Camera calibration and 3-D reconstruction

Julien FAUCHER

Project supervisor
Manuel BOISSENIN

Project coordinator
Bala AMAVASAI

Head of Lab
Jon R. TRAVIS

June 30, 2006

Contents

List of Figures

Acknowledgments

First of all I would like to thank Dr **Jon R. Travis** and Dr **Bala Amavasai** for giving me the opportunity and all the facilities to carry out this project to a succeful completion.

I would like to thank Mr **Manuel Boissenin** who supervised me for his commitment with me and his numerous pieces of advice to help me passing through the problems I encountered.

Mr **Arul N. Selvan** helped me on some parts of my project, as well as providing me some useful ideas to go further.

Thanks too to Mr **Kim Chuan Lim** for his technical help all along the year, and for the conception of the platform.

I am deeply in debt to Mr **Jan Wedekind** for his programming support and his patience in debugging some parts of my code.

Part I

Introduction

Chapter 1

Motivation

Computer Vision is the study and application of methods which allow computers to extract useful information from image content. In this domain, 3-D reconstruction is a major topic.

Cheap techniques to acquire 3-D representation of objects are desirable. For instance 3-D from photogrammetry (or stereoscopy) can do a 3-D reconstruction of an object using only camera(s), but the main problem of these techniques is that patches of uniform colours cannot be put in correspondence, and so precise geometric information of these regions cannot be extracted.

There are also other methods using ultrasounds or laser, but at the expense of sophisticated techniques that have the inconvenience to be both expensive and long to be mastered. Thus, to be accessible the experience and the learning of these techniques have to be embedded in reusable software components.

The large majority of 3-D reconstruction techniques use a camera and need common algorithmic components which can be grouped under the name of *Camera Calibration*. Camera calibration is a widely and well understood set of techniques that allows to get the intrinsic configuration of the camera. However there is not yet a satisfying - for our purpose - implementation of the technique.

In many ways, it can be considered that this work has already been carried out with several implementations. Indeed, there is a good software implementation of the camera calibration for *Matlab*, but Matlab is unfortunately a commercial software. There is also another free implementation in *OpenCV* [10] but the code is not objected oriented, which make the learning cost to understand the library quite high and the possibility to improve the work quite low. An alternative library is *Gandalf* [11], in which camera calibra-

tion source code is easier to understand, but which uses the calculus facilities of Gandalf which are different from those of MIMAS [9]. For these reasons it was decided that we implement our own camera calibration algorithm which is needed in MIMAS [9].

In spite of the knowledge available on the subject, the steps involved to implement the technique are quite numerous and required good programming basis. It took five months of serious work to be able to reach the stage of being able to accurately calibrate a camera. This work has been made in such a way that it is easily re-usable by the community and it should take now only a few hours to be able to create a personalised software to calibrate a camera using the developed library.

To demonstrate and validate the implementation a set up to recover the 3-D geometry of an object using a laser line has also been developed. There are already a few projects for 3-D reconstruction with a laser line (such as COPOS [18]) but their approach is not as flexible as ours and the precision is not as good, since their method depends for instance on how well you tune the position of your laser.

In our method, the 3-D reconstruction is based on a laser line, projected on the object of interest. The scanning of the object can either be done by a rotating or a translating relatively to the laser plane to get the 3-D points.

The developed project is available on-line, is open-source, and is known as the **Bright Project** [1] (Bright reconstruction with light).

The possible applications of the developed method are numerous :

- Museum : keep a digital copy of an art work
- Entertainment : 3-D model creations for video games
- Medicine : 3-D reconstruction of an anatomy
- Engineering : creation of prototypes
- Machine Vision : 3-D model creations for tracking

Chapter 2

Context, background

2.1 The MMVL lab

The Microsystems and Machine Vision Laboratory (MMVL) [2] is a research group within the Materials and Engineering Research Institute (MERI) at Sheffield Hallam University, UK.

The main research activities involve the design, development and implementation of machine vision techniques targeted at a variety of *real-time* and *non real-time* applications which include micro-robotic systems, biological applications, micro-manipulation and microscope imaging.

2.2 The MIMAS library

MIMAS [9] was originally conceived as a platform for *real-time machine vision research*. Its aim was and still is to reduce the turnaround time of new research into the application workspace. It is written in C++ and is released in source code form subject to the GNU Lesser General Public License (LGPL). Various optimization schemes (POSIX threads) are implemented in order to achieve the real-time objectives.

MIMAS was used to build a number of vision systems including two European Union sponsored projects, namely *MINIMAN* (completed in 2002) and *MiCRoN* (completed in 2005). MIMAS is also being used to build a number of customised vision solutions for academia and industry.

A partial list of algorithms and classes that are part of the vision toolkit are as follows :

- Localisation of objects using colours
- Variety of tracking methods

- Comprehensive matrix library with linear algebra algorithms
- Image capture with Video4Linux
- Low-level image processing algorithms
- edge and corner detection
- etc...

Chapter 3

Equipments

3.1 Hardware equipment

Following are the equipments used during the project, that was given to me at the beginning of the project, or ordered to be built in the university workshop.

- Desktop computer running Ubuntu GNU/Linux
- Camera : webcam Logitech 8K89 ITE (resolution : $640 * 480$)
- Laser line (class 2). Ref : UHL5-20G-650-45.
  Power : 20 mW, laser wavelength : 690 nm.
- Mechanical support and holder to hold the camera and laser.
- Rotating platform to rotate the object to be scanned.

3.2 Software

Software used in the development of the vision system include :

- languages:
  - C, used by OpenCV [10] and Gandalf [11].
  - C++, the object oriented facilities and performance of this language make it a good choice.
  - Fortran, used by LAPACK and MINPACK libraries.
  - Matlab for the camera calibration reference program.

  - Perl a scripting language, essentially for batch jobs.
  - For its documentation quality : LaTeX

- libraries:
  - Standard Template Library (STL) library
  - BOOST [14] portable C++ source libraries, and especially uBLAS for matrix handling.
  - LAPACK [12] and MINPACK [13] for matrix algebra
  - MIMAS [9] a vision library which also provides an interface to access earlier mentioned libraries.
- Compiler sets - GNU compiler collection.
- OpenOffice.org for presentations.

Chapter 4

Work accomplished

This chapter presents the different stages which led to the final project. The purpose of the chapter is to explain how I managed to find a subject fitting my needs and abilities, and how I decomposed the work.

4.1 Subject research

My objective since my arrival was to develop a 3-D reconstruction software, nevertheless no particular method was given.

So my first approach was to collect as much information as I could on the subject. I read some chapters of an introduction book [3] on machine vision to understand the subject. I also read some papers about 3-D scanning methods, as well as papers explaining different methods to reconstruct the surface of an object with some sample 3-D points.

I also tried some software implementation such as *PowerCrust* [17] to reconstruct an object with 3-D points. This software implements an algorithm to get the crust (surface) of an object given 3-D points sampled from its surface. The tests were conclusive; if the 3-D points are precise enough, the surface is given accurately.

4.2 Test of different methods

I needed to find a way to scan the 3-D points from the object, knowing that from these points I could retrieve the 3-D shape with softwares like *PowerCrust*.

4.2.1 COPOS

The first solution I found was the one which guided me to the follow-up of my project.

The COPOS [18] project is a French project, initiated by Ronan Billon, whose purpose was to create the cheapest 3-D laser scanner possible.

So I decided to build my own scanner following the recommendations of the website. Dr Bala Amavasai found me a cheap laser line (10 pounds), and Kim Chuan helped me to adapt an old rotating platform (from an abandoned project) to one fitting my needs. I installed everything along with the camera in a large cardboard box to protect from the daylight, and scanned a few objects.

The COPOS software provided from the scan a set of points, to which can be applied the Power Crust algorithm to reconstruct the surface.

The program was working reasonably well, but the result was not very satisfactory. The laser line detection on the object was not good enough (maybe because the line projected by my laser wasn't strong enough), and the global precision was very limited : the COPOS software requires the user to provide some distances (between the camera and the platform for instance) which are hard to measure accurately.

This is why I kept the COPOS project in mind while I was going further, but decided to change some parts of it.

4.2.2 ArcheoViz

I also tried a project name ArcheoViz [19]. This project is very different from COPOS, because it's working only with 2 pictures of the object. The method used is the photogrammetry (in its simplest form, because only using 2 images are used). The main advantage is that it can handle large object, such as small monuments (the project was designed for archaeological purposes). The problem is that the software needs information about the object (location of a few 3D points) to process other points and reconstruct the shape.

So the software was not suitable for my needs, but it helped me to discover another way to do 3-D reconstruction.

4.3 Camera calibration

Once I had chosen my method to scan the object, I understood that implementing a camera calibration software was necessary, because it was needed

for my method, and it was not available in MIMAS, although it was in the other computer vision libraries (OpenCV [10] and Gandalf [11]). So implementing it was both useful for my project and the follow-up of the MIMAS library, because it could be used for some other projects.

The implementation of camera calibration was the largest part of my project, because it took me five months to implement all the steps, from the algorithm to the GUI (Graphical User Interface).

4.4 3-D reconstruction

Once the camera calibration method was implemented, I could get back to my 3-D reconstruction project. I had to find a way to make my 3-D reconstruction more accurate than COPOS.

The COPOS solution requires the user to provide the distances and angles between the camera, the laser and the platform, which, unless you have an accurate solution to measure them, is bound to affect the precision.

The basic idea of my solution is to calibrate the rotating platform and the laser line according to their *relative position* to the camera, to avoid imprecise measurement.

To implement the above method I designed a stand which could hold both the webcam and the laser line, to make sure they are keep at the same relative position to one another.

This stand was built by the university workshop, and consists of a stem holding 2 rotating arms, one for the camera, and one for the laser line (see fig 4.1 page 15).

I also needed a rotating platform, for which Kim Chuan Lim helped me with the design and fabrication. It is made with up of a stepper motor to get a good precision on the angle, and a controller to be able to activate the motor with a computer, using a serial port.

Based on Manuel Boissenin's idea, I designed a software to calibrate the laser line (regarding to its relative position to the camera).

Once the laser was calibrated, the next step was to compute the 3-D points for a single projection of the laser line on an object. The laser line can be decomposed into points, and then the 3-D position of these points can be retrieved.

Then, if the rotating platform is calibrated (that is its relative position to the camera is known), it was possible to do the previous operation several times by rotating the platform. With the resulting 3-D points with the

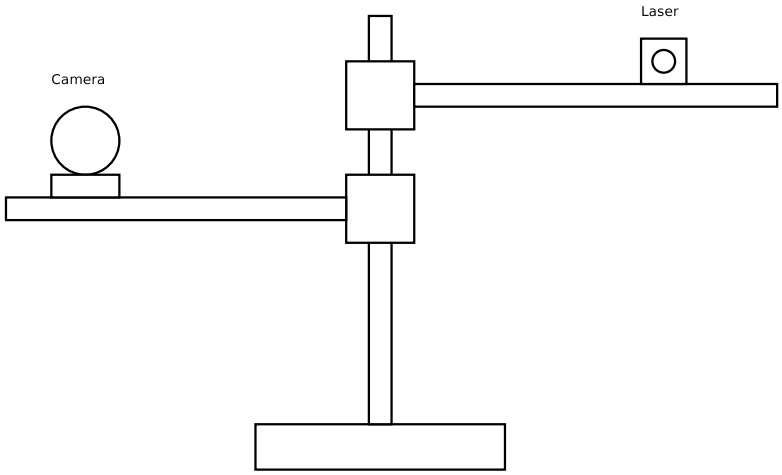

Figure 4.1: The laser and camera stand

corresponding angle of the platform, one can compute a full 3-D scan of the object.

Part II

Camera Calibration

Chapter 5

Introduction

In order to understand the purpose of camera calibration, it will be helpful to understand how the images are represented by the camera.

A practical model is to consider the camera as a pinhole (cf fig. 5.1)

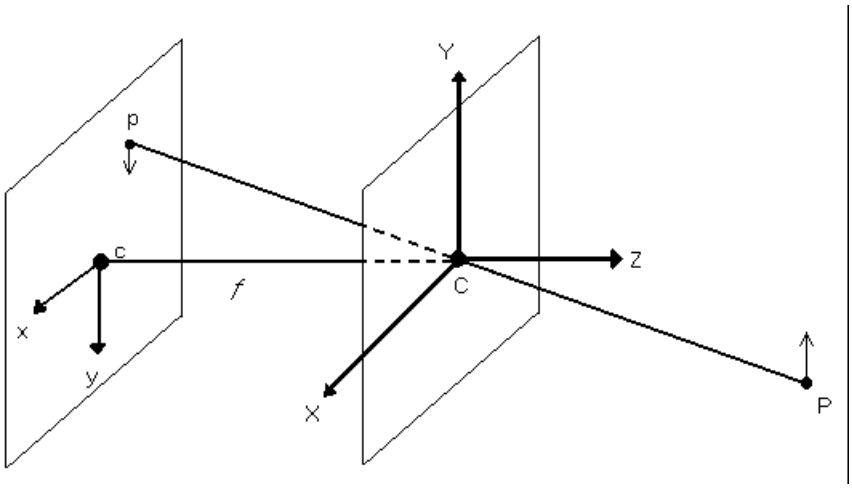

Figure 5.1: The pinhole camera imaging model

In this model, the camera maps 3D points to a 2D image. This can be modelled by a 3*4 projection matrix.

It has been proved [3] that the projection of a point in space with homogeneous coordinates X onto the image plane having the homogeneous coordinates x satisfy the equation :

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & fs_\theta & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.1}$$

which can also be written as :

$$\lambda x = K \Pi_0 g X \tag{5.2}$$

where

- f is the focal length of the camera
- (s_x, s_y) is the size of a pixel
- s_θ is the skew factor
- (o_x, o_y) is the optical centre
- K is the intrinsic matrix
- λ is a scalar factor.
- Π_0 is the canonical projection matrix.
- $g = (R, T)$ is a transformation composed of a rotation and a translation.

The camera calibration process consists of finding experimentally the matrix K, using some pictures captured by the camera.

Chapter 6

Algorithm

The basic idea of calibration is to take a picture of an object with an accurately known shape, and to use the properties of both the object and the image captured to compute the intrinsic matrix.

There are quite a lot of different algorithms to do the camera calibration. Some of them use only one picture, but with a lot of information on the scene taken. For instance [4] gives an algorithm to calibrate a camera with only one picture of a chessboard-faced cube. The main difficulty is to construct accurately this kind of calibration object.

The chosen algorithm is a quite recent one (December 2, 1998) [6], and one of the most accurate as well. This is the one used by the OpenCV [10] library, and by the camera calibration program [16] that is probably the most cited implementation of the algorithm by the community.

The algorithm uses a chessboard to calibrate. The advantage of this object is that it is simple to fabricate (one may just print a chessboard pattern and glue it on a plane surface such as a book). The algorithm is explained in [4] and in details in [6], and an overview of the algorithm is given in this chapter.

Since the world reference frame can be chosen freely, we choose it aligned with the chessboard, so that points on it have coordinates of the special form $X = [X, Y, 0, 1]^T$. The center of the world frame must be on the board, and the Z-axis of the world frame is the normal vector.

Then the projection equation (5.1) can be simplified to :

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = K[r_1, r_2, T] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{6.1}$$

where $r_1, r_2 \in \mathbb{R}^3$ are the first and the second column of the rotation matrix R.

Notice that the matrix

$$H = K[r_1, r_2, T] \in \mathbb{R}^{3\times 3} \tag{6.2}$$

is an homography between the chessboard plane and the image plane. It's possible to compute H, if we know at least 4 points in the world reference frame, and their projections. We can use the 4 extreme corners of the chessboard for that.

With the 4 corners, a system of equations is defined, which we want to find an approximated solution using an error-minimising algorithm such as a least-square function. More corners of the chessboard could be used to refine the solution, but experience has shown that using only the 4 extreme corners and all the corners of the chessboard leads to a very close result, so although the general version with n corners is implemented, the 4 corner version is used for a faster result. The solution is found using a least-square algorithm based on the SVD (Singular Values Decomposition) detailed in [4].

From 6.2, we have $[h_1, h_2] \sim K[r_1, r_2]$, which is equivalent to :

$$K^{-1}[h_1, h_2] \sim [r_1, r_2] \tag{6.3}$$

Since r_1 and r_2 are orthonormal vectors, we obtain 2 equations that the calibration matrix K has to satisfy :

$$\begin{cases} h_1^T K^{-T} K^{-1} h_2 = 0 \\ h_1^T K^{-T} K^{-1} h_1 = h_2^T K^{-T} K^{-1} h_2 \end{cases} \tag{6.4}$$

Using $S = K^{-T}K^{-1} \in \mathbb{R}^{3\times 3}$, we can take several images and evaluate the equations 6.4 for each image. As there are five unknown in K, and each image provides two equations, a minimum of three images is needed.

A solution of the system can be approximated in the least-square sense, and then we can retrieve K from S using the Cholesky factorisation.

What we get is just a first approximation of the intrinsic matrix K, because the previous algorithm doesn't consider the distortion of the camera or the noise on the picture.

Zhang explains in [6] how to optimise the result, using the Levenberg-Marquardt algorithm, and how to introduce the distortion.

The solution is obtained through minimising an algebraic distance which is not physically meaningful. We can refine it with a maximum likelihood

algorithm. We are given n images of a model plane and there are m points on the model plane. If we consider that the noise is independent and identically distributed, the maximum likelihood estimate can be obtained by minimizing the following functional:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \|m_{ij} - \tilde{m}(K, R_i, t_i, M_j)\|^2 \tag{6.5}$$

where $\tilde{m}(K, R_i, t_i, M_j)$ is the projection of point M_j in image i, and m_{ij} the measured point on the image.

The rotation and translation R_i and t_i are the extrinsic parameters of each image. Those parameters can be approximated for each image using K and H, and then R_i must be refined to respect the mathematical form a rotation matrix.

The Levenberg-Marquardt algorithm to work needs a first approximation of the solution K, which we have computed above. At each iteration, it will modify slightly the values of K, and try to minimise the sum given by the equation 6.5.

In the equation 6.6, the distortion of the image is not considered. But it's also possible to find the first two parameters (k_1 and k_2) of the distortion using the above method, simply by using a first approximation of (k_1 and k_2) as 0, and using the parameters in the projection :

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \|m_{ij} - \tilde{m}(A, R_i, t_i, M_j, k_1, k_2)\|^2 \tag{6.6}$$

This time, the distortion is considered in the projection $\tilde{m}$. In the equation 6.6, there are 7 unknowns, hence a minimum of four images must be captured.

More information about the repercussion of k_1 and k_2 on the projection, and how to compute the projection taking into consideration k_1 and k_2 can be found in Zhang's paper [6].

Chapter 7

Implementation in MIMAS

7.1 Software design

The software for the camera calibration was done in C++, using as well as updating the MIMAS library.

The basic idea was to implement classes to do the camera calibration, independent to some choices (like the GUI, the chessboard size, etc...) and subsequently to create a GUI in QT4 [15] using the above classes and provide a full camera calibration tool.

7.2 Presentation of the classes

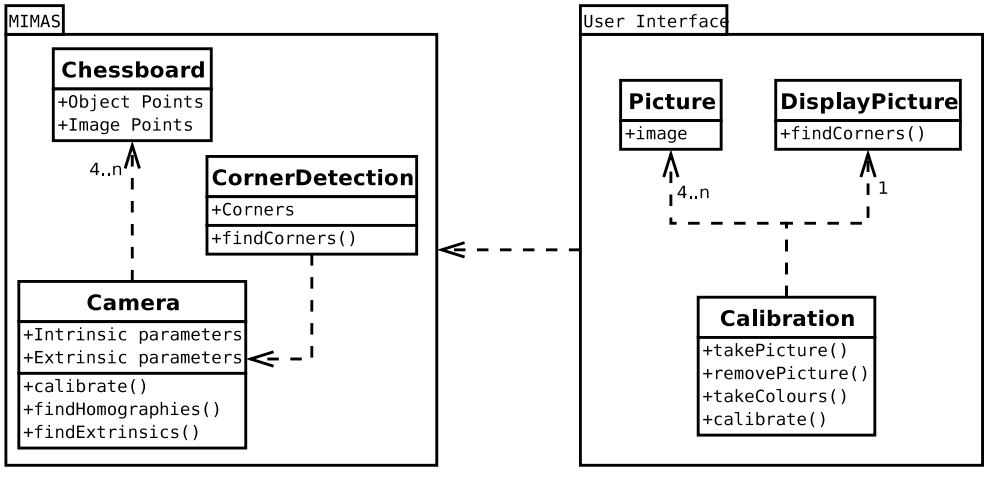

Figure 7.1: Class diagram

7.3 Extra functionality added to MIMAS

To implement the calibration algorithm, I needed to use some functions provided by MIMAS such as matrices, image manipulation, correlation algorithm etc... But MIMAS is not as comprehensive as OpenCV for instance, so some generic functions were missing. I implemented the following functions, and added them to the library.

7.3.1 Cholesky decomposition

For linear algebra computations, MIMAS provides a convenient set of wrappers functions for the open-source Fortran library LAPACK [12].

The Cholesky decomposition in LAPACK is the function `pptrf`. There was not wrap in MIMAS, so a wrapper function was written.

It is now available in the MIMAS library through the function :

```
template<typename T>
triangular_matrix< T > pptrf(symmetric_matrix< T > const &A);
```

7.3.2 Determinant of a matrix

As surprising as it may seem, the determinant function was not available in MIMAS. Since the first version of the calibration algorithm needed it, an implementation was done, using the LU factorisation which was already in wrapped in MIMAS from LAPACK.

The determinant is now available in the MIMAS library through the function :

```
template<typename T>
double determinant(matrix< T > const &M);
```

Note : a further improvement of the calibration algorithm allowed to bypass the use of this function.

7.3.3 Levenberg-Marquardt algorithm

As this algorithm is non-linear, it was not implemented in LAPACK, but is available in MINPACK [13], which is another Fortran library.

The Levenberg-Marquardt algorithm functions are `lmdif` (normal version) and `lmdif1` (version with a simplified calling sequence).

It is now available in the MIMAS library through the following functions having the same name.

```
lmdif_t::Vector lmdif( const lmdif_t::Vector &A,
                       void (*fnc)(int *m, int *n, double *x,
                                   double *fvec, int *iflag),
                       int nb,
                       int maxfev,
                       double tolerance = 1e-7,
                       int mode = 1,
                       int factor = 1,
                       double epsfcn = 0);

lmdif1_t::Vector lmdif1( const lmdif1_t::Vector &A,
                         void (*fnc)(int *m, int *n, double *x,
                                     double *fvec, int *iflag),
                         int nb,
                         double tolerance = 1e-7);
```

The main difficulty faced in implementing these functions it that I had to start from scratch, because there were no MINPACK wrapper function in MIMAS. Based on the implementation of the LAPACK wraps, wrapper functions and classes were provided for MINPACK. The detection of the MINPACK library is now included in the configuration file of MIMAS.

Chapter 8

The corner detection algorithm

8.1 Introduction

While the calibration algorithm needs to get the corners of a chessboard, implementing a good corner detection algorithm seemed to be an obvious condition to make a robust implementation of this algorithm.

In fact, the accuracy of the result of the calibration algorithm depends on the precision of the location of the corners. That's why the slightest imprecision on the location of the corner gives an imprecision on the calibration result.

A first approach would have been to let the user himself select the location of the corners, but the problem is that the human eye can't be as precise as the machine to make the choice. Where the eye will only see an area which seems to contain the corner, an algorithm will be able to choose a precise point in this area, even with a sub-pixel accuracy.

8.2 Choice of the algorithm

It would have been possible to implement an algorithm which needed no interaction with the user apart from taking the picture, but this algorithm would have needed a full scan of the picture to guess the position of the corners, and a post-treatment to remove unwanted corners. The resulting algorithm would have been very slow. But one may easily change my program to implement this algorithm.

That's why the algorithm I decided to choose needs the user to provide a first approximation of the four extreme corners of the chessboard, to avoid

useless computations, and to be sure there won't be any mismatch with another corner of the image.

8.2.1 The template based algorithm

There is an article [7] explaining a corner detection algorithm designed for chessboard based camera calibration algorithms. I took from this article not the full method, but only the main idea : using a cross template to detect the corners.

The cross template is like[1] (see fig. 8.1):

255	X	X	X	X	X	X	X	X	X	255
X	255	X	X	X	X	X	X	X	255	X
X	X	255	X	X	X	X	X	255	X	X
X	X	X	255	X	X	X	255	X	X	X
X	X	X	X	255	X	255	X	X	X	X
X	X	X	X	X	0	X	X	X	X	X
X	X	X	X	255	X	0	X	X	X	X
X	X	X	255	X	X	X	0	X	X	X
X	X	255	X	X	X	X	X	0	X	X
X	255	X	X	X	X	X	X	X	0	X
255	X	X	X	X	X	X	X	X	X	0

Figure 8.1: The black and white template, for the upper-left corner

The centre of the cross will correlate with the corner of the square. Because of the noise around the corner, the template must be big enough to correlate accurately. Furthermore, there are actually four templates, one for each type of corner (left or right, upper or lower)

The previous template is perfectly working on a black and white image, but when we take a picture of the chessboard, it will appear as an image in

[1] Remember that black is 0, white is 255, and X can be any color

levels of grey. And with the light, the black is not perfectly black, neither the white is.

This is why, in the final implementation, the user is requested to click in a black square and in a white area of the chessboard, to calibrate the corner detection.

Furthermore, as there is blur on the image taken by the camera, the template can not be used accurately with only 2 colours.

So I adapted the template to make it correlate with a generated image: a black square on a grey background, on which I applied a Gaussian blur filter, so I used this image to extract coefficients to apply on the template.

Finally, the final template is like (see fig. 8.2):

150	X	X	X	X	X	X	X	X	X	150
X	150	X	X	X	X	X	X	X	150	X
X	X	150	X	X	X	X	X	150	X	X
X	X	X	150	X	X	X	150	X	X	X
X	X	X	X	140	X	110	X	X	X	X
X	X	X	X	X	100	X	X	X	X	X
X	X	X	X	110	X	60	X	X	X	X
X	X	X	150	X	X	X	50	X	X	X
X	X	150	X	X	X	X	X	50	X	X
X	150	X	X	X	X	X	X	X	50	X
150	X	X	X	X	X	X	X	X	X	50

Figure 8.2: The final template, with white = 150 and black = 50

The main advantage of this template is that it can be applied even if the squares are inclined. Although the paper recommends to avoid angles over 30^o, experimentation has shown that the corner can be detected even if the square is rotated with an angle ranging from -40^o to 40^o.

The template is applied to all the pixels in a window around the first approximation (using a correlation based algorithm), and the best correla-

tion found, if superior to a given minimum correlation, gives the best pixel. Otherwise, the algorithm considers no corner is in the window.

8.2.2 The sub-pixel detection

The limitation of the previous algorithm is it will only find a corner with a pixel precision. If we want a robust calibration algorithm, we need a better precision.

Another paper [8] introduces the Blais and Rioux (among other algorithms, but this one - in its *fourth order* form - was presented as the most efficient).

In its original form, this algorithm is intended to work with the intensities of the pixel, and will find the peak intensity point with a sub-pixel precision.

Given a discrete representation of the intensities, the algorithm will be able to interpolate assuming the shape is parabolic around the peak, and find it.

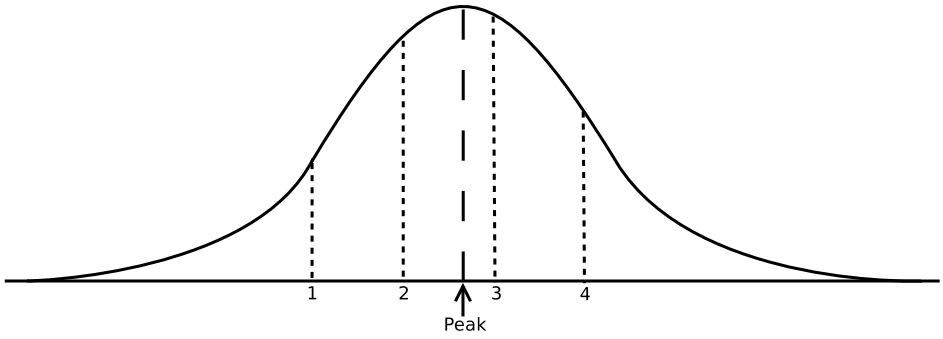

Figure 8.3: The Blais and Rioux detector

The algorithm uses the values of the intensities around the detected pixel to approximate the peak. For instance, in Fig 8.3, the best approximation of the peak is the point 3, so using the values of the intensities at the points around, the detector will approximate the peak between 2 and 3.

8.2.3 The final algorithm

I decided to adapt both of the previous algorithms to make my own. The Blais and Rioux detector is intended to work with the intensity of the pixel, but I decided to use it with the values of the correlation numbers instead.

First, we need a first approximation of all the corners of the chessboard. As we have an approximation of the four extreme corners, we can compute the

homography between the chessboard and the image. We also know the size (in number of squares) of the chessboard pattern, so it is possible using the homography to compute an approximation of the position of all the corners of the chessboard.

So for each corner, the algorithm finds the type of the corner (e.g. upper-left), computes the correlation numbers of all the points in a window around the approximated point, finds the maximum, and using its neighbors applies the fourth-order Blais and Rioux detector (BR4) on the X and on the Y axes.

The algorithm is explained in details in Algorithm 1.

Algorithm 1 Corner detection algorithm

```
1: Get a first approximation of the corners, and compute the homography.
2: for all corners of the chessboard do
3:    Compute an approximation of the corner using the homography.
4:    for all pixels in a window around the approximated corner do
5:       According to the corner location, find the matching template.
6:       Correlate with the corner template.
7:       Save the best correlation.
8:    end for
9:    if the correlation is greater than the minimum required then
10:      Save the best corner.
11:   else
12:      No corner found. Exit.
13:   end if
14:   Apply the Blais and Rioux detector on the best corner.
15:   Save the corner in a matrix.
16: end for
17: return Matrix of the results.
```

Chapter 9

Tests and validation

9.1 Test of the corner detection algorithm

First estimation of the homography (with a non accurate approximation of the 4 corners) :

$$\begin{bmatrix} 0.399123 & -0.0141567 & 0.488672 \\ 0.0310757 & 0.377969 & 0.676622 \\ 8.06638e-05 & -2.39378e-05 & 0.00751802 \end{bmatrix} \tag{9.1}$$

Homography found with all the points (80) :

$$\begin{bmatrix} 0.403101 & -0.00685318 & 0.465341 \\ 0.0318304 & 0.372436 & 0.693644 \\ 8.20428e-05 & -7.91863e-06 & 0.00750119 \end{bmatrix} \tag{9.2}$$

Second estimation of the homography, with the accurate 4 corners detected by the corner detection algorithm :

$$\begin{bmatrix} 0.403051 & -0.00697422 & 0.46662 \\ 0.031288 & 0.373242 & 0.692403 \\ 8.08968e-05 & -9.34388e-06 & 0.00752612 \end{bmatrix} \tag{9.3}$$

Conclusion : the homography found with 4 corners is almost the same than the one found with 80 corners, so to avoid useless calculations, we will only do the calculation with 4 corners.

9.2 Test of the convergence of the algorithm

I tested my calibration algorithm with a large number of images, to test the convergence.

We can see on the diagrams that the algorithm doesn't converge very well, even though I tried with more than enough images (80). The problem probably comes from the least-square approximation, with is not resistant to outliers. See the chapter "Possible extensions"for more details.

Nevertheless, the result I get is good enough for my 3-D reconstruction project, because I just need a good approximation of the calibration matrix, not a perfect result.

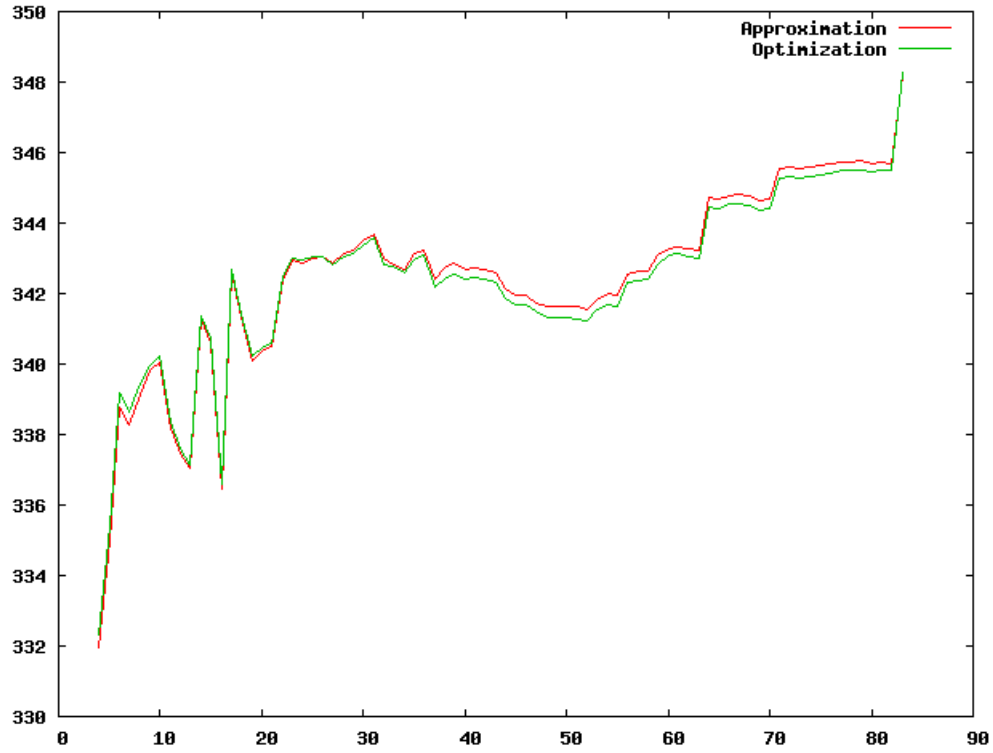

Figure 9.1: The ax parameter

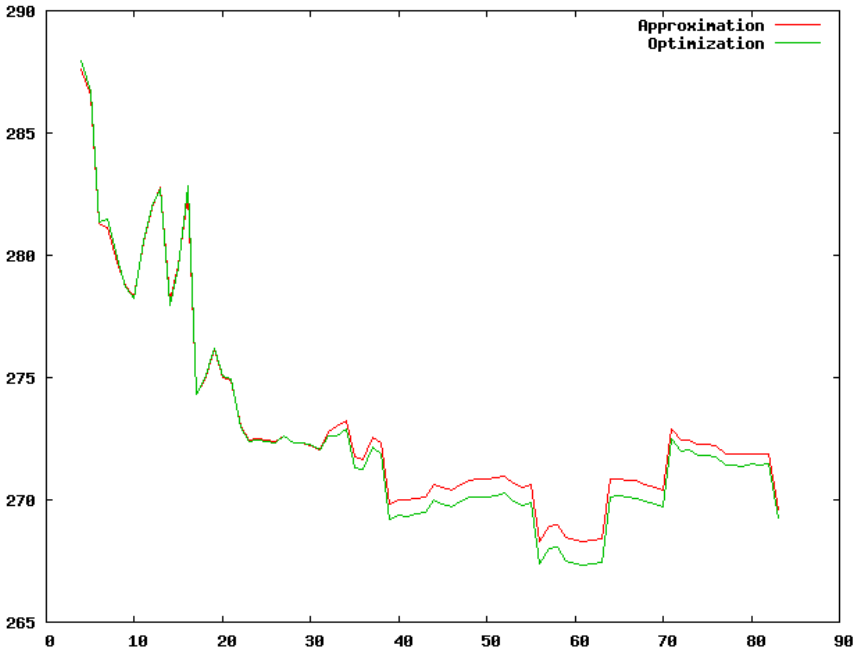

Figure 9.2: The ay parameter

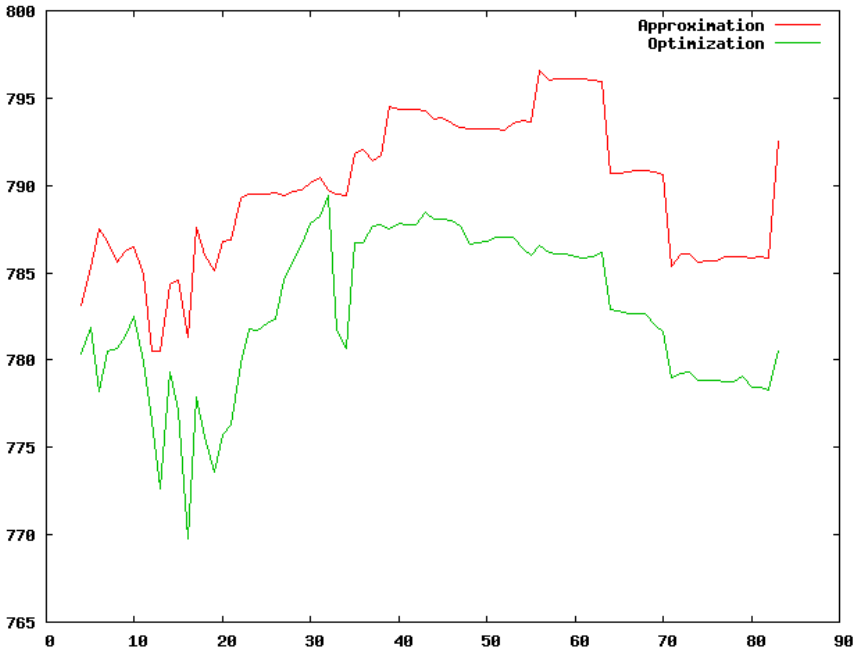

Figure 9.3: The fx parameter

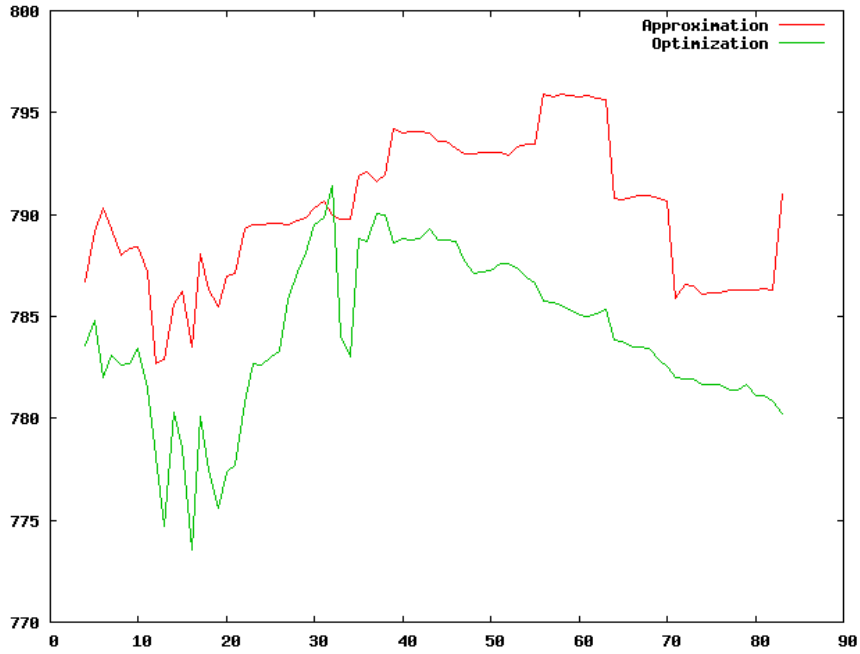

Figure 9.4: The fy parameter

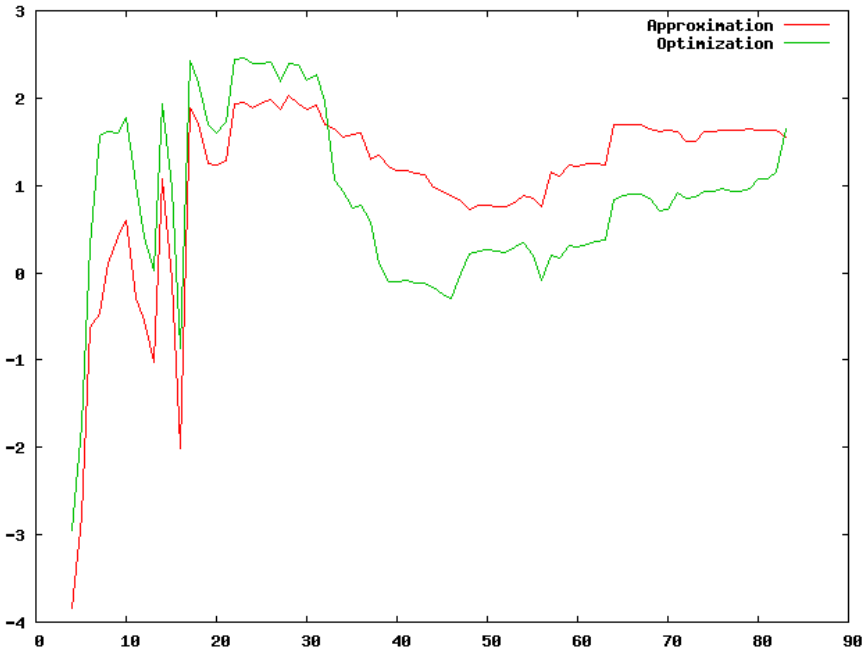

Figure 9.5: The skew parameter

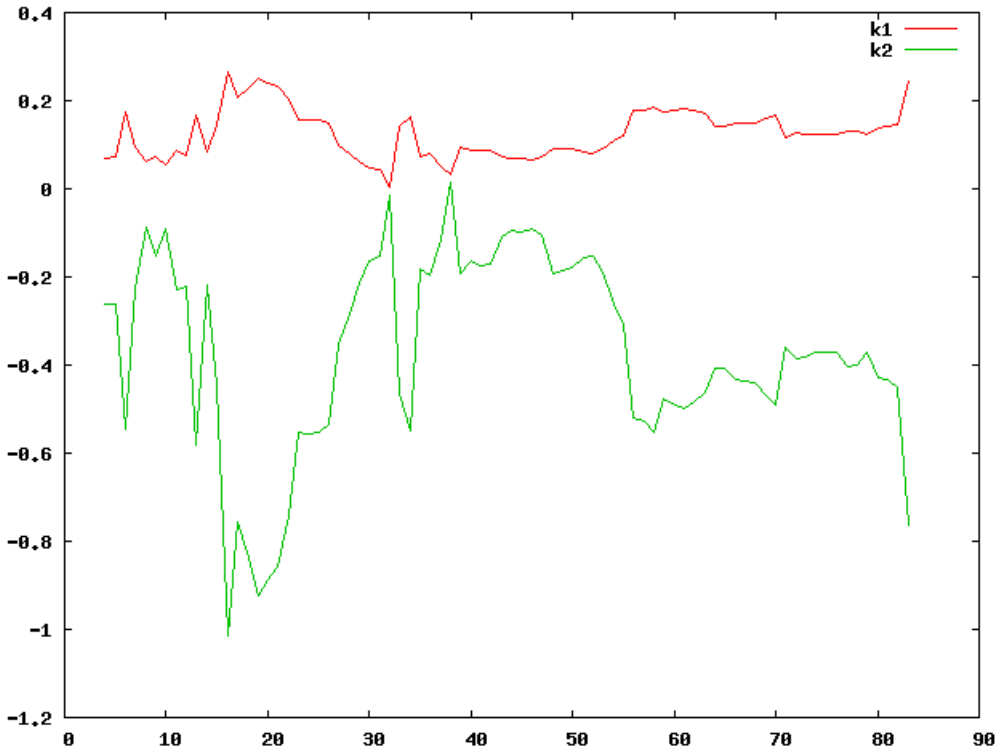

Figure 9.6: The distortion parameters

Part III

3D reconstruction

Introduction

Calibrating the camera was a first step to start the 3-D reconstruction, but it was also the biggest work to be done. We will see that once the camera is calibrated, using both the calibration result and the source code of the calibration software leads to a fast way to get the 3-D reconstruction of objects.

The method presented here requires only a laser line, a camera (we use a webcam, which is enough), a support for the camera and the laser line and either a rotating or translating system, to move the object to scan.

This method is more flexible than COPOS [18] as it doesn't require the user to provide the relative positions of the laser line and the camera, with the lack of precision it implies. Once the camera and the laser are fixed, the relative positions are found by calibration.

Chapter 10

Laser calibration

This section explains the method designed to calibrate the laser line with only two pictures. In the following, calibrating the laser means finding the equation of the laser plane in a given frame.

10.1 Theory

In this 3-D reconstruction method, we are using a laser line to scan the object. If the laser is fixed, the laser line defines a *plane* in the 3-D space.

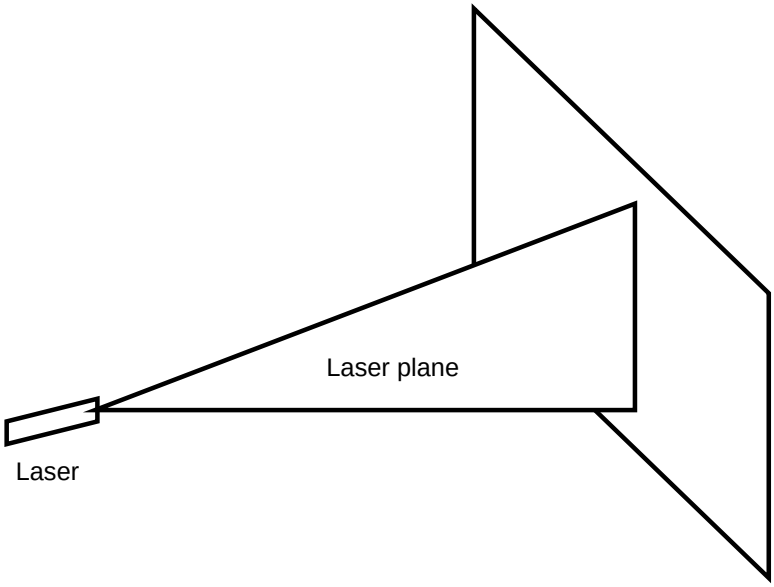

Figure 10.1: The laser plane

The equation of this plane is needed. The *world reference frame* is chosen to be the camera frame (the frame which centre point is the centre of the camera, and the z axis is normal to the image plane).

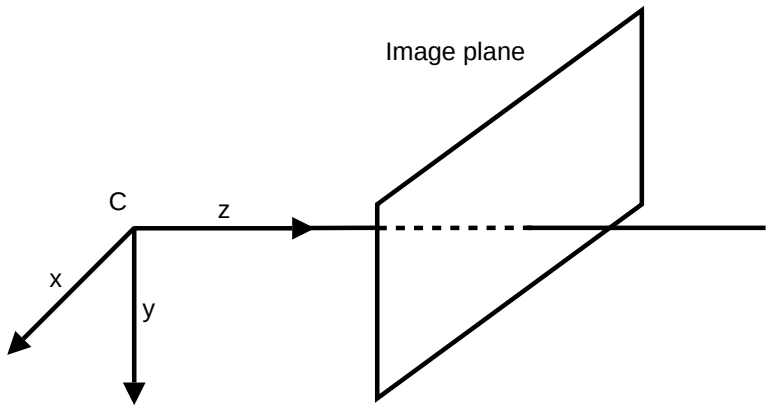

Figure 10.2: The camera frame

10.2 Overview of the method

In theory, 3 non-collinear points define a plane. Then it is possible to find the plane equation with those points using them to define 2 vectors, and to make the cross product in order to get a normal vector to the plane. A point of this plane and this normal vector define the plane equation.

We use the chessboard we have already been using to calibrate the camera. The main idea, once the camera and the laser line are both fixed, is to take 2 pictures of the laser line on the chessboard. Then using the intrinsic matrix (found when calibrating the camera), and for each picture, finding the extrinsic matrix lead to the ability to find the equation of the laser line in the world reference frame.

From those two lines, we will extract 3 non-collinear points, and thus find the laser equation.

10.3 Detailed description

For each picture, we find the homography using the same technique as for the calibration : providing the four extreme corners of the chessboard, and then computing the homography with the same method.

For the laser line, we use another method. The laser line can be detected on the chessboard by an appropriate filter. Using the homography the equation of the line in the chessboard frame can be found, and then using the extrinsic parameters the equation can be transformed into the camera frame.

The laser line is represented in the image by 2 points. If H is the homography matrix, X the homogeneous 2-D coordinates of a point in the chessboard frame, and x the homogeneous 2-D coordinates in pixels in the image frame, we have :

$$x = HX$$

Thus, as H is invertible, we have :

$$X = H^{-1}x \tag{10.1}$$

If we note X_C the homogeneous 3-D coordinates of a point in the camera frame and P the extrinsic matrix for homogeneous coordinates :

$$P = \begin{bmatrix} R & & & T \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

we have the relation :

$$X_C = PX \tag{10.2}$$

Thus, combining the equations 10.1 and 10.2, we can transform the points of the laser line in the image frame to the camera frame.

$$X_C = PH^{-1}x \tag{10.3}$$

To get the equation of the plane, 2 pictures of the laser line on the chessboard are taken. For these 2 pictures, the homography as well as the extrinsic parameters is computed (getting the extrinsic parameters is possible, because we know the camera calibration matrix. The method is explained in Zhang's paper [6] and has already been used in the calibration process).

2 points belonging to the laser line are taken from the first picture, and only 1 from the second pictures. These 3 points u_1, u_2, u_3 belong to the laser plane, its equation can be derived.

Using 10.3, the coordinates of the 3 points are known in the world reference frame.

Taking for instance u_1 as a point and the cross product $(u_2-u_1)\times(u_3-u_1)$ as a normal vector defines the plane.

10.4 Implementation with MIMAS

10.4.1 Detection of the laser line

The laser line is detected in 2 steps. The first one is to do a threshold. The laser line is red on the edges, but in its centre, where the intensity is the highest, the line is white. That's why a threshold on the high intensities of the image will provide a set of points belonging to the laser line.

I implemented in MIMAS a threshold function with 2 levels. It is now available using the functions :

```
template< typename T >
image< T > bilevel_double( const image< T > &im, T min, T max,
                           T val1, T val2 );

template< typename T >
image< T > &bilevel_doubleIt( image< T > &image, T min, T max,
                              T val1, T val2 );
```

These functions map the pixel x of *image* to *max* if $val1 < x < val2$, or to *min* otherwise.

The second step to find the laser line is to use the *Hough transform*. This transformation (well-known in the machine vision community) can, for a given set of points, identify a line passing in the neighbourhood of the maximum of these points.

This report won't develop this transformation, because it was already implemented in MIMAS.

10.4.2 Equipment

To get an accurate calibration of the laser line (meaning to get the laser line in a stable position), a stand is needed. There could be many ways of designing this stand. The one I chose is presented in Fig 4.1 page 15.

It is composed of a stem, and 2 rotating arms. One has a tray to hold the webcam, the other has a hole to set the laser in. The laser stand can slide on the arm, in order to reach the object according to its size and position. The arms can rotate about the stem, and slide up and down before being tightened to the stem with a screw.

Chapter 11

The 3-D reconstruction

Here is explained the method to get the 3-D line on an object, and to belonging to this line.

11.1 Equipment

The stand to hold the camera and the laser line must be used, as well as a rotating platform.[1]

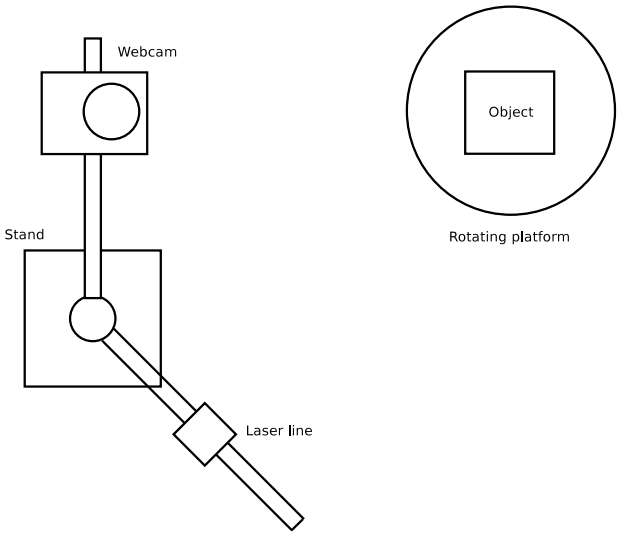

Figure 11.1: The full equipment

[1]The method could probably be adapted to a translating platform

11.2 Method of 3-D reconstruction

As Figure 11.2 shows, the 3-D point can be retrieved from the intersection between the laser plane and the line going through the camera centre and an enlighten pixel.

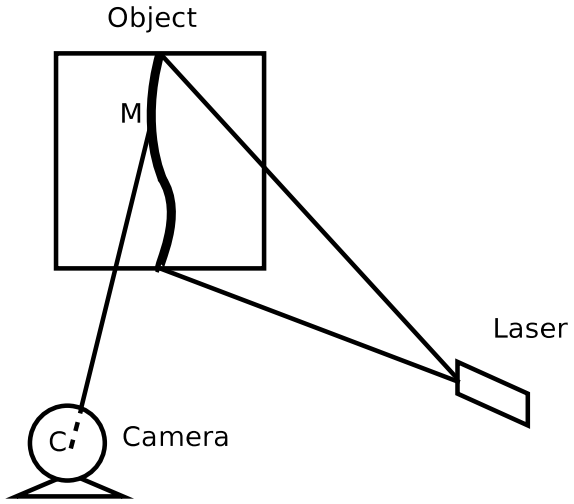

Figure 11.2: The 3-D reconstruction

11.3 Detection of the line and computation of the 3-D points

Here the line detection is presented as well as the computation of the 3-D location of the points

11.3.1 Laser line detection

The laser line is projected on the object, and a picture is taken by the camera. The object is placed in a dark environment, and a threshold on high intensities is used to extract the laser line from the picture.

11.3.2 3-D points computation

The last step is, from the 2-D points of the laser on the object, to compute their 3-D location.

From the theory of camera projection, it is known that a 2-D point is the projection on the image plane of a 3-D point by the ray that goes through this point and the camera centre.

So if the equation of the ray passing through the camera centre and the 2-D point can be computed, the intersection between the ray and the laser plane gives the 3-D location of the point.

The equation of the ray can be computed using the following statement: in the general case, if we note $P^+ = P^T(PP^{-1})^T$ the pseudo-inverse of P, the point P^+x will project to the point x by the ray. This is due to the usual projection equation 11.1 :

$$x = PX = K[R|T]X \tag{11.1}$$

where x is the 2-D point in homogeneous coordinates in the image frame, P the camera projection matrix, and X the 3-D point in homogeneous coordinates in the world reference frame, *i.e.* the camera frame.

In our case, there is no frame change, so $P = K[I_3|0]$. As K is invertible, the point $K^{-1}x$ will project to x, because $KK^{-1}x = x$

Thus, the equation in the world reference frame of the line passing through the centre of the camera (which is the origin of the frame) and the point x is given by the vector along the line $v(v_1, v_2, v_3)$, and is the set of points $M(x_1, x_2, x_3)$ such as :

$$\begin{cases} x_1 = kv_1 \\ x_2 = kv_2 \\ x_3 = kv_3 \end{cases} , k \in \mathbb{R} \tag{11.2}$$

If the plane is given by the point $P(p_1, p_2, p_3)$ and its normal vector $n(n_1, n_2, n_3)$, the intersection of the plane and the line is given by :

$$k = \frac{n_1p_1 + n_2p_2 + n_3p_3}{n_1v_1 + n_2v_2 + n_3v_3} \tag{11.3}$$

Combining 11.2 and 11.3, we find the 3-D point coordinates.

11.4 Utilisation of the rotating platform

With the rotating platform, we can scan the whole object using the previous method on several pictures of the object, rotated each time.

11.4.1 Platform calibration

The platform needs to be calibrated, as well as the camera and the laser. In this case, as the platform is intended to rotate the object, the axis of rotation must be know. So the purpose of the platform calibration is to find the axis equation in the world reference frame.

The platform calibration can be done with only one picture, using a special calibration object : a *folded chessboard*. This chessboard defines 2 planes, and if the chessboard is put on the platform such as the intersection of the 2 planes and the platform axis match, it is possible to compute the axis by computing the planes' intersection.

The computation of the axis is quite straightforward, because the planes equations can be computed with the homographies and the extrinsic parameters for each plane (meaning that the 4 extreme corners of the chessboards must be computed, such as for the camera calibration).

Let's $A(a_1, a_2, a_3)$ be the point and $u(u_1, u_2, u_3)$ the normal vector defining the plane P_1, and $B(b_1, b_2, b_3)$ and $v(v_1, v_2, v_3)$ defining the plane P_2.

A vector along the line L, intersection of P_1 and P_2, is given by $u \times v$, and a point can be found by solving the equations :

$$\begin{cases} u_1(x - a_1) + u_2(y - a_2) + u_3(z - a_3) = 0 \\ v_1(x - b_1) + v_2(y - b_2) + v_3(z - b_3) = 0 \end{cases}$$

Also, the axis of the platform is vertical, so there is a point of the line such as $y = 0$. To find this point, the equations are :

$$\begin{cases} u_1(x - a_1) + u_3(z - a_3) = 0 \\ v_1(x - b_1) + v_3(z - b_3) = 0 \end{cases} \tag{11.4}$$

The solution of the equations 11.4 is :

$$\begin{cases} x = \frac{v_1(u_1a_1+u_3a_3)-u_1(v_1b_1+v_3b_3)}{v_1u_3-u_1v_3} \\ z = \frac{v_3(u_1a_1+u_3a_3)-u_3(v_1b_1+v_3b_3)}{u_1v_3-v_1u_3} \end{cases}$$

11.4.2 Rotation formula

When the platform is calibrated, the rotation axis is known. The actual rotation can be computed using the *Rodrigues's rotation formula*. This formula, in its matrix form, provides a rotation matrix around an unit vector $w(w_x, w_y, w_z)$ of angle θ.

$$R = \begin{bmatrix} R(1,1) & R(1,2) & R(1,3) \\ R(2,1) & R(2,2) & R(2,3) \\ R(3,1) & R(3,2) & R(3,3) \end{bmatrix}$$

with :

$$\begin{cases} R(1,1) = cos(\theta) + w_x^2(1 - cos(\theta)) \\ R(1,2) = w_x w_y(1 - cos(\theta)) - w_z sin(\theta) \\ R(1,3) = w_y sin(\theta) + w_x w_z(1 - cos(\theta)) \\ R(2,1) = w_z sin(\theta) + w_x w_y(1 - cos(\theta)) \\ R(2,2) = cos(\theta) + w_y^2(1 - cos(\theta)) \\ R(2,3) = -w_x sin(\theta) + w_y w_z(1 - cos(\theta)) \\ R(3,1) = -w_y sin(\theta) + w_x w_z(1 - cos(\theta)) \\ R(3,2) = w_x sin(\theta) + w_y w_z(1 - cos(\theta)) \\ R(3,3) = cos(\theta) + w_z^2(1 - cos(\theta)) \end{cases}$$

The Rodrigues's formula was implemented, and added to MIMAS's algebra module :

```
template< typename T >
boost::numeric::ublas::vector< T > rodrigues
   ( boost::numeric::ublas::vector< T > const &u,
     boost::numeric::ublas::vector< T > const &v,
     double theta);
```

The previous function does the rotation of the vector v of angle θ around the unit vector u.

11.4.3 The 3-D reconstruction

Once everything above is implemented, the 3-D reconstruction is straightforward. The last thing needed, although it's not compulsory is an automated system to rotate the platform.

In my project, Kim Chuan Lim designed me a rotating platform using a stepper motor, which can be controlled by the serial port. Knowing the number of steps to make the motor do a full turn of the platform, a program to make the platform rotate of a given angle can be written.

The algorithm is explained in details in Algorithm 2.

In this algorithm, the function *get2Dpoints()* will find the laser line and extract 2-D points, the function *reconstruction()* will compute the 3-D points from the 2-D points for a line, and the function *rodrigues()* will apply the Rodrigues' rotation formula on a set of points.

Moreover, before applying the Rodrigues' equation, the points will be translated in a frame which centre is the point we know of the axis of the platform, because the rotation formula can only be applied if the axis goes though the centre of the frame. After the rotation is done, the points are translated back to its original frame.

Algorithm 2 3-D reconstruction algorithm

Require: $step \in]0, 360]$

$\theta \leftarrow 0$
$3Dpoints \leftarrow \emptyset$
while $\theta < 360$ **do**
  $image \leftarrow getImageFromCamera()$
  $line \leftarrow get2Dpoints(image)$
  $3Dpoints \leftarrow 3Dpoints + rodrigues(reconstruction(line), \theta)$
  $rotatePlatform(step)$
  $\theta \leftarrow \theta + step$
end while
return 3Dpoints

Chapter 12

Possible extensions

My project of 3-D reconstruction started almost from scratch, and I spent a lot of time on the calibration algorithm. So the final result is not as accurate as it could be.

This is why this section will present some ideas to improve the project. It is targeted as a start point to the next person who wants to go further in this project. The idea would be to make all the steps of the 3-D reconstruction as accurate as the camera calibration.

12.1 Camera calibration

Using robust statistic algorithms, it would be possible to improve the calibration with for instance a RANSAC (RANdom SAmple Consensus) algorithm to get the homographies (using all the corners of the chessboard) in that case, to remove the outliers.

And to make the calibration process converge, it would also be possible to use instead of a least-square algorithm a robuster estimator to outliers, such as the least median of squares and the least trimmed squares algorithms. (see [20] for information about robust estimators).

Another idea would be to make the corner detection fully automatic. As it is a difficult problem to find the chessboard corners in the general case without any information on their positions, we can simplify it by, for instance, adding a red line surrounding the chessboard, which would be easy to find with a colour filter, and look for the corners in the surface defined by the line.

12.2 3-D reconstruction

The laser calibration could be much more improved, because I use only 2 pictures to calibrate it. It must be possible to find an optimised algorithm (using for instance a least-squares algorithm) working with more pictures of the laser on the chessboard. Furthermore, the line detection could be improved as well, because for the moment it doesn't find all the time the accurate middle of the line.

Another point which must be improved is the laser detection on the object. For the moment the algorithm is very basic, it makes do with a simple average of the bright points, taken line by line. So it is not really precise, and doesn't work if there are 2 points on the same line.

A way of going round this problem would be to compute the connex set of the bright points, and to do the average for each connex part (the algorithm *"Connected Component Analysis"* is already implemented in MIMAS).

We should also be able to cope with different lightning conditions, and to proceed even if the object is not in a dark environment. A circle polariser or a red filter may help in that direction.

Part IV

Appendix

Bibliography

[1] The Bright project
http://vision.eng.shu.ac.uk:8080/3DReconstruction

[2] The MMVL laboratory
http://www.shu.ac.uk/mmvl/index.html

Books

[3] Machine Vision. Ramesh Jain, Rangachar Kasturi, Brian G. Schunck ISBN: 0-07-113407-7

[4] An invitation to 3-D vision. From images to geometric models. Yi Ma, Stefano Soatto, Jana Koseck, S. Shankar Sastry. Springer Editions. ISBN: 0-387-00893-4

[5] Multiple View Geometry in computer vision. Richard Hartley and Andrew Zisserman. Cambridge Editions. ISBN: 0521-54051-8

Articles

[6] A Flexible New Technique for Camera Calibration, by Zhengyou Zhang (December 2, 1998).
http://research.microsoft.com/~zhang/calib

[7] A novel corner detection algorithm for camera calibration and calibration facilities. Theodore Pachidis, John Lygouras, Vasilios Petridis.
http://users.otenet.gr/~pated/publicgr.htm

[8] A comparison of algorithms for sub-pixel peak detection. R. B. Fisher, D. K. Naidu.
http://citeseer.ist.psu.edu/fisher96comparison.html

Libraries

[9] The MIMAS library
`http://www.shu.ac.uk/mmvl/mimas`

[10] The OpenCV library
`http://www.intel.com/technology/computing/opencv`

[11] The Gandalf library
`http://gandalf-library.sourceforge.net`

[12] The LAPACK library
`http://www.netlib.org/lapack`

[13] The MINPACK library
`http://www.cisl.ucar.edu/softlib/MINPACK.html`

[14] The BOOST library
`http://www.boost.org`

[15] The QT4 library
`http://www.trolltech.com/products/qt`

Softwares

[16] A camera calibration toolbox for matlab, Jean-Yves Bouguet.
`http://www.vision.caltech.edu/bouguetj/calib_doc`

[17] Power Crust, Unions of Balls, and the Medial Axis Transform
`http://www.cs.utexas.edu/users/amenta/powercrust`

[18] COPOS : Cloud Of Points Of a Scanner. (in French)
`http://copos.berlios.de`

[19] ArcheoViz Stereoscopy system
`http://www.cis.upenn.edu/archeoviz`

Others

[20] Robust regression algorithms
`http://en.wikipedia.org/wiki/Robust_regression`