

3D Reconstruction Through Photographs

MASTER DISSERTATION

Vítor Miguel Amorim de Almeida

MASTER IN INFORMATICS ENGINEERING



UNIVERSIDADE da MADEIRA

A Nossa Universidade

www.uma.pt

september | 2013

3D Reconstruction Through Photographs

MASTER DISSERTATION

Vítor Miguel Amorim de Almeida
MASTER IN INFORMATICS ENGINEERING

SUPERVISOR
Eduardo Leopoldo Fermé

CO-SUPERVISOR
Luís Armando de Aguiar Oliveira Gomes

ABSTRACT

Humans can perceive three dimension, our world is three dimensional and it is becoming increasingly digital too. We have the need to capture and preserve our existence in digital means perhaps due to our own mortality.

We have also the need to reproduce objects or create small identical objects to prototype, test or study them. Some objects have been lost through time and are only accessible through old photographs.

With robust model generation from photographs we can use one of the biggest human data sets and reproduce real world objects digitally and physically with printers.

What is the current state of development in three dimensional reconstruction through photographs both in the commercial world and in the open source world? And what tools are available for a developer to build his own reconstruction software? To answer these questions several pieces of software were tested, from full commercial software packages to open source small projects, including libraries aimed at computer vision.

To bring to the real world the 3D models a 3D printer was built, tested and analyzed, its problems and weaknesses evaluated. Lastly using a computer vision library a small software with limited capabilities was developed.

Keywords: Computer Vision, OpenCV, Software Engineering, Epipolar geometry, Stereo Vision.

RESUMO

Os humanos compreendem o mundo que nos rodeia em três dimensões, e de facto o mundo físico é tridimensional e está cada vez mais a tornar-se um mundo digital. Além disso existe também a necessidade de capturar e preservar a existência humana através de meios digitais.

Existe igualmente a necessidade de reproduzir objetos, de criar cópias idênticas para prototipagem, para teste ou simplesmente para estudo. Alguns objetos foram perdidos e hoje em dia apenas são acessíveis através de fotografias antigas.

Com a geração robusta de modelos digitais através de fotografias pode-se utilizar um dos maiores bancos de dados que existe para reproduzir objetos digitalmente através de modelos informáticos e fisicamente através de impressoras 3D.

Qual é o estado atual de desenvolvimento da reconstrução tridimensional através de fotografias, tanto no mundo comercial como no mundo do código aberto? Que ferramentas estão disponíveis também para um especialista na área desenvolver o seu próprio *software* de reconstrução? No intuito de responder a estas questões vários *softwares* foram testados, desde comerciais até pequenos projetos “open source” incluindo bibliotecas especialmente desenvolvidas para visão computacional.

Para trazer para o mundo real os modelos 3D uma impressora 3D foi construída, testada e analisada, os seus problemas e fraquezas avaliados. E finalmente utilizando uma das bibliotecas testadas, um *software* com capacidades básicas foi desenvolvido.

Palavras-chave: Engenharia de Software, Geometria Epipolar, OpenCV, Visão Computacional, Visão Estéreo

I dedicate this work to all the people without whom this work would have not be possible, to all the scientist, engineers, professors and individuals that developed a piece of the puzzle that is now this document.

ACKNOWLEDGMENTS

First of all I would like to thank my supervisors Professor Doutor Eduardo Fermé and Professor Doutor Luis Gomes, whose experience and knowledge guided me through the development of this work.

To my parents, Antonieta Amorim and Victor de Almeida for their support in this endeavour through all the ups and downs, when I needed the most and, for helping me review the document.

I thank Jorge Lopes and Ruben Lubbes for their help and understanding of the RepRap 3D printer and for the help understanding how it works and how build it.

TABLE OF CONTENTS

I. Introduction	1
I.1. Motivation.....	2
I.2. Contribution	3
I.3. Organization.....	3
II. Background	5
II.1. Informal projective plane properties.....	6
II.1.1. 1 st Property – Parallel Lines converge to a vanishing point	7
II.1.2. 2 nd Property – Nearer Objects are lower in the image (objects in the same plane).....	7
II.1.3. 3 rd Property – Nearer objects look bigger	8
II.1.4. Consequences	8
II.2. The Pinhole Camera model	9
II.2.1. What it is	9
II.2.2. Conventions.....	10
II.2.3. Equation	10
II.2.4. Simplification.....	10
II.3. Homogeneous Coordinate system	11
II.3.1. Projective Geometry	11
II.3.2. Homogeneous Coordinates	12
II.4. Real World Camera	13
II.4.1. Displaced optical axis	13
II.4.2. Radial distortion.....	15
II.4.3. Tangential Distortion.....	15
II.4.4. Calibration and implementation.....	16
II.5. Epipolar geometry	18
II.5.1. Fundamental Matrix F.....	19
II.5.2. Essential Matrix E	20
II.5.3. Reprojection matrix Q	21
II.6. Getting the focal length from an image file.....	22
II.7. Computer Vision.....	23
II.7.1. Stereo Vision	24
III. Capturing in 3D	27
III.1. Best Recording.....	28

III.2. Effect of image quality.....	30
III.3. Hardware	31
III.4. Stereo Recording	32
IV. Existing Software	37
IV.1. Proprietary Software	38
IV.1.1. Autodesk Catch 123D.....	38
IV.1.2. Agisoft PhotoScan & Agisoft StereoScan.....	40
IV.1.3. Photo to 3D	42
IV.1.4. Vi3Dim	43
IV.2. Open Source Software	44
IV.2.1. Insight 3D.....	45
IV.2.2. Reconstructing Rome	45
V. Computer Vision Libraries	47
V.1. OpenCV	48
V.1.1. Introduction	48
V.1.2. Platform	48
V.1.3. Installation.....	49
V.1.4. Results.....	53
V.2. EmguCV	54
V.2.1. Introduction	54
V.2.2. Platform	55
V.2.3. Installation.....	55
V.3. BoofCV	57
V.3.1. Introduction	57
V.3.2. Platform	57
V.3.3. Installation.....	58
V.3.4. Results.....	59
V.4. SimpleCV	59
V.4.1. Introduction	59
V.4.2. Platform	59
V.4.3. Installation.....	60
V.4.4. Results.....	60
V.5. Caltech's camera calibration for Matlab	60
V.5.1. Introduction	60
V.6. Overall Library Results	61
VI. 3D printer	63
VI.1. Building the "Prussa Mendel"	64
VI.1.1. "Prussa Mendel" Frame.....	64
VI.1.2. "Prussa Mendel" Extruder head.....	65
VI.1.3. "Prussa Mendel" Electronics.....	66
VI.2. Stress points building the machine.....	67
VI.3. Results.....	68
VII. Implementation	69
VII.1. Process	70

VII.1.1. Image acquisition for calibration	70
VII.1.2. Image calibration.....	71
VII.1.3. Image Acquisition for 3D generation	73
VII.1.4. 3D generation.....	73
VIII. Conclusion And Future Work	77
Index	79
References	81

LIST OF FIGURES

Figure II.1 - An illustration of how the scenery of a straight road in the desert would be stored in an image taken by a photographic camera.....	6
Figure II.2 - A crop of Figure II.1 where the road and the horizon meet, showing how parallel lines converge into a point.	7
Figure II.3 - A crop of Figure II.1 with a portion of a plane α represented with a red trapezoid.....	8
Figure II.4 - The same crop and plane as in Figure II.3, but with the height of the poles highlighted-.....	8
Figure II.5 - Pinhole camera model illustration with the Euclidian coordinate system represented and the focal length f [3].	9
Figure II.6 - A similar model to the pinhole camera model where the image plane is in front of the pinhole (the user's eyes).	10
Figure II.7 - A ray of light straight through the optical axis represented by a yellow line collides with the image plane to the side of the image planes center (principal point).	14
Figure II.8 - Image x, y and camera $xcam, ycam$ coordinate systems [5].	14
Figure II.9 - Radial distortion: rays farther from the center of a simple lens are bent too much compared to rays that pass closer to the center [6].	15
Figure II.10 - Tangential distortion results when the lens is not fully parallel to the image plane [6].	16
Figure II.11 - Chessboard pattern used in calibration. This is a symmetric pattern.	16
Figure II.12 - Asymmetric circles grid pattern. This is an asymmetric pattern.	17
Figure II.13 - (a) C and C' are the two cameras centers and x and x' are the representation of the point X in C and C' cameras image planes. The camera centers, the point X and its images lie in the same epipolar plane π (b) The point X projects to a point x in the C camera. The x' point must lie on the epipolar line l' . Based on [5].	19
Figure II.14 - Essential matrix E geometry. The translation vector T and the rotation vector R are shown in relation to the left camera. The location of the right camera is depicted in relation to the left camera [6].	21
Figure II.15 - A point P is recorded in the left camera as p and in the right camera as p' . fl and fr are the left and right focal lengths of the cameras respectively. xl and xr are the distances to the left side of the projected plane in the left and right cameras respectively. B is the baseline of the cameras, this is, the distance between them.	25
Figure II.16 - Relation between object distance to the camera and the disparity [6].	26
Figure III.1 - Issues when creating a 3D model.	29
Figure III.2 - Keychain 808 #16 camera and a euro coin besides it for scale.	32
Figure III.3 - Model for mounting the stereo rig made with Autodesk Inventor.	32
Figure III.4 - 3D printed version of the model shown in Figure III.3.	33

Figure III.5 - Remote shutter and focus. Component view on the right. Schematic on the left.	34
Figure III.6 - Breadboard schematic for the simultaneous control of a pair of reflex cameras for a stereo rig using an Arduino duemilanove, two 2N2222 transistors, two 2,5mm audio jack, two 10k Ω resistors, and a piezo buzzer.	35
Figure IV.1 - A reconstructed helmet using Autodesk Catch 123D PC application and the camera positions of the used images.	39
Figure IV.2 - A reconstructed helmet using Autodesk Catch 123D PC application.	40
Figure IV.3 - a) Left side of the object, the reconstruction of the model was very good. b) The right side of the object is completely deformed and no recognizable. c) The reconstructed object from behind showing a part missing.	41
Figure IV.4 - Four different reconstructions using four different image pairs via photo-to-3d, with 0.9MPx photos.	43
Figure IV.5 - Reconstruction of one of the samples of Vi3D software.	44
Figure V.1 - CMake header for binary generation under Windows 8.	49
Figure V.2 - CMake searches the computer for the available compilers. For MVS 2012 the option to be chosen is <i>Visual Studio 11</i>	50
Figure V.3 - Add support for OpenGL in OpenCV before generating the binaries in CMake.	50
Figure V.4 - MVS 2012 Solution generated by CMake ,opened in MVS 2012.	51
Figure V.5 - A new OpenCV project under MVS 2012.	52
Figure V.6 - The additional Include Directories in MVS 2012 needed to compile a program with OpenCV.	52
Figure V.7 - The additional Library Directories in MVS 2012 needed to compile a program with OpenCV.	53
Figure V.8 - The additional Dependencies in MVS 2012 needed to compile a program with OpenCV	53
Figure V.9 - The colored picture on top is unrectified. The bottom black and white is the rectified picture using OpenCV C library in MVS 2012.	54
Figure V.10 - Creation of a new EmguCV C# project.	56
Figure V.11 - Referencing EmguCV library's in MSV 2012.	56
Figure V.12 - Adding OpenCV dll's to the C# project.	57
Figure V.13 - Creation of a new java project in NetBeans.	58
Figure V.14 - Adding the BoofCV library binaries to the NetBeans project.	58
Figure V.15 -Addition of java docs to NetBeans to shows the library's documentation and autocomplete.	59
Figure V.16 - a) Pictures analyzed using Caltech's Camera Calibration Toolbox for Matlab with the pattern corners highlighted in color. b) The position change of the calibration pattern is shown in relation to a static camera [21].	61
Figure VI.1 - "Prussa Mendel" 3D printer frame under construction, with its axis shown.	65
Figure VI.2 - Mounted extruder head and a magnified image of the gear. The gears do not match perfectly and this can lead to a fault with the plastic flow.	65
Figure VI.3 - On the left printed square with a hole of the size of a 1€ coin with the coin inserted. On the right the printed part and a 1coin. This test piece was used to check if the printer printed with correct height and correct hole diameter.	68
Figure VI.4 - Printing the stereo rig shown in Figure III.3.	68
Figure VII.1 Code of the file header exporting code in C++	75

LIST OF TABLES

Table 1 - The four different geometries, the transformations allowed in each and the measures that remain invariant under those transformations [4].	12
Table 2 - Specifications of the cameras used in the high quality rig.	34

LIST OF FLOWCHARTS

Flowchart II-1 – Calibration algorithm.	17
Flowchart II-2 – Focal length calculator algorithm, using a C++ jhead wrapper.....	23
Flowchart III-1 – Arduino program flowchart aimed at taking the calibration images for the high quality reflex images.	35

ACRONYMS

3D – Three dimensional

ABS – Acrylonitrile butadiene styrene

CCD – Charge-Coupled Device

CRL – Common Runtime Language

CMOS – Complementary Metal-Oxide-Semiconductor

CPU – Central Processing Unit

FFF – Fused Filament Fabrication

GPS – Global Positioning System

GUI – Guided User Interface

IDE – Integrated Development Environment

MPx – Mega Pixel

MVS – Microsoft Visual Studio

OO – Object Oriented

PCL – Point Cloud Library

PLA – PolyLactic Acid

RAMPS – RepRap Arduino Mega Pololu Shield

RepRap – Replicating Rapid Prototyper

Acronyms

USB – Universal Serial Bus

XML – EXtensible Markup Language

I. INTRODUCTION

*"If I have seen further it is by standing on ye sholders of
giants"*

Sir Isaac Newton, February 15th, 1676

To study a complex 3 dimensional (3D) object, that is, any real-world object, we have to be near to the object, we have to be in the same physical space as the object and the number of people that can analyze it is limited. Further with this there is always a possibility of degradation of the object due to exposure to the surrounding elements.

But a fully detailed and 3D model of the same object with both the physical and textural characteristics can be visualized and studied by an unlimited number of individuals and, since it's a digital object, it can be manipulated and replicated without degrading it. This is just one of the contexts where technological applications of 3D reconstruction can help preserve our culture and world, with not invasive tests and a low cost in equipment.

The amount of detail in the reconstruction can change a lot whether we wish to capture a small object with all the little details that it has or whether we wish to capture an entire city to stay in the archives and to create guided digital tours. To build an extremely detailed reproduction of a city can be quite difficult if not impossible at this stage and this cannot be done in a laboratory, we cannot take an entire city to a laboratory; on the other hand small objects can be taken to a laboratory, or the laboratory mounted around it. This shows that we can apply different technologies to different types of reconstruction according to its size and its detail.

A small object or even a room can be scanned with lasers, resulting in a very detailed model. This has been done but the larger the target the more difficult it can be to use these kind of techniques.

Photography is widely disseminated around the world and it is an hobby of many people. There is already an extensive amount of data from all the major tourist sites and monuments. If we could take this data and reconstruct the world we would have found the perfect reconstruction method because all the necessary data sets have already been collected and they are available online.

I.1. MOTIVATION

We live in a 3D world and for many centuries we have designed and idealized what we wanted to do in a 2 dimensional way, by the means of papyrus, paper or any other means of writing. This simple but very useful technology is still used today, and presumably will be continued to be used. Due to the advancements of technology, the difficulty of schematics, and even simply by the dawn of the digital era the 3D world has gained a substantial weight in the developer's life.

The ability to test, reproduce and make small scale model of what will be, can reduce future problems and create better results.

We want to extend our digital world, make it more interactive and real with less effort and we want to make our digital creations real 3D objects.

Another motivation behind this work is to assist and be a part of one of the cutting edges of computer technology, the capture of real world entities and the creation of accurate computer models. With this technology historic monuments can be preserved forever without any degradation, accurate terrain reconstruction can be used to simulate and test disaster situation and prevent losses among many other applications.

One side of the project was to digitalize the world, but we can already introduce digital pieces into the real world, bringing without the cost of transport and the wait, essential customized pieces, created with the expertise of people over the world by downloading an internet model file.

I.2. CONTRIBUTION

In this work the current state of mathematical models for 3D reconstruction are shown and studied, starting with how images are captured from the real world onto a static two dimensional image, and what this transformation can instill into the images – the deformations and how to mathematically treat this deformations.

Several methods to transform two dimensional images into 3D have been studied and these are summarized and explained along with how to obtain the best results in model generation. The whole process, from 2D images to a 3D printed model, is addressed in this work including the available software in this area.

This work intends on giving all the necessary mathematic basics of 3D reconstruction to an individual who has not studied this topic, to show the current state of software development in this area and to provide the best start possible on this area by knowing what libraries have been developed. Additionally we hope to help in the construction of plastic models, by means of a 3D printer.

I.3. ORGANIZATION

This thesis follows a simple approach in its organization. Chapter II. starts by describing all the mathematical fundamentals necessary to understand the passage of information from the real world to an image including the various techniques used today to capture and generate models based on the real world.

On the next Chapter III we discuss how to make the best of the captures in order to create better quality models.

Existing software to check what has been produced is analyzed in Chapter IV I. and various computer vision libraries are detailed in Chapter V.

Chapter VI, and Chapter VII, describe the implementation of the models of this thesis and the build up of the 3D printer, respectively.

Finally Chapter VIII, concludes this work and suggests the future work.

II. BACKGROUND

There are a few mathematical descriptions that are essential to understand computer vision and, although it is not in the scope of this project to explain in detail how the real world is transformed into the projective world, this chapter will provide all the necessary pieces to understand how many of the operations are made and from where the mathematical models arise.

First will be shown how light is captured and how this information is stored. This is a crucial part, because depending on how the light is transformed different mathematical models may arise. Our brains are calibrated on how our eyes capture light making all the internal necessary computations. By using a different focal length than what our brain is used to the difference in image perception is obvious.

A perfect mathematical model has no application in the real world and without a way of closing the gap between these two, computer vision has no real application. A way of handling real world image capturing devices is shown and also how to transform the images in order to resemble images taken with a perfect camera.

In computer vision the Euclidian space is not used due to the lack of the necessary space transformations. Another type of spatial representation is used, and it's called the projective plane, thus introducing the homogenous coordinate system. Images taken by a photographic camera are in fact in this projective plane.

II.1. INFORMAL PROJECTIVE PLANE PROPERTIES

Before formalizing mathematically how the images are captured and what type of transformations and deformations they are submitted to, there are a few properties that can be easily deducted and demonstrated. This will provide clues on how images can be treated and what properties are preserved when taking a picture of the real world.

For this analysis any generic camera can be used to take the pictures but instead we use a simple landscape drawing (Figure II.1) of what a camera would capture in order to simplify the comprehension of some properties. This image is based on the work of Professor Jitendra Malik [1]. Different cameras will generate slightly different images, due to the different hardware: different lenses will capture and direct the light differently, different sensors will interpret the light in a different way. These are just some examples, forward in this document the most common issues will be addressed.

Figure II.1 shows how a scenery is captured in a photographic camera. Just by looking at it is perceivable some properties and differences from what we humans see.

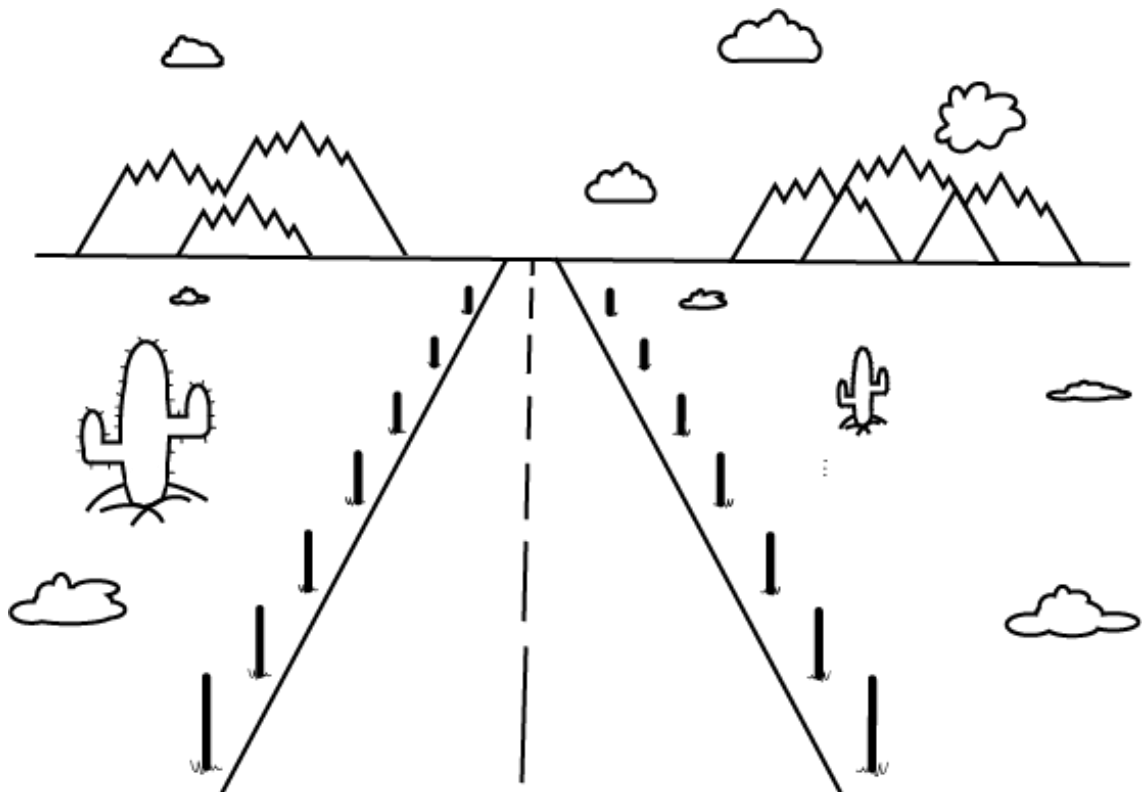


Figure II.1 - An illustration of how the scenery of a straight road in the desert would be stored in an image taken by a photographic camera.

II.1.1. 1st Property – Parallel Lines converge to a vanishing point

One property that is not preserved when taking pictures is the parallelism between lines, i.e. parallel lines in the real world are not parallel in the projected space. In fact they converge onto a vanishing point.

This is actually one of the definitions used to explain in layman's terms what the projective plane is: the projective plane can be thought of as an ordinary plane equipped with additional "points at infinity" where parallel lines intersect.

This implies that any two infinite lines, parallel or not in the projective plane will intersect once.

In the example shown, Figure II.2 the two road sides are in fact parallel but in the projective space (in the image) they are converging beyond the horizon.

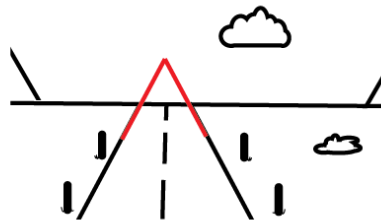


Figure II.2 - A crop of Figure II.1 where the road and the horizon meet, showing how parallel lines converge into a point.

II.1.2. 2nd Property – Nearer Objects are lower in the image (objects in the same plane)

In the projective plane for objects that are located in the same plane, or a part of the object, if they are lower in the image it means that they are also nearer to the observer.

Figure II.3 denotes some objects that stand in the same plane α , i.e. all the bases of the poles are located in the same plane. The poles that are lower in the image are in fact nearer to the observer.

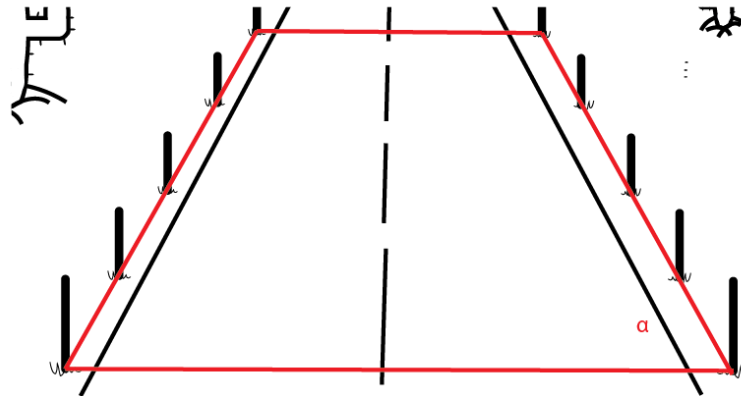


Figure II.3 - A crop of Figure II.1 with a portion of a plane α represented with a red trapezoid.

II.1.3. 3rd Property – Nearer objects look bigger

Nearer objects in the projective plane look bigger the nearer they are.

In II.1.2 it was established that objects lower in the image if they are located in the same plane, they are nearer. With that established pole 1 is nearer than pole 2 but pole 1 appears bigger than pole 2. In fact the poles in the road have all the same size but in Figure II.4 some appear bigger than others.

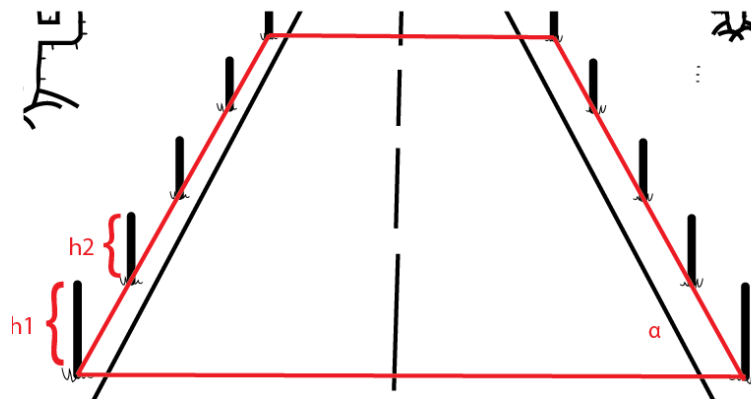


Figure II.4 – The same crop and plane as in Figure II.3, but with the height of the poles highlighted-.

II.1.4. Consequences

These somewhat informal properties show that many of the Euclidian plane properties – real world properties – are not preserved when working with the projective plane. A summary of these properties follows: [2]

- Points project to points;

- Lines project to lines;
- Planes project to the whole image or a half image;
- Angles are not preserved;
- Special cases:
 - Line through focal point projects to a point;
 - Plane through focal point projects to line;
 - Plane perpendicular to image plane projects to part of the image (with horizon);

II.2. THE PINHOLE CAMERA MODEL

II.2.1. What it is

The pinhole camera model is the simplest of camera models. There are no lenses and there is no distortion, such that in this model light enters through a minimal hole and a singular ray enters from any particular point. The image is projected in a plane called the *projective plane*.

The size of the projected image is given by the *focal length* (which is a camera parameter). The focal length is defined as the distance between the pinhole and the projective plane. The bigger the focal length the closer the object is in the projective plane.

The projected image is inverted in both x and y axis, consequently complicating a little bit the equations generated from this model. This is shown in Figure II.5.

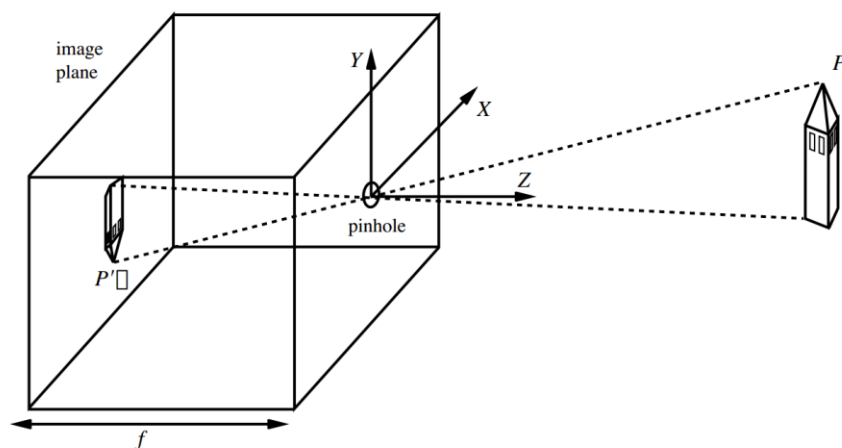


Figure II.5 – Pinhole camera model illustration with the Euclidian coordinate system represented and the focal length f [3].

II.2.2. Conventions

As a convention upper case letter represent points in the real world and lower case letters represent points in the projective plane.

II.2.3. Equation

Using the similar triangles theorem it is possible to get the following equations:

$$x = -\frac{fX}{Z}, \quad y = -\frac{fY}{Z}$$

Thus, a point $(X,Y,Z)^T$ in real world is mapped in the projective plane as $(x=-fX/Z, y=-fY/Z)$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} x = -\frac{fX}{Z} \\ y = -\frac{fY}{Z} \end{bmatrix} \quad \text{II.1}$$

II.2.4. Simplification

This model can be simplified through two ways.

- 1st by inverting the image in both axis through modern computational methods (something which is already made nowadays by our cameras).
- 2nd by using a similar model to the one represented in Figure II.6

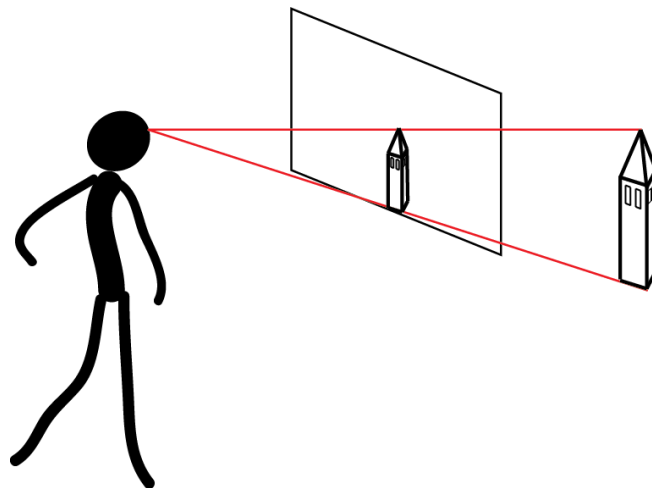


Figure II.6 – A similar model to the pinhole camera model where the image plane is in front of the pinhole (the user's eyes).

In both cases the deducted equation is:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} x = \frac{fX}{Z} \\ y = \frac{fY}{Z} \end{bmatrix} \quad \text{II.2}$$

II.3. HOMOGENEOUS COORDINATE SYSTEM

II.3.1. Projective Geometry

Homogeneous coordinates or projective coordinates are a system of coordinates used in projective geometry. Homogeneous coordinates are to the Projective geometry like Cartesian coordinates are to the Euclidian geometry.

In Euclidian geometry lengths, angles, intersecting and parallel lines do not change when Euclidian transformations are applied (translation and rotation). Considering the image of a photographic camera, it is easily identifiable that none of these properties are preserved. Hence Euclidian geometry is not adequate for these study just like it was described in section II.1.

Euclidian geometry is actually a subset of what is known as projective geometry. Projective geometry models the image processing of a camera very well because it allows a much larger class of transformations (besides translations and rotations it includes perspective projections). Projective transformations preserve type (points remain points and lines remain lines).

Just like Euclidian geometry projective geometry exists in any number of dimensions.

	Euclidean	similarity	affine	projective
Transformations				
rotation	X	X	X	X
translation	X	X	X	X
uniform scaling		X	X	X
nonuniform scaling			X	X
shear			X	X
perspective projection				X
composition of projections				X
Invariants				
length	X			
angle	X	X		
ratio of lengths	X	X		
parallelism	X	X	X	
incidence	X	X	X	X
cross ratio	X	X	X	X

Table 1 - The four different geometries, the transformations allowed in each and the measures that remain invariant under those transformations [4].

II.3.2. Homogeneous Coordinates

A point in an n -dimensional Euclidean space is represented as a point in an $(n+1)$ -dimensional projective space in most cases. So a point $(x, y)^T$ in a Euclidian Plane is represented by three coordinates in projective geometry $(x, y, w)^T$. This last value w is the scaling of the point and is unimportant, meaning that $(x, y, w)^T$ is the same point as $(\alpha \cdot x, \alpha \cdot y, \alpha \cdot w)^T$.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \alpha \cdot x \\ \alpha \cdot y \\ \alpha \cdot z \end{pmatrix}, \alpha \neq 0 \quad \text{II.3}$$

$w = 0$ is a special case that represents that a certain point is in the infinite space.

One of the biggest Advantages of using homogenous coordinates is that the coordinates of points, including points at infinity (there is a point at infinity for each direction), can be represented using finite coordinates and that formulas involving homogeneous coordinates are often simpler and more symmetric than their Cartesian counterparts. Another advantage is that it is a non-metrical geometry, independent from any metric structure.

II.4. REAL WORLD CAMERA

There are commercial versions of pinhole cameras and any camera with removable lens can be modified to be a pinhole camera. Commercial versions are scarce, not easily found, and were used essentially in the beginning of the photographic era.

A real world camera has some kind of lens, where the light is collected, and a surface where the image is recorded. Nowadays cameras have charge-coupled device (CCD) sensors or complementary metal-oxide-semiconductor (CMOS) sensors which are a cheaper version of the CCD. This is the type of light capturing device that will be discussed in this chapter.

Due to flaws in the construction of the lenses, the location of the sensor inside the camera or even the type of lenses the image can be slightly distorted. Additionally the pixels in the CCD can be non-square (unequal scale factor in the y and x directions) [5]. This of course does not match the mathematical models and before the projections can be analyzed and computerized all the flaws have to be minimized.

Here three major camera flaws will be discussed and will serve as a baseline to check the quality of the computer vision libraries. These flaws are a displaced optical axis, radial and tangential distortions.

With the homogeneous coordinates previously explained it is now possible to write the camera equation shown in II.3 in a more computational friendly way:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \text{II.4}$$

Equation II.4 can be written in a more compact form:

$$x = PX, P = \text{diag}(f \quad f \quad 1) [I \mid 0] \quad \text{II.5}$$

II.4.1. Displaced optical axis

The previously shown model – the pinhole camera model – assumes that all the camera parts are neatly and precisely located. This is rarely true, if not due to human error in

construction, due to automated construction procedures or to budget restrictions. A perfectly aligned camera is much more expensive to make than a slightly out-of-place one, and in the major range of applications there is no difference for the user.

Figure II.7 shows the how a displaced optical axis can affect the images and what differences in the image it can make. If they are not aligned all the light is misplaced slightly, this added with the other types of distortion that will be shown in the next two sub sections can generate a very different image than what is expected.

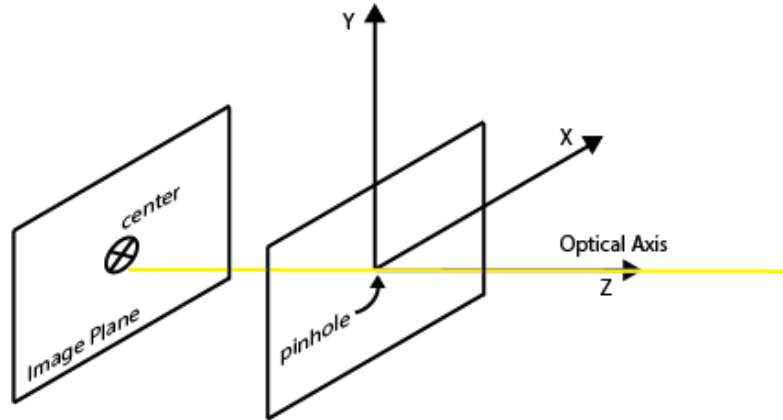


Figure II.7 – A ray of light straight through the optical axis represented by a yellow line collides with the image plane to the side of the image planes center (principal point).

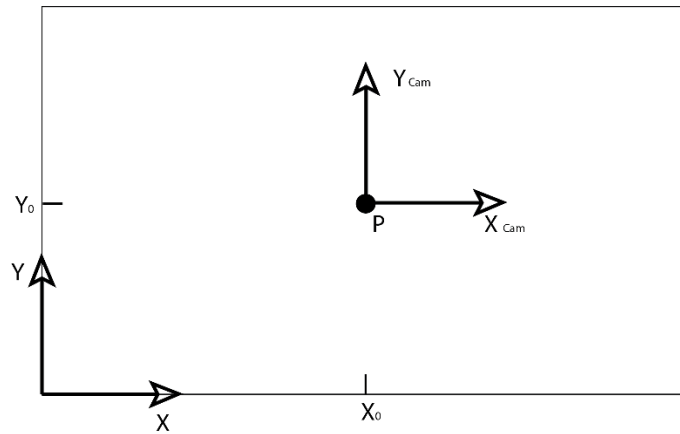


Figure II.8 – Image (x, y) and camera (x_{cam}, y_{cam}) coordinate systems [5].

This displacement of the principal ray generates a principal point offset. P_x and P_y represent the dislocation of the principal point in the image plane. This simple difference that does not in any way have an impact to common photography, but has an impact in the mathematical model of the camera. With these small changes the new camera equation is given in II.6.

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fX + P_x \\ fY + P_y \\ Z \end{bmatrix} = \begin{bmatrix} f & P_x & 0 \\ f & P_y & 0 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad \text{II.6}$$

II.4.2. Radial distortion

Here the distortion arises from how light is captured. The lens design can noticeably distort the light thus distorting the pixel location in the picture. Photographers take advantage of this type of distortion to make art, but in computer vision, the camera models do not take this into account. Some examples of these effects are the “barrel” and “fish eye”.

Some of these errors in projection can even come from hardware construction flaws, which are typical of inexpensive lens is. The effect is stronger as you get farther from the center. Barrel distortion is particularly noticeable in cheap web cameras but less apparent in high-end cameras, where a lot of effort is put into fancy lens systems that minimize radial distortion [6].

Figure II.9 shows how radial distortion affects the captured image.

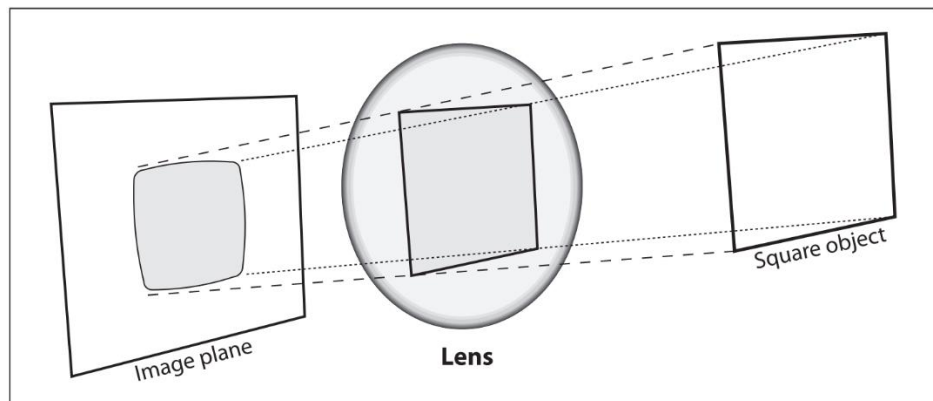


Figure II.9 - Radial distortion: rays farther from the center of a simple lens are bent too much compared to rays that pass closer to the center [6].

II.4.3. Tangential Distortion

Tangential distortion is due to miss positioning of the sensor that captures the light. It might not be aligned perfectly with the cameras lens or it can happen when the image sensors is glued to the back of the camera. Figure II.10 shows this.

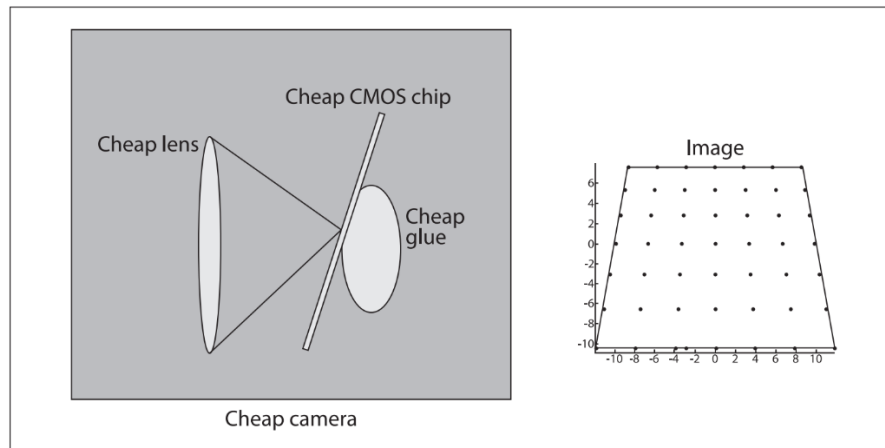


Figure II.10 – Tangential distortion results when the lens is not fully parallel to the image plane [6].

II.4.4. Calibration and implementation

There are a lot of cameras manufacturers, a lot of camera models, and a lot of different lens. Therefore each set of equipment has different calibration matrixes. And we have seen that many of the image flaws arise not only because of the design of the equipment but also because of the faults in the construction process. For this reason all camera sets have to be calibrated separately, and each calibration generated is unique to a camera.

These intrinsic camera parameters can be determined by software resorting to some kind of pattern, usually a chessboard, used to determine how the true projection should look like by extracting the chessboard squares corners and computing their positions. In this work the pattern used was the chessboard and the main reason behind this choice was the symmetry in this pattern, which makes it easy to detect and there is no additional computation needed when compared to an asymmetric pattern (an asymmetric pattern needs to have its asymmetry described in code). The chessboard pattern and an asymmetric pattern are presented in Figure II.11 and Figure II.12.

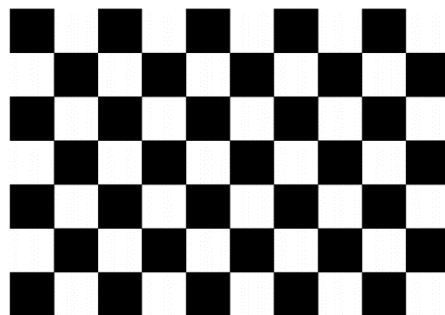


Figure II.11 – Chessboard pattern used in calibration. This is a symmetric pattern.

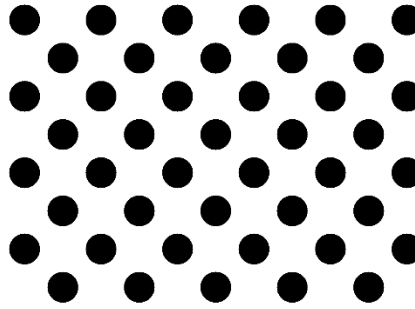
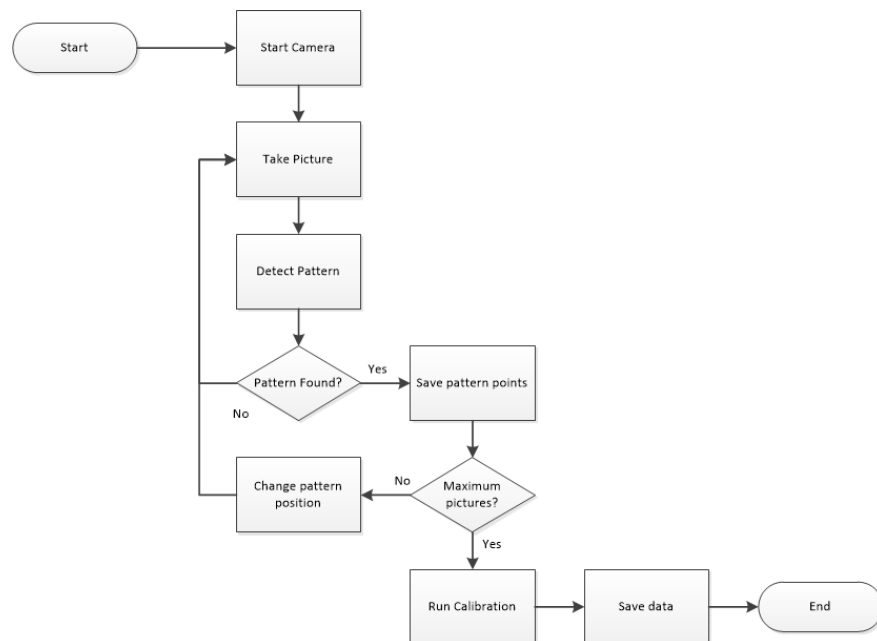


Figure II.12 – Asymmetric circles grid pattern. This is an asymmetric pattern.

All the necessary mathematical models and algorithms are already programmed and implemented in computer vision libraries, because this is the basic first step towards working with computer vision.

The image calibration algorithm is described in a very simple way in Flowchart II-1. This is the used implementation and it is library dependency free. This process is at the base of any good computer vision algorithm and this was the chosen implementation algorithm in chapter V.



Flowchart II-1 – Calibration algorithm.

When the program starts the first thing to do is to initialize the cameras, to make sure that they are ready to start recording and to initialize all the necessary variables. For a successful calibration at least 15 image should be used in order to produce a good calibration. A simple counter can be used to achieve this goal.

After this, and only after a successful camera start, an image should be analyzed. The image can be taken by extracting a frame from the cameras generated video or by remotely activating the shutter. This depends on the used type of camera.

For each frame/image extracted the calibration pattern is to be detected. In case of successful pattern detection the counter is incremented and the calibration points stored. Otherwise the image is discarded not incrementing the counter neither storing the pattern points.

When the number of calibration images is achieved the chosen calibration pattern will run and the data saved.

II.5. EPIPOLAR GEOMETRY

The basic geometry of a stereo rig is the epipolar geometry. This is basically two pinhole models (one for each camera) and an extra point called epipole or epipolar point. Epipolar geometry just like the pinhole model does not take in account the image distortion due to the lenses and as such the images are considered to already have been undistorted.

The epipolar geometry is the intrinsic projective geometry between two views [5]. In order to understand how stereo vision works it is essential to get the bases of epipolar geometry.

Due to of the importance of this topic it was decided to include some information on this work. The major equations and deduction will be shown and depicted. Further information and proof can be found at [5] and [6].

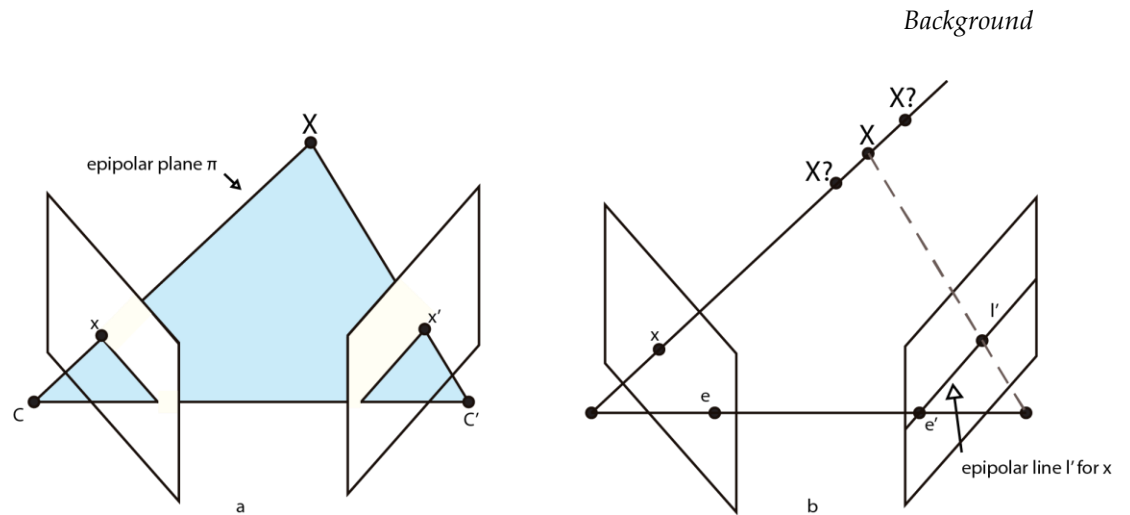


Figure II.13 – (a) C and C' are the two cameras centers and x and x' are the representation of the point X in C and C' cameras image planes. The camera centers, the point X and its images lie in the same epipolar plane π (b) The point X projects to a point x in the C camera. The x' point must lie on the epipolar line l' . Based on [5].

Gary Bradski and Adrian Kaehler in learning OpenCV [6] have summarized some facts about stereo camera epipolar geometry.

- Every 3D point in view of the cameras is contained in an epipolar plane that inter-sects each image in an epipolar line.
- Given a feature in one image, its matching view in the other image must lie along the corresponding epipolar line. This is known as the epipolar constraint.
- The epipolar constraint means that the possible two-dimensional search for matching features across two imagers becomes a one-dimensional search along the epipolar lines once we know the epipolar geometry of the stereo rig. This is not only a vast computational savings, it also allows us to reject a lot of points that could otherwise lead to spurious correspondences.
- Order is preserved. If points A and B are visible in both images and occur horizontally in that order in one imager, then they occur horizontally in that order in the other imager.

II.5.1. Fundamental Matrix F

Epipolar geometry can be represented geometrically and for such the Fundamental matrix F is used.

Assuming that a pair of images were taken using a stereo rig and that a point X is visible in both of the captured images: x for the left image and x' for the right image. Point x' must lie on the epipolar line l' , just like is shown in Figure II.13. The epipolar line is the projection in the second image of the ray from the point X through the camera center C of the first camera. Because of this the point can be mapped in the second camera C' .

$$X \mapsto l'$$

By mathematical deduction it can be proven that there is a singular correlation and that it is represented by the fundamental matrix F . This matrix is a calculated matrix that satisfies Equation II.7. The proof is beyond this work and can be found in [6] and [7].

$$x'^T F x = 0 \quad \text{II.7}$$

II.5.2. Essential Matrix E

There is a particular case, a special case of the fundamental matrix F when normalized image coordinates are used. This specialization is called the essential matrix E . This matrix encapsulates the translation, $[t]$, and rotation, R , between the two cameras and in fact it is defined by their external product. The proof of this property can be found at [5].

$$E = [t]_x R \quad \text{II.8}$$

The F matrix contains additional information about the intrinsic parameters of both cameras, and because of these additional parameters it can relate the two cameras in pixel coordinates i.e. the essential matrix E is only geometrical, knowing nothing about the camera properties relating the location in physical coordinates, while the fundamental matrix F relates the points on the image plane of one camera in image coordinates (pixels) to the points on the image plane of the other camera in image coordinates [6].

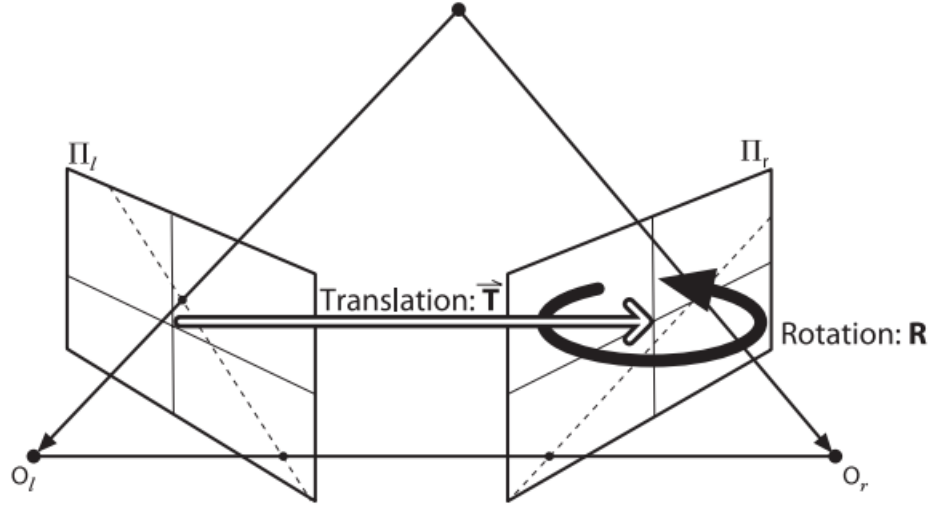


Figure II.14 – Essential matrix E geometry. The translation vector T and the rotation vector R are shown in relation to the left camera. The location of the right camera is depicted in relation to the left camera [6].

II.5.3. Reprojection matrix Q

There is also another specialization of the fundamental matrix that simplifies the reprojection [6]. This specialization allows us to obtain the 3D coordinates directly from the image and from a calculated value, obtained between the two images on the stereo system and is called Q matrix. This matrix is defined in equation II.9.

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{-1}{T_x} & \frac{(c_x - c'_x)}{T_x} \end{bmatrix} \quad \text{II.9}$$

All the parameters are from the left image except c'_x which is the principal point x coordinate for the right image. With a two dimensional homogeneous point and its calculated disparity, d , we can project the point into three dimensions [6]:

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \quad \text{II.10}$$

The 3D coordinates are $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$ since this is determined using homogeneous coordinates.

II.6. GETTING THE FOCAL LENGTH FROM AN IMAGE FILE

Pictures taken with an average digital camera leave an imprint in the photograph i.e., the camera writes in the header of the image file some additional information about that particular photograph. This information is inserted using an Exif tag. With these parameters it may be possible to determine the focal length in a unit that can be used in a computer vision algorithm. That unit is in pixels.

To determine the focal length in pixels four parameters must be available: width and height, focal length (mm) and camera model. Width and height can be extracted from other sources than the image header, but focal length and camera model cannot.

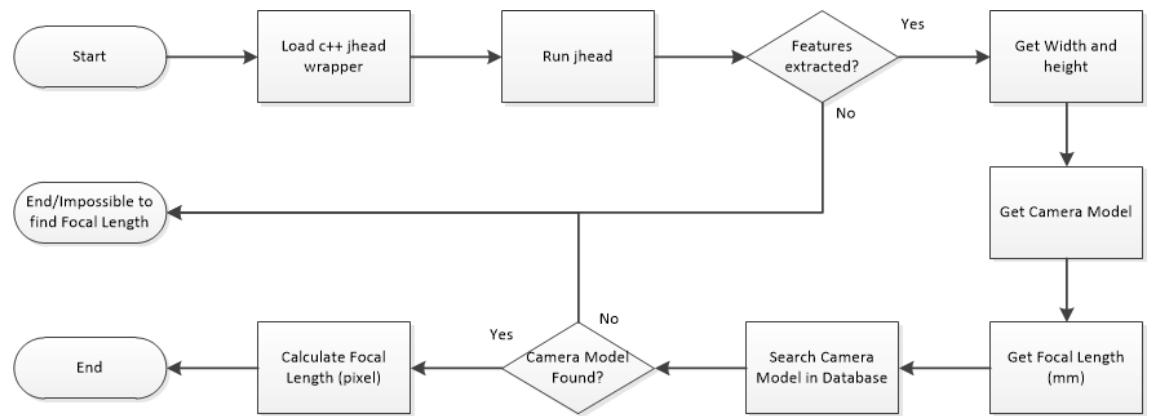
A simple tool to extract this information is provided at [8] by Matthias Wande. This is a command line tool that extracts all the available information from the picture header to be available elsewhere. As it is a command line executable, with a text output to the command line, and with no access to its source code, for this project a C++ wrapper had to be build and its output redirected into a class.

In order to calculate the focal length in pixels another piece of information is needed: the CCD width in mm. Fortunately there is extensive information available from the camera manufacturers and even some websites have all the information stored in a data base. Noah Snavely in his work [9] offers access to his source code. In his files there is an extensive database of camera CCD widths in mm, and although it's a bit outdated it has been used and extended.

The mathematical expression necessary to make the calculation is shown in equation II.12. This formula can be obtained using the similar triangles theorem defined in book [10] chapter IV.

$$Focal\ length\ (pixels) = \frac{Image\ width\ (pixel) \times FocalLength\ (mm)}{CCD\ width\ (mm)} \quad II.11$$

An algorithm to estimate the focal length, taking advantage of the previously described software and mathematical equations, is described in the Flowchart II-2. This algorithm was implemented in C++ under Microsoft Visual Studio (MVS).



Flowchart II-2 – Focal length calculator algorithm, using a C++ jhead wrapper.

Flowchart II-2 can be described in a more detailed way. Loading the wrapper and running it is simply running inside a program using a system function that evokes the command line. This program can be called and executed inside a C++ program. The arguments inserted in this program are created in order to match the image file name from which we want to extract the features. If no features are extracted the program will abort and show an error.

The jhead program returns a string, to get the information is just a question of parsing the string correctly. If the camera model is in the database it's possible to calculate the focal length using equation II.11. If not, the program will abort and show an error.

II.7. COMPUTER VISION

Several techniques have been developed in order to help reconstruct an object i.e., in order to create a very similar computer model.

Some approaches use a binocular stereo rig very much like how our, and brain works to perceive depth, while some other approaches only use a single camera. Both will be covered in this chapter, describing the necessary steps to produce a 3D digital object.

As was described in II.4, all the pictures taken from a digital camera have some kind of flaw, and to produce the best results these flaws should be minimized. Oliver Faugeras in [11] proves that to reconstruct a scene with a binocular stereo rig the intrinsic and extrinsic parameters of a camera do not have to be known prior to the experiment, and that by point matching these parameters can be derived. So both calibrated and uncalibrated cameras can be used and both a singular camera with pictures taken from different positions in relation to a scene/object or a binocular stereo rig. The difference

in these two methods is of course the computational effort needed to solve the problem. Without knowing the intrinsic and extrinsic parameters the time and number of processors is greatly increased.

With calibration it is possible to extract metric calibration data, i.e. after the reconstruction is done it is possible to relate the size of the digital object to the size in the real world. This is made possible because we can attribute a size to the calibration pattern. In the case of the chessboard pattern, it can be done by indicating the size of each side (in the metric desired system).

II.7.1.Stereo Vision

II.7.1.1. Introduction

The human brain works with stereo vision. We have two eyes and from them both we can infer depth. If we close one of our eyes our depth perception becomes severely crippled and our ability to accurately detect an object distance without moving around it is greatly reduced (we can identify sizes and distances still with some ability not just due to the vision but also to our own perception of the world, we know what size some objects should be and some relative distances).

Very much like our brain and with just one eye (assuming the other is closed or covered) acquiring depth from a still position is impossible without some knowledge of the surrounding world.

Gary Bradski and Adrian Kaehler, in learning OpenCV [6], defined a four step process for stereo vision:

1. **Undistortion** - Mathematically remove radial and tangential lens distortion. The effect was described in II.4.
2. **Rectification** - Adjust for the angles and distances between cameras;
3. **Correspondence** - Find the same features in both left and right images;
4. **Reprojection** - Calculate a disparity map;

Undistortion, rectification and correspondence are three much studied topics that can be extended by reading the mathematical basics of computer vision. We will focus on the last step – reprojection.

II.7.1.2. Reprojection

This is the fourth and final step, so in order to execute this step all the previous three steps are assumed have to been executed successfully, and both right and left images are undistorted, rectified and reprojected. If all this is meet, Figure II.15 shows the basic method for distance acquisition from a stereo rig and equations II.12 and II.13 its formalization. Besides this, it's assumed that:

- The principal points have been calibrated, in order to have the same pixel coordinates in the two images, as was explained in II.4.1;
- Both cameras have the same focal length $f_l = f_r$;
- Images are row-aligned and every pixel row of one camera aligns exactly with the corresponding row in the other camera;

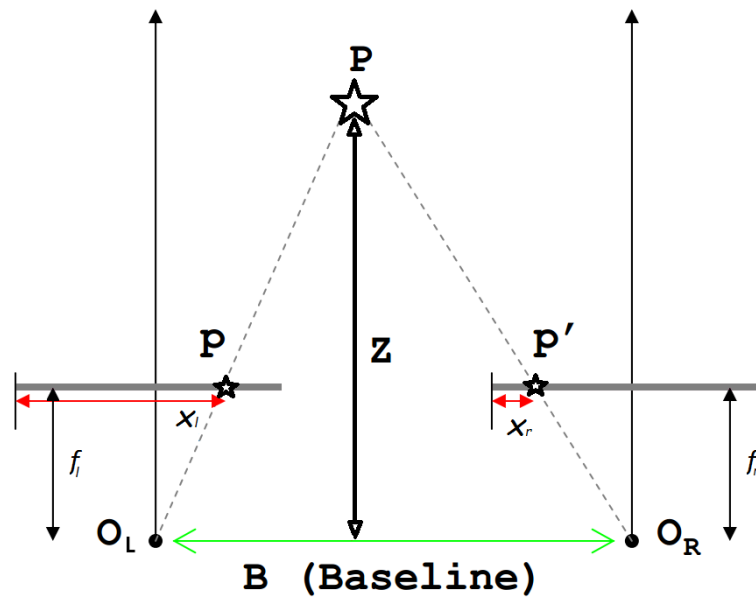


Figure II.15 – A point P is recorded in the left camera as p and in the right camera as p'. f_l and f_r are the left and right focal lengths of the cameras respectively. x_l and x_r are the distances to the left side of the projected plane in the left and right cameras respectively. B is the baseline of the cameras, this is, the distance between them.

$$\frac{B - (x^l - x^r)}{B - f} = \frac{B}{Z} \Rightarrow Z = \frac{fB}{x^l - x^r} \quad \text{II.12}$$

Where d is the disparity:

$$d = x^l - x^r \quad \text{II.13}$$

The represented model is the most simplified case and is shown here in order to explain how it works. The real algorithm used and equations are far more complex, but they are also out of the scope of this work.

With equation II.12 it's easy to realize that the disparity d is inversely proportional to the distance Z . This means the further away the object is the smaller the disparity d will be. This means that a stereo vision system have a higher depth resolution for objects nearer to the camera and that larger differences in objects farther away are more difficult to detect. Figure II.16 shows the relation between disparity and distance.

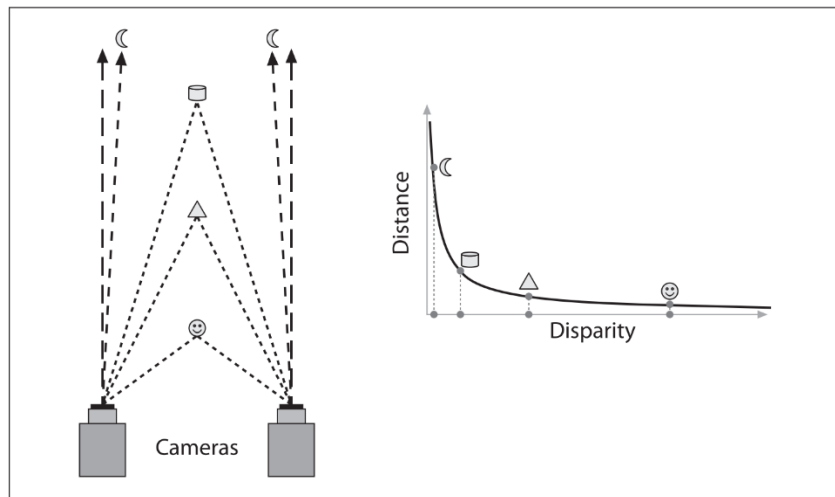


Figure II.16 – Relation between object distance to the camera and the disparity [6].

III. CAPTURING IN 3D

As was stated the scope of this project was to use digital photographic cameras to try and reconstruct objects. Many digital cameras are available on the market with a lot of different technologies both on the optics and on the capturing mechanism. And some even in the compression methods used to translate the electronic signal to digital information.

There are several methods to capture an object using only one camera, from the distance to the object, to the distance between each image taken and environment light, so the best way to obtain images to make 3D reconstruction was studied and will be completely depicted.

Fredrik Hollsten published a thesis [12] relating image quality and 3D reconstruction and by discussing his results better quality 3D reconstructions can arise.

By specifying the hardware and under which circumstances it was used it may be possible to discuss the result with more data and arrive at better conclusions. All the hardware used, variations and constructed rigs will be shown and explained.

All the tools necessary to obtain the best results in 3D reconstruction will be described in this chapter and it is expected they will allow us to get better results in 3D reconstruction from images.

Finally the specific tools employed in this project will be discussed and explained since some of them were built from the ground zero.

III.1. BEST RECORDING

There is still no perfect method to record the images, we can only minimize the errors. There are some that can be done in order to maximize the reconstruction quality but there is no flawless method. Fredrik Hollsten states *“The results of reconstructed models strongly depend on the material and lighting of the photographed objects and scenes.”* in [12].

Some algorithms work better under some circumstances and not so well under other circumstances. This is due to camera quality and/or the scene lighting. Fredrik Hollsten defined four major potential issues with the camera and studied their effect for 3D reconstruction. More detailed information can be found at [12].

1. Insufficient image quality;
2. Unfocused images;
3. Blooming [13]. Blooming is an effect specific to digital camera sensor, and it occurs when the charge in a pixel exceeds the saturation level and it starts to “leak” to adjacent pixel;
4. Lack of metadata. (Described in II.6);

Light changes and excess of light was a major issue during the course of this work, degrading significantly the quality of the reconstruction, calibration and rectification of the images.

Reconstruction is possible if every pair of images contains the same features but this is simply one of the requirements for a good result in reconstruction. Figure III.1 and the following list synthetizes all the major issues and necessities for reconstruction purely based on image acquisition (without hardware interferences).

1. The same features have to be present in at least a pair of images to attempt the reconstruction;
2. The object must not be occluded;
3. The object must be focused;
4. The two images must not be identical i.e., taken from almost the same place;

5. If there is a body covering the object we want to reconstruct, a part of the reconstruction will not occur due to the lack of information.

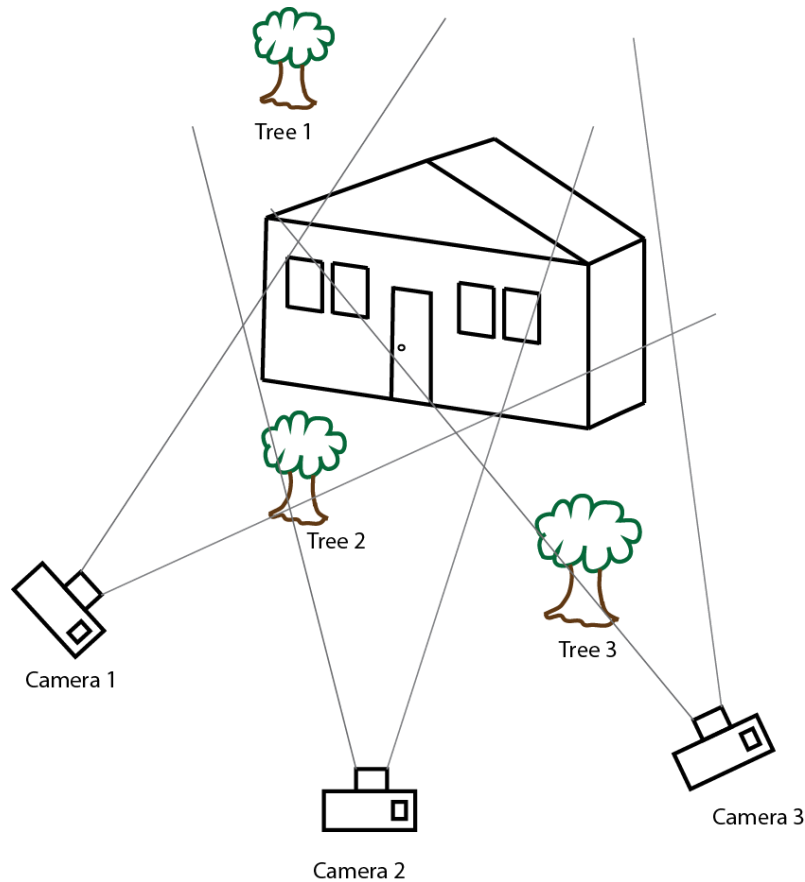


Figure III.1 – Issues when creating a 3D model.

Cameras 1, 2, and 3 have in their line of sight trees and a house, and although both of these objects can appear in each of the cameras pictures probably only one of these objects can be correctly focused (it all depends on the distance between the tree and the house, but let's assume the distance is such that only one of these can be focused at a time). *"Blurry images will not have any definable features"* [12].

Assuming that the user of the cameras chooses only to focus the trees the following statements are true:

- If camera 1 and camera 2 took each a picture from the represented stand, the reconstruction of the canopy of tree 2 would be possible because it is in the range of both cameras. This is only a partial reconstruction of the visible portion of the canopy, to reconstruct all the canopy a 360° capture would have to be taken.

- Although tree 1 is in both camera 2 and camera 3 line of sight it cannot be reconstructed and this is due to the house. The house is in front of both cameras and a covered object cannot be reconstructed.
- By stereoscopy tree 3 cannot be reconstructed, since only camera 3 can see it.

Assuming that the user of the cameras chose only to focus the house the following statements are true:

- Three pairs can be arranged with the shown cameras positions:
 - Camera 1 & camera 2;
 - Camera 1 & camera 3;
 - Camera 2 & camera 3;

III.2. EFFECT OF IMAGE QUALITY

A very recent study has been made in order to understand the effect of image quality on 3D reconstruction by [12]. This work focused in three major issues that can cause problems in 3D reconstruction:

- Noise;
- Blur;
- Image compression;

In order to test the effect that these issues can have, the author of [12] injected onto his dataset these issues artificially and with several levels of image degradation.

There is no satisfactory reconstruction quality scale and there is no way to compare the real world to the reconstruction. F. Hollsten took two factors to consider the quality of the reconstruction: -The point from where the reconstruction was no longer possible and, - The number of points found compared to the reference dataset.

By simulating a low quality image in a very large image the result to which he came can be contested. Assuming that the same camera takes two pictures of the same object from the exact same position, i.e. that both the images are exactly the same but that one

was taken with the 6MPx setting and the other was taken with the 1MPx setting, does a 6MPx image with one of the issues previously stated injected into its matrix have a better result than a 1MPx image? And what if a 1Mpx image had also those issues injected into its matrix?

This is an important question to ask, due to several reasons:

- A very good way to reconstruct (and sometimes the only way) is by using aerial reconstruction. The weight and size that a flying object can take is limited and because of that the quality of the cameras is much reduced.
- The processing time is increased with the size of the image, and due to the computational limitation this often means that the images are down sampled to accelerate the process. In [9] they down sampled images larger than 2 MPx and their experiments were run on a cluster of 62 nodes with, two quad core processors each, on a private network with 1 GB/sec Ethernet interfaces. Each node was equipped with 32 GB of RAM and 1 TB of local hard disk space. Even with all this computational power this took one day to process.

Even with a data set composed by very large area images F. Hollstein proved that image quality reduces the quality of the generated 3D model (even though it is still possible to reconstruct).

III.3. HARDWARE

A very important part, and something that will certainly influence the results, is the hardware used in the project, particularly the cameras that will capture the images. The better the quality of the image the better the generated model will be, but the bigger the image resolution the more time the software will take to process the images. As an example Reconstructing Rome [9], [14], reduced the resolution of all images above 2MPx in order to be able to make all the necessary program calculations within a single day.

The cameras used in this project are outdoor capable webcams with battery, that can support a maximum resolution of 1280 x 720 Px (0.9MPx) when working as a webcam directly connected a computer. The model is called keychain 808 #16 camera, and has a 1/4" CMOS WXGA HD Sensor. A picture of the hardware is shown in Figure III.2.



Figure III.2 – Keychain 808 #16 camera and a euro coin besides it for scale.

III.4. STEREO RECORDING

One of the techniques studied required a stereo rig to take the photographs so that the cameras would stay in the same relative position to each other (if the cameras changed their position the calibration done with them would have to be restarted). To support the stereo image process a stereo rig was created by creating a 3D model using Autodesk Inventor. This was then printed in the 3D printer that is discussed I. Figure III.3 shows the model and Figure III.4 shows the printed version of the model, with one of the cameras mounted.

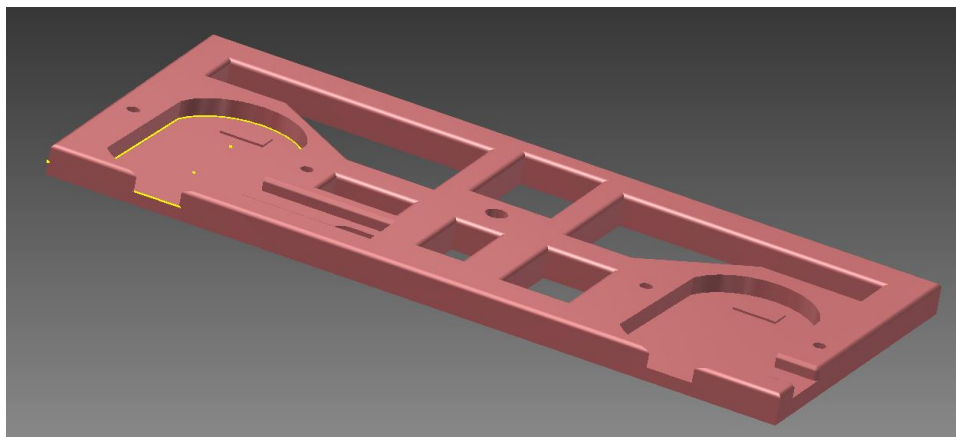


Figure III.3 – Model for mounting the stereo rig made with Autodesk Inventor.

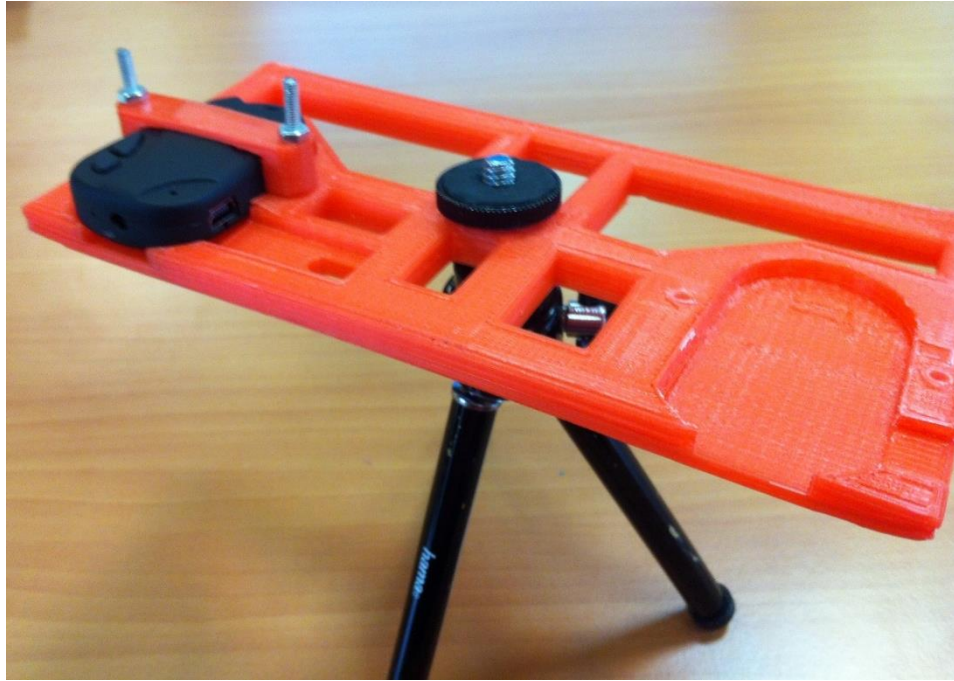


Figure III.4 – 3D printed version of the model shown in Figure III.3

Another rig was created using high quality reflex cameras. These are chunky cameras and it was very difficult to create a rig capable of supporting both their weight and the necessary electronic equipment to connect them to a computer.

Unfortunately it was not possible to have two reflex cameras of the same model. Instead two cameras of the same manufacturer, with the same optical lens mounted were used. Since one of the cameras had a much larger resolution, by using an additional step in the preprocessing of the images, the quality of the image was reduced to match the lower quality camera.

The specifications of the reflex cameras are shown in Table 2. The Lens can be set to have a focal distance between 18 and 55 mm and as such both of them were set to the same focal distance 22mm.

	Left Camera	Right Camera
Manufacturer	Cannon	Cannon
Model	1000D	500D
Resolution	10MPx	15MPx
Lens	Cannon zoom lens EF-S 18-55mm	Cannon zoom lens EF-S 18-55mm
Focal Distance	22mm	22mm

Table 2 - Specifications of the cameras used in the high quality rig.

These high quality cameras do support a computer connection that allows us to control them as you would with a webcam, so in order to overcome these difficulties an electronic circuit to control the camera shutters had to be created. This allowed the computer to communicate with a microcontroller, that had the capability of controlling the shutters.

Both cameras had a 2.5mm stereo audio connector that allows both the focus and the shutter to be controlled remotely by short circuiting the ground and the left channel or the ground and the right channel. Figure III.5 shows the remote shutter and zoom control implemented.

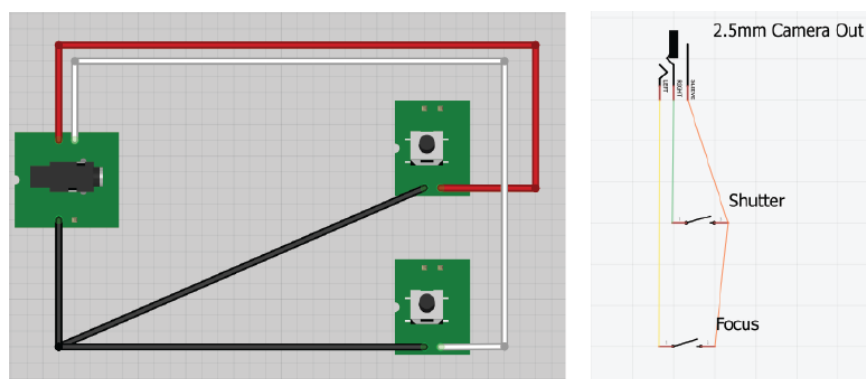


Figure III.5 - Remote shutter and focus. Component view on the right. Schematic on the left.

The chosen microcontroller was an Arduino [15] duemilanove due to its simplicity and efficiency. Another simpler controller could have been used, but this accepts a language

similar to C which is much simpler to program and control. To simulate the button a 2N2222 transistor was used. The created circuit, the C microcontroller program flux chart and the rig are shown in Figure III.6 and Flowchart III-1.

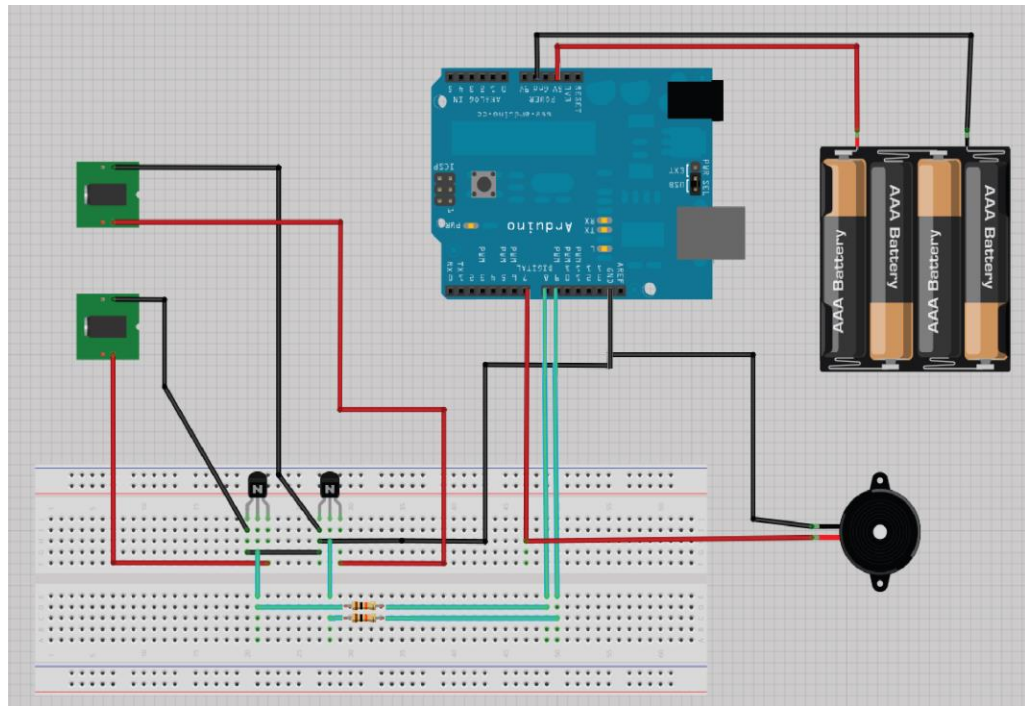
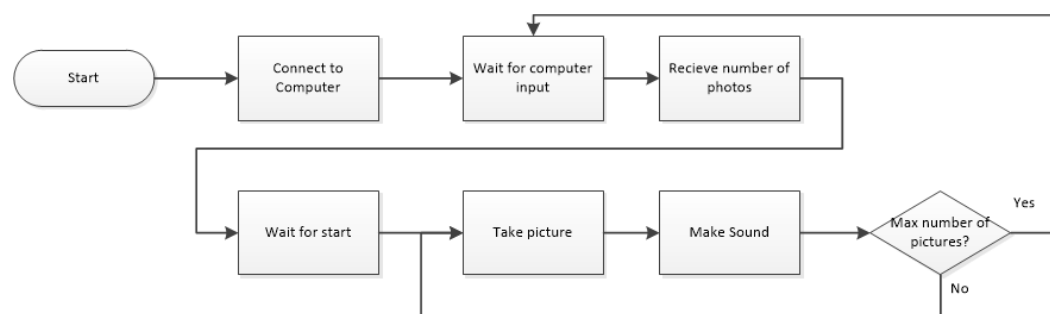


Figure III.6 – Breadboard schematic for the simultaneous control of a pair of reflex cameras for a stereo rig using an Arduino duemilanove, two 2N2222 transistors, two 2,5mm audio jack, two 10k Ω resistors, and a piezo buzzer.



Flowchart III-1 – Arduino program flowchart aimed at taking the calibration images for the high quality reflex images.

Flowchart IV-1 was implemented successfully using the pseudo C language. The microcontroller is plug and play, and the communication between the device and the computer was made using the COM ports. If the communication is successful, the user inserts the number of images to be captured and waits for the user to give the indication to start the routine (by pressing any key).

Capturing in 3D

When the user starts the microcontroller gives the signal to take the pictures and makes a sound. When the number of pictures taken reaches the users request the programs restarts and waits for users input. The images taken are stored in the reflex cameras memory cards.

IV. EXISTING SOFTWARE

As was seen in section II.7 there is much more involved in 3D reconstruction through photographs than just taking random photographs and expecting the software to be able to reconstruct the object or scene. If the camera is not calibrated, if there is no header information in the files or if the camera position in the scene is not correctly determined, then the reconstruction is severely crippled.

As if this was not enough, the range of application is very strict, being the foremost advanced and more studied the topological reconstruction. Unfortunately topological reconstruction has in many cases the advantage of knowing the camera positions through GPS coordinates.

In this chapter several 3D reconstruction softwares, both proprietary and open source, will be studied and dissected, their weak and solid options brought to light, thus enabling us to make a better and a more sturdy software.

An extensive investigation has shown that there is no available software that aims to collect pictures and generate a computer model equivalent with metric, and as such the whole process has been divided into smaller achievable parts and the available software categorized into that part.

It's only possible to compare results if the same datasets are used, so for comparison sake all the tested softwares will be supplied with the same pictures. These pictures will not use a stereo rig since only one of the presented softwares is able to take advantage of stereo images. Therefore all the pictures taken will come from a single camera. One of the keychain 808 #16 presented in III.3 will be used to take the pictures used in the proprietary section of this chapter.

IV.1. PROPRIETARY SOFTWARE

The main characteristic of proprietary software is that rarely the source code is available, thus we are only able to use the provided features as it was intended by the developers, without the ability to extend those features. That is their downfall, but also because they are proprietary and they usually ask for money for the software, they tend to work out of the box with less bugs.

With this in mind, proprietary software can be a good baseline for what has been already developed and what can be done. This type of software quality and functionalities can only be matched by cutting edge academic investigations.

The following list indicates all the tested proprietary 3D reconstruction related software and their home website.

- Autodesk Catch 123D - <http://www.123dapp.com/catch>
- Agisoft PhotoScan - <http://www.agisoft.ru/products/photoscan>
- Photo to 3D - <http://www.photo-to-3d.com/>
- Vi3dim - <http://www.vi3dim.com/>

All five of the proprietary software shown are easy to install or have no installation at all, so no information on how to install them is provided. In some cases the software is not built to be user friendly and tutorials are needed in order to learn how to make things work, but this section will focus mainly in the results.

IV.1.1. Autodesk Catch 123D

Autodesk Catch 123D is a cloud based 3D reconstruction software made by the famous AutoCAD developer Autodesk. It is cloud based because all the computational reconstruction effort is made by their own servers.

IV.1.1.1. Description

This software is available for free and the current available supported platforms are Windows (PC), IOS (iPhone and iPad) and browser. With the browser option this software is available on almost every single electronic device with access to an internet connection. The tested versions were the PC and the web application. Both of these were

used because they have slightly different user interfaces and model construction options. The web application is by far more advanced than the PC one.

This software generates the necessary files to open the model and work with it in other modeling softwares making it really easy to print it.

IV.1.1.2. Results

This application's process is mostly automated, and in the beginning only asks the user to select the group of photos he wishes to make a reconstruction of an object. The software uploads the pictures to the Autodesk server, computes the models and downloads the generated model. The system uses an automated point detection algorithm which finds the same point across multiple pictures and reconstructs the object.

Several data sets were inserted, but in none of them the software was able to completely reconstruct the object autonomously. In all the data sets no points were found in some of the images (perhaps due to different light and different focus) hence only half the object was reconstructed. However the same point in different images can be referenced manually and with several points inserted the model improved a lot in quality. After this process, called "manual stitching", the quality of the model was very good.

The reconstructed model shown in

Figure IV.1 shows a crude model of a helmet built by the software almost without user interaction.

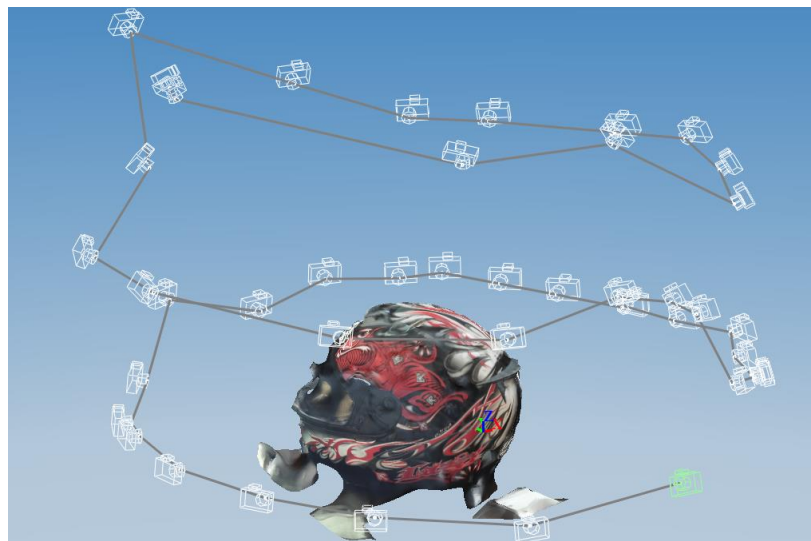


Figure IV.1 – A reconstructed helmet using Autodesk Catch 123D PC application and the camera positions of the used images.

By exporting this project and opening it in the web application it was possible to obtain a much better constructed model. This object has an open part in the front. The software allows us to select the mesh, adjust or delete it. Figure IV.2 shows the same object as

Figure IV.1 but treated with some additional tools available on the web application.



Figure IV.2 - A reconstructed helmet using Autodesk Catch 123D PC application.

IV.1.2. Agisoft PhotoScan & Agisoft StereoScan

Both Agisoft PhotoScan & StereoScan are available for the main desktop operating systems like windows, Mac OS X and Linux (both in x86 and x64) but there is no support for mobile versions at this point.

IV.1.2.1. Description

These softwares cover the two main types of reconstruction through images. StereoScan takes as input images taken from a stereo rig and it's the most simple of the software's presented in this sub chapter, and also the only one that is free to use. With the stereo pair images it can automatically calibrate the images, including camera positions and

lens distortions, being both image alignment and 3D model reconstruction fully automated.

PhotoScan comes in two flavors, a started edition and a professional edition. Both of them are paid but they are also much more powerful. Besides the ability to reconstruct image pairs this software can reconstruct through unordered images taken from a single camera and has the option to insert the XYZ information from where the images have been taken (this can be easily extrapolated from GPS coordinates). This is very useful in topographical reconstruction, where the images are taken from a plane which stores its GPS position.

This software also allows us to export the model and the point cloud generated, in order to treat it and create better results with other modeling software

IV.1.2.2. Results

Almost fully automated, with only one user input, this software generated almost exactly the same model as Autodesk Catch 123D did, without the additional user interface. The generated model was one half perfect and one half completely deformed, and because this software does not allow “manual stitching” it was not possible to obtain a better model.

Figure IV.3 shows the obtained model using Agisoft PhotoScan professional.

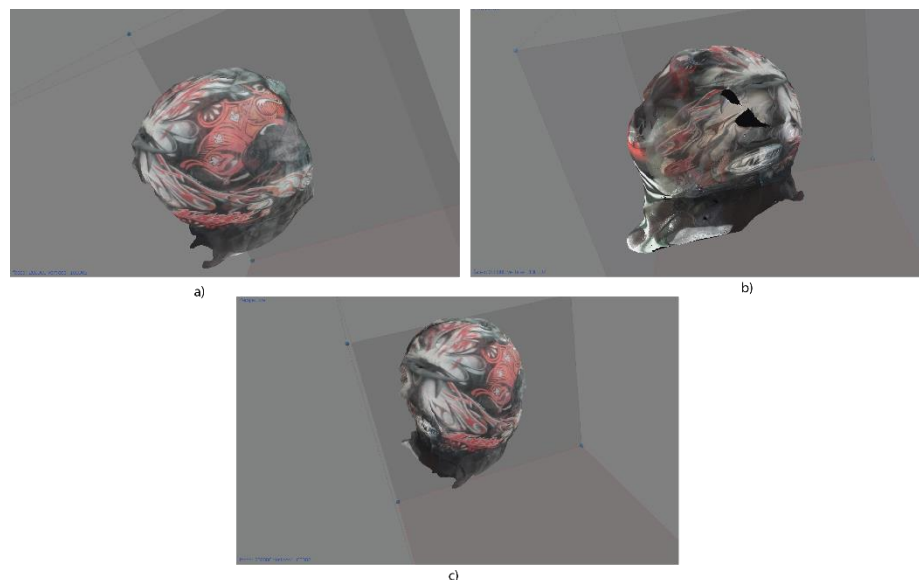


Figure IV.3 – a) Left side of the object, the reconstruction of the model was very good. b) The right side of the object is completely deformed and no recognizable. c) The reconstructed object from behind showing a part missing.

IV.1.3.Photo to 3D

A very simple software to use, available only as an online application

IV.1.3.1. Description

This web application has several formats for exporting, a large amount of working examples and a clear tutorial on how to be used, this software takes any two photographs taken by the user of a certain object from different angles and generates a 3D model.

The available exporting formats are: 3DS, Anim8or, Collada, VRML 2.0, X3D.

IV.1.3.2. Results

Again the same data set was used but because this software only accepts a pair of pictures at a time to create a model, several pairs of right and left pictures were selected and submitted. The obtained results were very poor as is displayed in Figure IV.4.

The examples detailed in the web page showed a pretty good reconstruction from only two pictures. To better test it a 5MPx and a 10MPx cameras were used. When computing the model using the 5MPx pictures, a very good model was generated just like the ones detailed in the webpage. However when the 10MPx pictures were inserted the program was unable to finish computing the model (a few hours passed without finishing the computation).

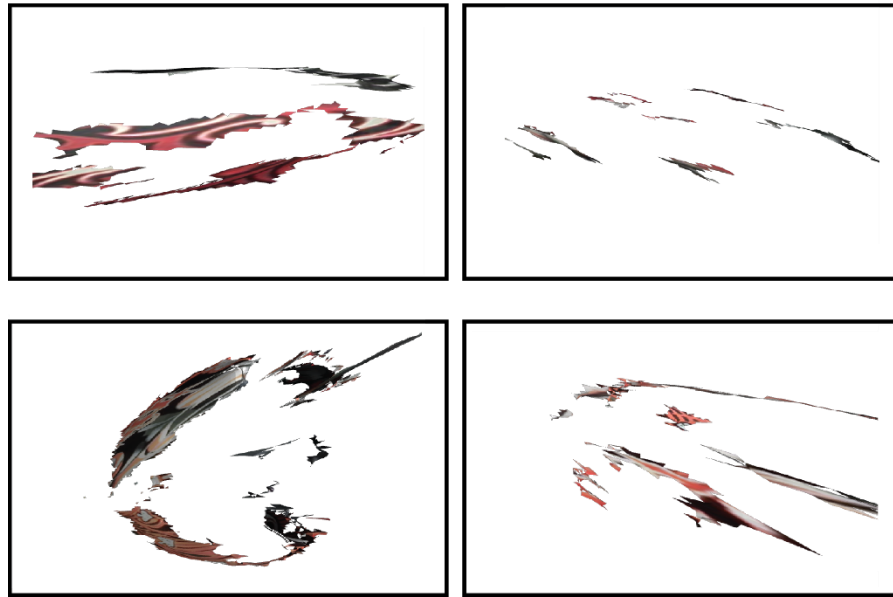


Figure IV.4 – Four different reconstructions using four different image pairs via photo-to-3d, with 0.9MPx photos.

IV.1.4. Vi3Dim

Vi3Dim is available in two packages, a demo and a full version requiring a paid license to have access to all the software functions for windows 7 (both x86 and x64) as a self-installing executable file.

IV.1.4.1. Description

Besides accepting pictures as input it also supports video input of the reconstruction objects (later decomposing the video as frames and saving them as pictures).

No shortcuts and no additional information is given to the user after install. The user has to navigate to the installation directory to find a group of batch files with no README attached to them. Without the online tutorial this would be a completely unusable software.

IV.1.4.2. Results

To work with this software one must follow a very restricted way of taking pictures. All of them have to be taken on the same plane and with the same spacing between them. There is no user interface and the program is a batch file that is run under the windows command prompt. Unfortunately there is no output to the user when the matching is complete and where the files are stored, although it was possible to find it by reading the manual.

Due to the very different method that this software uses, one of the samples provided by the manufacturer was used and even the video provided gave software errors and did not generate anything.



Figure IV.5 – Reconstruction of one of the samples of Vi3D software.

IV.2. OPEN SOURCE SOFTWARE

Open Source software has its source code available to read and modify, which is great, making it possible to extract parts of functional software and not needing to re-write everything. However with this great advantage comes also a great disadvantage, it rarely works out of the box, it has unknown or unsupported dependencies and many other problems may arise.

Computer Vision Open Source software doesn't have the same presence and is not as developed as proprietary software, tending to be more fragmented in the functions that it executes. Nonetheless libraries are extensively developed and are presented in Chapter V.

The following list indicates all the tested Open Source 3D reconstruction related software and their home website.

- Insight 3D - <http://insight3d.sourceforge.net>
- Reconstructing Rome [9] [14] - <http://www.cs.cornell.edu/~snaveley/bundler>

IV.2.1. Insight 3D

IV.2.1.1. Description

Searching for open source 3D reconstruction software reveals Insight 3D as the top option in the results, however it is in fact more of a 3D modeling software, based on images, than a working 3D reconstruction software. It requires a lot of user input to produce a 3D model and is more orientated to simple and full of edges objects (it will not work properly on rounded objects).

It starts with the same approach as any other reconstruction software, by matching all the photos of a real scene, calculating the positions in space from where each photo has been taken (including the camera's optical parameters) along with a 3D point cloud of the scene. Later it can create a textured polygonal model.

It's depicted here because of the lack of fully functional open source projects in this area that are not libraries with fragmented working parts making this the most complete open source project available (academic investigation projects are not included in this analysis).

IV.2.1.2. Results

Good when trying to reconstruct objects like houses and buildings with a fairly regular surface. Since this is a very user assisted software, it requires a lot of the user's interaction to make the reconstruction. The pictures are used as a guide in the reconstruction to help plan the reconstruction.

IV.2.2. Reconstructing Rome

IV.2.2.1. Description

The software used in Reconstructing Rome [9], [14] is called Bundler (available at [16]) and its first public appearance was made with an available version on August 2008. Still under current development with a repository on git hub, anyone can fork it and modify/use.

Reconstructing Rome Bundler is by far the more complete and complex reconstructing software available at this point. It is capable of using thousands of photographs, taken from multiple cameras, to generate a sparse point cloud and then a more dense point cloud, with an extra piece of software provided by them.

IV.2.2.2. Results

This software was developed in the period that occurred the transition between x86 and x64 architecture for Linux, and nowadays some of the necessary libraries and dependencies compiled in x86 are difficult to find. The installation process of this software is very difficult and required some effort.

The best results of this application are available, along with the published paper where they were able to reconstruct Rome with great efficiency, using a lot of computer power which could not be reproduced in this work [9], [14].

V. COMPUTER VISION LIBRARIES

Many have implemented computational algorithms to process real time computational vision, some have even created stable and fully functional libraries.

There are several libraries that aim at real-time computer vision, and most of them are directly or indirectly based on an Intel library, developed in the 90's, since some are based on each other.

There are many available libraries at the moment, some more progressive than others, some that are open source, that work in a wide range of architectures (x86, x64, ARM), that work in a wide range of operating systems (Windows, Linux, Mac OS X) and even in a wide range of languages. Some of them are even cross platform, and are implemented in several computer languages.

In this chapter the most common and well developed computer vision open source libraries available will be dissected, showing their ups and downs and how to install them. The installation part is somehow tricky, since there are a lot of dependencies and it involves some research.

A good, stable, and efficient library is very important to have the best results and without it the final product can suffer immensely.

The libraries that were studied in this project are:

- OpenCV;
- EmguCV;
- BoofCV;

- SimpleCV;
- Caltech's camera calibration toolbox for Matlab.

V.1. OPENCV

V.1.1. Introduction

OpenCV is a computer vision library born in 1999, with its alpha release. This is a library that descended directly from Intel's laboratories. It is open source and was developed in the early stages in optimized C, making it highly efficient. Nowadays this library supports C, C++, java, python and MatLab, and is available for the three major operating systems on the market: Windows, Linux and Mac OS X.

It supports Intel's Integrated Performance Primitives (IPP) libraries [17] which allow further optimization on the low level, i.e. on the processor itself. This can greatly increase the computer performance of the algorithm since it allows the usage of otherwise not accessible routines [6]. These libraries were not tested and we have no data to allow us to describe the efficiency. Nonetheless some functions are slow and cannot be used in real life.

The applications of this library are well beyond what is needed for this work, meaning that this is the most complete and extensive library that was studied. It contains over 500 functions, that span in many areas in vision [6].

The OpenCV version used in this work was the stable release 2.4.2 and is available at <http://opencv.org/>.

V.1.2. Platform

It was decided we would work with the Windows 8 operating system, Microsoft Visual Studio 2012 (MVS) as the development platform and C++ as the development language. The first thing to choose was the programming language. C++ is more powerful than C being already object oriented (OO) but still low level enough to be optimized.

Choosing the Integrated Development Environment (IDE) and the operating system derived from the language that was chosen. There are not many C++ IDE's and the newly launched MVS developing platform is clean and well organized.

V.1.3. Installation

Windows 8, MVS, CMake and Visual C++ Redistributable for Visual Studio 2012 are pre-requisites for installing this library.

By default the OpenCV binary generation doesn't support OpenGL. To generate the binaries there are two requisites: - Glut [18] must be installed (MVS by default should install this and) – and an option must be selected.

CMake is available at <http://www.cmake.org/>. This is a cross platform compiler and will generate the necessary MVS files.

Visual C++ Redistributable for Visual Studio 2012 is available at <http://www.microsoft.com/en-us/download/details.aspx?id=30679> and is necessary to compile the libraries.

The latest Windows version of OpenCV was download from <http://opencv.org/> and the file OpenCV-2.4.2.exe was extracted to the root of the C: drive.

CMake should be run with administrator rights. There we are asked for two directories, one with the source code and one where to build the binaries. The OpenCV source code is located in the OpenCV-2.4.2.exe extraction place and the binaries should be created in a build folder inside the OpenCV directory, for a good practice. For binary generation the CMake header should look like Figure V.1.

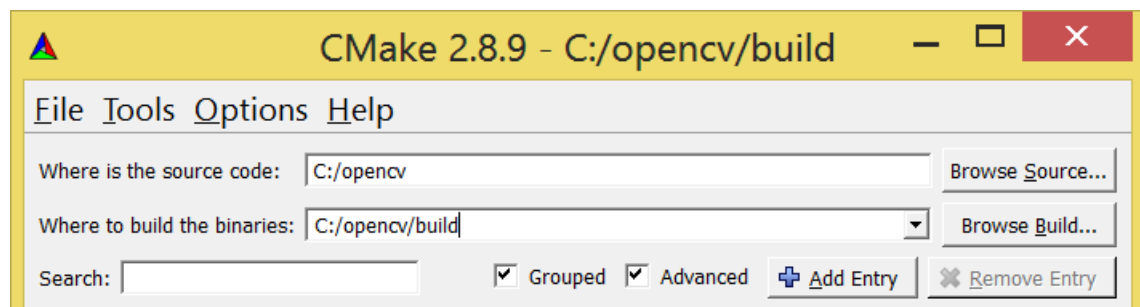


Figure V.1 – CMake header for binary generation under Windows 8.

Next is time to choose the compiler. For this the configure button is to be pressed and the compiler chosen. The generated files are platform specific and should be used later with the used compiler. MVS 2012 compiler is the eleventh version of MVS. It was chosen as the Figure V.2 shows.

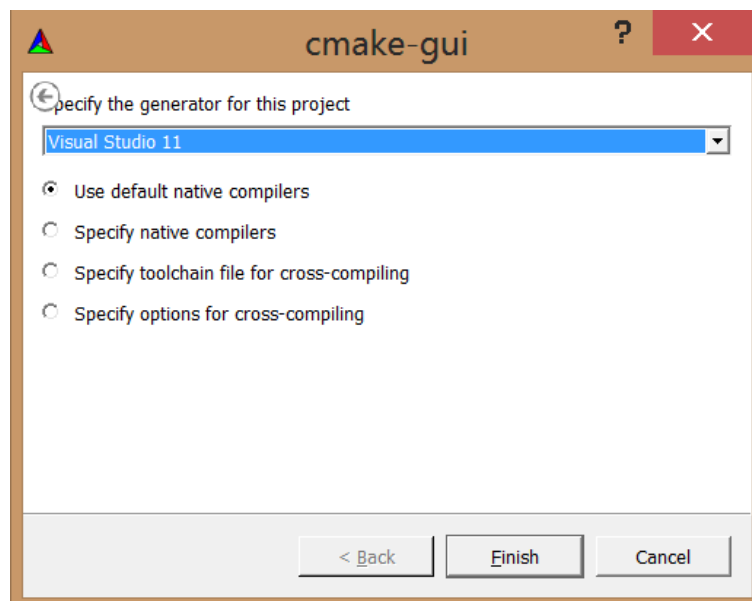


Figure V.2 – CMake searches the computer for the available compilers. For MVS 2012 the option to be chosen is *Visual Studio 11*.

Next CMake offers a wide range of configurations for the binary generation. The most common are already selected. If by any chance an option is not selected at this time the whole process must be repeated.

OpenGL is not selected by default. To select it the highlighted checkbox in Figure V.3 must be checked and OpenGL `gl_LIBRARY` and `glu_LIBRARY` must be found. These libraries will allow us to use the open GL OpenCV libraries i.e. to display the 3D data on the computer screen.

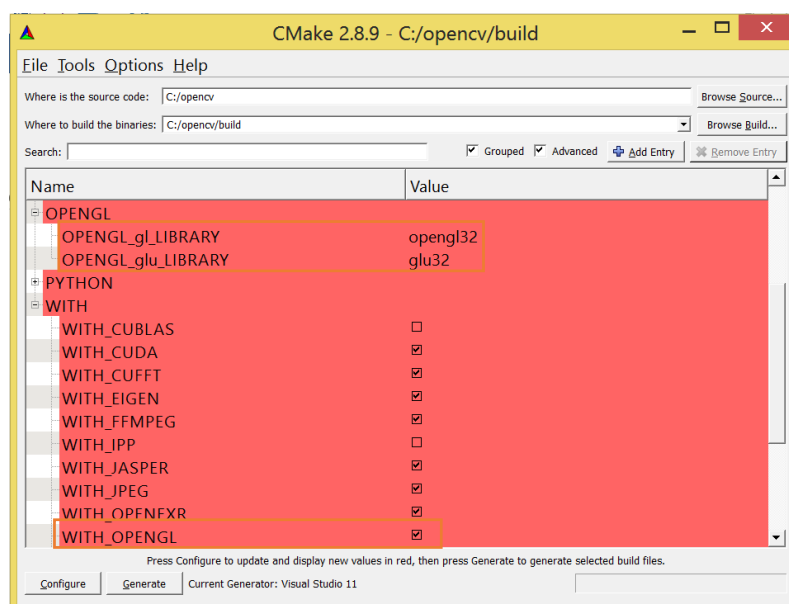


Figure V.3 – Add support for OpenGL in OpenCV before generating the binaries in CMake.

After this is done, it is only needed to press the generate button (bottom left of Figure V.3). This generates the OpenCV.sln file, which is a MVS solution file that contains all the libraries and its dependencies structure.

Figure V.4 shows the solution generated by CMake, opened in MVS.

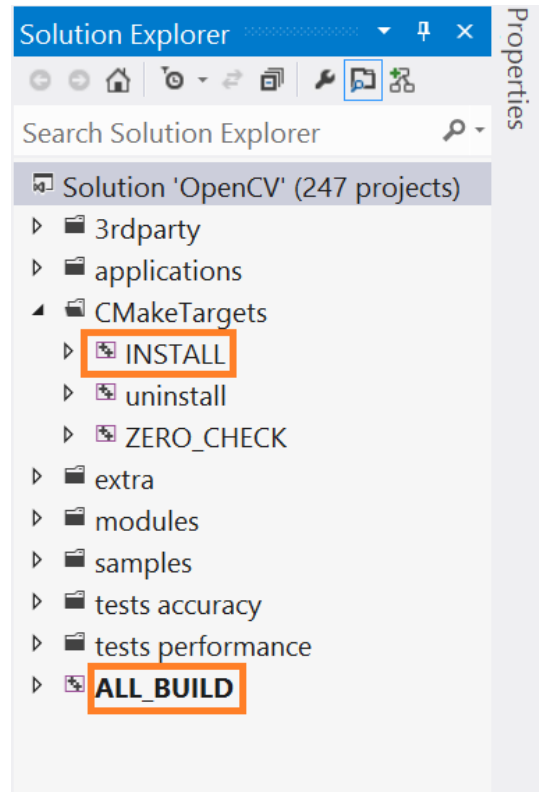


Figure V.4 – MVS 2012 Solution generated by CMake ,opened in MVS 2012.

MVS can build the library in two modes, Release and Debug. Their difference is straightforward, the Debug build offers Extended Debug Options. We generated both of them. Building ALL_BUILD will build the entire project an INSTALL will install it.

All OpenCV necessary files should be created by now, next they must be integrated into MVS.

First in MVS a new solution was created. The option selected was Win32 Console Application under Visual C++ as is shown in Figure V.5.

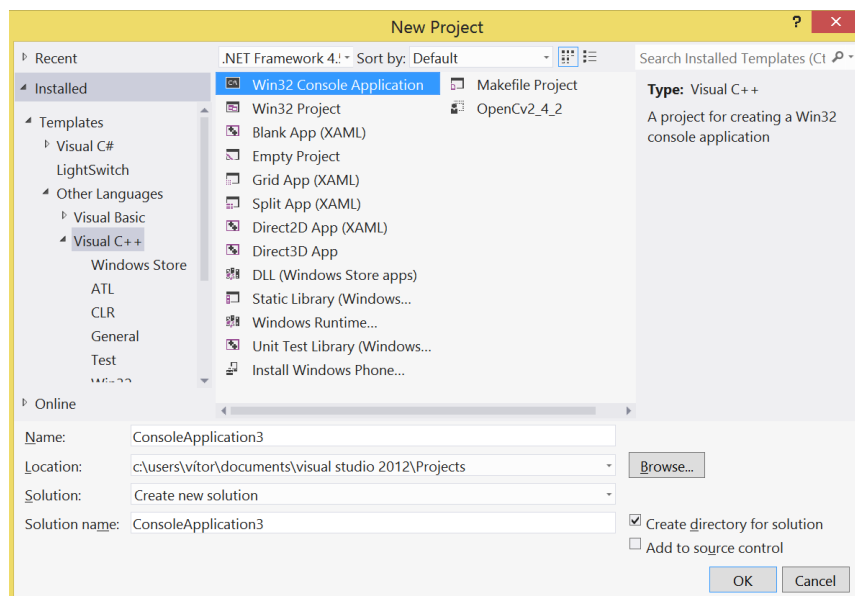


Figure V.5 – A new OpenCV project under MVS 2012.

By now only a simple console application, to be compiled with a C++ compiler, was created. For an OpenCV function to be used their include files (files written in C++ with library encoding) and the libraries files (.lib) must be included in the solution. These libraries files were generated previously when building the install project in OpenCV.sln solution. To include the necessary files the solution properties were selected.

Under C/C++ - General – Additional Include directories, we must insert the OpenCV include directories. These are under `C:\opencv\build\include`. Figure V.6 describes this procedure.

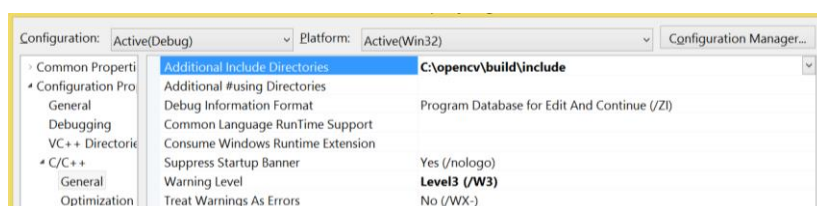


Figure V.6 – The additional Include Directories in MVS 2012 needed to compile a program with OpenCV.

Next, to finalize the configuration, the libraries were added. The library's directory is `C:\opencv\build\lib\Debug` for the debug files and is inserted in Linker – General – Additional Library Directories like it is presented in Figure V.7 and in Figure V.8.

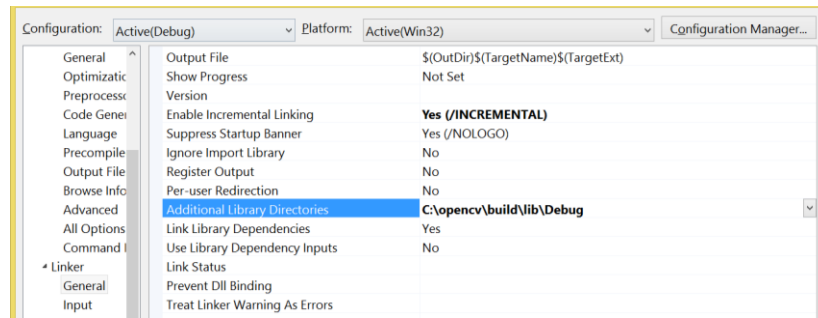


Figure V.7 – The additional Library Directories in MVS 2012 needed to compile a program with OpenCV.

The libs file names were inserted into Linker – Input – Additional dependencies. These libs are crucial to compile programs with OpenCV. Without them MVS will recognize all the classes and methods but will be unable to generate an executable. The necessary libraries are noted next.

```
opencv_core242d.lib
opencv_imgproc242d.lib
opencv_highgui242d.lib
opencv_ml242d.lib
opencv_video242d.lib
opencv_features2d242d.lib
opencv_calib3d242d.lib
opencv_objdetect242d.lib
opencv_contrib242d.lib
opencv_legacy242d.lib
opencv_flann242d.lib
opencv_ts242d.lib
```

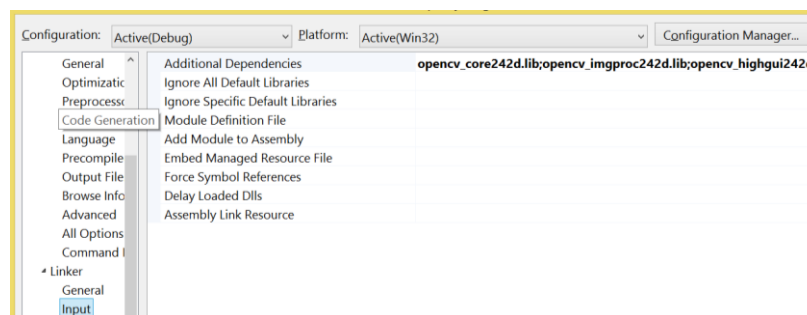


Figure V.8 - The additional Dependencies in MVS 2012 needed to compile a program with OpenCV

V.1.4. Results

OpenCV is still in migration to the new wrapper in C++, though the majority of functions have already been passed from older C to the new C++. However the tests show that the C library is more robust. This is because the same dataset and the same

program worked differently, just by changing the find corners function. Without a robust corner finding method there is no way to calibrate correctly the cameras and it is impossible to use more advanced library functions.

Besides the difference in the C and C++, this library generates highly distorted rectification matrixes and it has lots of problems when the images are highly saturated in terms of light i.e. if there is a bright source of light present in the image the generated matrixes are not usable.



Figure V.9 – The colored picture on top is unrectified. The bottom black and white is the rectified picture using OpenCV C library in MVS 2012.

V.2. EMGUCV

V.2.1. Introduction

Developed since 2008, this is a .Net wrapper for OpenCV. The library developers produce both an open source version and a more optimized commercial version.

This library allows OpenCV functions to be called from .NET compatible languages such as C#, VB, VC++, IronPython, etc.

The binary files are all provided for each of the platforms, so the users doesn't have to compile it by themselves and it is cross platform with support for Windows, Linux, Mac OS X, iPhone, iPad and Android devices.

Although distributed mainly as an open source library, some of the more advanced functions are paid, such as the Intel IPP libraries and the code can only be closed if the user buys this library. Otherwise all distributions are automatically open source.

V.2.2. Platform

As this is a .Net based library Windows and MVS are a clear choice for this library.

V.2.3. Installation

EmguCV developers provide two ways of installing this library: the source is available to download through git, or an installer containing all the necessary binaries to work under MVS 2008, 2010, 2012.

Through the git source it is possible to obtain the most recent code, since the EmguCV developers do not create binary packages for every single change of code and it is possible to use it under other operating systems. But this process is more complicated and there are no major differences between the current binary release and the most recent source.

The EmguCV developers provide a self-extracting executable file with all the binaries. After the extraction/installation, under MVS 2012, a solution was created. This was a Visual C# project with the Windows Forms Application template. Figure V.10 shows the creation of an EmguCV project in MVS.

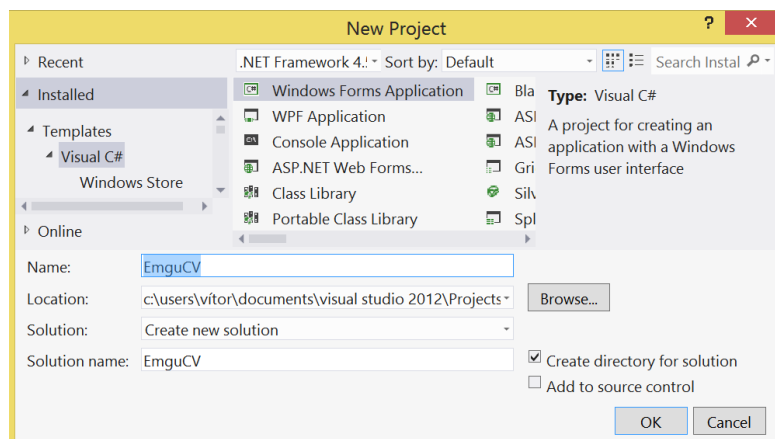


Figure V.10 – Creation of a new EmguCV C# project.

In order to use the EmguCV wrapper, both EmguCV and OpenCV libraries have to be referenced in the project.

First three dll files are needed: Emgu.CV.dll, Emgu.CV.UI.dll and Emgu.Util.dll. These EmguCV .dll's are under the bin folder in the installation folder, and to add them just go to Project – Add reference like is shown in Figure V.11.

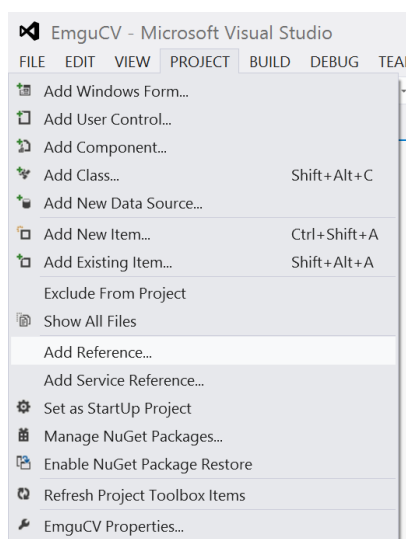


Figure V.11 – Referencing EmguCV library's in MSV 2012.

Next we need OpenCV dll's. These need to be copied into the project folder and when the executable is generated they have to be copied into the executable folder. This involves two steps. First adding the OpenCV .dll files under bin/x64 (opencv_*****.dll, npp64_***.dll, cublas***.dll, cudart64***.dll and finally the cufft64_***.dll). To do this just right click on the solution – Add – Existing Item... just like Figure V.12 describes. This process copies all the files into the solution folder.

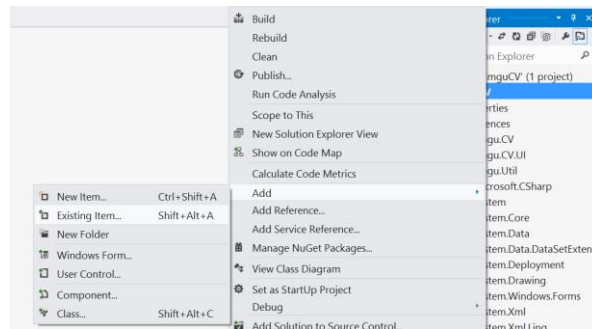


Figure V.12 – Adding OpenCV dll's to the C# project.

Lastly, the files have to be copied into the executable folder. For every single file in its properties, the “Copy if newer” has to be selected.

V.3. BOOFCV

V.3.1. Introduction

BoofCV is another open source library. It is written in Java and was built from scratch. According to its lead developer this boosts its performance and ease of use [19]. It even claims that it is faster than the native libraries.

Not as extensively developed as OpenCV this library is still in its early stages of development, being still in alpha. Nevertheless it has most of the base image transformation procedures encoded.

Since it is programmed entirely in Java its library binaries can be used in an Android project.

V.3.2. Platform

This library is Java specific and as such is cross platform. For Java development there are two very good and well known IDEs: Eclipse and NetBeans. These two IDEs are cross platform and work in Windows, Linux and Mac OS X, thus making the operating system irrelevant for the use of these libraries.

NetBeans has a cleaner interface and from past experience crashes less than Eclipse making this the preferred platform.

V.3.3. Installation

The author provides both the binaries and the java docs. This improves the installation process by a great deal. In the previous library – OpenCV – the installation is a tedious process and it was very poorly documented.

This is a regular Java program with additional libraries making it extremely easy to install. A Java project was created (Figure V.13) the .jar binary files added (Figure V.14) and javadocs added (Figure V.15).

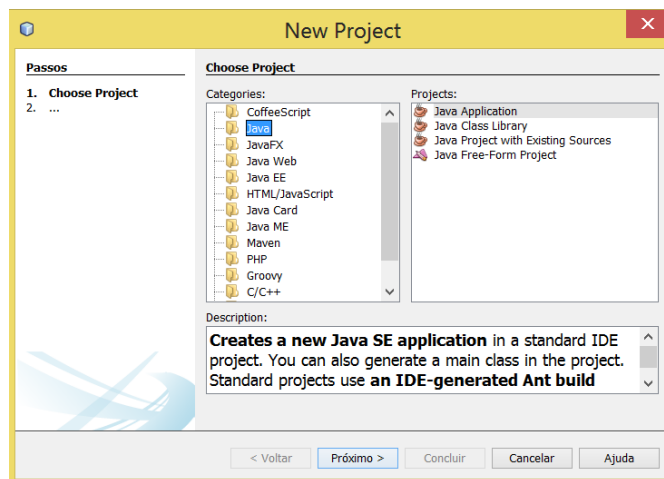


Figure V.13 – Creation of a new java project in NetBeans

After having a project created it is time to add the libraries of the BoofCV library. These are provided by the author in binary form, in a .jar file. To add the binaries the project properties and the Libraries tab were selected. This will show an “Add Library” option.

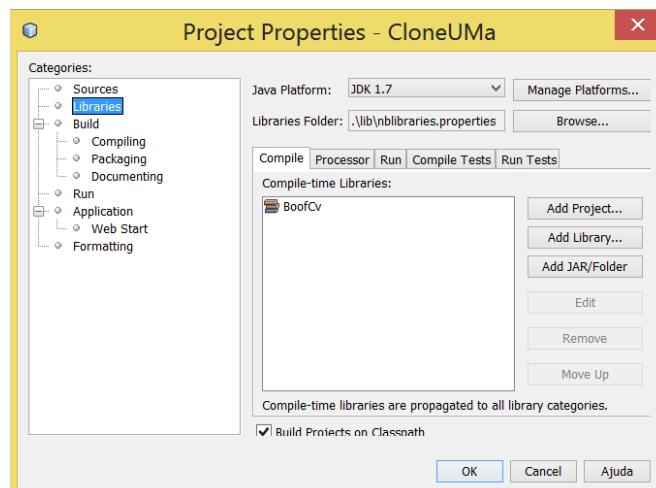


Figure V.14 – Adding the BoofCV library binaries to the NetBeans project.

With this NetBeans will be able to compile and run the project with the BoofCV classes and methods. To ease the programmers work, if we can add the java docs additional information and autocomplete of classes and methods, as is shown.



Figure V.15 –Addition of java docs to NetBeans to shows the library’s documentation and autocomplete.

V.3.4. Results

In the BoofCV webpage there are a few applets that show the functionalities of the library, making this a very promising library. However this was not the case. The applets provided in the webpage can be found at [20].

The library’s creator provides a set of simple programs and data sets to run the simplest functions of this library together with the distribution of the binary files.

With the provided datasets the library was able to find and rectify the images in perfection. But it had several difficulties with other datasets. Thus leading to conclude that this library has a very low quality pattern detection.

A good calibration and rectification is the beginning of any stereo vision program and without this part working perfectly the library is useless. But this is a young library and it’s still in its alpha version.

V.4. SIMPLECV

V.4.1. Introduction

SimpleCV is another cross platform wrapper for OpenCV, written in python. The aim of this library, as the name says, is to simplify the computer vision process that is often over complicated with libraries such as OpenCV.

There is no information regarding the ability to run this library code under other than x64 and x86 machines, that being a great disadvantage of this library.

V.4.2. Platform

All the previous tests were taken under Microsoft’s Windows 8, and since this library is able to run under this operating system and there is no obvious gain, identified by

the library's creators, for choosing any other operating system the chosen platform is once again Windows 8.

V.4.3. Installation

As any open source project the source code is available online but for simplicity they offer a bundled version (SimpleCV Version 1.3 Superpack) with all the necessary files to run the installation.

An executable file is provided which installs everything in its place, including all dependencies.

V.4.4. Results

This library has all the resources necessary to detect and proceed with the image calibration and rectification, except the actual algorithms developed into it. Everything would have to be written in order to test its capabilities and this is beyond this work.

V.5. CALTECH'S CAMERA CALIBRATION FOR MATLAB

V.5.1. Introduction

Jean-Yves Bouguet developed a Matlab toolbox in the early 2000s that is freely distributed online at [21]. It is important to note that this is a very powerful toolbox for Matlab with a lot of features and that it is often used in academic applications.

It offers a Guided User Interface (GUI) with all its options and all the basic standard options to get started in 3D imagery. The extrinsic and intrinsic parameters of the cameras can be calculated and it uses the same kind of pattern as shown in Figure V.16, meaning that the images can be undistorted and calibrated.

An interesting feature of this toolbox is that it can show the position of the cameras relatively to the pattern, for each of the photographs taken in the calibration process (or the opposite, the pattern change to a static camera).

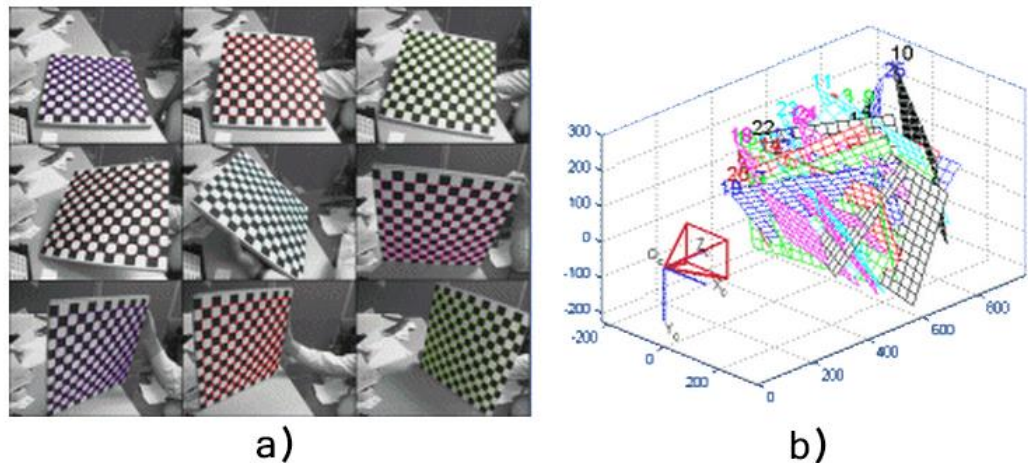


Figure V.16 – a) Pictures analyzed using Caltech’s Camera Calibration Toolbox for Matlab with the pattern corners highlighted in color. b) The position change of the calibration pattern is shown in relation to a static camera [21].

This toolbox was not tested due to the lack of a Matlab license.

V.6. OVERALL LIBRARY RESULTS

Out of the four libraries two had very bad results and thus they were discarded promptly as unsuitable for the problem at hand. This was unfortunate because the languages that they work under are high level and programming under these would reduce the time and effort for the encoder.

OpenCV and EmguCV are basically the same library, but in the case of EmguCV there is as translation layer to use the OpenCV binaries thus making this library less efficient than OpenCV. This is due the fact that when we use the same function in OpenCV, EmguCV goes through a translation process to call this function.

Since they are the same library, with an extra coating as the major difference, the choice stands between simplicity in encoding the program or fastness and better resource usage. Working with images is a very resource consuming area, and the larger the images the more processor and memory is necessary. Consequently it was chosen to work with OpenCV. With this choice it is expected that the programs run faster and become more reliable, than its counterpart in EmguCV would be.

VI. 3D PRINTER

3D printers have been available in the market for a few years but the machines were very expensive and the materials used were also very expensive and controlled by the manufacturers, in order to make a bit of extra profit.

The RepRap (replicating rapid prototyper) project is an open source project that aims for the distribution and construction of a free 3D printer that anyone can build at home. Free here is free of extra cost due to author's copyright. It's a self-replicating machine since most of its core parts are made of plastic and can be printed in a working machine.

The technique used to make the model is based on a variant of fused deposition modelling, an additive manufacturing technique, called Fused Filament Fabrication (FFF) and uses thermopolymers such as Acrylonitrile butadiene styrene (ABS) and Polylactic acid (PLA).

Since its start in 2007 this project has release four major printers:

- "Darwin" in 2007;
- "Mendel" in 2009;
- "Prussa Mendel" in 2010;
- "Huxley" in 2010.

As a part of creating replicas of real world objects a way to take them out of the digital world is needed. This open source project is perfect to bring out the digitally recreated objects.

This project has already several years of existence and counts with several specialists in the area, and thus its bugs and problems have been minimized. There is already a vast

construction manual, therefore in this work only a general construction order, the stress points of the construction, and the improvement points will be depicted.

There are several construction kits available for sale in the internet, due to the expansion of this project. One such construction kit was bought, of the most recent 3D printer to - the iteration 2 of the RepRap printer, the “Prussa Mendel”.

VI.1. BUILDING THE “PRUSSA MENDEL”

The model that was built was the most recent iteration (iteration 2) of the RepRap printer, the “Prussa Mendel”.

The construction of this printer can be divided into separated steps and when each all parts are completed the whole printer can be assembled. The main parts are:

- Frame;
- Extruder head;
- Electronics.

VI.1.1. “Prussa Mendel” Frame

The printer frame is the support for the printer. If it is not well assembled and calibrated the printed objects will more likely be flawed, or even in some cases the printer will not finish the object.

The frame is all made of 8mm steel studding (threaded bars), with plastic printed pieces and nuts maintain everything in position. An assembled frame is shown in Figure VI.1. The black parts were printed by another printer and are made of plastic.

Originally the supplied rods were the X and Y axis of the printer moved, were not made of hardened steel. With use, the bearings carve a path in the rods generating problems with the printing. So it was decided to buy and use hardened rods. Figure VI.1 shows the printer under construction with the axis X,Y,Z drawn in red.

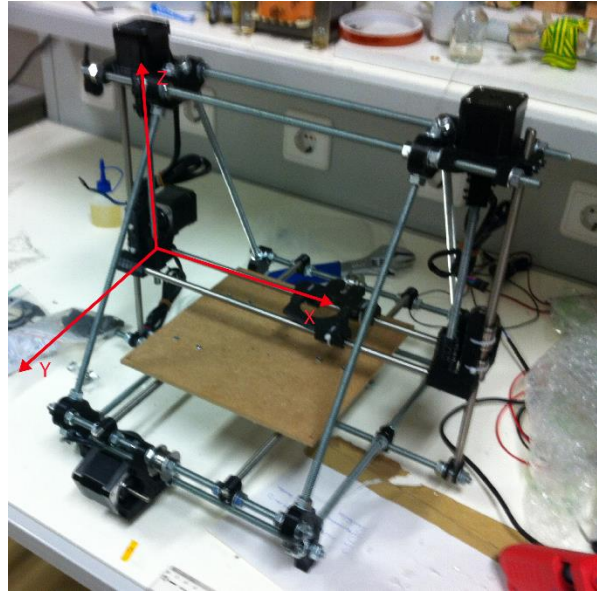


Figure VI.1 – “Prusa Mendel” 3D printer frame under construction, with its axis shown.

VI.1.2. “Prusa Mendel” Extruder head

The extruder head function is to control the flow of plastic, to melt it, and to deposit it on the heat bed. This printer works with a single thread of plastic at a time and it deposits, layer by layer, a single line of plastic in the heat bed.

If the flow of plastic is not well controlled there might be excess plastic deposited in some parts and in other parts no plastic at all, so it is crucial to check the assembled parts. Figure VI.2 shows a pair of gears that do not match well together. This was a problem that should be solved for better results.

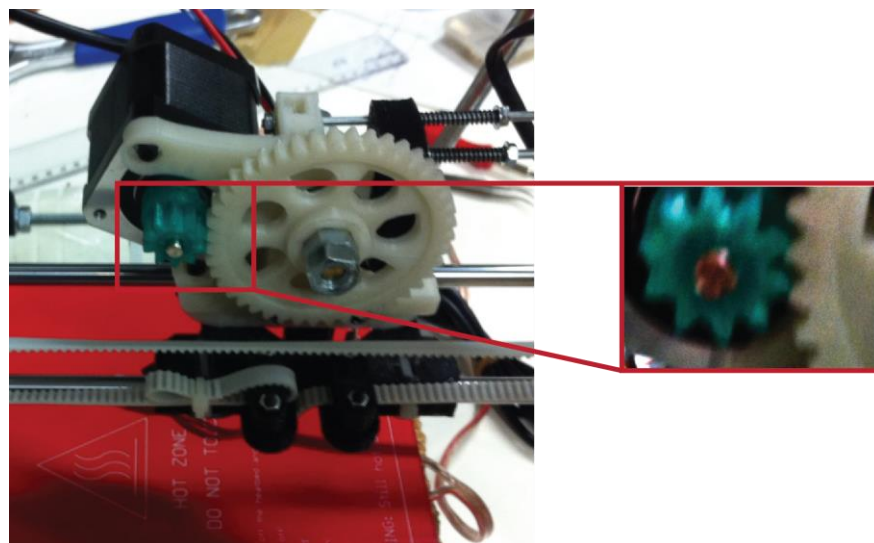


Figure VI.2 – Mounted extruder head and a magnified image of the gear. The gears do not match perfectly and this can lead to a fault with the plastic flow.

VI.1.3. “Prussa Mendel” Electronics

The “Prussa mendel” electronics include:

- Arduino Mega – This is the heart and head of the machine. This microcontroller is programed with one of the available firmware versions;
- RAMPS [22], [23] – An expansion shield for the Arduino Mega board that contains all the necessary parts to control the RepRap such as the stepper motor controllers. *“It is designed to fit the entire electronics needed for a RepRap in one small package for low cost”*[23]
- Stepper motors – The motors take care of the moving parts of the machine by moving the heat bed location (Y axis), the extruder head (X axis) both trough belts, also the extruder head position through threaded rods (Z axis) and finally the flow of plastic through gears;
- Heated bed [24] – The heated bed improves printing quality by helping to prevent warping. The reason behind this is as extruded plastic cools, it shrinks slightly. When this shrinking process does not occur evenly throughout a printed part, the result is a warped part;
- Heated bed thermistor – Used to control the heated bed temperature in order to maintain it stable.
- Nozzle resistor – The plastic filament is pushed into a nozzle through a system of gears and hobbled bolt. Then to melt this pushed plastic a resistor is used. The resistor heats up when a current passes through it;
- Nozzle thermistor – This thermistor is located near the nozzle resistor and is used to control the flow of current that passes through the resistor. By controlling the flow of current it is possible to maintain the temperature stable and optimal for the type of plastic used;
- X, Y, Z end stops – These can be either mechanical or electronic depending on the users preference. These are used to control the $(0,0,0)^T$ position of the machine. In the constructed model X and Y used mechanical end stops and Z used an electronic light sensor end stop.

The core of the RepRap is the program that is inserted into the Arduino, and as such this should be a carefully and thought decision. The firmware used was Sprinter Firmware [25] due to is capabilities, expansion and from other users experience. This

firmware allows us to calibrate the nozzles temperature (to obtain a perfect heating curve) and it also permits to disconnect the heating of the nozzle (in case the nozzle thermistors temperature is turned off or the signal is lost somehow). This is a particularly important aspect, because if the nozzle temperature goes unregulated it can melt the nozzle and lead to the destruction of the hardware.

VI.2. STRESS POINTS BUILDING THE MACHINE

After the printer was built it didn't start printing perfectly after the mounting. Some adjustments had to be made. The issue that surged the most was the alignment of the frame.

It was very difficult to align the frame, and when changing the surface where the printer was, it discalibrated slightly. Thus every time the printer is moved to another place it should be recalibrated and checked if everything is in place.

Verifying the quality of the printer assembly can be done by printing several test pieces and checking some stress points. The quality is measured by checking:

- Correct head position after several layers, i.e., precision after several layer of plastic have been placed;
- Correct temperatures, i.e., if the extruder head temperatures are correct for the plastic used. Different plastics have different temperatures requirements;
- Correct plastic flow, i.e., if the flow of plastic through the extruder head is correct;
- Correct size of the objects, i.e., if the objects have the correct measurements;
- Correct sensor position, i.e., if the X, Y, and Z, end stops are working correctly, especially the Z end stop which can become uncalibrated very quickly. The Z end stop should be calibrated with the extruder head already heated, due to the dilatation of the metal;
- Full movement of the heat bed and head. If the frame is slightly off position the Z position movement can be seriously compromised and the motor might not have enough power to bring up the extruder head.

VI.3. RESULTS

With the machine completely calibrated the obtained results were good and the major issues on the printed pieces were due to the different plastic filaments used throughout the test. As is possible to see, in Figure VI.3 and in Figure VI.4, the color of the plastic is different, and in fact they are two types of plastic. These two plastics were used in printing the test pieces.

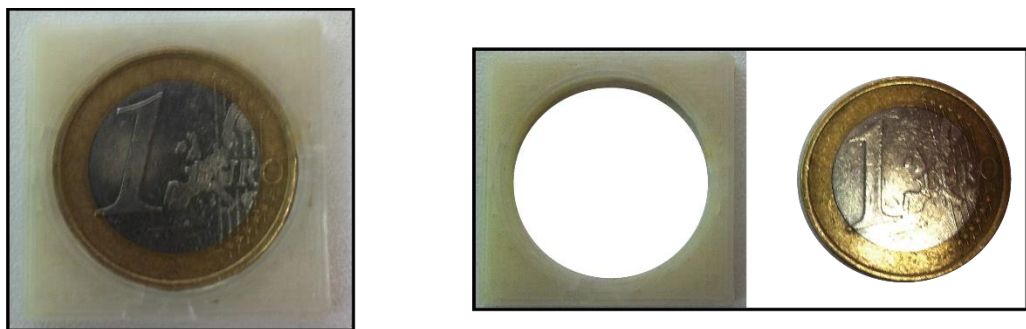


Figure VI.3 - On the left printed square with a hole of the size of a 1€ coin with the coin inserted. On the right the printed part and a 1€ coin. This test piece was used to check if the printer printed with correct height and correct hole diameter.

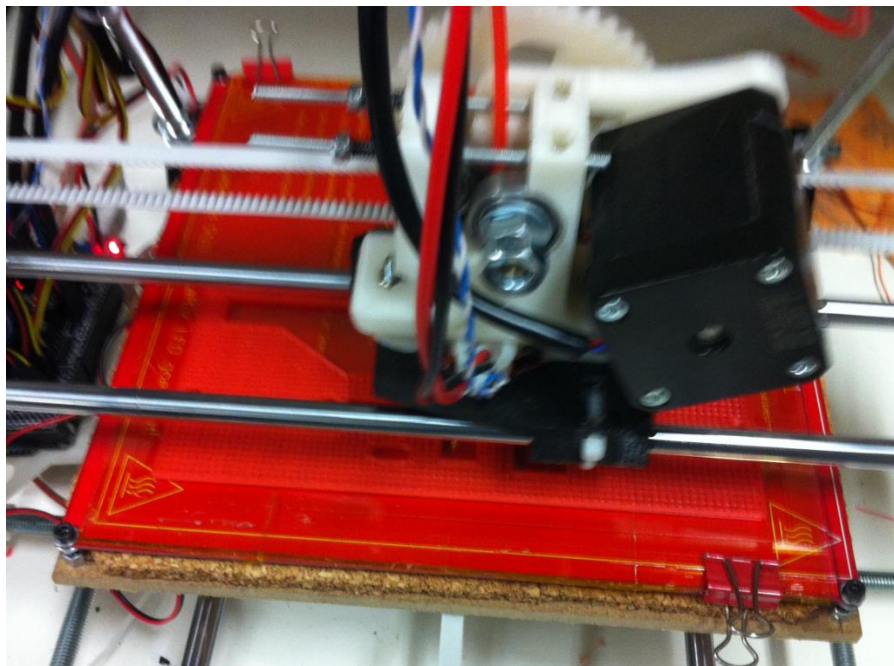


Figure VI.4 - Printing the stereo rig shown in Figure III.3.

VII. IMPLEMENTATION

As was previously demonstrated, this is a field that requires a lot of background work, in order to create a functional prototype. There are a lot of pieces that need to be put together in order to accomplish 3D reconstruction through images. Chapter V. addressed some of the best available libraries, one of which will be the basis for the implementation.

Studying different the techniques revealed that a solid and robust library of computer vision was required to produce an implementation. A library was required due to the high complexity of the mathematical models and due to the reduced time span of this project. Another reason is that the scope of this project was not to study already known functional models and to rewrite an already written code.

Based on the tests made on Chapter V. the best library to use was the one described in section V.1 OpenCV. Cross platform, integrated interfaces, high performance and other factors, made this the perfect choice. Again, as was with the library test, the platform environment for this implementation was very similar:

- OpenCV was pre-compiled with MVS 2012 C++ compiler, with support for MVS 2008, 2010 and 2012;
- Windows 8 was the operating system;
- MVS 2012 was the chosen programming IDE;
- Additional Common Language Runtime (CLR) support was added in the project;

The previous list describes the basic project created within MVS. This has a particular difference from the test project in chapter V.1, which is the added CLR capability. This allows for the use of the Windows graphic libraries.

In order to implement any code, a lot of libraries and programs were installed. These necessary libraries and software (the noncommercial versions) are attached to this project in digital form. Also all the code developed and implemented in this project and the results obtained, are attached in digital support.

VII.1.PROCESS

The implementation was stripped down to simpler modules, in order to simplify the development, and only when one module was working as intended and had the necessary results the next module started to be developed. The next list will show the implemented modules and a brief description of each:

- Image acquisition for calibration – This part covered the capture of the calibration pattern from different angles and distances.
- Image Calibration – This part intended to take the captured images and take them through the calibration process.
- Image acquisition for 3D generation – A simpler image acquisition than the one required for calibration.
- 3D Generation – Taking the captured images, generating a 3D model and exporting it to a file.

VII.1.1. Image acquisition for calibration

Capturing an image for calibration is more than just capturing what is being shown in front of the cameras, some tasks have to be fulfilled before committing and saving the image to disk. The requirements for this module are described in the following list:

- Connect to the camera rig and start receiving images;
- Capture an image from both cameras at the same time;
- Check for the pattern in the image;

- Inform the user that a pattern has been found;
- Multithread the image acquisition process, because the pattern finding solution is heavy in CPU usage;
- Wait a predetermined time before proceeding to the next image;
- Save the image if the pattern has been found and discard it otherwise;
- Stop the connection to the camera rig;

All the requirements have been met and were successfully implemented using the OpenCV library and C++. OpenCV had already the necessary camera modules implemented and was able to use several USB (Universal Serial Bus) cameras at the same time.

This module is an automated process that takes the images without any direct user interaction. What the user is doing is holding the calibration pattern (hopefully) in the line of sight of the cameras and changing its position and orientation.

Calibration is only possible if the calibration pattern is fully visible and identifiable by the program, thus this is why there is a necessity to discard the images where the pattern is not identifiable. Besides the obvious reason, which is the pattern not being captured at all because it was not in the line of sight, reflection and weird light sources can influence the identification of the pattern.

A more primitive and earlier implementation of this module did not have multithreading therefore all the processes were made by a single line of execution. Although it worked very well under most circumstances, when the pattern was nonexistent or was difficult to find the program locked for a few seconds in order to verify if there was really no pattern in the picture. This was due to the heavy processing needed to detect the calibration pattern. By adding multithreading the program can continue to take picture at the predefined pace and storing them if the pattern is found, while emitting a small beep in order to inform the user that the software was able to acquire a valid stereo pair.

VII.1.2. Image calibration

This module assumes that the previous module has been run at some point and that the images acquired are all in the same folder as the application or that there are stereo images in the folder with the patterns identifiable. The requirements for this module are described in the following list:

- Detect the calibration pattern;
- Load all the calibration pattern points into the appropriate format;
- Detect calibration pattern corners sub pixels;
- Calculate the Q matrix;
- Calculate the un-distortion matrix;
- Save the calculated camera parameters to an XML (eXtensible Markup Language) file;
- Report the calibration error;

If the previous module has been run all the necessary files are available for this module to run. The input files necessary are the stereo rig images.

The calibration pattern points have to be detected again in this phase not to know if they are in the line of sight (as was the case in the previous module), but in order to extract their location in the image. These extracted location points (in fact they are the corners of the chess board squares) are the input of an OpenCV algorithm that will calculate their sub pixels positions.

The better the calculated position of the pattern corners the better will be the calibration of the stereo pair. The sub pixels corner position algorithm, available with the OpenCV library, allows us to obtain the best possible positions of these corners to a sub pixel range. A detailed description of how this algorithm works and its mathematical basis can be found at [6] (page 319-321) and [26] .

The next step was to calculate the intrinsic and extrinsic parameters of the camera described at II.4.4. OpenCV has a function that takes as input the already computed calibration pattern corners, and that outputs the R, T, E, and F matrixes. This function is called stereoCalibrate. A detailed description of how this algorithm works and its mathematical basis can be found at [6] (page 427-430) and [27].

“It is easiest to compute the stereo disparity when the two image planes align exactly”[6]. All has been done in order to have a perfectly aligned camera rig, starting with the construction of a camera holder described in III.4, and with the use of twin cameras. However, unfortunately, it is very difficult to align the planes exactly. This step will solve this problem and with the cameras calibrated it is now possible to calculate what process the images have to be put through, in order to obtain a rectified

image pair and a stereo camera in standard form. The used algorithm was Bouguet's algorithms and it's implemented in the OpenCV function StereoRectify [28]. A detailed description of how this algorithm works and it's mathematical basis can be found at [6] (page 433- 436). Again, OpenCV has the built in machinery necessary to rectify the stereo rig.

With the Q matrix calculated (by StereoRectify) next is the un-distortion matrix M. By feeding InitUndistortRectifyMap with the parameters of the stereoRectify it is possible to get the matrix.

VII.1.3. Image Acquisition for 3D generation

Very similar to VII.1.1, this module connects to the stereo rig and saves the captured images. This module can be run before VII.1.2, storing the images for later.

This is a simple take and store the pictures for future use, and there is no need to verify anything in the picture.

VII.1.4. 3D generation

All the necessary steps have been made to create a 3D model of the photographed object. Calibration, rectification, and image acquisition have all been run and the only step left is to take all the information and create a 3D model. First the images taken in VII.1.3 have to be rectified and calibrated using the matrixes acquired in VII.1.2 Image calibration and only after this can a disparity map be generated. The disparity map will contain the distances of the object to the camera. The requirements for this module are described in the following list:

- Load the Calibration and rectification parameters;
- Calibrate and rectify the images;
- Load the disparity algorithm parameters;
- Calculate the disparity between the images;
- Generate the 3D sparse point cloud;
- Export the point cloud;
- Display the point cloud;

The calibration and rectification parameters were stored in an XML for simple reading. OpenCV has a small library integrated for reading and writing formatted matrices in XML and YML files so the loading is straight forward. This is a huge file containing all the transformations for each pixel that have to be made in order to obtain a calibrated and rectified image.

After the calibration and rectification it is now possible to calculate the disparity map. OpenCV has three built in algorithms to create a disparity map. The chosen one was StereoSGBM which is based on H. Hirschmuller algorithm [29],[30]. This algorithm offered a compromise between efficiency, with the ability to compute de disparity map almost in real time (with a ≈ 1 second delay).

The documentation is somewhat sparse and poor, and only the previously described algorithm is well documented. The three algorithms for disparity generation that OpenCV has implemented are described in the next list with a small description (the other two options were both tested, producing two very distinct results and this results are described as well):

- StereoBM [31]
 - Computes the disparity map using block matching algorithm;
 - The fastest of all with near real time generation making it exceptional to use in applications that require very fast processing, such as robotics;
 - Very poor results. The generated disparity map was a blur with nothing distinguishable;
 - Relies heavily on previous calibration by tweaking its internal values (it has some predefined values);
- StereoSGBM
 - Computes the disparity map using H. Hirschmuller algorithm [29],[30];
 - Relatively fast with a ≈ 1 second processing;
 - The best results were obtained with this algorithm;
 - Just like StereoBM it relies heavily on a previous calibration by tweaking its internal values (it has some predefined values);

- FindStereoCorrespondenceGC [32]
 - Computes the disparity map using graph cut-based algorithm;
 - Extremely slow, it took about 3 hours to generate a disparity map from a single image pair;
 - Produced a reasonably good disparity map;
 - This algorithm has disappeared in the most recent version of the library and the reasons are unknown;

3D generation can be done by applying directly the theory, using the Q matrix II.5.3 or by using the OpenCV included function `reprojectImageTo3D` [33]. Reports were found of problems when using this OpenCV function to generate the 3D point cloud saying that the generation was not working as supposed. Based on this assumption the two approaches were implemented, a custom 3D generation class from the disparity map was created allowing the user to choose which one to run, and thus making it possible to make some kind of comparison.

The chosen file format for exporting was the PLY which is a computer file format known as the Polygon File Format or the Stanford Triangle Format. A very simple non binary ASCII file format that has a configurable header to define its properties, which makes it perfect to this export.

Figure VII.1 shows the file header exporting code, in C++, and how simple it was to export the point cloud to a file named *"Exporting_file.ply"*, with *n* pints. Both the point spatial position and the point color in RGB are stored in this file format.

```
ofstream fout("Exporting_file.ply");
fout << "ply" << endl;
fout << "format ascii 1.0" << endl;
fout << "element vertex " << n << endl;
fout << "property float x" << endl;
fout << "property float y" << endl;
fout << "property float z" << endl;
fout << "property uchar red" << endl;
fout << "property uchar green" << endl;
fout << "property uchar blue" << endl;
fout << "end_header" << endl;
```

Figure VII.1 Code of the file header exporting code in C++

To display the previously generated point cloud three very different solutions were found:

- OpenGL
 - Simple and low level;
 - From OpenCV 2.4.5+ (Early 2013) using OpenGL was deprecated making it very difficult to compile because of the automatically generated error;
- PCL – Point Cloud Library
 - Very powerful library to display and process point clouds.
 - Unfortunately the binaries for this platform (Windows and MVS 2012) are not available, only up to MVS 2010 is supported.
 - Extensive research and tests proved that this library cannot be compiled under MVS2012.
- Third party software
 - No effort was made to directly connect two softwares, instead a more direct approach was taken. Simply open the exported file into a dedicated software of the area.
 - Several softwares are available with the capability to read and display the generated point cloud.
 - Mesh Lab[34] was the chosen software, because it is:
 - Very easy to use;
 - Open Source;
 - Can generate a mesh from the point cloud.

VIII. CONCLUSION AND FUTURE WORK

This work was intended to attest the current state of development in the field of reconstruction through photographs. The first step was to verify the development of the current mathematical model that substantiates the transformation from the real world projections to the projective plane. The mathematical models are well defined and the process of how images are captured and what transformations they suffer has been approached repeatedly in the past years.

With the mathematical model defined the next stage was to attest the algorithms that implement these models and the problem found with these algorithms is still in how to make this an automated process, limiting the need of user interaction with the software.

The process is not fully automated, due to the major issue found in both the available software and in the libraries. This issue was not in the transformation per say - the mathematical model applied - but in the problem of finding the same points across multiple images. If the same point is not found across the several images it is very hard to detect the movement that has occurred and the model is not generated. The difference between this was clearly shown with Autodesk Catch 123D and with Agisoft PhotoScan & Agisoft StereoScan, where one was able to provide far better results due to the user introduction of the same points across the several images.

The previously described issue was also found with the libraries, where even the camera calibration pattern was sometimes not found. This problem arises due to the differences of focus and of light across the images. In the real world rarely an object is lighted from all sides with the same intensity, so this is a problem to take into account.

With high quality images this effect can be reduced but the computation time also increases a lot.

For small scale and well defined problems the available software can supply the necessary models, not requiring any need to develop any special software. This includes small objects about the size of the helmet used to test the applications. And these models can be printed and utilized as almost identical models, with a few modelling tools using the rep rap 3D printer.

For extremely large reconstructions, such as buildings, the open source software Bundler can be used, but it's still impractical because it requires a lot of processing power and with a regular computer the computation time is illogical.

The available libraries provide a very good start in creating a reconstruction software, by laying all the basis in image treatment, calibration and, rectification, however they are not enough to produce a full reconstruction software without a very specialized team in the area of 3D reconstruction. They allow simple reproduction of a side of an object, with its built in method, but not much beyond it.

As future work, this project implementation can be extended to create a complete reconstruction software. All the necessary pieces have been implemented and deployed within the projects code and the library chosen has all the basic functions needed, the major issue now lies with computational power.

INDEX

3

3D printer, 63

A

Agisoft PhotoScan, 38, 40
Arduino duemilanove, 35
Autodesk Catch 123D, 38

B

Blur, 30
BoofCV, 57

C

Correspondence, 24

D

disparity map, 24

E

EmguCV, 54, 61
Epipolar geometry, 18
Essential Matrix, 20
Exif tag, 22
extrinsic parameters, 60

F

Fundamental Matrix, 19

H

Homogeneous Coordinate system, 11
Homogeneous Coordinates, 12

I

Image compression, 30
intrinsic parameters, 60

K

keychain 808 #16, 31, 37

M

Matlab, 60

N

NetBeans, 57, 58
Noise, 30

O

OpenCV, 48, 61

P

Photo to 3D, 38, 42
Pinhole Camera model, 9
Projective Geometry, 11
Prussa Mendel, 64

R

Radial distortion, 15
Real World Camera, 13
Rectification, 24
reflex cameras, 33
RepRap, 63
Reprojection, 24

S

SimpleCV, 59
Stereo Vision, 24

T

Tangential Distortion, 15
transistor, 35

U

Undistortion, 24

V

Vi3Dim, 38, 43

REFERENCES

- [1] J. Malik, "Computer Vision: The Fundamentals," 2012. [Online]. Available: <https://www.coursera.org/instructor/~626>. [Accessed: 25-Sep-2013].
- [2] R. Duraiswami, "CMSC: 426 Computer Vision -Image formation -Lecture 2." Maryland, 2005.
- [3] "Pinhole camera model." [Online]. Available: <http://rriai.org.ru/illustr/ai5-704.jpg>. [Accessed: 31-Jul-2013].
- [4] S. Birch, "An Introduction to Projective Geometry for computer vision - The Projective Plane." Palo Alto, pp. 1-22, 1998.
- [5] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd Editio. Cambridge University Press, 2003, p. 655.
- [6] G. Bradski and A. Kaehler, *Learning OpenCV*, First. O'Reilly Media, 2008.
- [7] S. N. Sinha and M. Pollefeys, "Multi-view reconstruction using photo-consistency and exact silhouette constraints: a maximum-flow formulation," *Tenth IEEE International Conference on Computer Vision ICCV05 Volume 1*, vol. 1, pp. 349-356 Vol. 1, 2005.
- [8] M. Wande, "Jhead." [Online]. Available: <http://www.sentex.net/~mwandel/jhead/>. [Accessed: 25-Jun-2013].
- [9] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, "Building Rome in a day," *2009 IEEE 12th International Conference on Computer Vision*, pp. 72-79, Sep. 2009.
- [10] A. Berele and J. Goldman, *Geometry: Theorems and Constructions*. Pearson, 2001, p. 224.
- [11] N. Paragios, Y. Chen, and O. Faugeras, *Handbook of Mathematical Models in Computer Vision*, First. Springer, 2006.
- [12] F. Hollsten, "The effect of image quality on the reconstruction of 3D geometry from photographs," Aalto University - School of Science, 2013.
- [13] P. A. (Kendall P. N. Levine, "Blooming control for charge coupled imager," 3,931,4651976.
- [14] S. Agarwal, Y. Furukawa, N. Snavely, B. Curless, and S. M. Seitz, "RECONSTRUCTING ROME," *ieee*, vol. 43, no. 6, pp. 40 - 47, 2010.

References

- [15] "Arduino," 2013. [Online]. Available: <http://www.arduino.cc/>. [Accessed: 13-Aug-2013].
- [16] Noah Snaveley, "Bundler." [Online]. Available: <http://www.cs.cornell.edu/~snaveley/bundler/>. [Accessed: 01-Jul-2013].
- [17] Intel, "IPP - Intel Integrated Performance Primitives." [Online]. Available: <http://software.intel.com/en-us/intel-ipp>.
- [18] "Glut." [Online]. Available: <http://user.xmission.com/~nate/glut.html>.
- [19] "BoofCV." [Online]. Available: http://boofcv.org/index.php?title=Main_Page.
- [20] P. Abeles, "BoofCV Applets," 2013. [Online]. Available: http://boofcv.org/index.php?title=List_of_Applets. [Accessed: 22-Aug-2013].
- [21] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab." [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html. [Accessed: 05-Sep-2013].
- [22] J. Russell, "RAMPS 1.4," 2013. [Online]. Available: http://reprap.org/wiki/RAMPS_1.4. [Accessed: 13-Aug-2013].
- [23] J. Russell, "RAMPS." [Online]. Available: http://reprap.org/wiki/Arduino_Mega_Pololu_Shield. [Accessed: 13-Aug-2013].
- [24] R. dev Team, "Heated Bed," 2013. [Online]. Available: http://reprap.org/wiki/Heated_Bed. [Accessed: 13-Aug-2013].
- [25] Kliment, "Sprinter Firmware," 2013. [Online]. Available: <http://reprap.org/wiki/Sprinter>. [Accessed: 13-Aug-2013].
- [26] "Corner Sub Pixel," 2013. [Online]. Available: http://docs.opencv.org/modules/imgproc/doc/feature_detection.html#cornersubpix. [Accessed: 15-Jul-2013].
- [27] "Stereo Calibrate (OpenCV function)," 2013. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=calibratecamera#stereocalibrate. [Accessed: 15-Jul-2013].
- [28] "Stereo Rectify," 2013. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#stereorectify. [Accessed: 15-Jul-2013].
- [29] "Stereo SGBM," 2013. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=sgbm#stereosgbm. [Accessed: 16-Jul-2013].
- [30] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 2, pp. 328–341, Feb. 2008.
- [31] "Stereo BM," 2013. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=sgbm#stereobm. [Accessed: 17-Jul-2013].

- [32] "Stereo GC," 2013. [Online]. Available: http://opencv.willowgarage.com/documentation/python/camera_calibration_and_3d_reconstruction.html#findstereocorrespondencegc. [Accessed: 17-Jul-2013].
- [33] "Reproject Image to 3D," 2013. [Online]. Available: http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=sgbm#reprojectimageto3d. [Accessed: 17-Jul-2013].
- [34] "Mesh Lab," 2013. [Online]. Available: <http://meshlab.sourceforge.net/>. [Accessed: 17-Jul-2013].