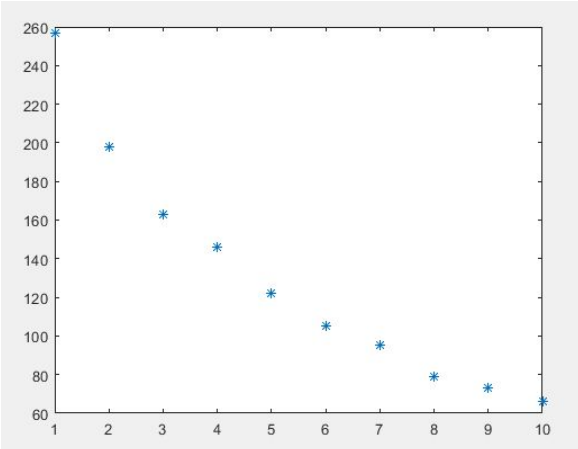
4. Distance

To obtain an estimate of the B * f product, several disparity measurements were made with the laser pointer placed at a different distance from the StereoCamera.

Measurements	**Distance (± 0.2 cm)**	**Disparity**
1	20cm	257
2	25cm	198
3	30cm	163
4	35cm	146
5	40cm	122
6	45cm	105
7	50cm	95
8	60cm	79
9	70cm	73
10	80cm	66



(distance at different moments of time)

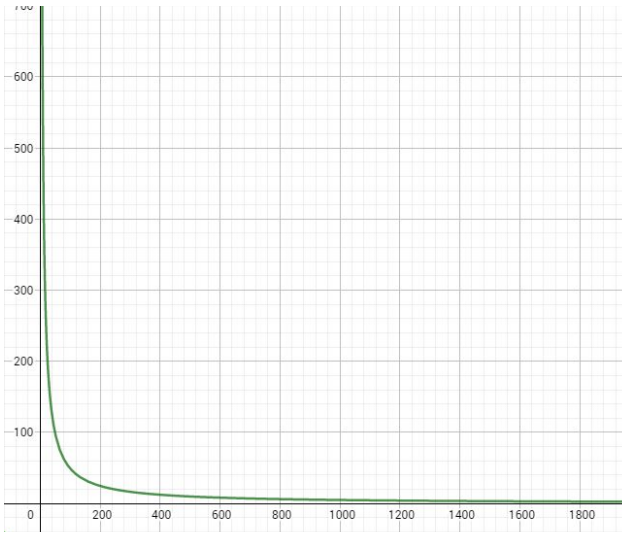


(disparity at different moments of time)

According to paragraph 6, for small distances (objects close to the camera) the disparity values are high, as we move away from the camera (objects away from the camera) the values of disparity increase.

The value of **C** obtained is: **4942.**

The formula become: z= 4942 / d



Once we have obtained the value of c we can implement the formula:

$$z = \frac{c}{d}$$

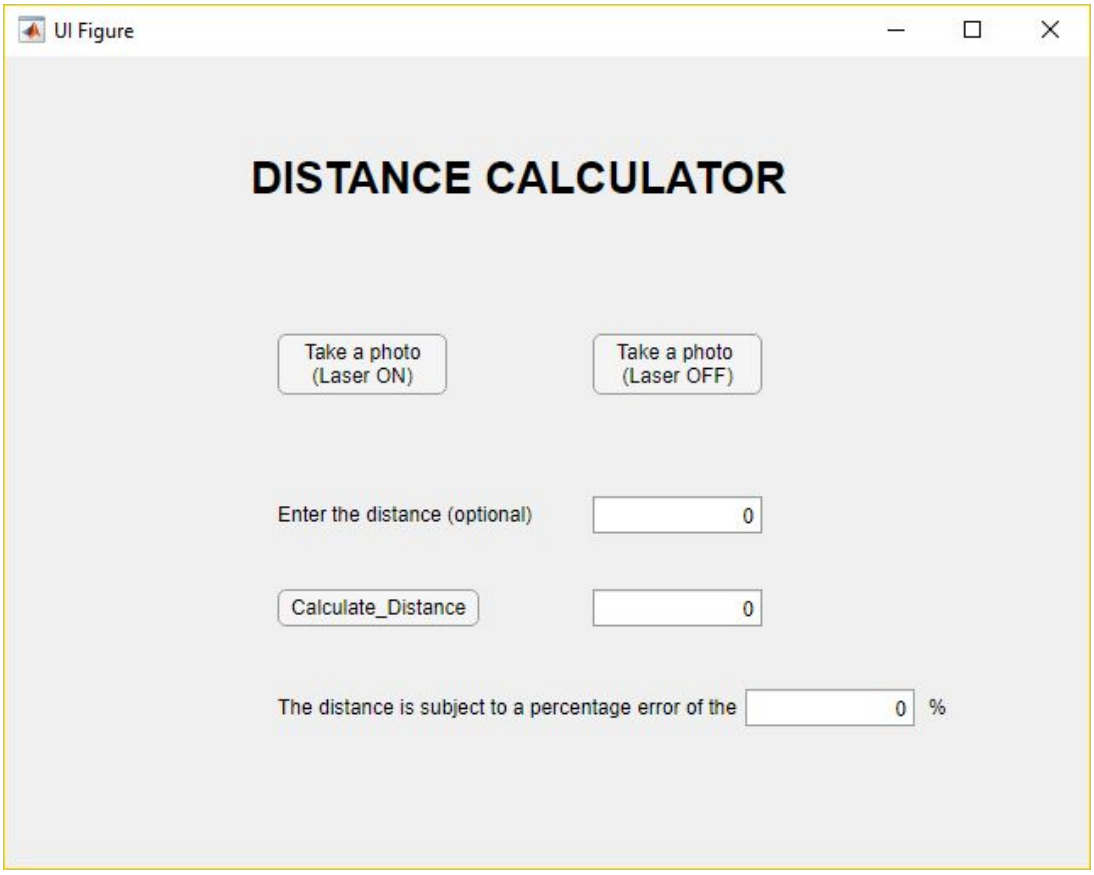
as follow:

```
# Calculate distance
distance = 4942 / disp
```

9. Graphics user interface

For a user-friendly experience, graphical interfaces have been developed that allow the control of specific tasks.
The interfaces developed for the MatLab code and for the python code will be presented and discussed below.

9.1 Gui on MatLab



The development of the graphical interface took place with the use of the app *AppDesigner.*
App Designer integrates the two main phases of the creation of applications (creation of visualization components and programming of the behavior of the app). Drag and drop visual components into the design area and use alignment tips to get a precise layout. App Designer automatically generates object-oriented code, which specifies the layout and design of your application.

The code implemented by the application is, for completeness, the following

```
classdef app < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        TakeaphotoLaserONButton     matlab.ui.control.Button
        TakeaphotoLaserOFFButton    matlab.ui.control.Button
        Calculate_DistanceButton    matlab.ui.control.Button
```

```
        Calculate_DistanceEditField    matlab.ui.control.NumericEditField
        EnterthedistanceoptionalLabel  matlab.ui.control.Label
        DISTANCECALCULATORLabel        matlab.ui.control.Label
        EnterthedistanceoptionalEditField  matlab.ui.control.NumericEditField
        ThedistanceissubjecttoapercentageerroroftheLabel  matlab.ui.control.Label
        Label                          matlab.ui.control.Label
        ThedistanceissubjecttoapercentageerroroftheEditField
matlab.ui.control.NumericEditField
    end

    methods (Access = private)

        % Button pushed function: TakeaphotoLaserONButton
        function TakeaphotoLaserONButtonPushed(app, event)
            cam= webcam('Pc Camera');
            cam.Resolution= '1280x480'
            cam.WhiteBalanceMode= 'manual';
            cam.WhiteBalance= 2800;
            img= snapshot (cam);
            imshow(img)
            stereoParams= importdata ('C:\Users\Utente\Desktop\stereoParams.mat')


            left = img(1:480,1:640,1:3);
            right = img(1:480,641:1280,1:3);
            [J1,J2]=rectifyStereoImages(left,right,stereoParams);
            frameLeftGray  = rgb2gray(J1);
            frameRightGray = rgb2gray(J2);
            imwrite(frameLeftGray,'frameLeftGray.png')
            imwrite(frameRightGray,'frameRightGray.png')

            pause(3);
            clear cam

        end

        % Button pushed function: TakeaphotoLaserOFFButton
        function TakeaphotoLaserOFFButtonPushed(app, event)
            camera= webcam('PC Camera');
            camera.Resolution= '1280x480'
            cam.WhiteBalanceMode= 'manual';
            cam.WhiteBalance= 2800;
            img1= snapshot (camera);
            imshow(img1)
            stereoParams= importdata ('C:\Users\Utente\Desktop\stereoParams.mat')

            left1 = img1(1:480,1:640,1:3);
```

```
            right1 = img1(1:480,641:1280,1:3);
            [J11,J21]=rectifyStereoImages(left1,right1,stereoParams);
            frameLeftGray1  = rgb2gray(J11);
            frameRightGray1 = rgb2gray(J21);
            imwrite(frameLeftGray1,'frameLeftGray1.png')
            imwrite(frameRightGray1,'frameRightGray1.png')

            pause(3);
            clear camera
        end

        % Button pushed function: Calculate_DistanceButton
        function Calculate_DistanceButtonPushed(app, event)
          frameLeftGray= imread('C:\Users\Utente\Desktop\frameLeftGray.png')
          frameLeftGray1= imread('C:\Users\Utente\Desktop\frameLeftGray1.png')
          frameRightGray= imread('C:\Users\Utente\Desktop\frameRightGray.png')
          frameRightGray1= imread('C:\Users\Utente\Desktop\frameRightGray1.png')

           immagine=(frameLeftGray- frameLeftGray1);
           imwrite(immagine,'immagine.png')
n=1;
m=1;
for i=1:1:409
    for j= 1:1:554
        if immagine(i,j)> 100
            a(n)=i;
            b(m)=j;
            n=n+1;
            m=m+1;
        end

    end
end

%calcolo posizione del laser nella foto di sinistra
contatore_righe=0;
contatore_colonne=0;

for j= 1:1:m-1
    contatore_colonne=contatore_colonne+b(j);
end
for i=1:1:n-1

 contatore_righe= contatore_righe+a(i);


end
```

```
posy=floor(contatore_righe/(n-1));
posx=floor(contatore_colonne/(m-1));

%calcolo la posizione del laser nella immagini destra

z=1;
c=1;
immagine1=(frameRightGray- frameRightGray1)
imwrite(immagine1,'immagine1.png')
for i=1:1:409
   for j= 1:1:554
      if immagine1(i,j)> 100
         s(z)=i;
         t(c)=j;
         z=z+1;
         c=c+1;
      end

   end
end

%calcolo posizione del laser nella foto di sinistra
contatore_righe1=0;
contatore_colonne1=0;

for j= 1:1:c-1
   contatore_colonne1=contatore_colonne1+t(j);
end
for i=1:1:z-1

 contatore_righe1= contatore_righe1+s(i);


end


posy1=floor(contatore_righe1/(z-1));
posx1=floor(contatore_colonne1/(c-1));
differenza=posx-posx1;

distanza=5458/abs(differenza)

app.Calculate_DistanceEditField.Value= distanza

errore=abs((app.EnterthedistanceoptionalEditField.Value-distanza)/app.Enterthedistanceoptio
```

```
nalEditField.Value)
app.ThedistanceissubjecttoapercentageerroroftheEditField.Value= errore*100

        end
    end

    % App initialization and construction
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure
            app.UIFigure = uifigure;
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'UI Figure';

            % Create TakeaphotoLaserONButton
            app.TakeaphotoLaserONButton = uibutton(app.UIFigure, 'push');
            app.TakeaphotoLaserONButton.ButtonPushedFcn = createCallbackFcn(app,
@TakeaphotoLaserONButtonPushed, true);
            app.TakeaphotoLaserONButton.Position = [162 281 100 36];
            app.TakeaphotoLaserONButton.Text = {'Take a photo'; '(Laser ON)'};

            % Create TakeaphotoLaserOFFButton
            app.TakeaphotoLaserOFFButton = uibutton(app.UIFigure, 'push');
            app.TakeaphotoLaserOFFButton.ButtonPushedFcn = createCallbackFcn(app,
@TakeaphotoLaserOFFButtonPushed, true);
            app.TakeaphotoLaserOFFButton.Position = [348 281 100 36];
            app.TakeaphotoLaserOFFButton.Text = {'Take a photo '; '(Laser OFF)'; ''};

            % Create Calculate_DistanceButton
            app.Calculate_DistanceButton = uibutton(app.UIFigure, 'push');
            app.Calculate_DistanceButton.ButtonPushedFcn = createCallbackFcn(app,
@Calculate_DistanceButtonPushed, true);
            app.Calculate_DistanceButton.Position = [162 144 119 22];
            app.Calculate_DistanceButton.Text = {'Calculate_Distance'; ''};

            % Create Calculate_DistanceEditField
            app.Calculate_DistanceEditField = uieditfield(app.UIFigure, 'numeric');
            app.Calculate_DistanceEditField.Position = [348 144 100 22];

            % Create EnterthedistanceoptionalLabel
            app.EnterthedistanceoptionalLabel = uilabel(app.UIFigure);
            app.EnterthedistanceoptionalLabel.Position = [162 196 181 28];
            app.EnterthedistanceoptionalLabel.Text = 'Enter the distance (optional)';
```

```
            % Create DISTANCECALCULATORLabel
            app.DISTANCECALCULATORLabel = uilabel(app.UIFigure);
            app.DISTANCECALCULATORLabel.FontSize = 26;
            app.DISTANCECALCULATORLabel.FontWeight = 'bold';
            app.DISTANCECALCULATORLabel.Position = [146 373 327 73];
            app.DISTANCECALCULATORLabel.Text = 'DISTANCE CALCULATOR';

            % Create EnterthedistanceoptionalEditField
            app.EnterthedistanceoptionalEditField = uieditfield(app.UIFigure, 'numeric');
            app.EnterthedistanceoptionalEditField.Position = [348 199 100 22];

            % Create ThedistanceissubjecttoapercentageerroroftheLabel
            app.ThedistanceissubjecttoapercentageerroroftheLabel = uilabel(app.UIFigure);
            app.ThedistanceissubjecttoapercentageerroroftheLabel.Position = [162 85 394 22];
            app.ThedistanceissubjecttoapercentageerroroftheLabel.Text = 'The distance is subject
to a percentage error of the';

            % Create Label
            app.Label = uilabel(app.UIFigure);
            app.Label.Position = [546 85 25 22];
            app.Label.Text = '%';

            % Create ThedistanceissubjecttoapercentageerroroftheEditField
            app.ThedistanceissubjecttoapercentageerroroftheEditField = uieditfield(app.UIFigure,
'numeric');
            app.ThedistanceissubjecttoapercentageerroroftheEditField.Position = [438 85 100 22];
        end
    end

    methods (Access = public)

        % Construct app
        function app = app

            % Create and configure components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)
```

```
        % Delete UIFigure when app is deleted
        delete(app.UIFigure)
    end
  end
end
```

The user, thanks to the click on the appropriate buttons, will be able to choose the desired functionality.
The features available are:

1. Take a photo (laser ON)
2. Take a photo (laser OFF)
3. Enter the distance (optional)
4. Calculate_Distance
5. The distance is subject a percentage error of…

Let's explain the functionality of the components mentioned above:

1. The Take a photo (Laser ON**)** button allows you to take pictures of the scene characterized by the presence of the laser.
   At the end of the shot the user will be notified with the photo obtained.
   The program divides the image in left and right images and then after having rectified them, it converts them to grayscale.
   Once the program has finished the images converted in grayscale will be saved.

2. The Take a photo (Laser OFF**)** button allows you to take pictures of the scene without the presence of the laser.
   At the end of the shot the user will be notified with the photo obtained.
   The program divides the image in left and right images and then after having rectified them, it converts them to grayscale.
   Once the program has finished the images converted in grayscale will be saved.

3. The Enter the distance (optional) button allows you to enter the effective distance of the measured object to evaluate the effective effectiveness of the app or to calculate the percentage error on the measurement.

4. The Calculate_distance button allows you to measure the distance of a object. In the last button after calculating the distance print the percentage of error on the measurement.

9.2 Gui on Python

The development of the graphical interface took place with the use of the *tkInter library.*
The code is implemented in Python modules, one for each of the application's functionalities.
The main module that is launched by the user is *Main.py*.
The user, thanks to the click on the appropriate buttons, will be able to choose the desired functionality.

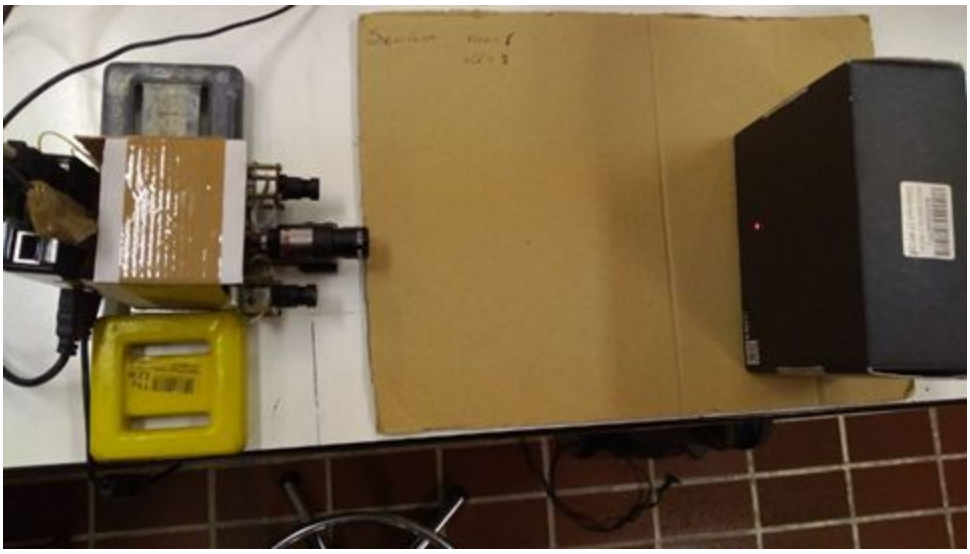
The features available are:

6. Scatta
7. Preview
8. Accendi laser
9. Spegni laser
10. Misura

10. Measurements on MatLab and Python

We have acquired a series of measurements, following different criteria, in order to provide a more accurate estimate of the accuracy and accuracy of the device. To obtain this information we have created a series of samples to validate the model or not.
The support (camera and laser) was placed on a fixed position, which remained unchanged during all measurements. We then took an object and placed it at an

incremental distance. The table light would not reflect the laser light. Finally, the support has been covered sideways and in the upper part with some pieces of cardboard to prevent the lens from being exposed to light. With artificial neon light.
For each distance value, eight measurements were repeated and a statistical analysis was performed on these samples. We are not interested in the hardware, we decided to approximate our first digit after the decimal point (millimeter).



Following are the results obtained.

10.1 MatLab results

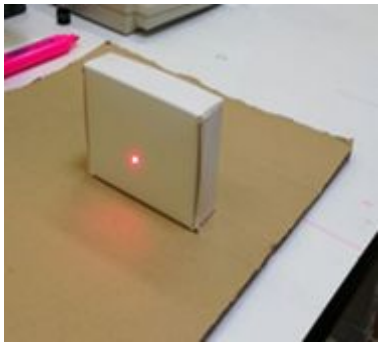
1. *Object at 20 cm (± 0.2 cm) distance*

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)² (cm²)	
20.8	0	0	
20.8	0	0	
20.8	0	0	
20.8	0	0	
20.8	0	0	
average = 20.8 cm		variance = 0 cm²	standard deviation = 0 cm

Absolute error = | real distance – average |= 0.8 cm

Relative error % = 4%

We realized that, keeping everything static, the measurements showed a zero standard deviation.At this point we decided to modify our evaluation method and, instead of repeating the calculation on the same object, we evaluated how much our algorithm was sensitive to object variation.For each distance considered, we performed three evaluations using:

a) a black cardboard box (*range = 100*)*	**b)** a white cardboard box (*range = 80*)*	**c)** a piece of wood (*range = 100*)*
		

We consider the results obtained:

2. **Real distance = 20 cm (± 0.2 cm)**

	Black object	White object	Piece of wood
Distance detected	20.8 cm	20.5	20.5 cm
Absolute error	0.8 cm	0.5 cm	0.5 cm
Relative error %	4.0 %	2.5 %	2.5 %

3. **Real distance = 40 cm (± 0.2 cm)**

	Black object	White object	Piece of wood
Distance detected	41.0 cm	41.0 cm	41.3 cm
Absolute error	1.0 cm	1.0 cm	1.3 cm
Relative error %	2.5 %	2.5 %	3.3 %

4. **Real distance = 50 cm (± 0.2 cm)**

	Black object	White object	Piece of wood
Distance detected	50.5 cm	50.5 cm	51 cm
Absolute error	0.5 cm	0.5 cm	1 cm
Relative error %	1.0 %	1.0 %	2.0 %

5. **Real distance = 60 cm (± 0.2 cm)**

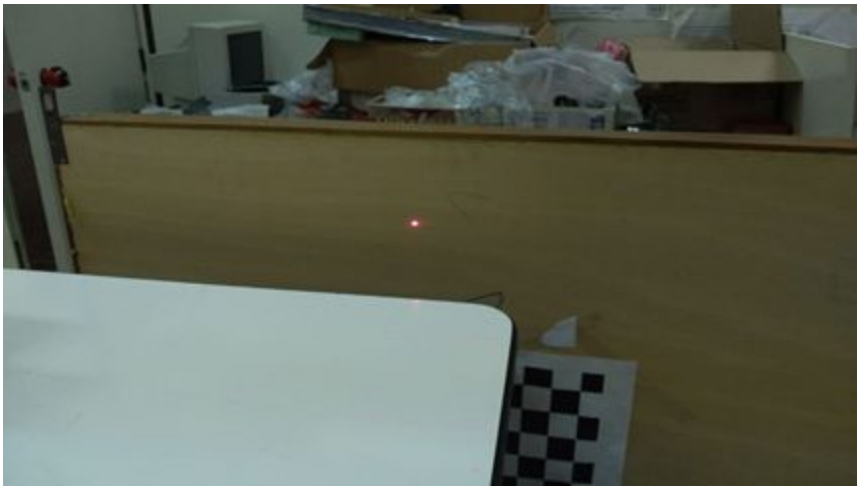
	Black object	White object	Piece of wood
Distance detected	59.9 cm	59.9 cm	59.9 cm
Absolute error	0.9 cm	0.9 cm	0.9 cm
Relative error %	1.5 %	1.5 %	1.5 %

6. **Real distance = 80 cm (± 0.2 cm)**

	Black object	White object	Piece of wood
Distance detected	80.3 cm	80.3 cm	80.3 cm
Absolute error	0.3 cm	0.3 cm	0.3 cm
Relative error %	0.4 %	0.4 %	0.4 %

Proceeding in this mode we realized that, even if we changed the objects, the distance measured did not change.

For the measurements at: *100 cm (± 0.2 cm)*, *120 cm (± 0.2 cm)* and 1*40 cm (± 0.2 cm)*, only one object was used (*a wood panel*).



The results obtained are the following:

7. **Real distance = 100 cm (± 0.2 cm)**

	Piece of wood
Distance detected	99.2 cm
Absolute error	0.8 cm
Relative error %	0.8 cm

8. **Real distance = 120 cm (± 0.2 cm)**

	Piece of wood
Distance detected	116 cm
Absolute error	4 cm
Relative error %	3.3 cm

9. **Real distance = 140 cm (± 0.2 cm)**

	Piece of wood
Distance detected	136.5cm
Absolute error	3.5 cm
Relative error %	2.5 cm

Considerations

We now report in a general table the relative error values of the previous experiments.

Real distance	20 cm	40 cm	50 cm	60 cm	80 cm	100 cm	120 cm	140 cm
Relative error %	2.5 – 4.0 %	2.5 % - 3.3 %	1.0 % - 2.0 %	1.5 %	0.4 %	0.8 %	3.3 %	2.5 %

The only thing we can deduce is that it seems that in the extremes, very close distances and very far distances, the algorithm is getting worse. The most reliable values belong to an intermediate range. This could be caused by the fact that in the estimation phase of the parameter *c* we considered this very range (*we remember c as product B * f, distance = c / disparity*).

In order to improve the evaluation of the distance it would be convenient to estimate c considering even longer distances, this would improve our results at a greater distance, but it would make those at intermediate distances worse.

10.2 Python results

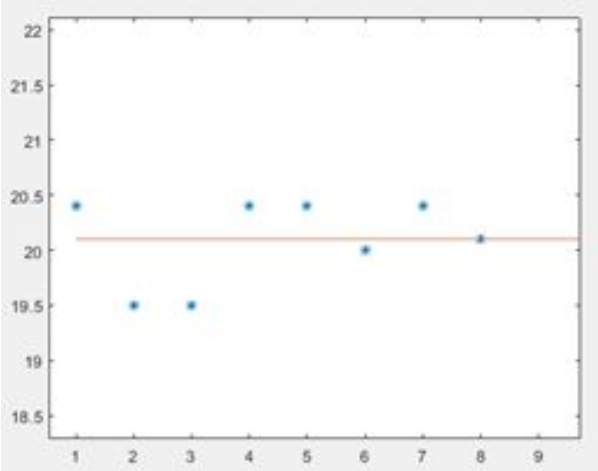
1. Object at 20 cm (± 0.2 cm) distance

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)^2 (cm^2)	
20.4	0.3	0.9	
19.5	-0.6	0.4	
19.5	-0.6	0.4	
20.4	0.3	0.1	
20.4	0.3	0.1	
20	-0.1	0	
20.4	0.3	0.1	
20.1	0	0	
average = 20.1 cm		variance = 0.25 cm^2	standard deviation = 0.5 cm

Absolute error = | real distance – average | = 0.1 cm

Relative error % = 0.5 %

Dispersion of distances detected by their mean value:



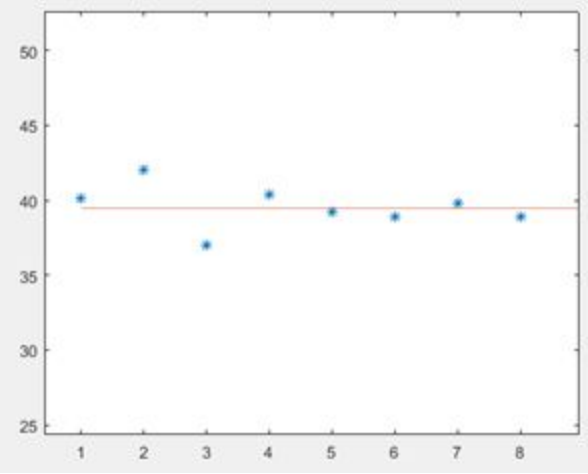
2. Object at 40 cm (± 0.2 cm) distance

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)^2 (cm^2)	
40.1	0.6	0.4	
42	2.5	6.3	
37	-2.5	6.3	
40.4	0.9	0.8	
39.2	-0.3	0.1	
38.9	-0.6	0.4	
39.8	0.3	0.1	
38.9	-0.6	0.4	
average = 39.5 cm		variance = 1.85 cm^2	standard deviation = 1.4 cm

Absolute error = | real distance – average |= 0.5 cm

Relative error % = 1.3 %

Dispersion of distances detected by their mean value:



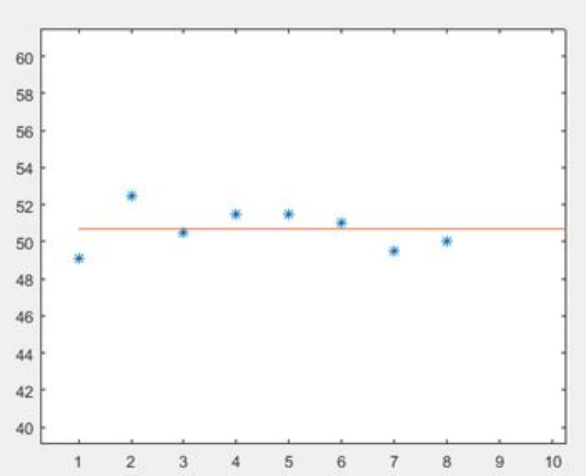
3. Object a 50 cm (± 0.2 cm) distance

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)^2 (cm^2)	
49.1	-1.6	2.6	
52.5	1.8	3.2	
50.5	-0.2	0	
51.5	0.8	0.6	
51.5	0.8	0.6	
51	0.3	0.1	
49.5	-1.2	1.4	
50	-0.7	0.5	
average = 50.7 cm		variance = 1.13 cm^2	standard deviation = 1.1 cm

Absolute error = | real distance – average |= 0.7 cm

Relative error % = 1.4%

Dispersion of distances detected by their mean value:



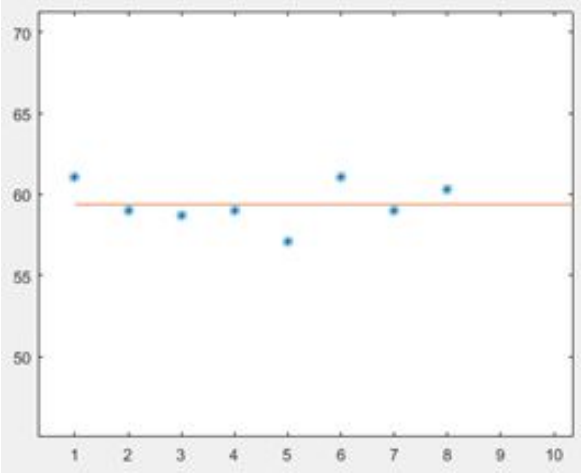
4. Object at 60 cm (± 0.2 cm) distance

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)^2 (cm^2)	
61.1	1.7	2.9	
59	-0.4	0.2	
58.7	-0.7	0.5	
59	-0.4	0.2	
57.1	-2.3	5.3	
61.1	1.7	2.9	
59	-0.4	0.2	
60.3	0.9	0.8	
average = 59.4 cm		variance = 1.63 cm^2	standard deviation = 1.3 cm

Absolute error = | real distance – average |= 0.6cm

Relative errore % = 1.0%

Dispersion of distances detected by their mean value:



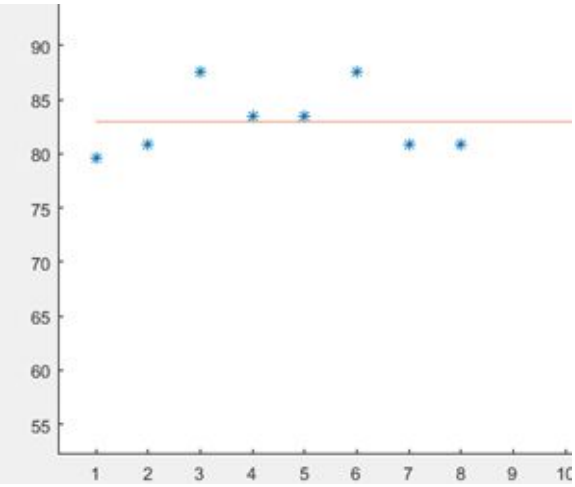
5. Object at 80 cm (± 0.2 cm) distance

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)^2 (cm^2)	
79.6	-3.4	11.6	
80.8	-2.2	4.8	
87.5	4.5	20.3	
83.4	0.4	0.2	
83.4	0.4	0.2	
87.6	4.6	21.2	
80.8	-2.2	4.8	
80.8	-2.2	4.8	
average = 83.0 cm		variance = 8.49 cm^2	standard deviation = 2.9 cm

Absolute error = | real distance – average |= 3 cm

Relative error % = 3.8%

Dispersion of distances detected by their mean value:



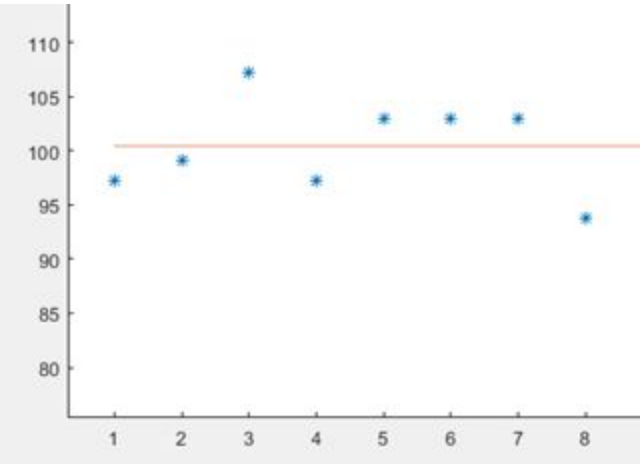
6. Object at 100 cm (± 0.2 cm) distance

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)^2 (cm^2)	
97.2	-3.3	10.9	
99.1	-1.4	2.0	
107.2	6.7	44.9	
97.3	-3.2	10.2	
103.0	2.5	6.3	
103.0	2.5	6.3	
103.0	2.5	6.3	
93.8	-6.7	44.9	
average = 100.5 cm		variance = 16.48 cm^2	standard deviation = 4.1 cm

Absolute error = | real distance – average |= 0.5 cm

Relative error % = 0.5%

Dispersion of distances detected by their mean value:



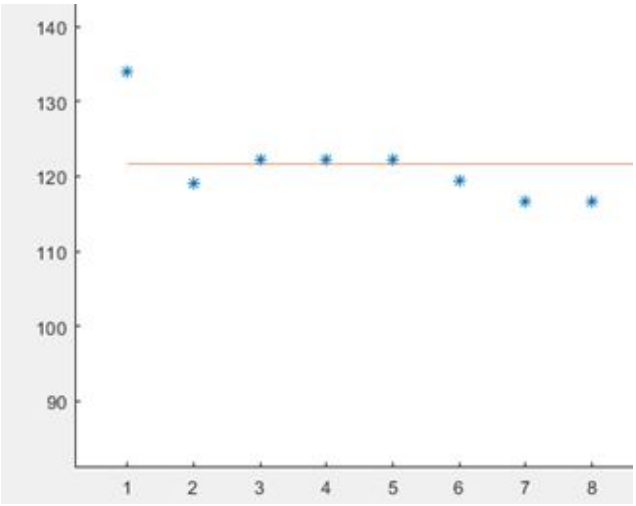
7. Object at 120 cm (± 0.2 cm) distance

distance detected (cm)	distance detected – average (cm)	(distance_detected – average)^2 (cm^2)	
134.0	12.4	153.8	
119.0	-2.6	6.8	
122.2	0.6	0.4	
122.2	0.6	0.4	
122.2	0.6	0.4	
119.4	-2.2	4.8	
116.7	-4.9	24.0	
116.7	-4.9	24.0	
average = 121.6 cm		variance = 26.83 cm^2	standard deviation = 5.2 cm

Absolute error = | real distance – average |= 1.6cm

Relative error % = 1.3%

Dispersion of distances detected by their mean value:



8. Object at 140 cm

By bringing the object to a distance of 140 cm, it is no longer possible to detect the laser as is often not detected by one of the two cameras.

Considerations

The results are in line with expectations:

Real distance (±1mm)	Relative error	Standard deviation
20 cm	0.5 %	0.5 cm
40 cm	1.3 %	1.4 cm
50 cm	1.4 %	1.1 cm
60 cm	1.0 %	1.3 cm
80 cm	3.8 %	2.9 cm
100 cm	0.5 %	4.1 cm
120 cm	1.3 %	5.2 cm
140 cm	//	//

In fact, although the relative error does not seem to vary with a criterion, it is possible to notice how the standard deviation, the dispersion of the measurements around their mean value, increases with increasing distance.

Our model is therefore more reliable for limited distances. This can be caused by the fact that the laser is more difficult to detectably in the distance, due to possible light interference caused by the surrounding environment.

11. Comparison between MatLab e Python

We can now make a comparison between the two methods used, considering the only relative error:

Real distance	20 cm	40 cm	50 cm	60 cm	80 cm	100 cm	120 cm	140 cm
Relative error % Python	0.5 %	1.3 %	1.4 %	1.0 %	3.8 %	0.5 %	1.3 %	//
Relative error % MatLab	2.5 – 4.0 %	2.5 % - 3.3 %	1.0 % - 2.0 %	1.5 %	0.4 %	0.8 %	3.3 %	2.5 %

Considering these results and the only relative error, the algorithm in Python is better in short distances. By increasing the distance the algorithm in Python is no longer able to detect the laser. Although the results in Matlab have relatively high relative errors, it is still possible to use it at greater distances.

12. The problem of range on MatLab environment

Study of laser in default position

The algorithm developed in MatLab environment presents a phase in which, after performing a subtraction of the matrices representing the photos:

left in gray scale with laser - left in gray scale without laser

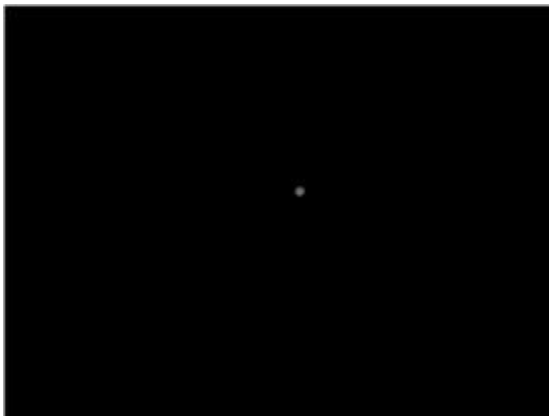
right in gray scale with laser - right in gray scale without laser

we go to detect the laser (let's see te function *Matching stereo* at pag. ancora boh) Also the images obtained from the subtraction are in *gray scale*. They are composed of pixels that have a color that goes from white to black. In the *gray scale* the white pixels are represented by a coefficient equal to 255, while the black pixels by a coefficient equal to 0.

In order to detect the laser we set a range for the coefficient that represents the pixel gradation in the *gray scale*.

We reset all the pixels that have a gradation below this range and that do not represent the laser but at most the noise around it. By modifying the value of the RANGE we are going to modify which of the pixels we consider as representative of the laser.

In the following photos we can see how the color of the pixels changes for an object placed at 20 cm from the camera (white pixels) and for a place at 80 cm (gray pixels).

20 cm (± 0.2 cm)	80 cm (± 0.2 cm)
	

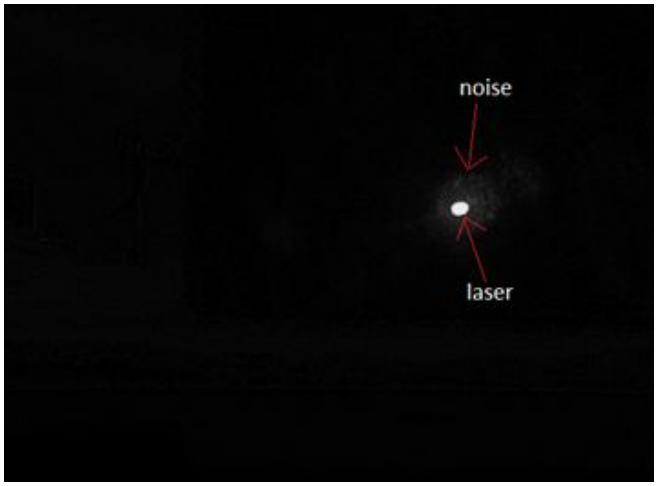
So an adaptation of the range is necessary based on the distance the object is located.It must be lowered in case of greater distance.The adaptation of the range according to the distance at which the object is located from the camera also allows us to eliminate the noise, as previously described.

We report below an analysis of the range related to two acquisitions:

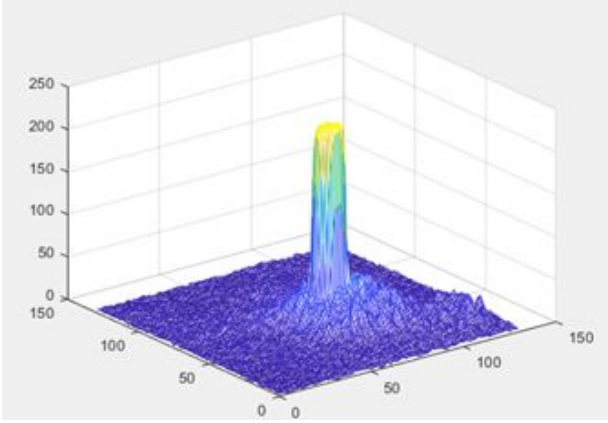
- one relative to an object placed at a distance of 20 cm from the camera
- one relative to object placed at 80 cm

The acquisitions were made with the same criteria described in the section 1.1 of this report.

1. **object at 20 cm (± 0.2 cm) distance**



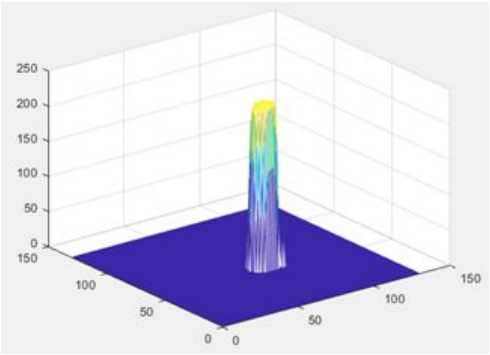
This picture represents the image obtained from the subtraction of the matrix relative to the left frame with laser (*in gray scale*) and the matrix relative to the left frame without laser (*in gray scale*). It was acquired by setting the range to 0. This means that we initially consider all the pixels of any grayscale gradation. In the image the white pixels represent the laser. Around them it is possible to notice some noise, represented by some gray pixels that form a halo around the white ones. At this point we have cut out a window around the pixels that represent the laser and we have built the 3D image.



It is possible to observe the noise around the laser, represents by the ripples.

Now we begin to increase the range to discard an increasing number of pixels and focus only on those considered representative.

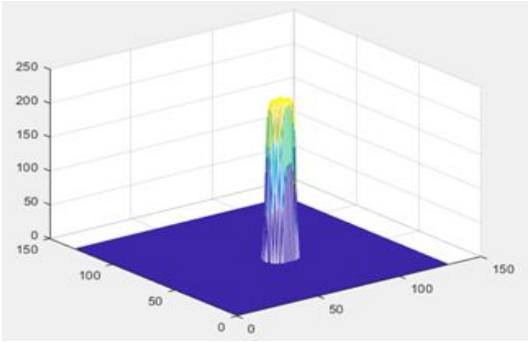
- **range: 60**



Number of pixels representing the laser: *173.*

The noise has almost disappeared.

- **range: 61**

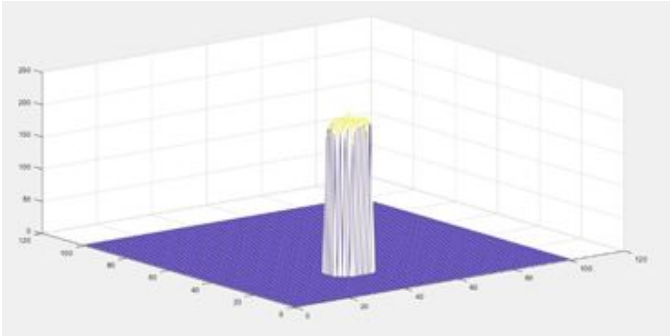


Number of pixels representing the laser: *172.*

We can consider the noise disappeared.

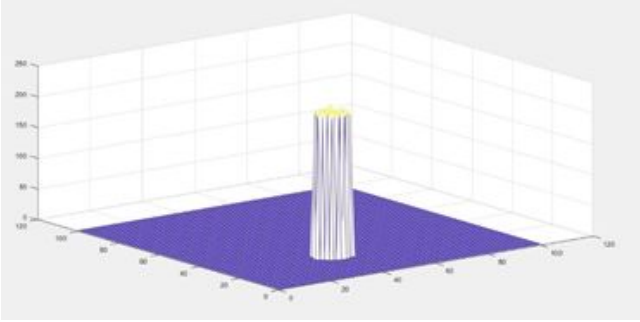
Increasing the range we are going to evaluate if the number of representative pixels of the laser decreases drastically.

- **range: 200**



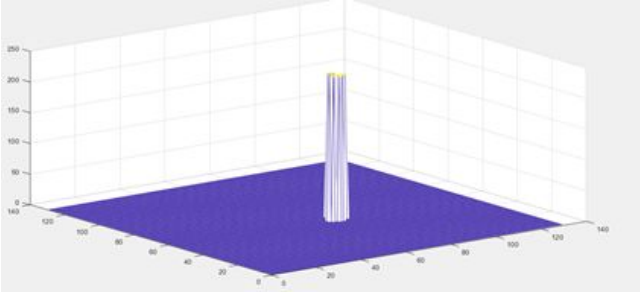
Number of pixels representing the laser: *115*

- **range: 230**



Number of pixels representing the laser: *65*

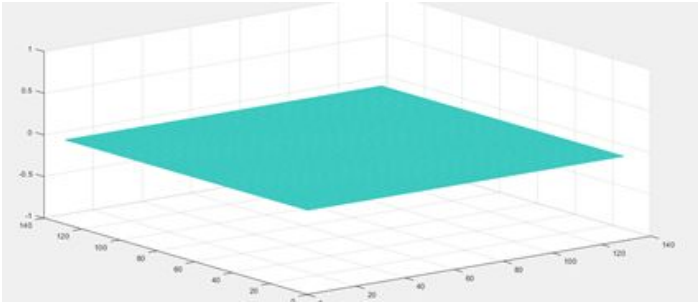
- **range: 234**



Number of pixels representing the laser: *7*

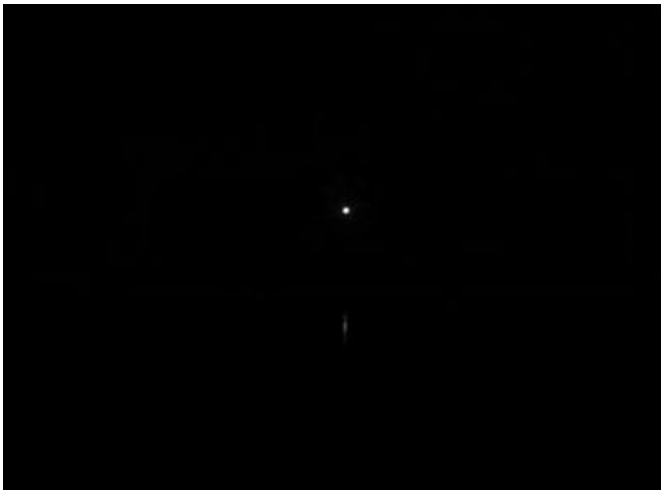
At this point we can consider that the change in the number of representative pixels of the laser is quite substantial, in fact, by increasing the range slightly, we are discarding all the pixels.

- **range: 237**



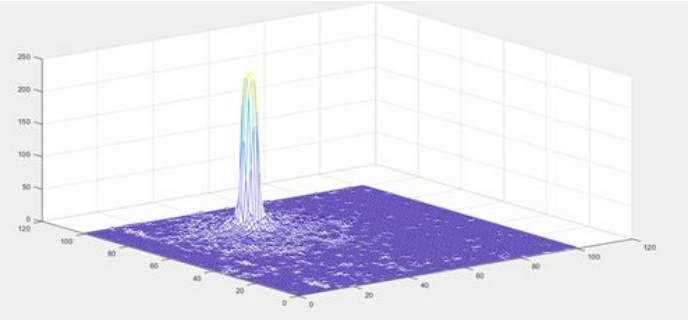
At this point we repeated the same measurements considering the object at a greater distance, equal to 80 cm (± 0.2 cm).

2. object at 80 cm (± 0.2 cm) distance



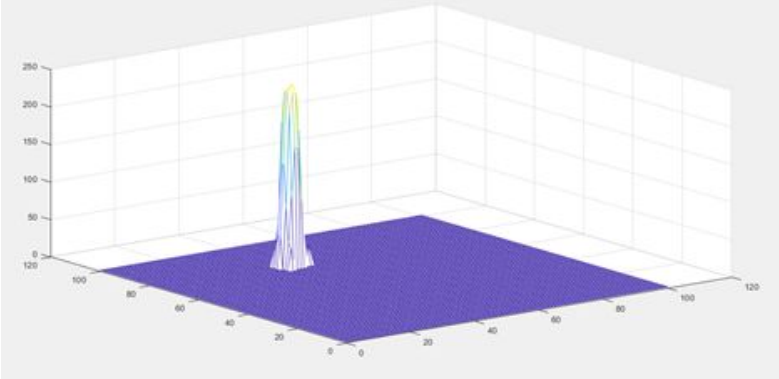
This picture represents the image obtained from the subtraction of the matrix relative to the left frame with laser *(in gray scale*) and the matrix relative to the left frame without laser *(in gray scale*). It was acquired by setting the range to 0. We can see how, with increasing distance, the noise is reduced, in fact it is also difficult to distinguish it from the laser in the previous image.

Cutting out a window around the pixels that represent the laser, we have built the 3D image.



Now we begin to increase the range to discard an increasing number of pixels and focus only on those considered representative.

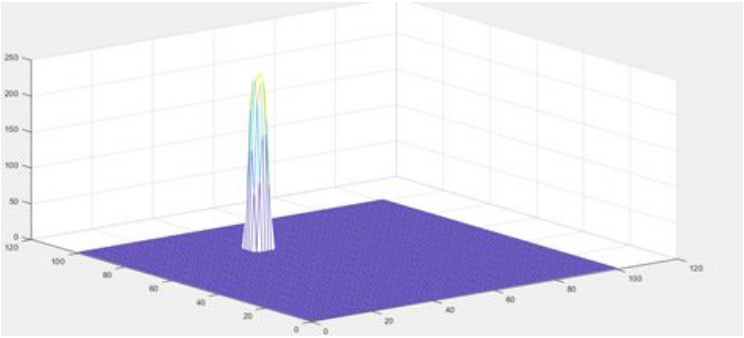
- **range: 26**



Number of pixels representing the laser: *71*

Some noise is still observable.

- **range: 45**

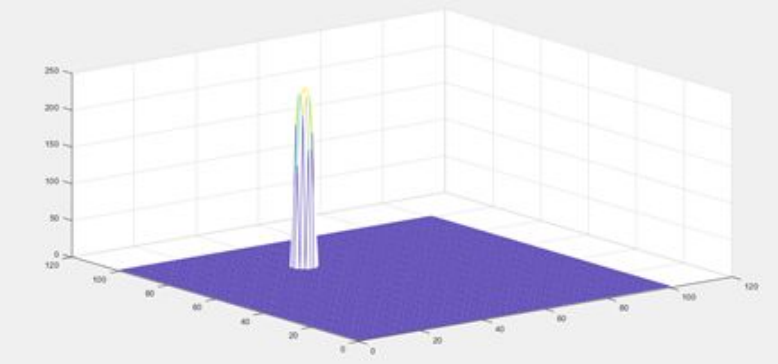


Number of pixels representing the laser: *47* .

We can consider the noise disappeared.

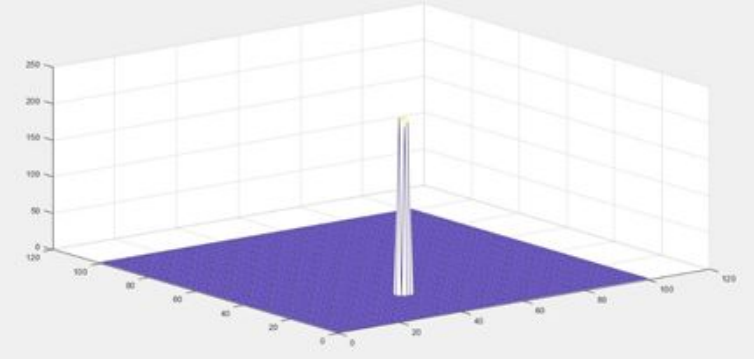
Increasing the range we are going to evaluate if the number of representative pixels of the laser decreases drastically.

- **range: 100**



Number of pixels representing the laser: *23*

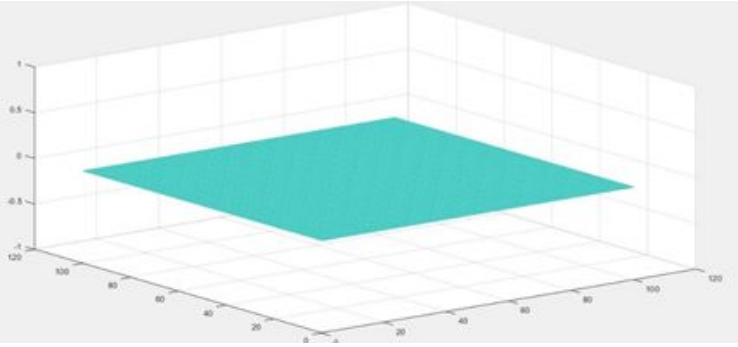
- **range: 230**



Number of pixels representing the laser: *8.*

At this point we can consider that the change in the number of representative pixels of the laser is quite substantial, in fact, by increasing the range, we are discarding all the pixels.

- **range : 250**



Comparing the values obtained in test 1) and 2), we can say that at shorter distances the range can be increased more than what can be done at greater distances.This is caused by being closer, the laser is mapped into the images acquired with a greater number of pixels that have gradation tending to white. At this point it is possible to reset a greater number of pixels and therefore increase the coefficient we have defined as a *range.*

12.1 Laser inclination

As we said in the section 3.2 the beam of light coming from the laser can be modified in its direction. The laser position is modified by screwing and unscrewing the hexagonal screws on the side of the laser.

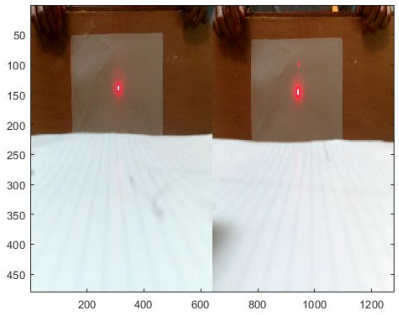
Following some steps we studied the possible inclinations x-y evaluating the maximum laser travel in angles and then we repeated the study of *range* considering the inclination.

The procedure we followed consists of the following steps:

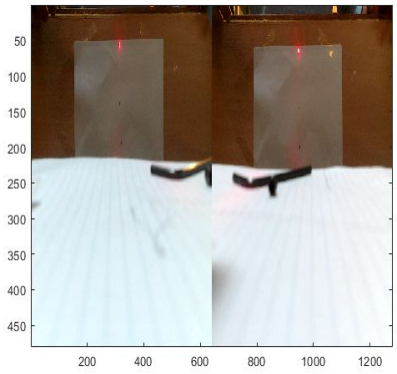
1. we measured the distance between our subject and our camera. The distance we calculated amounts to 49 cm
2. we took a picture with the laser positioned exactly in the middle *(phase shift 0 on both the x and the y)*
3. we increased the y up to the maximum value and we took the photo;
4. we decreased the y up to the minimum value and we took the picture;
5. we reported the y in its centered position;
6. we moved the x to the left up to its limit and we took the picture;
7. we moved the x to the right up to its limit and we took the picture;

Here there are the images obtained from this work in succession:

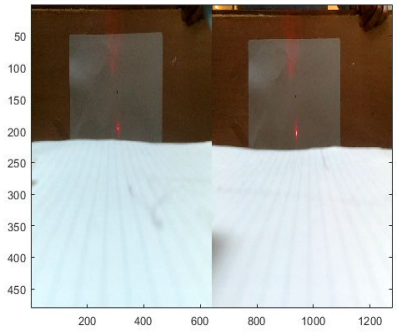
1)



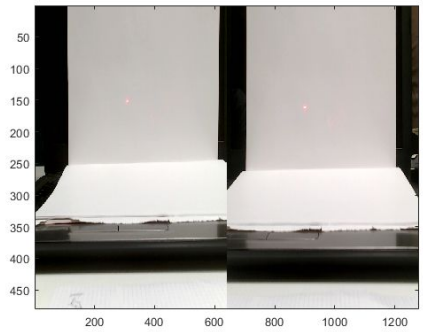
2)


50
100
150
200
250
300
350
400
450
200
400
600
800
1000
1200

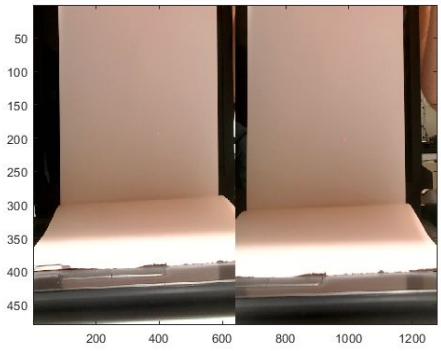
3)


50
100
150
200
250
300
350
400
450
200
400
600
800
1000
1200

4)


50
100
150
200
250
300
350
400
450
200
400
600
800
1000
1200

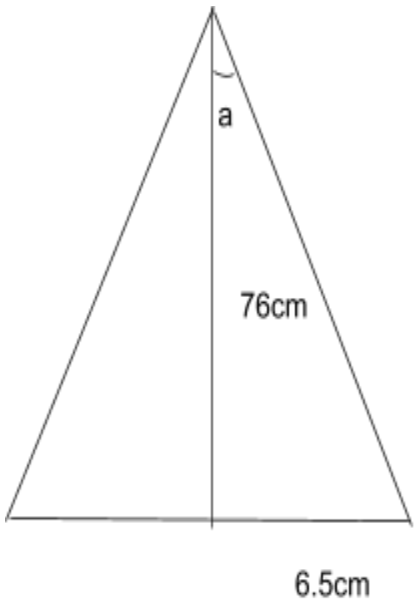
5)



The results we obtained are the following:

1. the distance between the maximum point of the y and the minimum point is 13 cm;
2. the distance between the maximum point of x and the minimum point is 13 cm;

At this point we have succeeded in modeling the opening angle of our laser through trigonometry.
We report below the calculations we conducted on the y that will obviously be mirrored to those performed on the x.



The accounts executed are as follows:

$$6.5 = 76 * tga$$

$$a = arctg\ (6.5/76)$$

$$a = 0.0853\ rad$$

Our result, however, is expressed in radians, so what we need to do is to make a conversion of our result into degrees.

$$x = 0.0853 * 180\ /pi$$

$$x = 5°$$

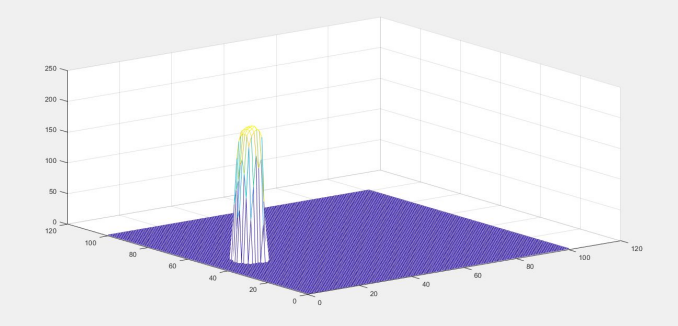
12.2 The problem of range, changing the laser direction

We study how the range and the number of pixel, representing the laser in the frames, are modified by making a displacement along the axis and ungo the axis, and placing the objects at different distances from the camera (*20 cm and 80 cm*).

1) **object at 20 cm (± 0.2 cm) distance**

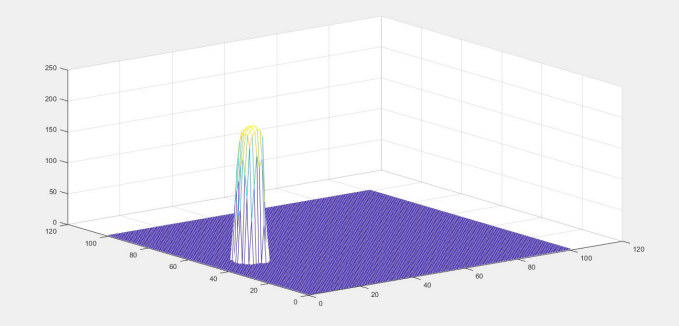
Laser displacement upward (y axis)

- **range: 60**



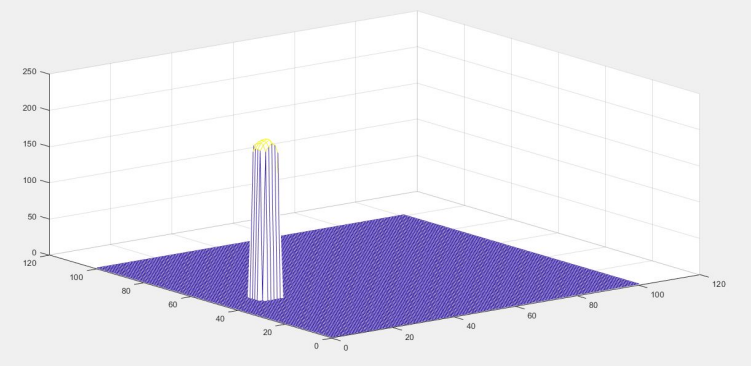
Number of pixels representing the laser: *64.*

- **range: 61**



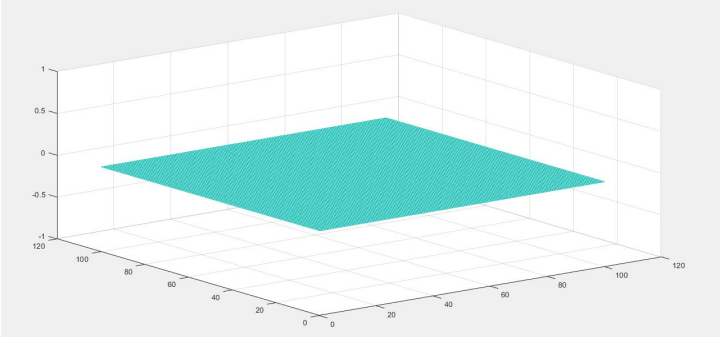
Number of pixels representing the laser: *64.*

- **range: 200**



Number of pixels representing the laser: *28.*

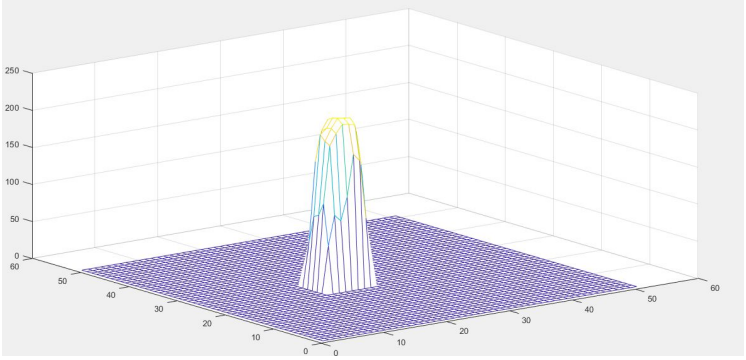
- **range:230**



Number of pixels representing the laser: *23.*

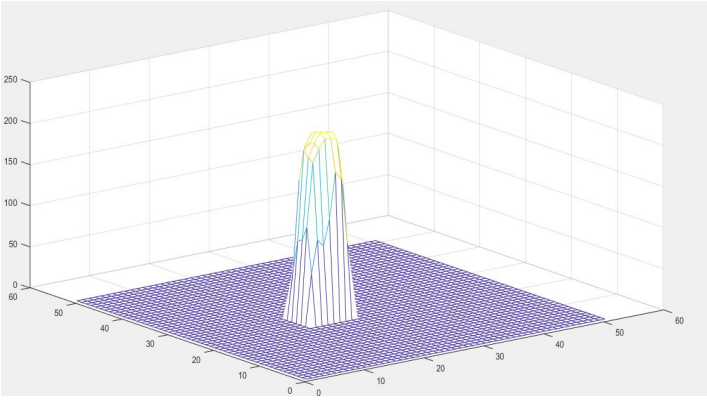
Laser displacement down (y axis)

- **range: 60**



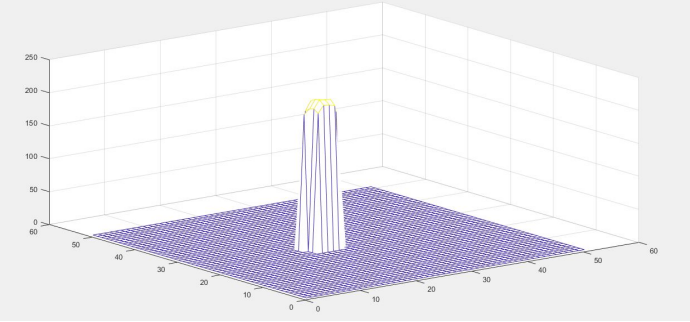
Number of pixels representing the laser: *59.*

- **range: 61**



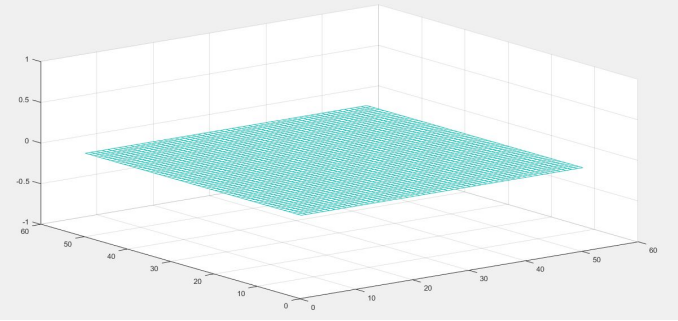
Number of pixels representing the laser: *59.*

- **range: 200**



Number of pixels representing the laser: *21.*

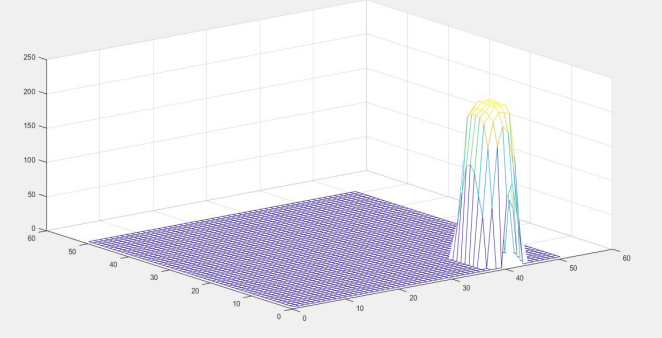
- **range:230**



Number of pixels representing the laser: *0.*

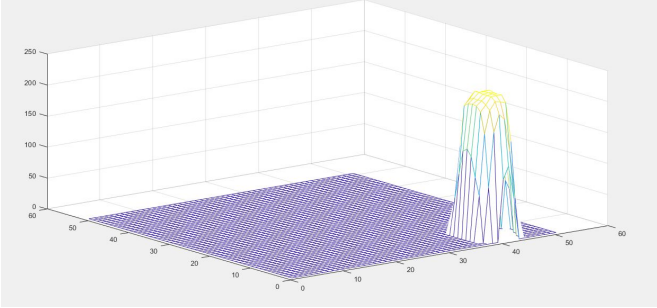
Laser displacement to the left (x axis)

- **range: 60**



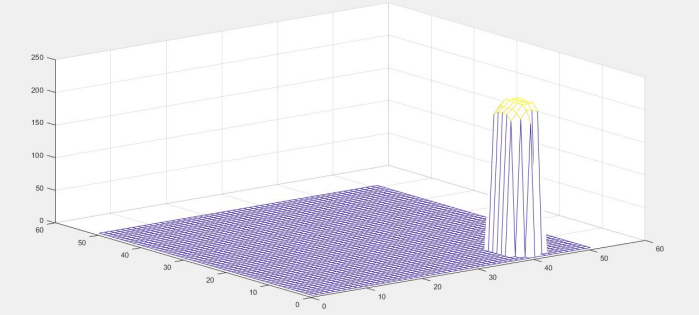
Number of pixels representing the laser: *71.*

- **range: 61**



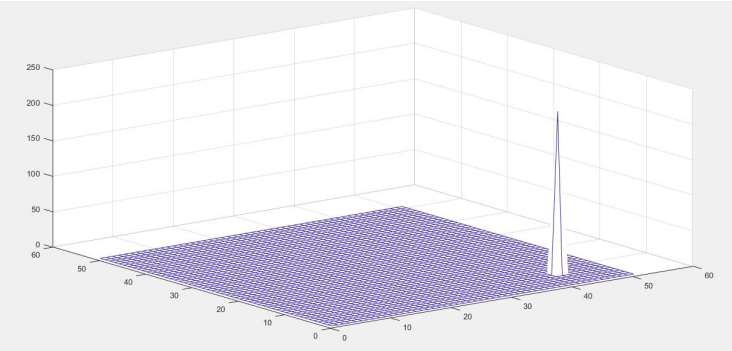
Number of pixels representing the laser: *70.*

- **range: 200**



Number of pixels representing the laser: *39.*

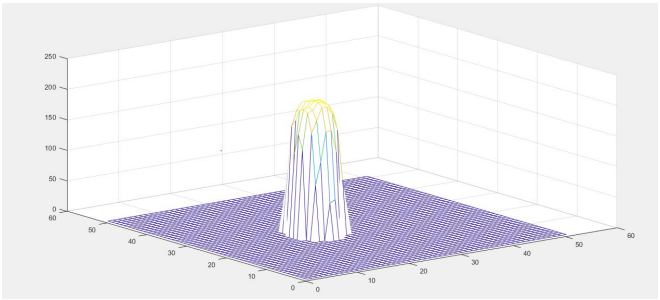
- **range:230**



Number of pixels representing the laser: *2.*

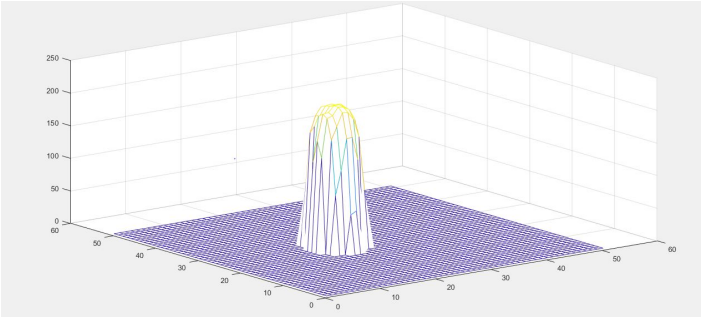
Laser displacement to the right (x axis)

- **range: 60**



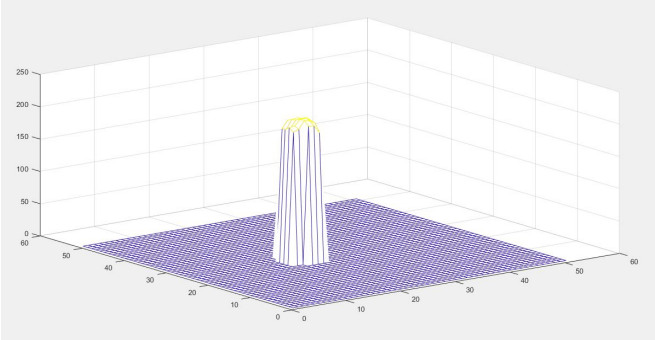
Number of pixels representing the laser: *59.*

- **range: 61**



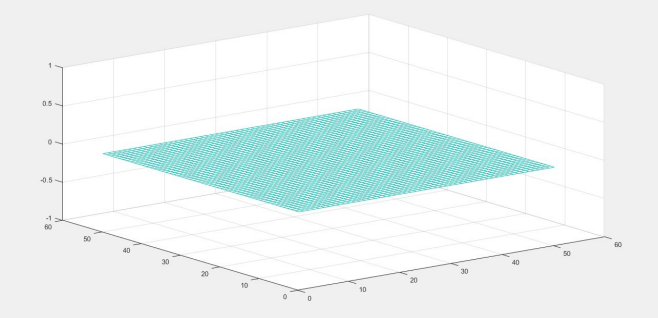
Number of pixels representing the laser: *59.*

- **range: 200**



Number of pixels representing the laser: *30.*

- **range:230**



Number of pixels representing the laser: *0.*

As we expected, the number of points decreases with the change of the range used in position of the laser.
The following table expresses this:

	60(range)	61(range)	200(range)	230(range)
upward (y axis)	64	64	28	0
down (y axis)	59	59	21	0
left (x axis)	71	70	39	2
right (x axis)	59	59	30	0
no changing	173	172	115	65

13. Problems encountered and limits

In this section will present both the hardware and software problems encountered

Problems hardware:

1. Base of the dual camera not perfectly parallel to the ground
2. Offset between the y axis of the two cameras
3. Defective USB connector
4. The laser can be powered only via Raspberry

Problems software:

5. Library used in Python does not allow to eliminate white balance limiting the repeatability of measurements
7. MatLab is not supported by Raspberry
8. Impossibility to detect the laser perfectly through the algorithm due to the possible light pollution of the environment.

We explain in detail each of the problems listed above:

1.Base of the dual camera not perfectly parallel to the ground

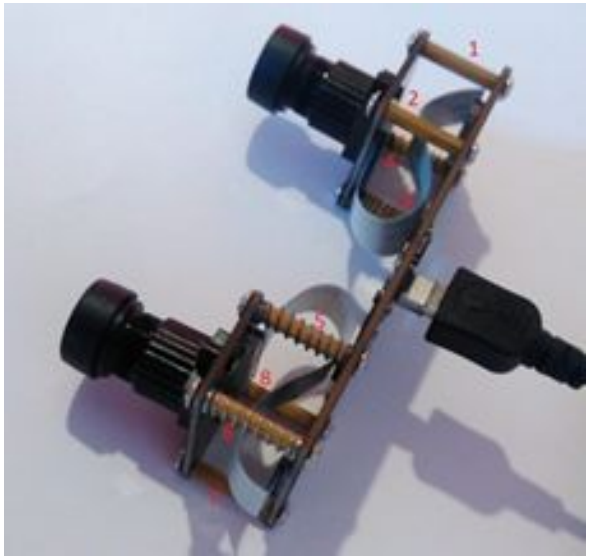
In order to solve this problem we made sure that the dual camera was not resting on the ground, but fixed on the support at a certain distance from the ground.

2.Offset between the y axis of the two cameras

The camera presented an offset problem between the y coordinates of the two cameras. Inserting springs: two in the upper supports of the left one and two more in the lower supports of the right one we can solve this problem and re-align the cameras.
Through a gauge we measured the distance at which we fixed the two cameras to their support to solve the problem of offset.

Here are more detailed measurementes:

1) 1.3 cm
2) 1.3 cm
3) 1.5 cm
4) 1.5 cm
5) 1.4 cm
6) 1.4 cm
7) 1.5 cm
8) 1.3 cm



3.Defective USB connector

We simply applied scotch to the USB connector so that it can not move.

4.The laser can be powered only via Raspberry

This represents a big limit on the code implemented on matlab. There is no function that can turn the laser on and off. In order to switch it on or off it is necessary to access the code developed in Python and control the laser via the GPIO pins of the Raspberry, otherwise you have to control it manually.

5.Library used in Python does not allow to eliminate white balance limiting the repeatability of measurements

The library used in Python to perform distance measurements does not allow us to access the parameters of the dual camera and modify them. In particular, it is not possible to reset the white balance. This meant that the algorithm developed in the MatLab environment was not reportable in Python. We have therefore developed a different algorithm for the two development environments. The technique used to detect the laser in Python uses different principles. While in MatLab we perform a subtraction between matrices representing the frames, in Python we exploit hue, saturation and brightness features in order to detect laser.

6.MatLab is not supported by Raspberry

As we have seen previously, although the code in Python seems to be more precise at short distances, the Matlab code allows us to have a larger work space.
Not being supported by Raspberry, the code developed in MatLab environment can not be exported in Raspbian operating system.
Therefore this problem creates limits on the operating space of the device developed in Python, which is smaller than that in Matlab.

7.Impossibility to detect the laser perfectly through the algorithm due to the possible light pollution of the environment.

This problem creates limits regarding the repeatability of our measurements, especially in Python because in this environment the laser is detected also by a parameter related to brightness. If we change the initial conditions (in particular the environmental conditions: sunlight, artificial light, neon light ...) we will get a bit different results.

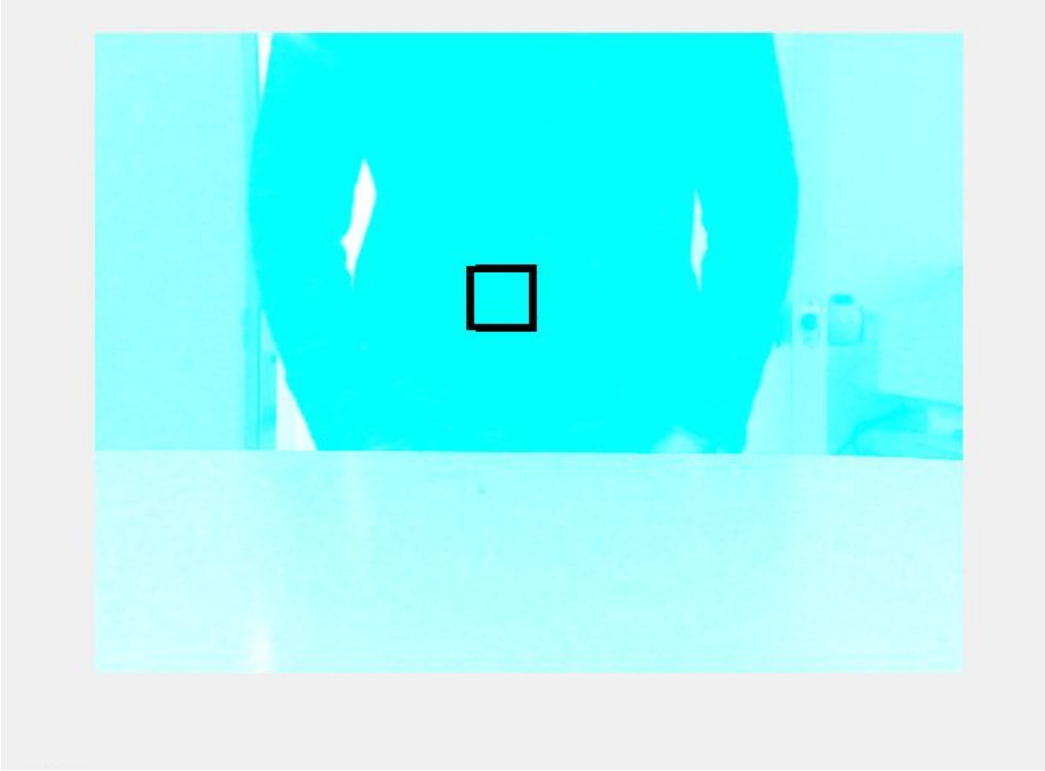
14. Possible future developments

As a future development we have tried to calculate the distance of an object without using the laser.

The principle of use is as follows:

1. The operator must insert the object to be evaluated inside the rectangle visible in the interface
2. Read the runtime distance visible in the workspace

The interface implemented is as follows:



(The real distance was 122)

Let's see now the implemented code:

```
while(true)

    img = snapshot(cam);   % it takes one picture

     left = img(1:480,1:640,1:3);
     right = img(1:480,641:1280,1:3);
     [J1,J2] = rectifyStereoImages( left, right, stereoParams);
     frameLeftGray  = rgb2gray(J1);
     frameRightGray = rgb2gray(J2);
     disparityMap = disparity(frameLeftGray, frameRightGray);
     disparityRange=[0 128];

     numeratore=0;
     n=0;
     for i=155:1:186
        for j= 242:1:270
            if disparityMap(i,j) > 20
                numeratore = disparityMap(i,j) + numeratore;
                n=n+1;
            end
          end
      end

     definitiva1gray=rgb2gray(definitiva1);
    %imshow(stereoAnaglyph(disparityMap,single(definitiva1gray)));
    imshow(stereoAnaglyph(J1,definitiva1));
    media = numeratore/n

    distanza = 5458/media

end
```

Let's see the explanation of the main functions used:

rectifyStereoImages	Undistorts and rectifies left and right, a pair of stereo images. stereoParams is a stereoParameters object containing the parameters of the stereo camera system. J1 and J2 are the rectified images.

rgb2gray	Converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.
disparity	Returns the disparity map for a pair of stereo images, frameLeftGray and frameRightGray. FrameLeftGray and frameRightGray must have the same size and must be rectified such that the corresponding points are located on the same rows. This rectification can be performed using the rectifyStereoImages function. The returned disparity map has the same size as frameLeftGray and frameRightGray.

We do not have an accurate analysis of precision and efficiency of this method. According to us, a future development is to evaluate the difference with the method that has been widely analyzed before, trying to understand if it worsens, trying, in this case, to improve it.

15. Conclusions