Université Libre de Bruxelles
Faculté des Sciences
Département d'Informatique

Master en sciences informatiques

MEMO–F508 Masters thesis

Cartographie 3D du système lymphatique par imagerie IR

Nascimento Gustavo
Promoteur: Prof. Olivier Debeir

Année académique 2013-2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The lymphatic system plays an important role in the human body. Not only it is responsible for fat absorption but also for fluid balance and immunological defense. The lymphatic system is part of the circulatory system, like the blood system. The blood is composed of blood cells and plasma, which contains proteins and carries the white cells. The plasma goes through the capillary vessels in order to reach all the human body cells therefore all cells bath in this filtered blood called lymph.

This lymph has to be cleaned and recirculated, which is why the lymphatic system is used for. It takes back the lymph, makes it circulate through the human body and then returns it to the blood. In addition, the lymphatic system is composed of lymph nodes, which filter the lymph and create lymphocytes to combat invaders and protect the body.

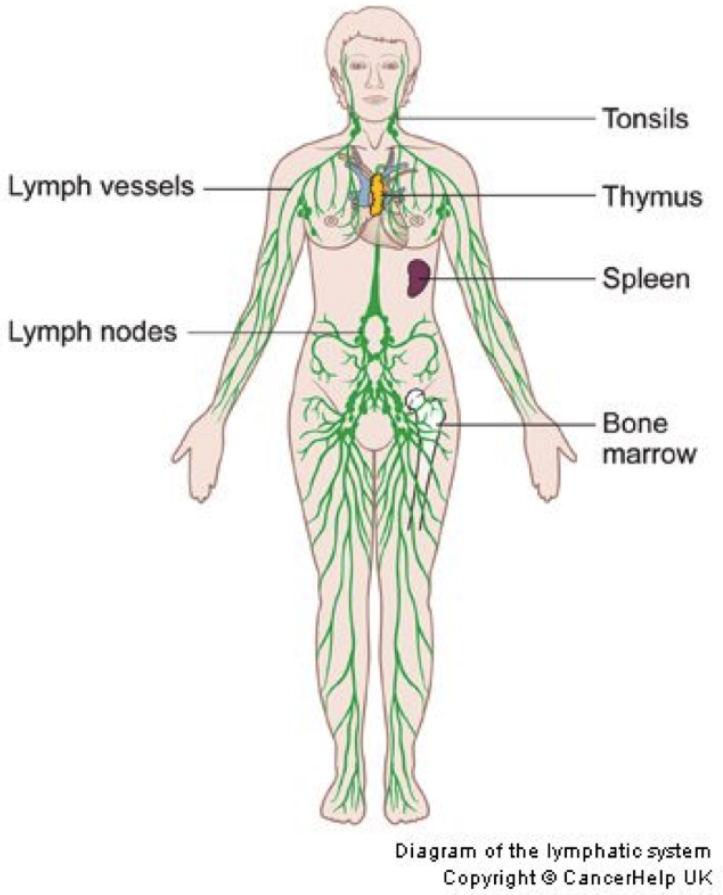
As opposed to the blood circulatory system, the lymphatic system does not have a pump to make the lymph circulate; it uses muscles and breathing motion to pump the lymph through the body.

The study of the lymphatic system is important for cancer diagnoses and treatments. Indeed, metastasis can circulate quickly in the lymphatic system and spread the tumor cells to other parts of the body. In addition, tissue

swelling called lymphedema can occur after cancer surgery, it is especially the case after breast cancer surgeries [34]. In order to treat the lymphedema, massage can be applied to drain the lymph fluid through the patient body and increase the lymph circulation.

The lymphatic system has a complex structure and varies greatly among individuals. Therefore, visualizing it correctly is a tremendous task, which requires the use of methods able to deliver accurate structural information. The lymphatic system structure can be seen in Figure 1.1 [12].

Figure 1.1: The lymphatic system



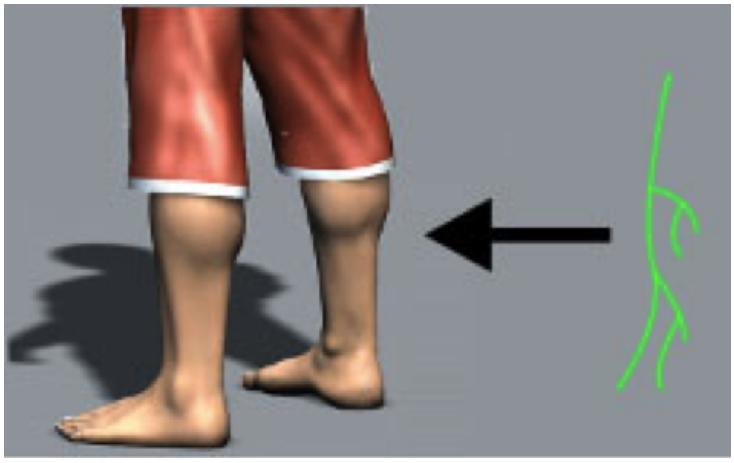
According to [34] "The lack of imaging to obtain structural or functional information currently limits our understanding of the role of lymphatics in disease and impedes the development of therapeutic interventions" therefore

this paper introduces a novel imaging method for the lymphatic system, using infrared lights coupled with a 3D camera.

1.2 Purpose

The purpose of this project is to develop a 3D cartography tool to visualize the lymphatic system of an individual. In order to do that, a green dye called indocyanine green is injected into patients' lymphatic system. This dye has the property to be fluorescent after being exited by infrared lights therefore depth information can be acquired by using infrared cameras. However, the 3D structure only is not helpful because semantics has to be given to that 3D model, i.e. a mapping has to be applied between the lymphatic system's 3D model and the part of the human body corresponding to that 3D model (see Figure 1.2 and Figure 1.3 [9]).

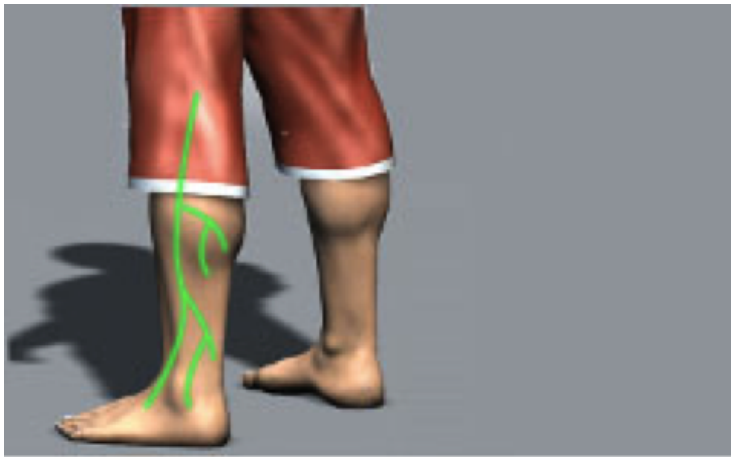
Figure 1.2: Mapping of the lymphatic model with the corresponding body part



In order to do that, a 3D model of the human body is acquired through a Microsoft Kinect camera. Given an accurate 3D model of a patient and his lymphatic system, the mapping can be easily done and physicists and physiotherapists can therefore operate more easily on the patients (see Section 2.3).

Furthermore, because cameras do not have a wide field of vision, image fusion and camera pose estimation have to be used. Image fusion consists

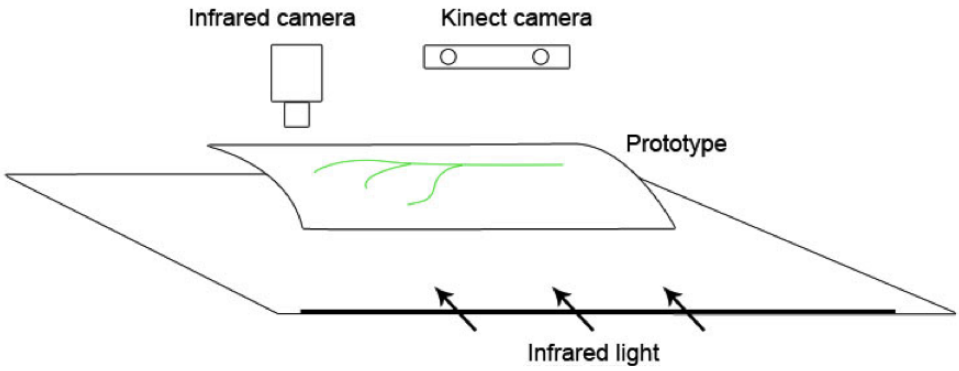
Figure 1.3: Mapping done



of combining multiple images into one while camera pose estimation consists of translating the camera coordinate system into a world coordinate system. That is, determining the camera pose inside a world coordinates system.

This Master Thesis consists of creating a proof of concept for a novel 3D imaging acquisition system. In the first time, a basic Kinect camera coupled with a normal webcam will be used to acquire a 3D representation of an object. The Kinect will acquire a 3D surface of the object while the webcam will take pictures that will be applied on this 3D surface. If this step is successful, an infrared camera will later be introduced with the use of indocyanine green. Also, note that for testing purposes, a prototype model can be used (see Figure 1.4). This prototype model will be a basic representation of a human body limb.

Figure 1.4: Prototype model



Chapter 2

State of the art

2.1 Introduction

This literature review uses a bottom-up approach in order to describe the whole system. To construct a 3D representation of the lymphatic system, different disciplines have to be combined. First, this paper describes the biochemical process of the near-infrared fluorescence principle, a brief explanation is given and the indocyanine green solution is introduced. In addition, this paper reviews in which applications the indocyanine green is used and in what purposes. Third, the different lymphatic system imaging techniques are reviewed. Furthermore, an introduction to the Kinect, its solution to 3D surface reconstruction and the camera pose estimation problem are presented. Finally, the OpenCV library is presented. Indeed, an infrared camera will be used in addition of the Kinect camera therefore its pose estimation has to be computed too, OpenCV provides many useful functions to solve the pose estimation problem. Moreover, OpenCV offers solutions to other problems such as camera calibration and image stitching (see later).

2.2 Fluorescence principle

The fluorescence principle is a phenomenon in which molecules (called fluorochromes) emit light after being excited with light from another source. Fluorescence imaging offers many advantages over other imaging techniques such as ultrasound and x-ray fluoroscopy [42]. In addition, fluorochromes can

be repeatedly exited even after returning to their ground state [34]. However, not all wavelengths of light are suitable. Bellow 700nm the depth of tissue penetration by light is too shallow, above 900nm water absorption interferes with the signal-to-background ratio [28]. The region between 700nm and 900nm corresponds to the near-infrared region (NIR) therefore NIR fluorochromes are commonly used for fluorescence imaging. Since the NIR is invisible for human eyes and the emitted light is from a different wavelength of the projected light, the location of the fluorochromes have to be determined using a specific optical imaging device such an infrared camera.

The indocyanine green (ICG) is a commonly used NIR fluorochrome and it is the only one used in human studies [34]. ICG is an excellent vascular agent for the blood and lymphatic systems and it does not need to be used in large quantities. However, it has weak fluorescence properties but can be exited at a range between 760nm and 785nm with an emitted light between 820 and 840 nm.

Finally, ICG presents only a low toxicity [33] because it is not absorbed by the intestinal membrane but rejected via the liver.

2.3 ICG Applications

The ICG has been used in the medical field for more than 50 years. Cherrick et al. (1960) [18] used the ICG as an agent to estimate the hepatic blood flow on patients. However, it is only recently that ICG is used as a contrast agent when using NIR cameras. Murawa et al. (2009) [36] used ICG for lymphatic mapping and sentinel lymph node biopsy in breast cancer. Handa et al. (2009) [23] used it to capture near-infrared images during coronary artery bypass graft.

Miyashiro et al. (2008) [35] concluded that ICG is a promised tool to detect sentinel node in gastric cancer surgery. According to Marshall et al. (2012) [34] "To date, clinical applications range from (i) angiography, intraoperative assessment of vessel patency, and tumor/metastasis delineation

following intravenous administration of ICG, and (ii) imaging lymphatic architecture and function following subcutaneous and intradermal ICG administration".

All these studies were conducted using ICG as a 2D imaging tool. In a recent study, Liu et al. (2011) [31] stated that "2-D imaging systems do not provide sufficient depth information to guide an operation that is essentially three-dimensional". Therefore, this master thesis will investigate this problem using ICG and NIR cameras.

A 3D cartography of the lymphatic system can be used in 2 different applications:

- Locate and cut lymph node during surgery to prevent metastasis spread.
- Post-surgery application: treat lymphedema with massages.

2.4 Lymphatic system imaging techniques

The traditional imaging technique for the lymphatic system was the lymphography [22], which consists of injecting a radio contrast agent into a patient's body and then taking a X-Ray picture. Nowadays, common techniques are computed tomography (CT) and magnetic resonance imaging (MRI) [22, 39]. CT and MRI can be done with or without contrast agents [32]. In addition to these techniques, newer ones exist such as positron emission tomography, dynamic contrast-enhanced MRI (DCE-MRI) and color Doppler ultrasound (CDUS), which can provide more functional information than the CT and MRI scan [15].

It is only recently that fluorochromes such as ICG are used to obtain images of the lymphatic structure. According to Rasmussen et al. (2009) [38] "NIR fluorescence imaging has the opportunity to provide more rapid imaging of lymphatic function". However, they stated that "Although ICG has successfully been used to image the lymphatics, it possesses limitations such as a low quantum efficiency (0.016), which may limit the signal strength

for deep interrogation of tissues and future tomography applications [33–36]" therefore in the future new dyes have to be developed and tested on humans. The following table shows the review study conducted by Rasmussen et al. [38]

Table 2.1: Review of fluorescence imaging of the lymphatics.

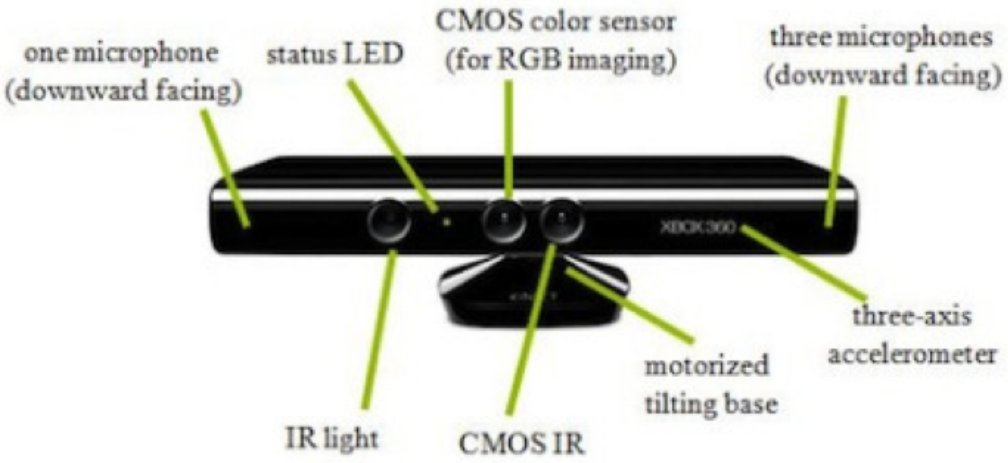
Author, Year [ref]	# of Subjects	Study Aim	Dosage	Comments
Kitai et al., 2005,[28]	18	Sentinel lymph node mapping and resection in breast cancer patients.	Unspecified amount of 5mg/ml ICG	Observed subcutaneous lymphatic vessels in all patients and identified sentinel nodes in 17 patients. No active propulsion reported.
Ogata et al., 2007, [31]	5	Intraoperative guidance for persons with lymphedema.	1 mg ICG	Used fluorescence imaging to intraoperatively guide lymphaticovenular anastomoses for treatment of lymphedema.

Unno et al., 2007, [25]	22	Diagnose lymphedema using fluorescence imaging.	1 mg ICG	Imaged and compared the lymphatics of 10 normal subjects and 12 subjects with secondary lymphedema of the leg. Identified characteristic fluorescent features in lymphedema subjects and compared to lymphoscintigraphy.
Fujiwara et al., 2008, [29]	10	Sentinel lymph node mapping and resection in skin cancer patients.	Unspecified number of injections containing 0.5 mg ICG	Successfully observed and identified lymphatic vessels and sentinel nodes in all subjects. No active propulsion was reported.

Sevick-Muraca et al.,2008, [26]	24	Dose escalation study for sentinel lymph node mapping in breast cancer patients.	0.31–100 μg ICG	Established minimal dosage of ICG needed to observe lymphatic trafficking to sentinel nodes. To our knowledge is the first time active lymphatic propulsion was observed in humans.
Ogasawara et al., 2008,[27]	37	Evaluate breast lymphatic pathways in patients with breast cancer.	25 mg ICG	Monitored lymph drainage pathways from different areas of the breast to the axilla.

Unno et al., 2008, [24]	27	Measure transit times of dye from injection to knees and inguinal region.	1.5 mg ICG	Imaged and compared transit times to knee and groin of 10 normal subjects and 17 subjects undergoing abdominal aortic aneurysm treatment. Correlated fluorescent velocities with lymphoscintigraphy velocities. Reported pulsatile flow in one lymphatic vessel.
Sevick-Muraca and Rasmussen, 2008, [32] Sharma et al., 2008, [6]	44	An ongoing study to quantitatively compare lymph function between normal and lymphedema subjects.	100–400 μg ICG	These papers provided snapshots of an ongoing clinical trial of lymphatic function. Velocity and period of propulsive flow were calculated, and the structure of normal and diseased lymphatics investigated.

Figure 2.1: The Kinect camera



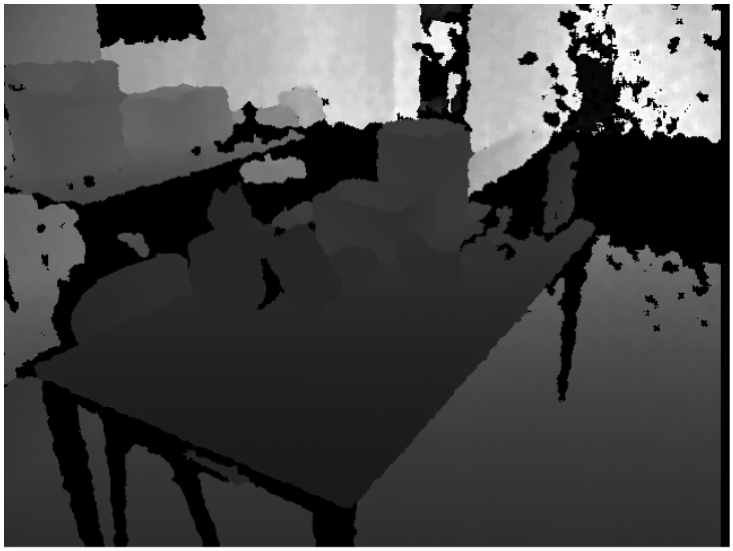
2.5 3D surface reconstruction

A 3D camera Kinect will be used to construct a 3D representation or the patient's limb. A Kinect camera is constituted of a RGB camera, a depth sensor and an infrared light emitter (see Figure 2.1 [5]).

Because of its depth camera and infrared light emitor, the Kinect is able to provide a live depth map using a technique called structured light [20], which consists of projecting a known pattern of infrared dots into the scene and determine the depth by analyzing the pattern deformation. The kinect hardcodes all the information about the pattern (the dots and their relative distances) thus it can compare what it sees with what it stores and provide a live depth map of the scene (see Figure 2.2 [30]).

The Microsoft Kinect has the advantage to be relatively low cost and offers many useful API included the KinectFusion toolkit. KinectFusion provides fast 3D surface reconstruction by scanning the object with the Kinect camera only. In addition, KinectFusion is fast and takes care of low level problems such as camera calibration (see section 2.6), camera pose, surface reconstruction...

Figure 2.2: A depth map from the Kinect camera



The researches about KinectFusion started in 2011 by Shahram Izadi et al. [24]. Explaining in details the implementation of KinectFusion is beyond the scope of this state of art and more details can be found in [37] and [24]. However, an interesting problem is the camera pose problem. The camera pose problem consists of transforming the camera coordinate system into a global coordinate system. In order to solve the camera six degrees of freedom (6DOF) pose, KinectFusion uses the iterative closest point (ICP) algorithm.

The ICP algorithm is an algorithm used to find the transformation that minimizes the distance between two sets of points. ICP is normally used to reconstruct 3D surfaces from different views of the same object. However, KinectFusion uses it to obtain a 6DOF transform that aligns the current points with the ones from the previous depth frame. This transform therefore gives the Kinect global camera pose. At the first frame the camera pose will be set as an identity matrix, at the next frames the ICP algorithm will give the transformation to apply to the camera pose matrix in order to get the new pose of the Kinect camera.

One of the key features of the KinectFusion algorithm is its parallel execu-

tion [24] on the GPU thus fast camera tracking and real-time 3D reconstruction can be obtained. In consequence of that, only compatible DirectX11 graphics cards can meet the requirement to run KinectFusion [11].

Using 3D surface reconstruction tools like the KinectFusion solves the problem of acquiring a 3D surface of the prototype to map the lymphatic system on it. In addition, image fusion, camera calibration and 6DOF pose solutions are offered by the KinectFusion toolkit. However, the Kinect camera cannot be used alone to obtain information about the lymphatic system. Indeed, fluorochromes will emit infrared lights as well and even though those infrared lights will be in a different range of the ones emitted from the Kinect camera, they can perturb the collected data. Therefore, a separated infrared camera will be used (see Section 3.10 for more details about it).

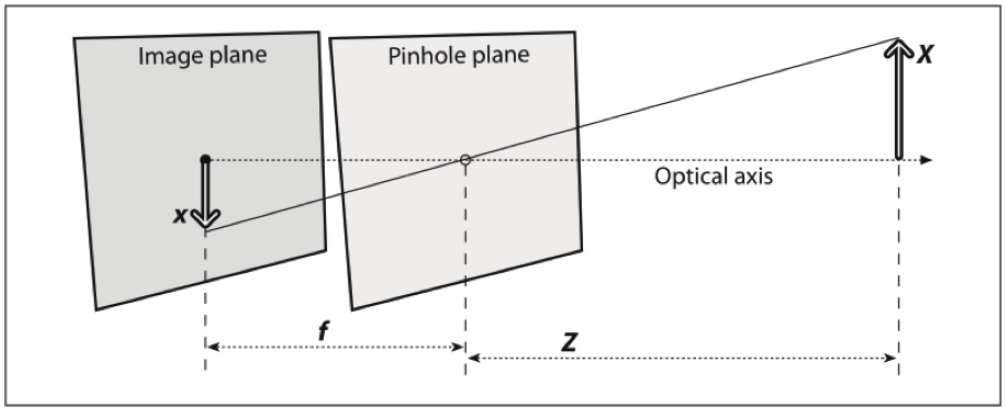
2.6 3D cartography method

The 3D cartography of the lymphatic system will be the most complex step of this master thesis. It will require solving the camera pose (see Section 2.5), camera calibration and image stitching problems. Fortunately, all these problems can be solved more easily by using the OpenCV (Open Source Computer Vision) library.

OpenCV is a free library used in computer vision and real-time applications. It is written in C++ and has C, python and java interfaces. In addition, OpenCV is cross-platform and provides more than 2500 optimized algorithms [13] which can be used for camera tracking, face recognition, image stitching, augmented reality, etc.

The following paragraphs will cover briefly how OpenCV can be used to solve the previous mentioned problems. A deeper explanation will be given afterwards when the project will start.

Figure 2.3: The Pinhole camera model



2.6.1 Camera calibration

Before explaining what camera calibration is and how OpenCV does it, some explanations about basic camera functioning must be given. The figures and formulas in the following paragraphs come from the work of Bradski et al. (2008) [17], the reader is invited to consult their book for further information.

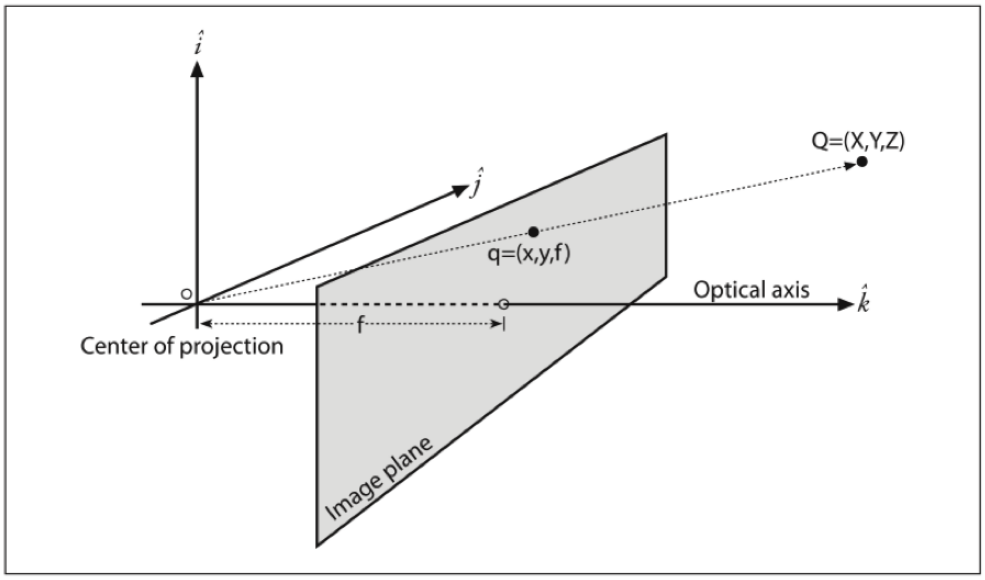
First of all, the simplest camera model is the pinhole camera, in which the light going through a small hole is projected in the opposite surface of that hole. See Figure 2.3 [17, p. 372]. X is the length of the object, x is the projected object image, Z is the distance from the camera to the object and f is the focal length, i.e. the distance where converging/diverging rays are in a focus point.

Like with the human eye, the projected image is inverted, which gives the following equation taken from [17, p. 371] (see the relationship between the two triangles):

$$-x = f\frac{X}{Z} \tag{2.1}$$

Now, in order to eliminate the negative sign, the following model [17, p. 372] can be used instead of the one showed in Figure 2.3 (notice that the relationship between the two triangles is still valid):

Figure 2.4: Simplified pinhole camera model



Next, two parameters c_x and c_y must be introduced. Indeed, one might think that the intersection of the projection center (see Figure 2.4) with the image plane is exactly at the center of the image plane. However, it is often not the case therefore these two parameters are used to adjust this displacement. The following formulas [17, p. 373] give the coordinates of a point in the image plane.

$$x = f_x \left(\frac{X}{Z}\right) + c_x \quad y = f_y \left(\frac{Y}{Z}\right) + c_y \tag{2.2}$$

Finally, the following matrices [17, p. 374] give the transformation that maps a 3D point in the physical world to a 2D point in the image plane. Q is a one-column matrix representing a 3D point in the physical world while q represents the image plane point; q is a 3 dimensions vector because homogeneous coordinates are used. M is called the camera intrinsic matrix.

$$q = MQ \tag{2.3}$$

$$q = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.4}$$

The pinhole camera model is simple but unfortunately the real cameras use lenses in order to gather more light. Lenses introduce some distortions, which have to be corrected. Two different distortions exist, radial distortions and tangential distortions [17, p. 375].

Radial distortions can be corrected by the following formulas:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.5}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.6}$$

Tangential distortions can be corrected by the following formulas:

$$x_{corrected} = x + [2p_1 y + p_2(r^2 + 2x^2)] \tag{2.7}$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 x] \tag{2.8}$$

Thus, in addition with the intrinsic matrix, a 5-by-1 matrix containing the k_1, k_2, p_1, p_2 and k_3 parameters has to be solved. Determining these two matrices is exactly what the calibration process is used for.

OpenCV uses the *cvCalibrateCamera2()* function to do the calibration. The user has to target the camera to a known pattern/structure, usually a chessboard [44]. By taking multiple snapshots of the chessboard at different angles, OpenCV can determine the intrinsic parameters [7].

In conclusion, camera calibration is used to correct distortions and obtain the different intrinsic parameters of the camera.

2.6.2 Camera pose estimation

As seen in the previous section, the equations 2.3 and 2.4 provide a way to map a 3D point in the world to a 2D point in an image taken by a camera. However, those equations consider only the intrinsic camera parameters (the matrix M) and not the extrinsic parameters. A real camera is moving while recording multiple view of an object therefore the camera position is different at each frame. In consequence, for each frame, a transformation that determines the position of the camera has to be found. This transformation is a combination of a rotation and a translation matrix.

The equations become [6]:

$$q = M\langle R|T\rangle Q \tag{2.9}$$

$$q = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \langle R|T\rangle = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.10}$$

The matrix M has to be computed only once but the transformation matrix $\langle R|T\rangle$ has to be computed every frame. In that purpose, some markers can eventually be placed inside the camera vision field (see Section 3.1). Those markers will be easily identifiable by the camera (usually a known pattern is used) therefore their position q will be known. In addition, they will never change of position therefore the position Q will also be known thus determining the camera pose will only consist of solving a system of equations.

2.6.3 Image stitching

As mentioned before, because the camera does not have a wide field of vision, multiple overlapping images will have to be combined together to form only one. This process called image stitching can also be done with OpenCV.

One way OpenCV deals with it is by using SURF descriptors [16] to find correspondences between two images targeting the same scene. Once the matching has been found, OpenCV finds the homography matrix to match the two images together.

2.7 Conclusion

The lymphatic system plays an important role for the body. Obtaining structural information about it is an important task in the medical field. This state of art gave a brief overview of the different problems that can occur during this project. The remaining part of this Master Thesis will consist of building a working prototype for testing purposes, obtaining a 3D construction of it through a Kinect camera and finally, obtaining 3D information from NIR fluorochromes. In that purpose, the OpenCV library and its algorithms will be examined in more details.

Chapter 3

Proof of concept

3.1 Introduction

The main goal of this thesis was to provide a proof of concept for a novel 3D imaging technique. Multiple tools and libraries have been examined in order to determine which one provides the best out of the box solutions for the studied problems. A general description of the solution will be given, then the following chapters will cover in details the pros and cons of each library used.

As a reminder, the purpose of this thesis was to provide a tool to visualize the lymphatic system in a three dimensional space. Since the focus is on building a proof of concept, the following solution has been chosen:

A Microsoft Kinect camera will scan the desired object (typically a patient's limb) and output a 3D surface representation of it. At the same time, pictures will be taken from an external infrared camera (for simplicity, a regular webcam is used, see Figure 3.1) at some desired locations. Afterwards, the pictures will be projected on the 3D surface at the corresponding locations, resulting in a 3D representation of one patient's limb with the exact emplacement of his lymphatic system on it.

As the reader can guess, projecting a picture at the correct position on a 3D surface is not an easy task and relies on several image processing techniques and linear algebra tools. In order to do so, acquiring the global camera

Figure 3.1: Cameras configuration

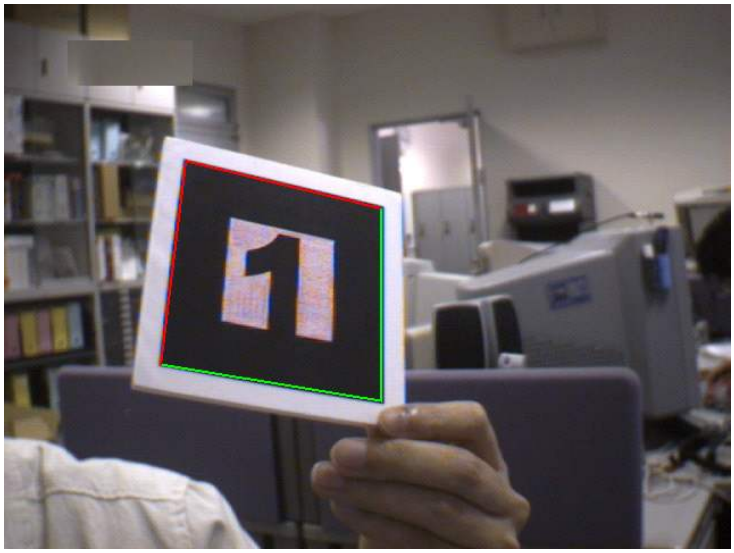
position (of both the Kinect and the webcam) is a necessary task. Indeed, the camera position is important because thanks to it, one can determine the position of a camera (in a given coordinate system) but also the orientation of it. The position and orientation of the cameras are typically given by a translation and rotation matrix (referred as the $\langle R|T \rangle$ matrix). Those matrices are involved in the formulas given in 2.6.2 (the pinhole camera model formulas) and are used to project a 3D point into its corresponding 2D point, which can be useful to texture the 3D surface with the corresponding IR pictures.

Therefore, the focus was first emphasized on finding a suitable library that can give fast and easy camera poses. The following chapters contain a discussion of severals existing tools available on the market.

3.2 ARToolKit

The ARToolKit library is a free library that can be used for non-commercial use (GNU general public license) and provides good solutions for augmented reality (AR) applications. As their official website states: "The ARToolKit video tracking libraries calculate the real camera position and orientation

Figure 3.2: ARToolKit marker

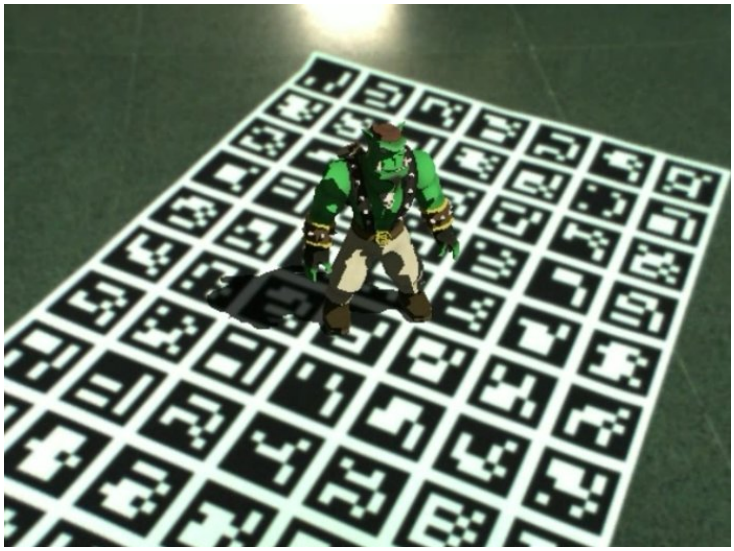


relative to physical markers in real time." [1]. That is, with the help of adequate markers, one can easily determine the pose of a camera in real-time. Figure 3.2 [1] gives an example of a typical marker used in AR applications.

Markers are often used in computer vision because they present patterns that can be easily recognized by imaging software and therefore their position on the image can be detected with more ease. Of course, one inconvenient is that the markers will have to be physically present on the scene. Also, multiple markers have to be combined together. Indeed, one marker is not enough because lighting condition and/or noise can perturb the detection of it therefore using multiple markers solves this issue but makes the code more complex.

At first sight, the ARToolKit library seemed to be a good solution. However, the library has not been updated since 2007, which is a problem since the installation process is not compatible anymore with recent operating systems and hardware. Several installation attempts have been made on different configurations (Windows 7, Ubuntu 12.04 LTS and Mac OSX 10.9 Mountain Lion) but none succeeded.

Figure 3.3: ArUco board



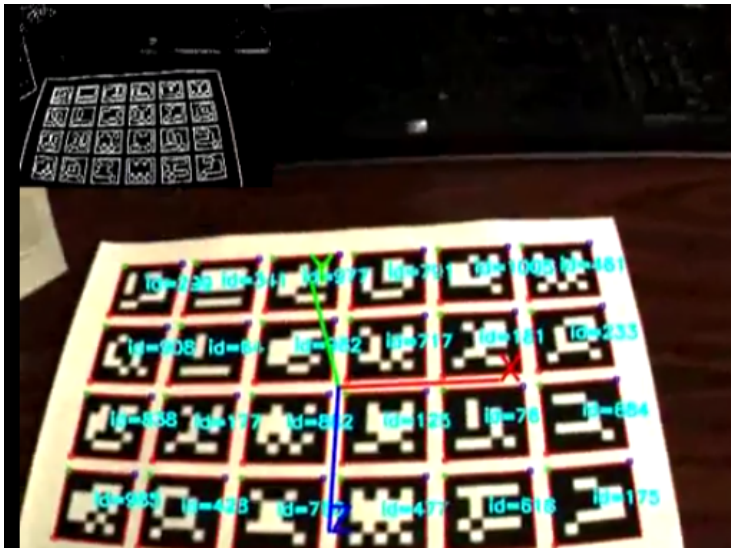
3.3 ArUco

ArUco is a C++ library based on OpenCV. Since OpenCV is used throughout this project, ArUco seemed an interesting alternative to ARToolKit. Like ARToolKit, ArUco uses markers/tags in order to get the camera pose in real time. A great feature of ArUco is that it allows the user to create board of tags. As seen in Figure 3.3 [2], a board consists of multiple markers put together.

Those boards can easily be configured (adjusting the number of horizontal/vertical markers for instance) and their creation is provided by simple scripts from the library.

The advantage of using boards over multiple markers is that one can print out the entire board and only one marker from it has to be seen from the camera in order to determine its pose. Also, with multiple markers on a board it is more difficult to lose them all due to lightning conditions or noise. Finally, it provides an easy setup for real world situations. For example, a board (big enough to surround the object of interest) can be printed out and placed on the scene. Then, the ArUco library deals with all the detection

Figure 3.4: ArUco demonstration



and computation steps. Figure 3.4 [2] shows an attempt made where several markers are detected in real-time and the camera pose can be retrieved from it.

ArUco has to be installed with cmake, an installation guide is provided on the official website [2] and on the sourceforge page [3].

3.4 123D Catch

123D Catch is an online tool (also available on PC desktop version) developed by Autodesk, it generates a 3D model from 2D pictures. Since only 2D pictures are necessary to obtain a 3D model, 123D Catch seemed to provide easy and simple solutions for the problem. Indeed, if only pictures taken from the external camera are enough to obtain a satisfying 3D model, then this Master Thesis problem would be solved. Therefore, several tests have been conducted in order to determine if 123D Catch can be useful for achieving the desired goal.

The different tests must reflect real-world situations. For example, a leg, an arm or any other part from a patient's body do not contain a high level

(a) High resolution

(b) Low resolution

Figure 3.5: Statue

of texture details (especially if no body hair are present). Therefore, first a high textured object had been tested, then, tests have been conducted on a less textured object.

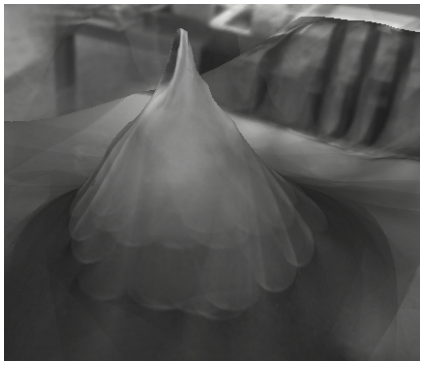
In addition to that, another issues can arise. Indeed, the IR camera will typically produce low resolution pictures with noise on it, which will alter the overall quality of the pictures. In order to get results approaching a real-world situation, two kinds of images have been used:

- High quality images from a digital camera.
- Low resolution grayscale images with noise.

At first, tests have been conducted on a highly textured object (a statue), from Figure 3.5 the reader can see that 123DCatch performs well in both a high resolution and low resolution scenarios.

On the opposite, Figure 3.6a shows that low textured objects produce poor quality models. Moreover, when low resolution pictures are used (which

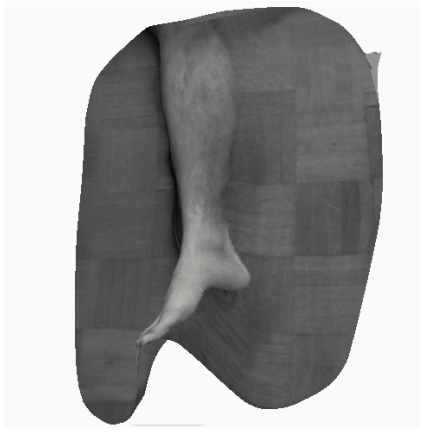
(a) High resolution



(b) Low resolution

Figure 3.6: Bottle

(a) High resolution



(b) Low resolution

Figure 3.7: Leg

will be the case in real world scenarios), the results are even worst (see Figure 3.6b) therefore more investigations have been conducted. The Figure 3.7 shows tests conducted on a leg, which represents a good example for the future use of this 3D scan tool. As can be seen from Figure 3.7a, high resolution pictures produce good quality model. As for the low resolution version, (Figure 3.7b) it seems to not affect much the model.

In conclusion, it can be observed than 123D Catch can be a good, cheap and easy solution for our problem.

However, a main concern about 123D Catch is that the 3D generation takes a lot of time (usually more than 15 minutes depending on the pictures resolution), a Kinect acquisition could provide faster results. Also, as said earlier, some models do not provided good results therefore some cautions should be raised (it is especially the case when pictures with a level of quality similar of IR images are used). Finally, 123D Catch does not provide any API to manipulate nor change the data therefore this solution is neither flexible nor suitable for personal adaptation.

3.5 Kinect Fusion

Kinect Fusion is a project developed by Microsoft researchers. It is a set of libraries being part of the Kinect SDK, it provides solutions for real-time 3D model scanning using the sensors of the Kinect. Several applications samples are provided freely by Microsoft. Those applications are available in C++ and `C#` native code.

One big feature of Kinect Fusion is that only a Kinect is necessary to construct a 3D model of the desired object. There is no need of markers or any other visual information. The scan is done in real-time and several export format exist (*.obj*, *.stl*, *.ply*).

However, the reader should be aware that naturally, only windows development is supported. Also, as said in Section 2.5, a decent hardware is

necessary in order to run the Kinect Fusion Toolkit smoothly. The official website recommends the following minimal configuration [11]:

Table 3.1: Minimal configuration to run Kinect Fusion

Desktop PC
Processor 3GHz multi-core
DirectX11 compatible graphics card with 2GB or more of dedicated on-board memory.

Also, the reader should keep in mind that running Kinect Fusion on a weaker configuration is possible (as long as a DirectX11 compatible graphics card is used). However, it will produce low frame rate and it will make the use of the Kinect Fusion Toolkit tedious (the Kinect camera will have to be maneuvered slowly in order to not lose its camera pose).

As said in 2.5 the Kinect Fusion Toolkit relies on the iterative closest point (ICP) algorithm. Be it reminded that the ICP algorithm finds the transformation that minimizes the distance between two sets of points. At first, the $\langle R|T \rangle$ matrix of the Kinect is set as the identity matrix and then, the ICP algorithm finds the relative transformation between the current frame and the next one and updates this matrix.

3.6 Discussion

This section will discuss the implementation choice made. Indeed, each library has advantages and disadvantages. The choice of the good one to use is an important one and would impact a lot the development of this Master Thesis.

As seen earlier, the ARToolkit library was directly eliminated. At first, this library seemed to provide good solutions for the studied problem. However, due to its old last update date, its installation was considered as hazardous for recent hardware and systems. Also, the lack of support in the long term makes it be an arguable choice.

The next library examined was the ArUco library, its development is recent and based on OpenCV. The latter constitutes a solid argument in favor of its use because as it will later be shown, a lot of steps during this Thesis are solved using the OpenCV library. Also, as noticed earlier, the library solves the camera pose problem using boards which constitutes several advantages over single markers.

123D Catch seems to offer an easy-to-use solution. However, the results can not always be guaranteed (see Figure 3.6b). Moreover, 123D Catch acts as a *black box* and no API are available to provide further controls or personalizations to the solution.

Finally, one last option is the Kinect Fusion Toolkit. The Kinect Fusion Toolkit can provide a simple API to obtain the camera pose in real-time. However, only the Kinect camera pose can be obtained although the set-up requires two cameras (the Kinect and one external camera for IR images acquisition, see Figure 1.4). Since the pictures to map on the 3D surface will be taken from the external camera, one needs to find a suitable solution to obtain the pose of the external camera and not only the Kinect.

As the reader can see, obtaining the Kinect camera pose is an easy task (thanks to the Kinect Fusion Toolkit). However, obtaining the IR camera pose is more difficult because it often relies on markers (or boards in the case of the ArUco library) placed into the scene.

An elegant solution would be to obtain the IR camera pose without relying on external markers. In order to do so, the following idea has been elaborated:

The external IR camera needs to be fixed with the Kinect in a way that when the Kinect will move, the external camera will execute the same movement as well. Since the camera moves together, only their relative position is needed.

Indeed, the relative position will be a $\langle R|T \rangle$ matrix, multiplying the matrix describing the Kinect pose by this matrix will give the IR camera pose.

The relative position can be computed before running the scan therefore the 3D acquisition is not slowed down. Also, no markers have to be used therefore this solution is the easiest and the most elegant one.

An important remark is that at the time of writing this master thesis (Mai 2014), Microsoft is releasing their new Kinect camera for PC. The Kinect v2 provides several improvements over the first version released in 2011. These improvements include:

- A high-definition color camera able to record in 1080p and capture colors more accurately.
- Able to detect heart rate from the facial details of an individual.
- An expanded field of view, which allows objects of bigger size and taller people to be closer from the camera.
- An improved skeletal tracking. The Kinect sensor can now track six skeletons at the same time. Moreover, the precision has been improved, thumbs and fingers can also be tracked yielding to more accurate skeletal tracking.
- Another new feature is what is called the new *active infrared* capability. The new active IR allows the Kinect to capture data even when the light is off and/or the lightning conditions vary. Therefore, the Kinect can run in any lighting conditions making computer-vision-based applications easier to use.

An important point that has not been mentioned yet is that the technology used to acquire live depth map with the Kinect v2 differs from the one present in the first Kinect. Indeed, the first generation of Kinect used a

technique called *structured light* (see Section 2.5). In the new Kinect, time-of-flight (ToF) technology is used.

ToF technology works differently from structured light techniques because in a ToF camera, IR light is emitted then reflected by the objects present in the frame of the camera. The time that the reflected light takes to come back is measured, yielding to the acquisition of the depth information [26].

The Kinect v2 offers higher accuracy and better tracking capabilities than its predecessor. At this time, Microsoft has not released it yet therefore conducted any test on it is not possible. However, the technology looks promising and further investigations should be made when it will be available for the public.

3.7 Solution Description

As said in Section 3.1, a Kinect camera will scan the desired object while an external camera will take pictures which will be then projected on the 3D surface. To project those pictures at the correct locations, one need to performs several steps before. The following subsections describe these steps and give a detailed overview of the solution.

3.7.1 General formula

The general formula that maps a 3D point to its corresponding 2D point in the pinhole camera model is the following (as a reminder, the reader is invited to consult the Section 2.6.2):

$$q = M\langle R|T\rangle Q \tag{3.1}$$

$$q = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \langle R|T\rangle = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}, Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.2}$$

This formula describes the relationship between a 3D point in the real world and its corresponding 2D point inside an image. This information is helpful because the 2D point indicates on which pixels of the picture correspond the 3D point associated with them. If these *(u,v)* coordinates are known, then the color of the associated pixel can be applied on the 3D point, giving a texture mapped on a 3D surface.

The following sections will first focus on how the different information constituting the formula can be obtained and then, how to project a simple picture on its corresponding emplacement given a camera pose. Finally, several pictures will be used and results will be given.

3.7.2 Intrinsic matrix

The matrix of intrinsic parameters is characterized by the following matrix:

$$Intrinsics = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

This matrix describes several inner geometric properties of the camera. The following sections will describe each of them.

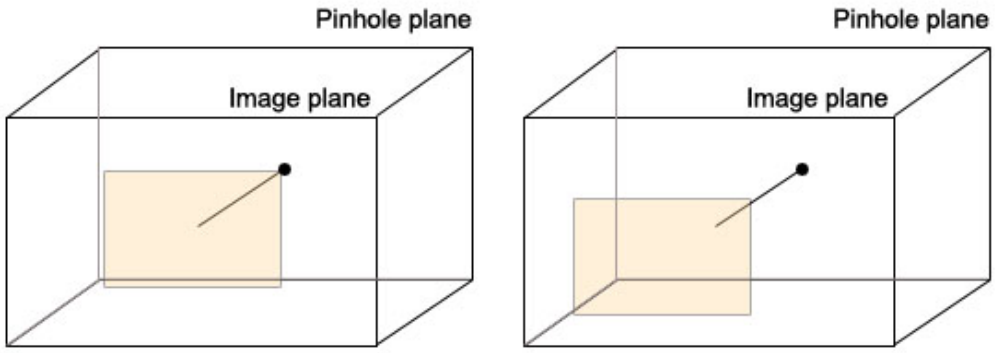
3.7.2.1 focal length (f_x, f_y)

The focal length corresponds to the distance between the pinhole plane and the image plane (see Figure 2.3).

In a perfect pinhole camera model, the f_x and f_y are the same for both axis. However, it is often not the case. This can be due to: camera's lens distortions, camera's sensor flaws, error in camera calibration, etc...

The focal length is measured in pixels.

Figure 3.8: Principal point offset



3.7.2.2 Principal point offset (c_x, c_y)

As seen in Section 2.6.1 the intersection of the projection center with the image plane is rarely at the exact center of the image plane (see Figure 3.8) therefore two parameters c_x and c_y are used to adjust this displacement.

3.7.2.3 Axis skew (s)

Axis skew causes skew/shear distortions on an object by a specified angle from an axis. See figure 3.9.

Remark: axis skew s often equals to zero.

3.7.2.4 Intrinsic matrix acquisition

The OpenCV library offers several ready-to-use samples and solutions for common imaging problems. Once OpenCV is installed, the samples can be found in the *opencv/samples* folder. For this project, the *calibration* sample located in the C++ folder (*cpp/*) has been used. As said in Section 2.6.1, the calibration process relies on a chessboard and several pictures of it to determine the intrinsic matrix.

The reader is invited to consult [44] for more details about how OpenCV determines the intrinsic parameters from a chessboard. Also, more informa-

Figure 3.9: Skew distortion

tion about the acquisition process can be found in section 3.8.1.

3.7.3 Extrinsic matrix

The extrinsic matrix is expressed by the following $\langle R|T\rangle$ matrix:

$$\langle R|T\rangle = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{3.4}$$

It represents the orientation and position of the Kinect camera into a given coordinates system. R is a rotation matrix describing the Kinect orientation while T is a translation vector representing the location of the camera's center inside the coordinates system used.

Obtaining the Kinect camera pose is important for two reasons:

- Testing the formula 3.1 with only the Kinect in order to verify that the projected texture is valid and hence the formula as well.
- As mentioned earlier, an easy approach to obtain the IR camera pose is to first calculate the relative transformation between the IR camera and the Kinect. Once this relative transformation is known, only the Kinect camera pose is needed.

The KinectFusion Toolkit provides straightforward methods to obtain the $\langle R|T \rangle$. Moreover, several samples applications are provided by Microsoft offering out of the box 3D surface acquisition tools. More details about this step can be found in Section 3.8.2.

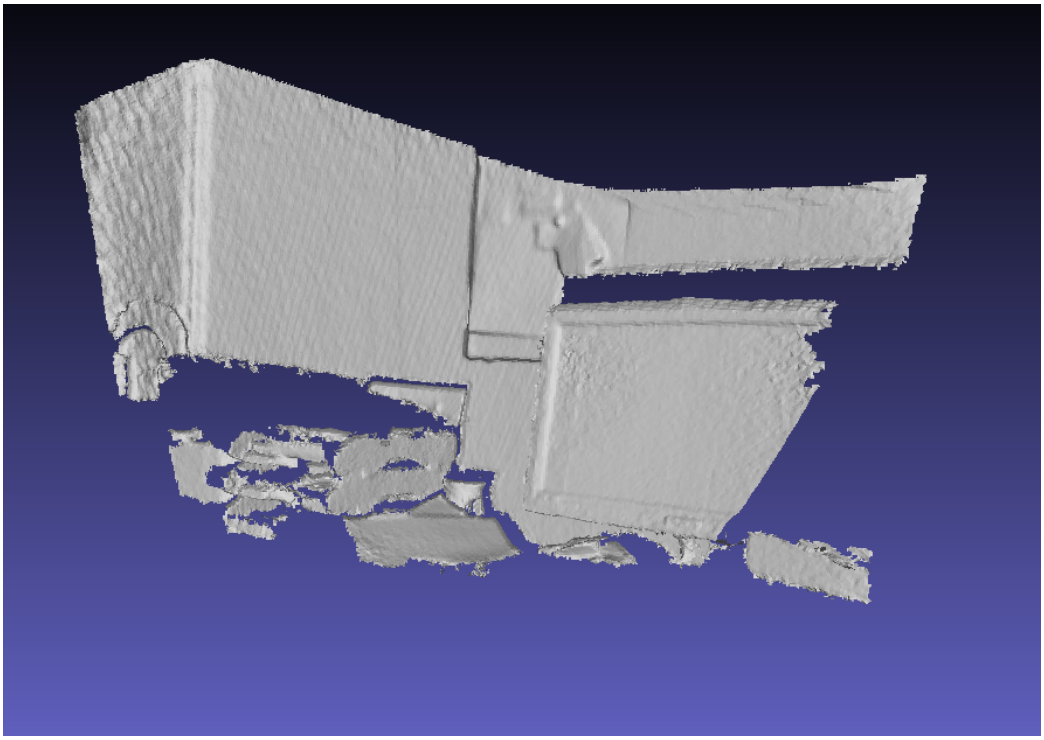
3.7.4 Results using only the Kinect

First, in order to prove the validity of the equation 3.1, no external cameras are used. Indeed, if good results can already be obtained with pictures taken directly from the Kinect, then it means that both the intrinsic matrix and the $\langle R|T \rangle$ matrix acquisition methods are correct therefore the general solution of this master thesis can proceed with pictures taken from an external camera.

If the pictures are taken from the Kinect camera, then all the information from the equation 3.1 can already be found. Indeed, as described in the previous sections, the intrinsic matrix can be obtained via the OpenCV API while the $\langle R|T \rangle$ matrix describing the pose of the Kinect camera can easily be obtained with the KinectFusion Toolkit API. Since, in this case, the pictures are taken directly from the Kinect, no further data are required.

Figure 3.10 shows a mesh obtained through the Kinect Fusion toolkit, Figure 3.12 show this mesh textured with the picture in Figure 3.11. As the reader can see, the projected texture is placed at the correct location thus the data acquired so far are correct. Since the texture mapping is correct, one may wonder why one bothers with the external camera and not just stop here? The problem is, that in real situations, a fluorochrome called indocyanine green (ICG) will be used (recall section 2.2), the ICG emits infrared light after being exited by infrared light from another source. Since the Kinect camera also uses infrared light to obtain depth information [20], some noise/interferences can occur. In one case, the IR light emitted by the external camera could be seen from the Kinect and produce errors. In another case, pictures taken with the external camera could contain the IR dots that the Kinect uses to obtain depth information. Section 3.10 discuses

Figure 3.10: Mesh of a desk obtained through Kinect Fusion



about these issues and presents a theoretical study of the Hamamatsu camera (an external camera that can be used to obtain IR information).

3.7.5 External IR camera pose

As mentioned earlier, an external IR camera will be fixed with the Kinect therefore it is necessary to obtain its extrinsic matrix as well. However, only the extrinsic matrix of the Kinect can be obtained easily (thanks to the KinectFusion Toolkit API). Obtaining the $\langle R|T \rangle$ matrix directly from the external camera is not easy and often relies on physical markers present on the scene (see Section 3.2). Therefore, a more convenient solution is to use the Kinect $\langle R|T \rangle$ matrix obtained through the KinectFusion API and multiply it by a matrix representing the relative transformation between the Kinect and the external camera fixed to it.

Figure 3.11: Picture taken from the Kinect camera

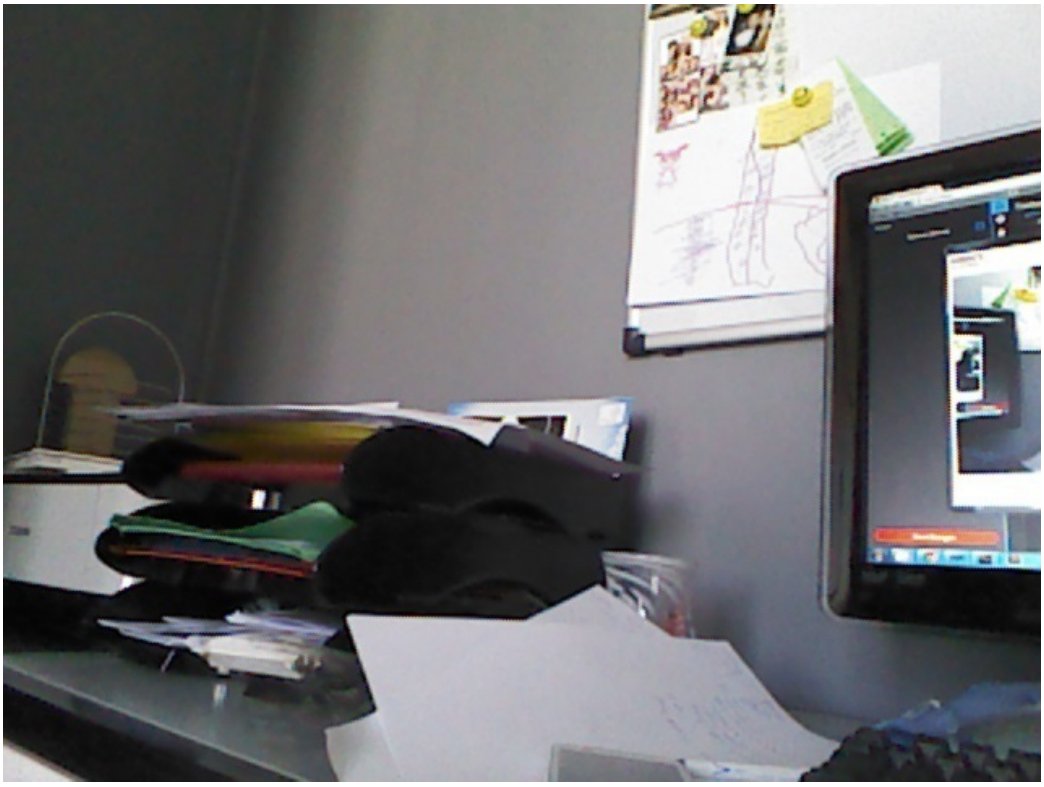
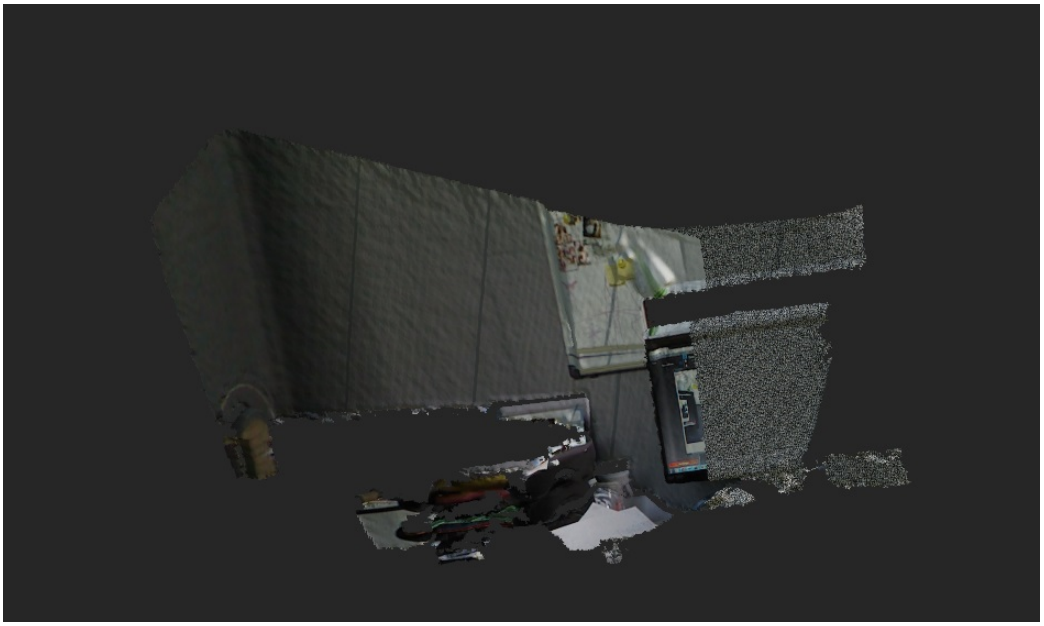


Figure 3.12: Mesh textured with a picture taken from the Kinect camera



To do so, one needs to obtain the individual position and rotation of each camera inside a common coordinates system and then simply calculate the difference between them (see Section 3.8.1.2).

3.8 Implementation details

This master thesis relies on many different libraries and tools. Indeed, as said earlier not only the Kinect Fusion Toolkit is used, but also the OpenCV library. The resolution method consists of combining each of these libraries and tool in an elegant and efficient way. Moreover, often the different steps of the resolution have been written with different languages and API's therefore good communication between them was crucial. Each of these steps was designed for a specific purpose and relies on existing tools and solutions. Developing a good structure and efficient algorithms was important in order to reduce the execution time and provide an easy and automated solution.

This section will elaborate what has been discussed before. First, the general structure of the resolution method will be presented. Then, more details will be given for each step of the solution.

The general structure and design of the solution is given by Figure 3.13. Before capturing and scanning a 3D model, several computations have to be carried out in order to obtain the different cameras information such as the intrinsics and distortion parameters. Only when all the data are gathered, one can start the scanning phase. During the scanning phase several pictures are taken and their corresponding camera positions are registered as well. Finally, the mapping of the pictures on the 3D surface can be operated and the corresponding output files can be viewed by any mesh viewer available on the market (MeshLab was used during this project).

The following subsections will give more details about each phase. Note that all the source code is available on github [10].

3.8.1 Cameras parameters gathering phase

The first important phase consists of obtaining the different parameters of the cameras (labeled with the number 1 in Figure 3.13). As said earlier these parameters are:

- Intrinsics and distortion parameters.
- Relative position between the Kinect and the external camera.

3.8.1.1 Intrinsics and distortion parameters

This section corresponds to the first (1.1) part in Figure 3.13.

Once OpenCV is installed, the intrinsics and distortion parameters can easily be obtained thanks to the sample program present in the OpenCV installation folder. Since obtaining the intrinsics and distortion parameters is a common task in computer vision, there was absolutely no need to create

Figure 3.13: General structure of the solution

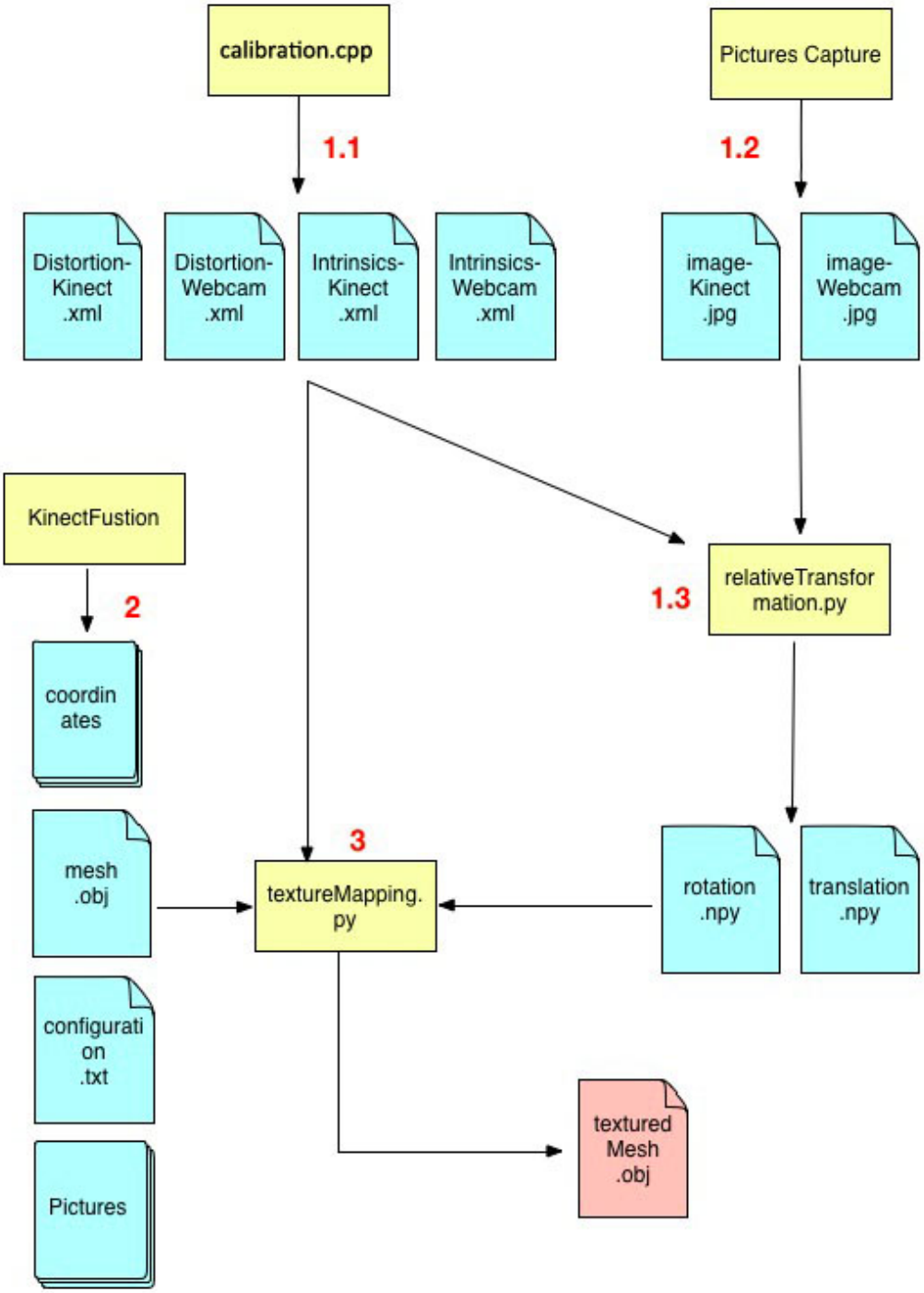
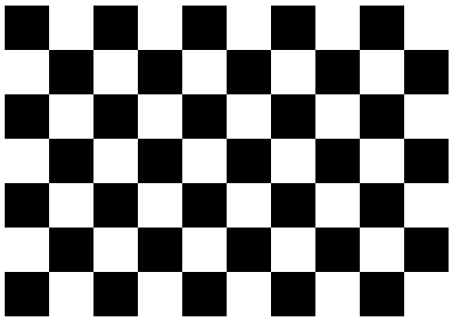


Figure 3.14: Chessboard



an application from scratch to do so. Indeed, OpenCV is a reliable library and thanks to its sample programs, this step could be carried out with ease (using the *calibration.cpp* sample).

The sample program provided by OpenCV requires 25 images in order to output the intrinsic and distortion matrices. Note that this number can be changed in the *default.xml* file together with other information such as the chessboard size and the input file containing the pathname of the different pictures to be used.

Those pictures represents the chessboard in different position and orientation. For this project, a 6 x 9 chessboard and 25 pictures samples have been used (see Figure 3.14 and Figure 3.15).

Another important setting present in the *default.xml* file is the size of the chessboard's squares. Indeed, the calibration process uses no metrics, i.e. one can set the size of the squares in millimeters, meters or any other unit that one finds suitable. After some tests, it has been shown that the Kinect Fusion Toolkit uses meters as unit therefore meters have been used for the calibration step (the chessboard's squares have a 0.024 m size).

Once the calibration is done, it outputs the intrinsic matrix but also the distortion matrix in a *.xml* file. However, one important remark is that the

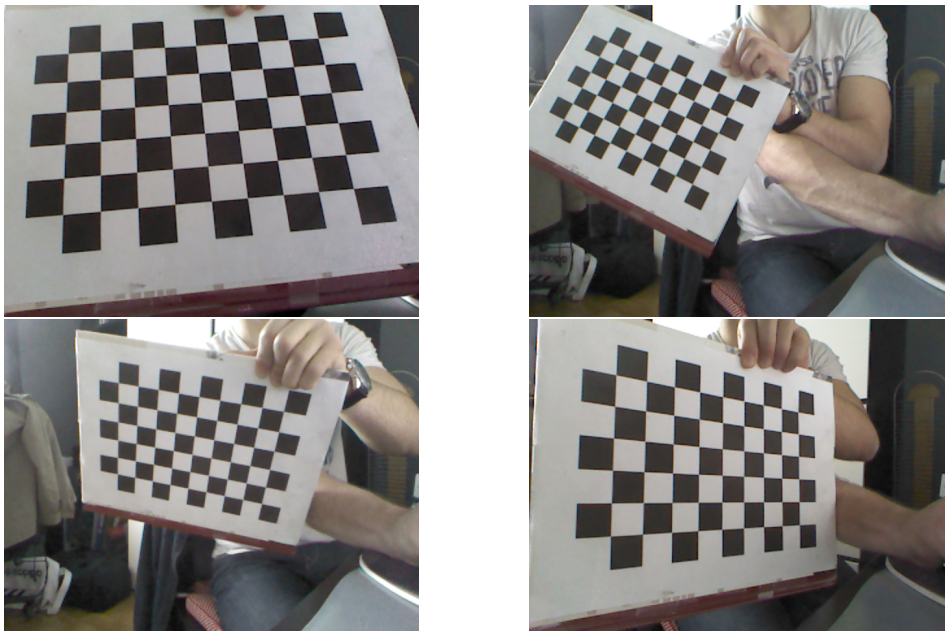


Figure 3.15: Calibration pictures sample

sample program provided by OpenCV save the intrinsics and distortion parameters in the same file with other information that are not necessary for this project. Therefore, a slight modification had to be made in order to save only the necessary information in an adequate file structure. As can be seen from Figure 3.13, XML files have been used. Indeed, OpenCV can easily store/load matrices and vectors if they are stored in an adequate XML format therefore modifying the original sample program was necessary for the next steps.

In addition to that, one should be careful with the orientation of the different pictures used. Indeed, during the tests it has been noticed that the external camera and the kinect camera axis convention is not the same. That is, a picture taken by the Kinect camera will have its x axis flipped if taken by the external camera. This symmetrical difference along the horizontal axis may distort the results therefore before running any computation, one should flip all the pictures along the x axis of one of the two cameras in order to be sure that all the pictures follow the same axis convention.

Remark: pictures were captured from the Kinect thanks to the sample `C#` *Color Basics* program provided by Microsoft when installing the Kinect SDK. As for the pictures taken from the webcam, a simple OpenCV program was coded. It only consists of displaying the video flow of the camera and taking a picture when the space bar is pressed. The source code of it can be found on Github as well [10].

After running the calibration program included with OpenCV, the following results have been obtained:

$$Webcam\ intrinsics = \begin{bmatrix} 828.272 & 0 & 319.5 \\ 0 & 828.272 & 239.5 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$Kinect\ intrinsics = \begin{bmatrix} 510.965 & 0 & 319.5 \\ 0 & 510.965 & 239.5 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$Webcam\ distortion = \begin{bmatrix} 0.01167 \\ 0.30229 \\ 0 \\ 0 \\ -2.3152 \end{bmatrix} \tag{3.7}$$

$$Kinect\ distortion = \begin{bmatrix} -0.00471 \\ 0.010791 \\ 0 \\ 0 \\ -0.17281 \end{bmatrix} \tag{3.8}$$

Note that the distortion parameters are used for further steps. Also, the calibration program output an undistorted image at the end of its execution, which is helpful to determine if the calibration step went well and if the parameters are correct. Figure 3.17 and Figure 3.16 show the undistorted output images for the webcam and the Kinect respectively.

Figure 3.16: Calibration result for the webcam

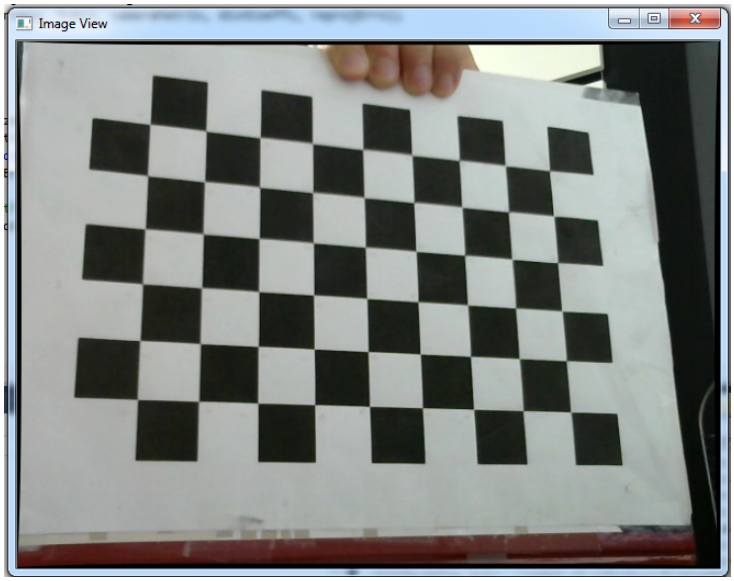
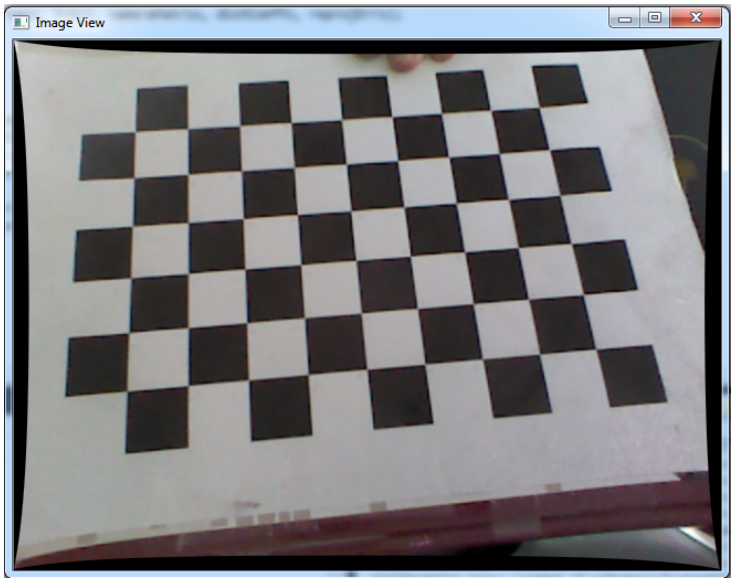


Figure 3.17: Calibration result for the Kinect



3.8.1.2 Relative position between the Kinect and the external camera

This section corresponds to the first (1.2 and 1.3) part in Figure 3.13.

As said earlier, the main idea to obtain the relative transformation between the Kinect and the external camera is to get the individual pose of each camera. Once these poses are known, determining their relative pose consists of finding the unknown in an equation (see later).

To obtain the individual pose of each camera, the *solvePnp()* function of OpenCV is used. This function requires the intrinsics/distortion parameters as well as four images points and their corresponding real-world point. As seen in section 3.8.1.1, the intrinsics/distortion matrices can be obtained by using one of the samples provided with the OpenCV library. As for the image/real-world points, one can target the two cameras at the same chessboard and take a picture of it. For each camera, four points of the chessboard have been selected (see Figure 3.18). The reader can observe the coordinate system used: the origin is at the upper left corner and meters are used again. This configuration is convenient because the same axis convention is used for 2D images by OpenCV.

Now that four real-world points (expressed in meters) are selected, one needs to find their corresponding image coordinates. Once again OpenCV comes in handy because it can easily detect (i.e. find their coordinates in pixels) the inner corners of a chessboard structure using the *findChessboard-Corners()* function (see Figure 3.19). Since the corners can be detected, the coordinates of the four previous points can be found (expressed in pixels).

Having now all the required parameters, the *solvePnp()* function can be used. Here is the pseudocode describing all the previous steps.

Figure 3.18: Four points and axis convention

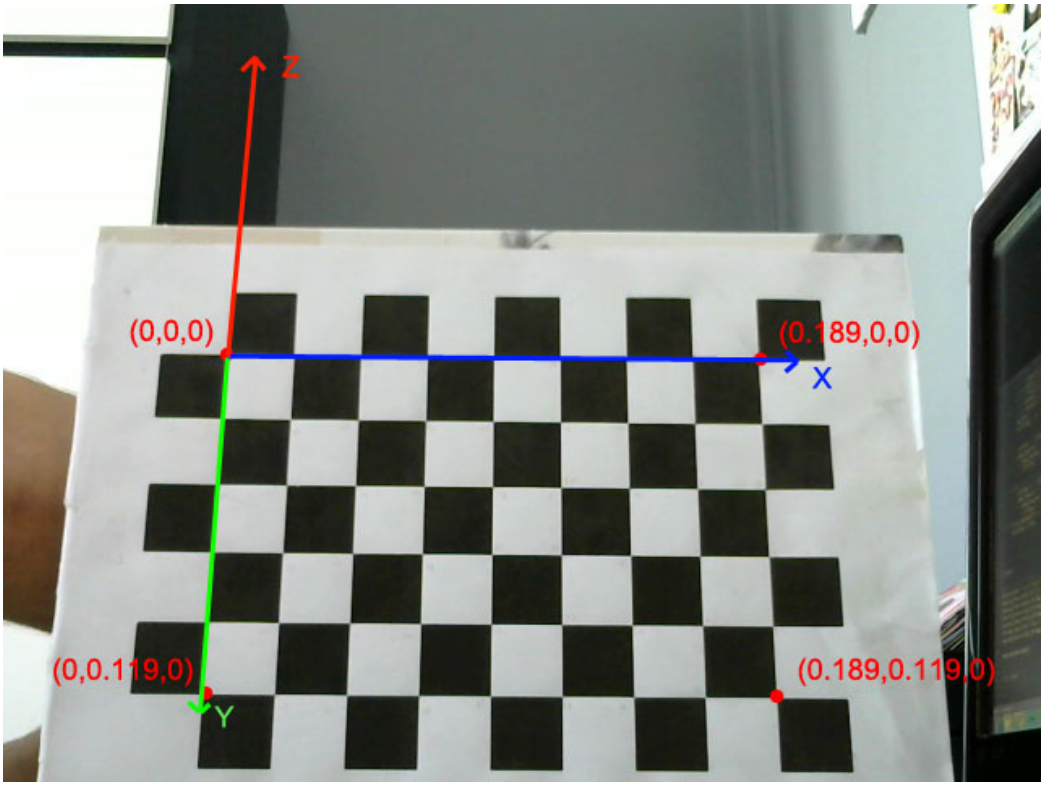
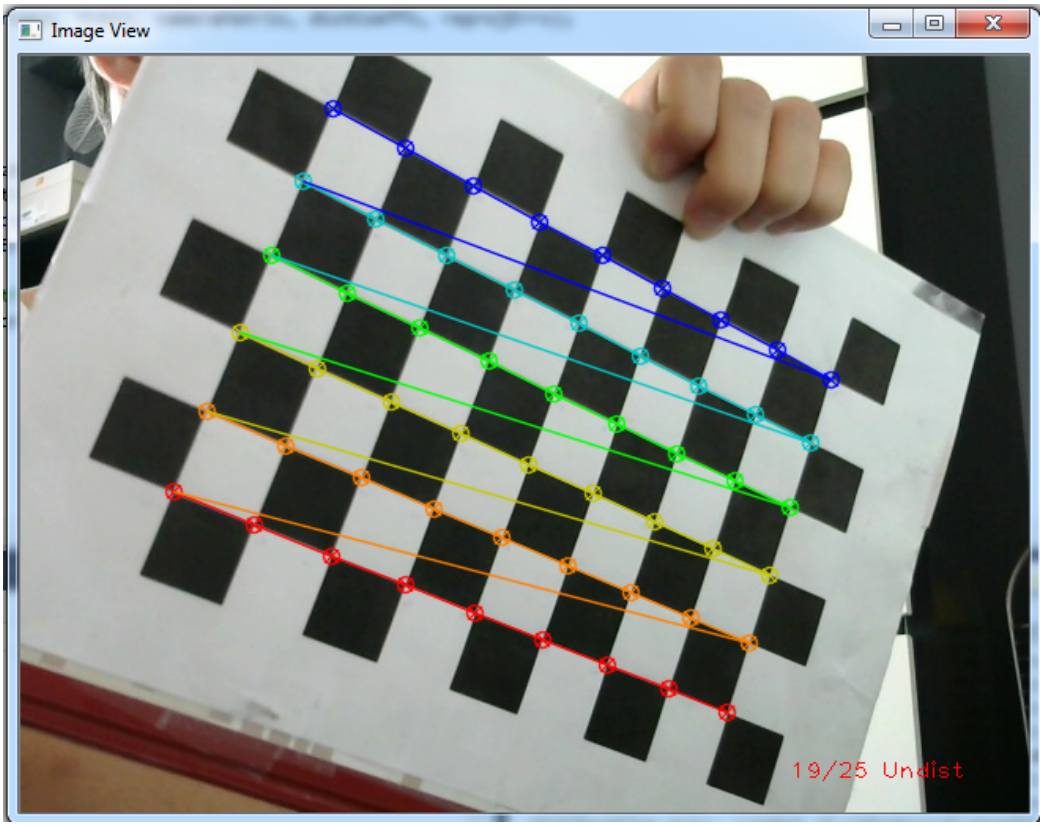


Figure 3.19: Inner corners detection of a chessboard



Algorithm 1 Getting the pose of a camera

$intrinsic \leftarrow load(Intrinsics.xml)$
$distortion \leftarrow load(Distortion.xml)$
$imageChessboard \leftarrow imageRead(image.jpg)$
$objPoint1 \leftarrow (0, 0, 0)$
$objPoint2 \leftarrow (0.189, 0, 0)$
$objPoint3 \leftarrow (0, 0.119, 0)$
$objPoint4 \leftarrow (0.189, 0.119, 0)$
$objPoints \leftarrow createArray(objPoint1, objPoint2, objPoint3, objPoint4)$
$corners \leftarrow findChessboardCorners(imageChessboard)$
$imgPoint1 \leftarrow corners[48]$
$imgPoint2 \leftarrow corners[0]$
$imgPoint3 \leftarrow corners[53]$
$imgPoint4 \leftarrow corners[5]$
$imgPoints \leftarrow createArray(imgPoint1, imgPoint2, imgPoint3, imgPoint4)$
$rvec, tvec \leftarrow solvePnp(objPoints, imgPoints, intrinsic, distortion)$

Note that *corners* is a vector containing all the corners coordinates (expressed in pixels). Therefore, in order to obtain the image points corresponding to the real world corresponding points, one only needs to know how the corners are stored inside the vector and use the appropriate indexes (see Figure 3.20).

An important remark is that the *solvePnp()* function returns a 3-elements *tvec* (translation) and *rvec* (rotation) vector. Contrary to what one might think, the *tvec* vector does not describe the camera position in the world coordinates system. Indeed, the *tvec* vector is expressed in the camera coordinates system and not in the world coordinates system. In other words, *tvec* is the position of the world origin in camera coordinates. The camera coordinates system always moves with the camera (and the camera is at the origin of it) while the world camera coordinates system is static (with the camera moving within it).

This poses a problem since both of the cameras (the Kinect and the webcam) have to be expressed in the same coordinates system in order to find their relative pose. To remedy this issue, the following code is used:

Algorithm 2 Converting camera coordinates to world coordinates system

$rvec, tvec \leftarrow solvePnp()$
$rotationMatrix \leftarrow rogrigues(rvec)$
$cameraPosition \leftarrow -rotationMatrix.Transpose() * tvec$

This code will transform a camera coordinates system into a world coordinates system allowing the two cameras poses to be expressed inside a common world coordinates system. How does this exactly work ?

As a reminder, equation 3.4 represents the camera orientation and location inside the world coordinates system. However, in this case a camera coordinates system is used and not a world coordinates one. The following matrix represents the camera orientation and its location but inside the camera coordinates system (an additional (0,0,0,1) vector has been added to make the matrix square):

$$\left[\begin{array}{c|c} R & T \\ \hline 0 & 1 \end{array}\right] \tag{3.9}$$

The issue is that this matrix represents how the world is transformed relative to the camera coordinates system and not the opposite. Indeed it is fairly more intuitive to have a matrix that represents how the camera is transformed relative to the world coordinates system. Suppose we have a R_c and a C representing respectively the rotation and position of the camera inside a world coordinates system. The inverse of the $\langle R_c|C\rangle$ matrix should give the equation 3.9 (the world-to-camera transformation is simply the inverse of the camera-to-world transformation and vice versa).

$$\left[\begin{array}{c|c} R & T \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R_c & C \\ \hline 0 & 1 \end{array}\right]^{-1} \tag{3.10}$$

The right part of the formula can be decomposed into two matrices describing the rotation and translation.

$$\left[\begin{array}{c|c} R_c & C \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} I & C \\ \hline 0 & 1 \end{array}\right] \times \left[\begin{array}{c|c} R_c & 0 \\ \hline 0 & 1 \end{array}\right] \tag{3.11}$$

$$= \left[\begin{array}{ccc|c} 1 & 0 & 0 & c_1 \\ 0 & 1 & 0 & c_2 \\ 0 & 0 & 1 & c_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \times \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & 0 \\ r_{2,1} & r_{2,2} & r_{2,3} & 0 \\ r_{3,1} & r_{3,2} & r_{3,3} & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.12}$$

Distributing the inverse on these two matrices gives the following result (note that $(AB)^{-1} = B^{-1}A^{-1}$):

$$\left[\left[\begin{array}{c|c} I & C \\ \hline 0 & 1 \end{array}\right] \left[\begin{array}{c|c} R_c & 0 \\ \hline 0 & 1 \end{array}\right]\right]^{-1} = \left[\begin{array}{c|c} R_c & 0 \\ \hline 0 & 1 \end{array}\right]^{-1} \left[\begin{array}{c|c} I & C \\ \hline 0 & 1 \end{array}\right]^{-1} \tag{3.13}$$

Finally, applying the inverse and the multiplication gives:

$$\left[\begin{array}{c|c} R_c & 0 \\ \hline 0 & 1 \end{array}\right]^{-1} \left[\begin{array}{c|c} I & C \\ \hline 0 & 1 \end{array}\right]^{-1} = \left[\begin{array}{c|c} R_c^T & 0 \\ \hline \mathbf{0} & 1 \end{array}\right] \left[\begin{array}{c|c} I & -C \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{3.14}$$

$$= \left[\begin{array}{c|c} R_c^T & -R_c^T C \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{3.15}$$

This results comes from the fact that the inverse of a rotation matrix is its transpose while the inverse of a translation vector is simply its negation. Here is a recap of the full development (note that the explanations and formulas were taken from [8]):

$$\left[\begin{array}{c|c} R & \boldsymbol{T} \\ \hline \mathbf{0} & 1 \end{array}\right] = \left[\begin{array}{c|c} R_c & C \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1} \tag{3.16}$$

$$= \left[\left[\begin{array}{c|c} I & C \\ \hline \mathbf{0} & 1 \end{array}\right]\left[\begin{array}{c|c} R_c & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]\right]^{-1} \quad \text{(decomposing rigid transform)} \tag{3.17}$$

$$= \left[\begin{array}{c|c} R_c & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1}\left[\begin{array}{c|c} I & C \\ \hline \mathbf{0} & 1 \end{array}\right]^{-1} \quad \text{(distributing the inverse)} \tag{3.18}$$

$$= \left[\begin{array}{c|c} R_c^T & 0 \\ \hline \mathbf{0} & 1 \end{array}\right]\left[\begin{array}{c|c} I & -C \\ \hline \mathbf{0} & 1 \end{array}\right] \quad \text{(applying the inverse)} \tag{3.19}$$

$$= \left[\begin{array}{c|c} R_c^T & -R_c^T C \\ \hline \mathbf{0} & 1 \end{array}\right] \quad \text{(matrix multiplication)} \tag{3.20}$$

The reader can see that the relationship between the camera coordinates and the world coordinates can be expressed by the following formulas:

$$R = R_c^T \tag{3.21}$$

$$T = -RC \tag{3.22}$$

Therefore the position of the camera inside the world can be given by:

$$C = -R^{-1}T \tag{3.23}$$

$$= -R^T T \tag{3.24}$$

Which is why the algorithm 2 uses the same development in order to get the positions of the two cameras inside the same world coordinates system.

As for *rvec*, it is a 3-elements rotation vector, the *rodrigues()* method transforms this rotation vector into a 3x3 rotation matrix. Note that rotation

vectors define a more compact way to represent a rotation matrix. As an example, a rotation of $\pi/2$ radiants around the x axis would be represented by the following vector:

$$\begin{bmatrix} \pi/2 & 0 & 0 \end{bmatrix} \tag{3.25}$$

Sometimes it is more convenient to represent a rotation with a vector in order to simplify further computations. Indeed, once the rotation and translation vectors are obtained for both the Kinect and the external camera, getting their relative transformation is straightforward:

Algorithm 3 Getting the transformation between the Kinect and the webcam

$R \leftarrow rotationCamera - rotationKinect$
$R \leftarrow rogrigues(R)$
$T \leftarrow positionCamera - positionKinect$

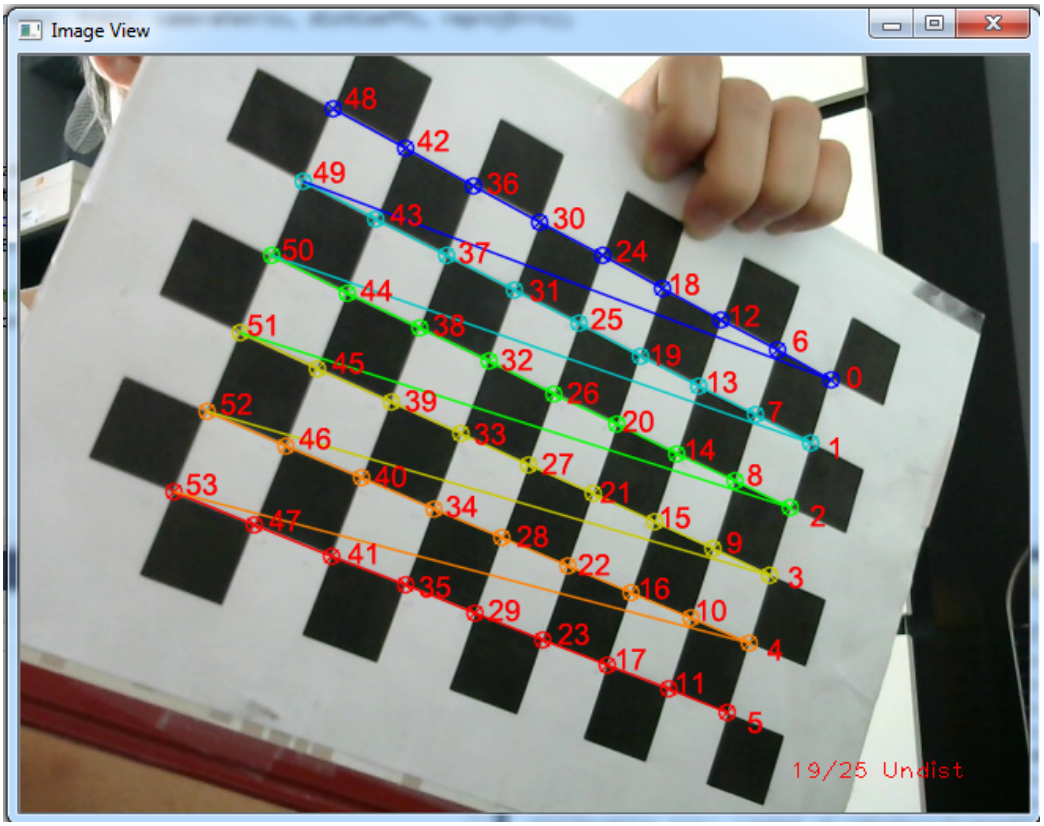
The reader can notice that in this case, using rotation vectors instead of rotation matrices was more convenient since it facilitates the acquisition of the rotation matrix between the Kinect and the external camera.

Now that the relative transformation is obtained, one needs only to multiply the $\langle R|T \rangle$ matrix from 3.1 with the $\langle R|T \rangle$ matrix representing the relative transformation between the Kinect and the webcam.

In order to do so, homogeneous coordinates are used. Homogeneous coordinates consists of adding a fourth row to the $\langle R|T \rangle$ matrices in order to transform them in square matrices, which is necessary for the matrix multiplication (see equation 3.26).

$$\langle R|T \rangle = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.26}$$

Figure 3.20: Corners order



3.8.2 Scanning phase

This section corresponds to second part in Figure 3.13.

Once the Kinect development tools are installed, one can find several sample programs in the installation folder. One of them is the *KinectFusion-Explorer - WPF* application. It consists of an easy and user friendly application made by Microsoft to perform 3D surface acquisition (see Figure 3.21).

This sample application also provides a way to save the 3D surface obtained into a *.obj/.ply/.stl* file. For this project the *.obj* file has been chosen because its structure is easy to modify and to associate with an *.mtl* file (the file responsible for texture mapping).

Figure 3.21: Kinect Fusion Toolkit

Concerning the Kinect extrinsic matrix, it can easily be obtained by using the provided API from Microsoft (the method *worldToCameraTransform()* returns it). For this project, the sample *KinectFusionExplorer - WPF* application has been modified, a button *save camera coordinates* has been added in the user interface allowing the user to save the Kinect camera pose in an external file when it is necessary (see Figure 3.21). The modified program will produce *coordinates.txt* files and a *configuration.txt* file as well, which indicates the number of coordinates taken during the 3D surface acquisition phase.

An important remark is that each time the *save camera coordinates* button is pressed, the user also needs to take a picture with the external camera. A separate OpenCV program *DisplayCamera.py* has been created for that purpose.

Once the scanning is over, the 3D surface can be saved by using the adequate file format (*.obj*). Figure 3.13 shows a recap of the different files produced during this step.

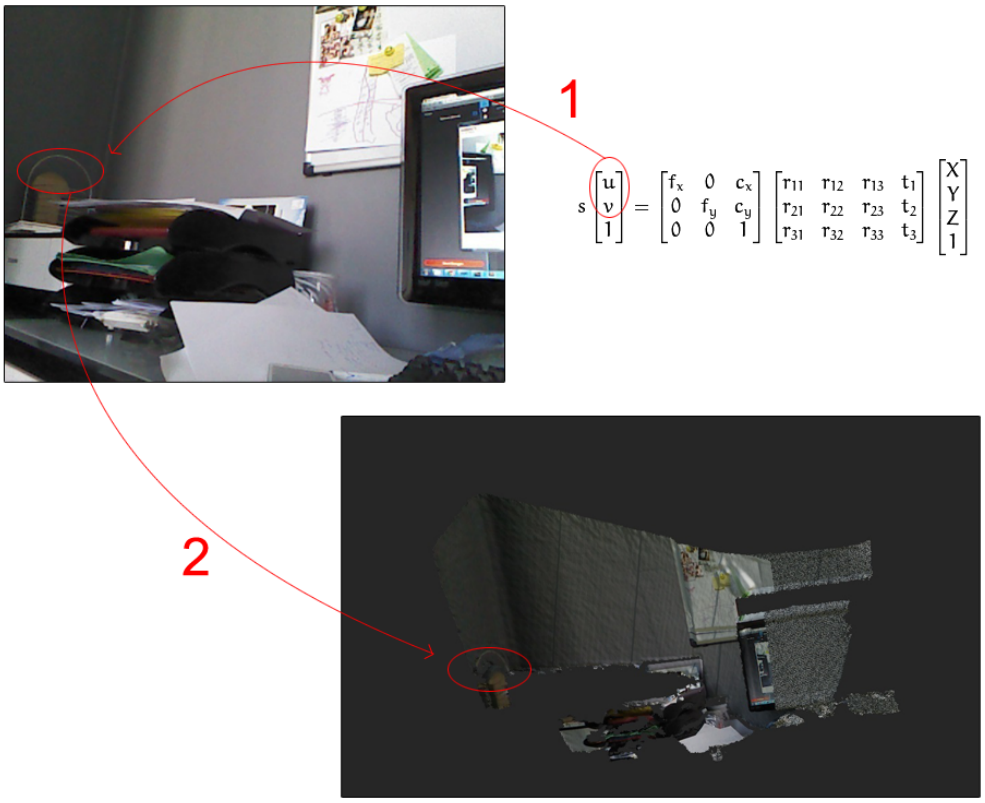
3.8.3 Texture mapping phase

Now that the intrinsic parameters, the Kinect extrinsic matrix and the relative transformation between the cameras are known, the final step of the resolution can begin.

The general Equation 3.1 outlines the different steps needed to map a 3D point into its corresponding 2D point. Once the corresponding 2D point is known, its 2D coordinates indicates which pixel will be used inside the adequate picture in order to texture the 3D surface. Figure 3.22 illustrates this and as the reader might notice, one information is still missing. Indeed, the $[\,X\ Y\ Z\,]$ vector still needs to be obtained.

This vector corresponds to the coordinates of a point of the 3D surface. Recall that those coordinates are given by the *.obj* file acquired during the

Figure 3.22: Texture mapping description



previous step. An *.obj* file has the following structure (lines starting with # are comments):

```
#List of vertices
v 0.538 0.207 -0.361
v 0.535 0.207 -0.353
v 0.538 0.214 -0.361
...

#List of texture coordinates (see later)
vt 0.0 0.0
vt 0.0 0.5
vt 0.5 0.0
...

#List of normals
vn 0.375 -0.366 0.850
vn 0.240 -0.259 0.935
vn 0.217 -0.119 0.968
...
#List of faces
f 1//1 2//2 3//3
f 4//4 5//5 6//6
f 7//7 8//8 9//9
...
```

A 3D mesh is often formed by triangles, each corner of the triangle is a vertex. The *.obj* file lists the coordinates of all the vertices and normals but it also contains a list of faces definitions. A face is defined by its vertices, normals and texture coordinates (see later). Each corner of a face is defined by the following:

f Vertex index/Texture coordinates index/Normal index

Therefore, the following examples defines a triangle by using the first,second and third vertex/normal listed in the *.obj* file:

f 1//1 2//2 3//3

Such definition does not take into account the texture associated with the face. Indeed, a *.obj* file of a textured mesh needs to contain texture coordinates inside it. Each mesh can contain several texture materials associated with it. A picture can be a texture material, if it is the case, the texture coordinates inside the *.obj* file indicate the coordinates of specific pixels inside the picture:

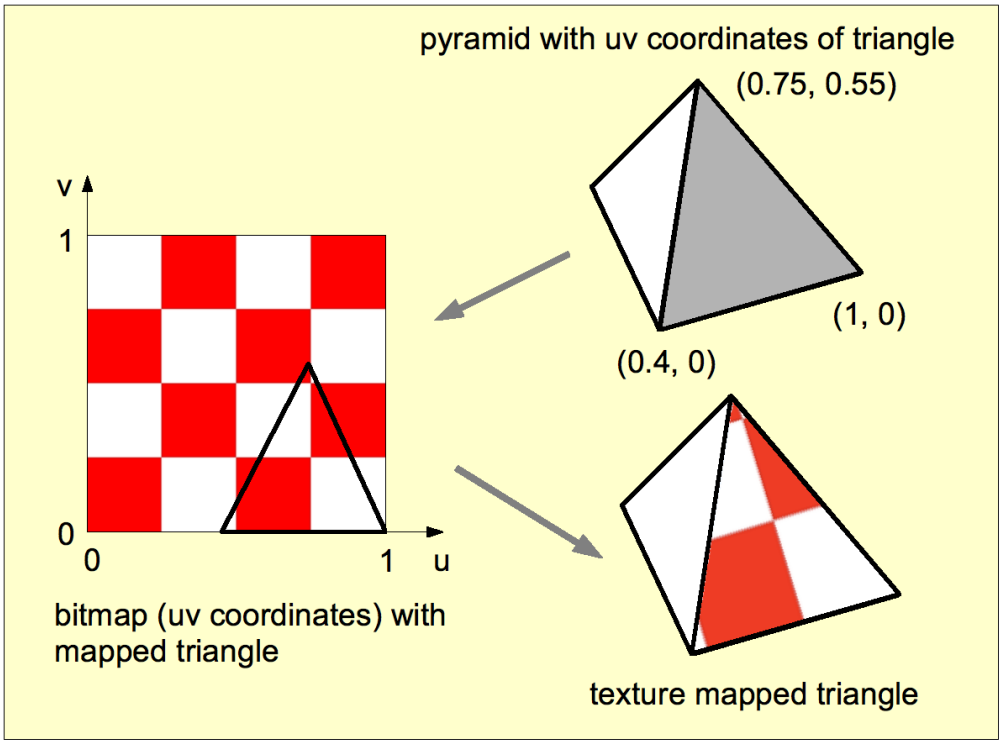
vt u v

Where u and v are normalized (their value is between 0 and 1) and describe image coordinates of a texture (see Figure 3.22 and Figure 3.23 [4]). When using textures, a *.mtl* file needs to be associated with the *.obj* file. A *.mtl* file contains material information, each material can be specified by its color, texture, illumination, etc. *.mtl* files are convenient since multiple materials can be defined inside the same file therefore all the pictures taken by the external camera can be regrouped in a single *.mtl* file:

```
newmtl texture0
Ns 10.0000
Ni 1.5000
...
illum 2
Ka 0.0000 0.0000 0.0000
Kd 0.5880 0.5880 0.5880
...
map_Ka Images/image0.jpg
map_Kd Images/image0.jpg

newmtl texture1
```

Figure 3.23: Texture mapping example



Ns 10.0000
Ni 1.5000
...
illum 2
Ka 0.0000 0.0000 0.0000
Kd 0.5880 0.5880 0.5880
...
map_Ka Images/image1.jpg
map_Kd Images/image1.jpg

The following paragraphs will outline the general solution found to produce these texture coordinates and effectuate the texture mapping.

The *.obj* files produced by the Kinect are usually big (more than 100Mb)

therefore before starting any computation one important step is to preprocess the entire file and put its data inside lists since a disk access is usually slower than a direct memory access. Therefore, the first step is to create lists containing the vertices and normals coordinates. Note that after running some tests it has been estimated that preprocessing the entire file reduces 50% of the execution time.

Another optimization is to use the Numpy library. The Numpy library is a python library used for scientific computing, it provides several tools for linear algebra and matrix manipulation therefore its utilization facilitated many steps during the resolution of this master thesis.

The next step consists of applying the formula 3.1 on each vertex coordinates. If the results are inside the pictures frame (for example, if the pictures have a 640 x 480 size then the coordinates *(u,v)* must be inside these dimensions), then the results are kept and a new texture coordinates is created. If the results are not inside the pictures frame, then they are discarded. The algorithm runs on all the vertices coordinates and create the texture coordinates associated with them one after the other.

Algorithm 4 Texture mapping algorithm

```
intrinsics ← loadIntrinsics()
vertices_list, normals_list ← preprocessFile()
for picture i do
    ⟨R|T⟩ ← getRT(i)
    for vertex coordinates j do
        [ X Y Z ] ← vertices_list.getCoordinates(j)
        [ U V W ] ← intrinsics * ⟨R|T⟩ * [ X Y Z ]
        [ U V 1 ] ← normalizeUV([ U V W ])
        if u and v inside picture i dimension then
            uv_texture_coordinates_list.append(vt + u + v)
            faces ← createTexturedFace()
            if faces contains 3 faces then
                faces_list[i].append(faces)
                faces.reinit()
            end if
        else
            faces ← createUntexturedFace()
            if faces contains 3 faces then
                untextured_faces_list.append(faces)
                faces.reinit()
            end if
        end if
    end for
end for
```

Algorithm 4 shows the main steps of the texture mapping process. One important remark is how the different faces are defined. Indeed, recall that the *.obj* file structure defines two ways to declare faces. One with the texture information and one without it. The program maintains several indexes indicating: the current vertex being treated, its corresponding normal coordinates and the current texture coordinates created. When the *(u,v)* coordinates are inside the picture frame, a face is defined like this:

Vertex index/Texture coordinates index/Normal index

When the *(u,v)* coordinates are not inside the picture frame, a face without texture is created simply like this:

Vertex index//Normal index

Therefore, the *createTexturedFace()* and *createUntexturedFace()* procedures create several faces and when three of them are created in a row, they are put inside some lists containing all faces definitions to write in the final textured *.obj* file.

Finally, at the end of the execution after that all the pictures were treated, the different lists contain all the information needed to define a textured *.obj* file, i.e. the vertices coordinates, the normals coordinates, the texture coordinates and the different faces definitions (the ones with textures associated with them and the ones without). The watchful reader can notice that the *faces_list[i].append(faces)* instruction gathers the faces belonging to the same picture since one needs to specify which material to use for each picture (see Figure 3.24). Also, Figure 3.24 gives an overall view of the resulting textured *.obj* file.

3.9 Results

The previous sections described in details the implementation of the solution, this section will describe and discuss the results obtained with the final application.

As a reminder, Figure 3.13 shows the work flow and the different steps needed to obtain a textured 3D surface. In opposition with the results showed in Section 3.7.4, where only pictures taken directly from the Kinect where projected on the 3D surface, this section describes the different results obtained through the external camera. During the scanning phase, several pictures where taken with a webcam and its corresponding positions were also acquired thanks to the modified KinectFusionExplorer application (see Section 3.8.2). Figure 3.25, Figure 3.26, Figure 3.27 and Figure 3.28 show the different results obtained, as the reader can notice some projection errors can be observed at some parts on the scene. Indeed, this is due because

Figure 3.24: Multiple textures inside a .obj file

```
usemtl texture0

f 51604//51604 51605//51605 51606/5/51606
f 51634/7/51634 51635/8/51635 51636/9/51636
f 51637/10/51637 51638/11/51638 51639/12/51639

....

usemtl texture1

f 1264/32616/1264 1265/32617/1265 1266/32618/1266
f 1267/32619/1267 1268/32620/1268 1269/32621/1269
f 1270/32622/1270 1271/32623/1271 1272/32624/1272

....
```

the camera acquiring the 3D surface is not aligned with the one taking the pictures. That is, the cameras do not target the same area therefore the texture mapping can not be operated accurately everywhere. This phenomenon is called the *parallax problem* and often takes place in cameras where the viewfinder is located above the lens taking the picture. One way to reduce the *parallax problem* would be to use wide angle lens but unfortunately the current infrared cameras do not have this advantage.

3.10 Hamamatsu camera

For test purposes, so far only a webcam had been used as an external camera for this project. In real situations, a camera able to obtain IR information has to be utilized. The following paragraph describes such cameras and also discusses the potential issues that can arise when the Kinect is combined with another IR camera.

Figure 3.25: Result 1

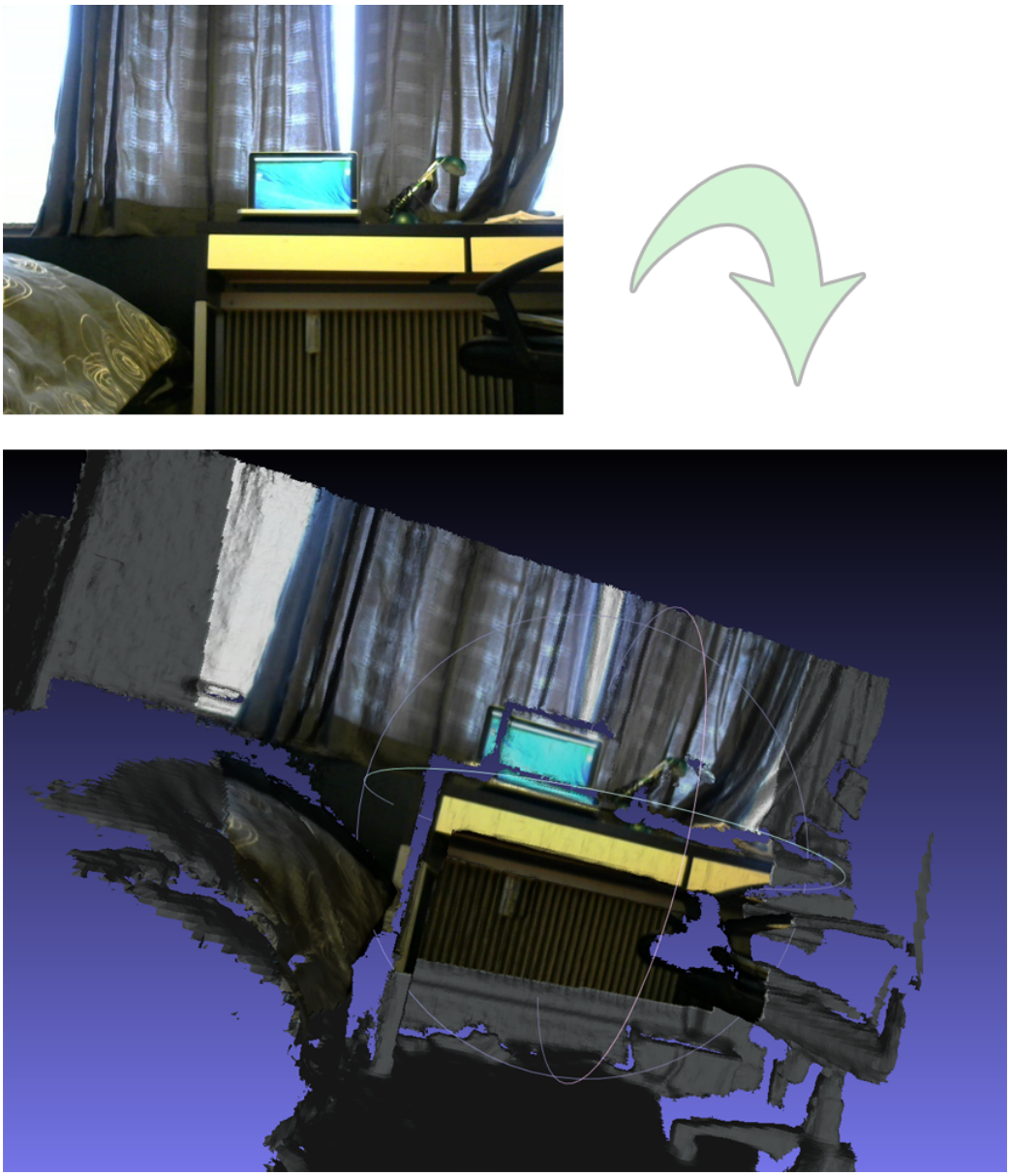


Figure 3.26: Result 2

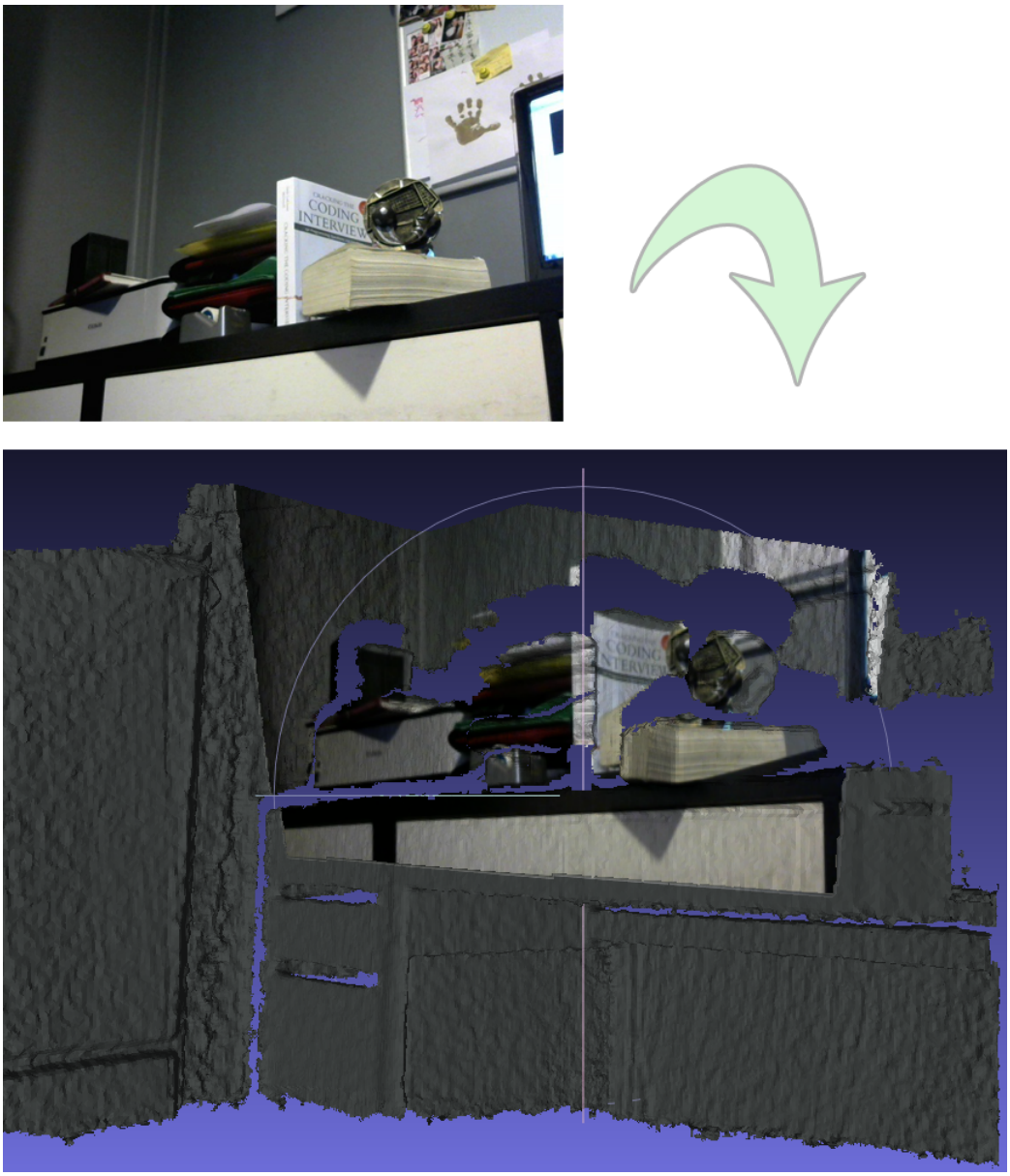


Figure 3.27: Result 3

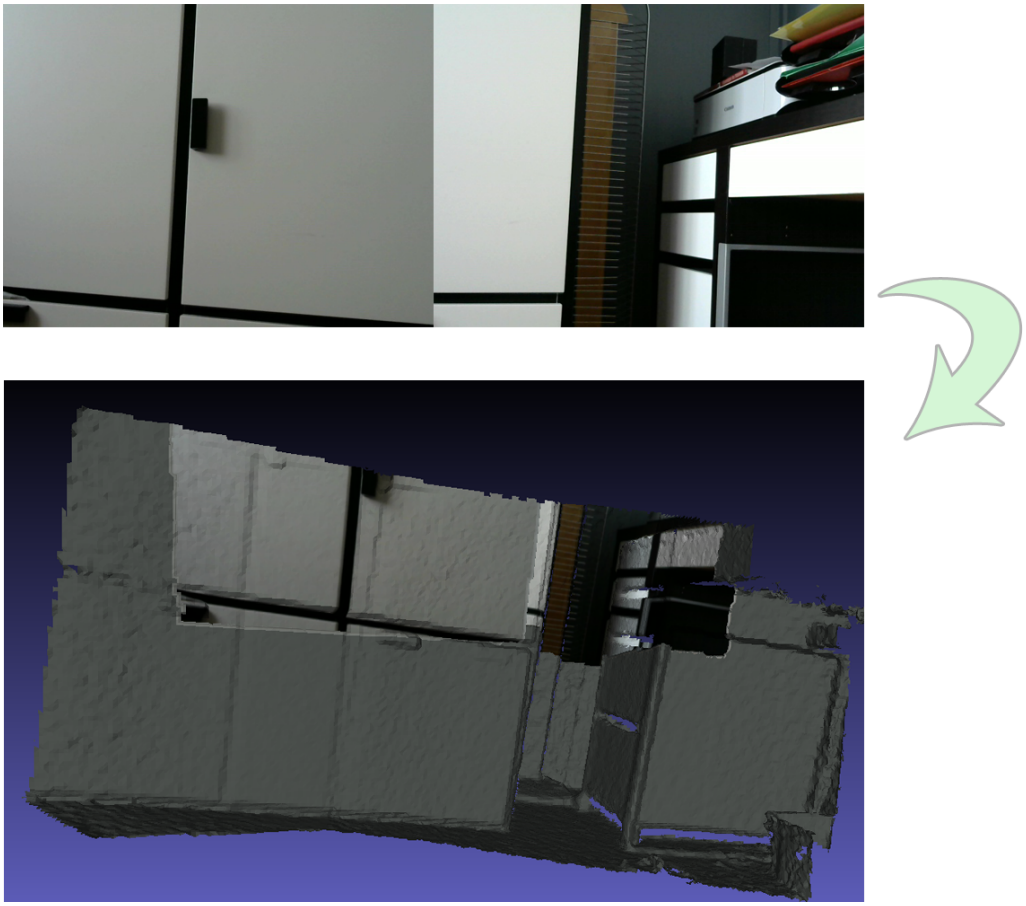


Figure 3.28: Result 4

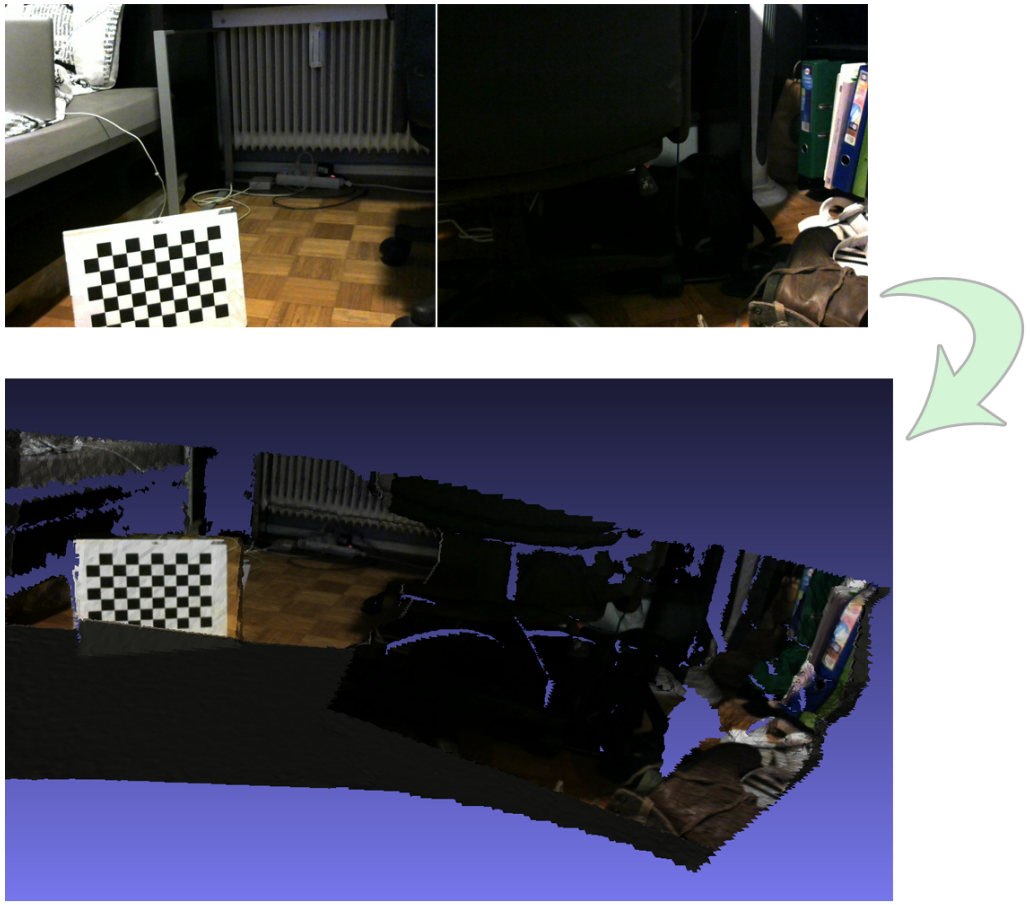


Figure 3.29: Photodynamic (PDE) Hamamatsu camera

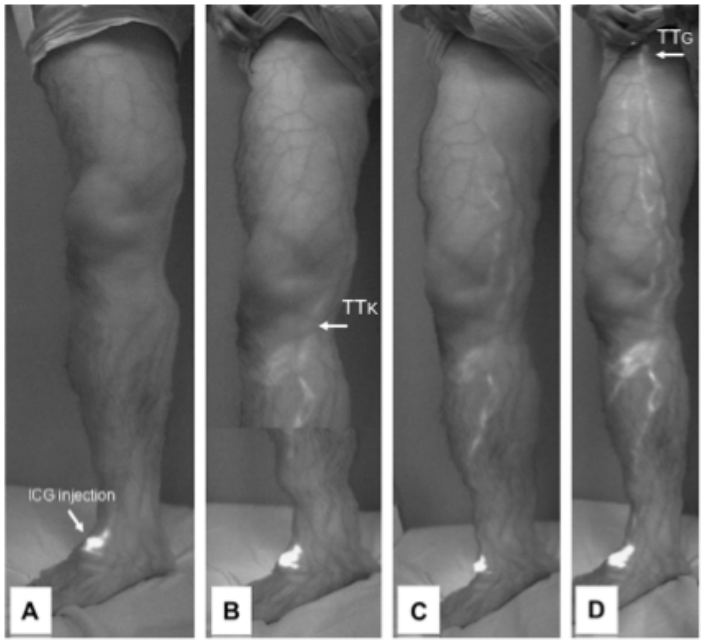
As seen in Section 2.3, the use of indocyanine green (ICG) coupled with IR cameras is recent. One of the cameras of interest is the Photodynamic Eye (PDE) Hamamatsu camera (see Figure 3.29 [14]). Indeed, the PDE camera has already been used in several studies.

Unno et al. (2007) [43] employs it as a diagnostic imaging technique to assess lymph function. They injected 0.3 ml of ICG at the dorsum of the feet and observed the time required by the ICG to reach the groin area. The visualized imaged can be seen in Figure 3.30. Gotoh et al. (2009) [21] used the PDE camera to make the distinction between tumor and normal tissues. More recently, Tagaya et al. (2010) [41] utilized it to identify sentinel lymph nodes in patients with breast cancer. In all those studies, the PDE camera provided satisfactory images to produce good results.

The PDE camera uses the fluorescence principle described in section 2.2. The camera excites the ICG with an IR light at a wavelength of 760 nm [41] and filters out light below 820 nm [21].

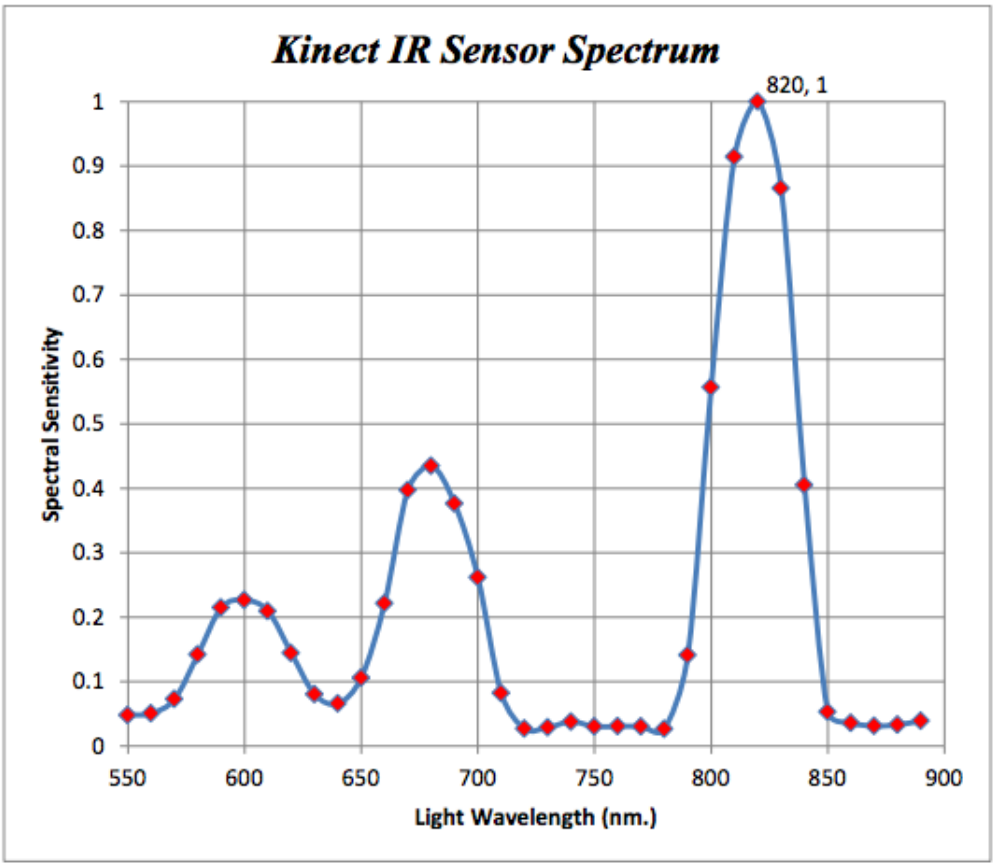
Note that the ICG can be exited at a range between 760nm and 785nm and it emits light between 820 and 840 nm [34]. As said earlier, issues can arise with the Kinect camera since it also produces IR light. Figure 3.31

Figure 3.30: Panoramic views of ICG fluorescence lymphography taken by the PDE Hamamatsu camera



[27] shows the spectral sensitivity of the Kinect as a function of the light wavelength. As it can be seen, lights at a wavelength of 760 nm (the wavelength used by the PDE Hamamatsu camera) does not affect much the Kinect therefore the PDE Hamamatsu camera will not affect the performances of the Kinect. However, the IR emitter from the Kinect produces IR dots at a wavelength of 830 nm [29] therefore those dots might be seen by the PDE hamamatsu camera. Indeed, some tests have been conducted using both the Kinect and the PDE camera, Figure 3.32 shows a picture taken from the PDE camera when no Kinect was in used while Figure 3.33 shows the same area after that is has been targeted by a Kinect camera. As it can be seen, the IR light emitted by the Kinect greatly affects the measurements of the PDE camera. A possible solution would be to use some filters or to turn off the IR emitter of the Kinect when pictures are taken by the PDE camera. If the later is used, one should be careful that the Kinect does not reset its camera pose during the process otherwise the surface acquisition will be prone to errors. Future investigations should be carried out in order to solve this issue.

Figure 3.31: IR Kinect sensor spectrum



Note that other cameras exist, however studying them all is beyond the scope of this thesis, the interested reader can consult the article written by Marshall et al. (2012) [34] for a more exhaustive study of other cameras.

Figure 3.32: PDE Hamamatsu camera in used with no Kinect

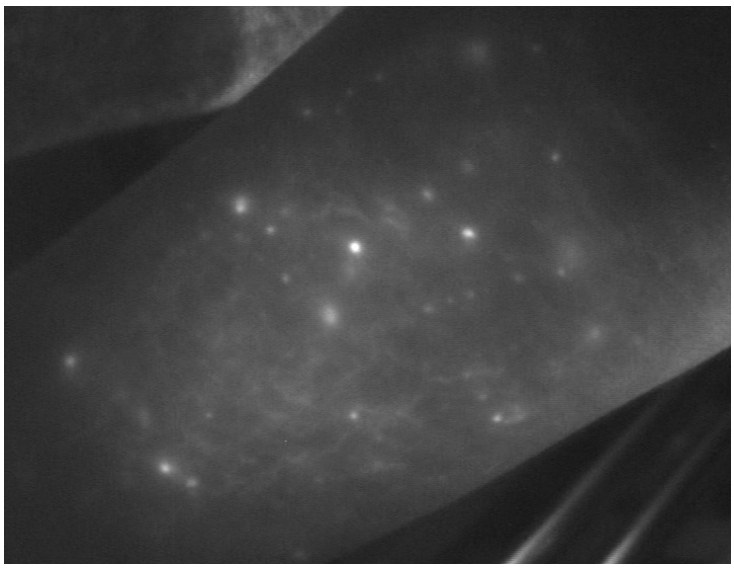
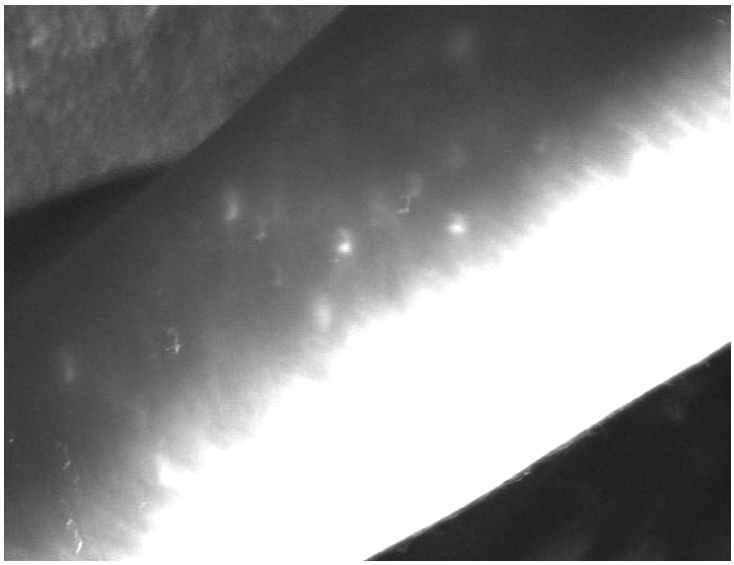


Figure 3.33: PDE Hamamatsu camera in used with a Kinect



Chapter 4

Conclusion

4.1 Conclusion

The purpose of this Master Thesis was to build a proof of concept for a novel 3D imaging system for the lymphatic system. As the reader could notice, such system relies on several computer vision techniques and various common imaging problems needed to be solved such as: camera calibration, camera pose estimation, texture mapping, 3D reconstruction and so on. Since the goal was to produce a proof of concept, only a Kinect camera coupled with an external webcam were utilized. The Kinect was used to acquire a 3D surface of the scene while the webcam was used to obtain pictures of it. Once the pictures were acquired, they could be projected at the correct locations on the 3D surface. Results showed that such concept is feasible. However, one needs to be careful about the parallax issues as discussed in Section 3.9 and how badly it could affect the solution in a real life scenario where the cameras will be used in a medical context.

Also, in a real world scenario, the external webcam will be replaced by an infrared camera coupled with the use of fluorochromes such as the Indocyanine Green. A feasibility study has been conducted by using the Hamamatsu PDE camera as an external camera. Results showed that the IR light emitted by the Kinect perturbed a lot the image acquisition carried out by the PDE camera (see Section 3.10). Future works should try to solve this issue by eventually using filters on the cameras or turning off the Kinect IR emitter

when pictures are acquired by the other IR camera.

In conclusion, this Master Thesis built the basics for a new imaging technique. However, this 3D imaging tool is not mature enough to be used in a medical context therefore future works on it should focus on reducing the parallax and fluorescence issues.

Bibliography

[1] ARToolKit home page. http://www.hitl.washington.edu/artoolkit/. Accessed April 9, 2014.

[2] ArUco: a minimal library for augmented reality applications based on OpenCv | aplicaciones de la visión artificial. http://www.uco.es/investiga/grupos/ava/node/26. Accessed April 9, 2014.

[3] Augmented reality library - browse files at SourceForge.net. http://sourceforge.net/projects/aruco/files/. Accessed April 9, 2014.

[4] blog.tartiflop » blog archive » first steps in Away3D : Part 3 – texture mapping. http://blog.tartiflop.com/2008/11/first-steps-in-away3d-part-3-texture-mapping/. Accessed May 6, 2014.

[5] Build a kinect bot for 500 bones. http://hackaday.com/2011/11/09/build-a-kinect-bot-for-500-bones/. Accessed July 19, 2013.

[6] Camera calibration and 3D reconstruction — OpenCV 2.0 c reference. http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html. Accessed August 2, 2013.

[7] Camera calibration with OpenCV — OpenCV 2.4.6.0 documentation. http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html. Accessed August 2, 2013.

[8] Dissecting the camera matrix, part 2: The extrinsic matrix. http://ksimek.github.io/2012/08/22/extrinsic/. Accessed May 5, 2014.

[9] Gianluca Miragoli makehuman 3d characters | makehuman. http://www.makehuman.org/content/gianluca_miragoli_makehuman_3d_characters.html. Accessed July 19, 2013.

[10] Github account for the project. https://github.com/gusta07/Master-Thesis. Accessed May 18, 2014.

[11] Kinect fusion. http://msdn.microsoft.com/en-us/library/dn188670.aspx. Accessed July 18, 2013.

[12] The lymphatic system : Cancer research UK : CancerHelp UK. http://www.cancerresearchuk.org/cancer-help/about-cancer/what-is-cancer/body/the-lymphatic-system. Accessed July 18, 2013.

[13] OpenCV | OpenCV. http://opencv.org/. Accessed July 24, 2013.

[14] PDE – photodynamic eye « international HealthScreen technologies. https://www.iht-ltd.com/pde-photodynamic-eye/. Accessed July 18, 2013.

[15] Tristan Barrett, Peter L. Choyke, and Hisataka Kobayashi. Imaging of the lymphatic system: new horizons. *Contrast Media & Molecular Imaging*, 1(6):230–245, 2006.

[16] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006*, page 404–417. Springer, 2006.

[17] Gary R Bradski and Adrian Kaehler. *Learning OpenCV: computer vision with the OpenCV library*. O'Reilly, Sebastopol, CA, 2008.

[18] Gilbert R. Cherrick, Samuel W. Stein, Carroll M. Leevy, and Charles S. Davidson. Indocyanine green: observations on its physical properties, plasma decay, and hepatic extraction. *Journal of Clinical Investigation*, 39(4):592, 1960.

[19] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, page 303–312, 1996.

[20] Barak Freedman, Alexander Shpunt, Meir Machline, and Yoel Arieli. Depth mapping using projected patterns, April 2008.

[21] Kunihito Gotoh, Terumasa Yamada, Osamu Ishikawa, Hidenori Takahashi, Hidetoshi Eguchi, Masahiko Yano, Hiroaki Ohigashi, Yasuhiko Tomita, Yasuhide Miyamoto, and Shingi Imaoka. A novel image-guided surgery of hepatocellular carcinoma by indocyanine green fluorescence imaging navigation. *Journal of Surgical Oncology*, 100(1):75–79, July 2009.

[22] Ali Guermazi, Pauline Brice, Christophe Hennequin, and Emile Sarfati. Lymphography: An old technique retains its usefulness1. *Radiographics*, 23(6):1541–1558, January 2003. PMID: 14615563.

[23] Takemi Handa, Rajesh G. Katare, Shiro Sasaguri, and Takayuki Sato. Preliminary experience for the evaluation of the intraoperative graft patency with real color charge-coupled device camera system: an advanced device for simultaneous capturing of color and near-infrared images during coronary artery bypass graft. *Interactive CardioVascular and Thoracic Surgery*, 9(2):150–154, January 2009. PMID: 19423513.

[24] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, and Andrew Davison. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, page 559–568, 2011.

[25] Toshiyuki Kitai, Takuya Inomoto, Mitsuharu Miwa, and Takahiro Shikayama. Fluorescence navigation with indocyanine green for detecting sentinel lymph nodes in breast cancer. *Breast cancer (Tokyo, Japan)*, 12(3):211–215, 2005. PMID: 16110291.

[26] Andreas Kolb, Erhardt Barth, Reinhard Koch, and Rasmus Larsen. Time-of-flight cameras in computer graphics. In *Computer Graphics Forum*, volume 29, page 141–159. Wiley Online Library, 2010.

[27] Andrei A. Kolomenski. *Realization of a spatial augmented reality system – A digital whiteboard using a kinect sensor and a PC projector*. PhD thesis, Texas A&M University, 2013.

[28] Joy L. Kovar, Melanie A. Simpson, Amy Schutz-Geschwender, and D. Michael Olive. A systematic approach to the development of fluorescent contrast agents for optical imaging of mouse cancer models. *Biochemistry–Faculty Publications*, page 9, 2007.

[29] Jeff Kramer, Nicolas Burrus, Florian Echtler, Herrera C. Daniel, and Matt Parker. Introducing the kinect. In *Hacking the Kinect*, page 1–9. Springer, 2012.

[30] Antoine Lejeune, Sébastien Pierard, Marc Van Droogenbroeck, and Jacques Verly. A new jump edge detection method for 3D cameras. In *International Conference on 3D Imaging (IC3D)*, 2011.

[31] Yang Liu, Adam Q. Bauer, Walter J. Akers, Gail Sudlow, Kexian Liang, Duanwen Shen, Mikhail Y. Berezin, Joseph P. Culver, and Samuel Achilefu. Hands-free, wireless goggles for near-infrared fluorescence and real-time image-guided surgery. *Surgery*, 149(5):689–698, May 2011.

[32] Alain Luciani, Emmanuel Itti, Alain Rahmouni, Michel Meignan, and Olivier Clement. Lymph node imaging: Basic principles. *European Journal of Radiology*, 58(3):338–344, June 2006.

[33] Hope-Ross M, Yannuzzi La, Gragoudas Es, Guyer Dr, Slakter Js, Sorenson Ja, Krupsky S, Orlock Da, and Puliafito Ca. Adverse reactions due to indocyanine green. *Ophthalmology*, 101(3):529–533, March 1994. PMID: 8127574.

[34] Milton V. Marshall, John C. Rasmussen, I.-Chih Tan, Melissa B. Aldrich, Kristen E. Adams, Xuejuan Wang, Caroline E. Fife, Erik A. Maus, Latisha A. Smith, and Eva M. Sevick-Muraca. Near-infrared

fluorescence imaging in humans with indocyanine green: a review and update. *Open surgical oncology journal (Online)*, 2(2):12, 2010.

[35] Isao Miyashiro, Norikatsu Miyoshi, Masahiro Hiratsuka, Kentaro Kishi, Terumasa Yamada, Masayuki Ohue, Hiroaki Ohigashi, Masahiko Yano, Osamu Ishikawa, and Shingi Imaoka. Detection of sentinel node in gastric cancer surgery by indocyanine green fluorescence imaging: Comparison with infrared imaging. *Annals of Surgical Oncology*, 15(6):1640–1643, April 2008.

[36] D Murawa, C Hirche, S Dresel, and M Hünerbein. Sentinel lymph node biopsy in breast cancer guided by indocyanine green fluorescence. *The British journal of surgery*, 96(11):1289–1294, November 2009. PMID: 19847873.

[37] Richard A. Newcombe, Andrew J. Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. KinectFusion: real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, page 127–136, 2011.

[38] John C. Rasmussen, I-Chih Tan, Milton V. Marshall, Caroline E. Fife, and Eva M. Sevick-Muraca. Lymphatic imaging in humans with near-infrared fluorescence. *Current opinion in biotechnology*, 20(1):74–82, February 2009. PMID: 19233639 PMCID: PMC2692490.

[39] Michael J. Reinhardt, Claudia Ehritt-Braun, Dagmar Vogelgesang, Christian Ihling, Stefan Högerle, Michael Mix, Ernst Moser, and Thomas M. Krause. Metastatic lymph nodes in patients with cervical cancer: Detection with MR imaging and FDG PET1. *Radiology*, 218(3):776–782, January 2001. PMID: 11230656.

[40] Eva M. Sevick-Muraca, Ruchi Sharma, John C. Rasmussen, Milton V. Marshall, Juliet A. Wendt, Hoang Q. Pham, Elizabeth Bonefas, Jessica P. Houston, Lakshmi Sampath, Kristen E. Adams, Darlene Kay Blanchard, Ronald E. Fisher, Stephen B. Chiang, Richard Elledge, and

Michel E. Mawad. Imaging of lymph flow in breast cancer patients after microdose administration of a near-infrared fluorophore: Feasibility study1. *Radiology*, 246(3):734–741, January 2008. PMID: 18223125.

[41] Nobumi Tagaya, Aya Nakagawa, Akihito Abe, Yoshimi Iwasaki, and Keiichi Kubota. Non-invasive identification of sentinel lymph nodes using indocyanine green fluorescence imaging in patients with breast cancer. *Open Surgical Oncology Journal*, 2010.

[42] Susan L. Troyan, Vida Kianzad, Summer L. Gibbs-Strauss, Sylvain Gioux, Aya Matsui, Rafiou Oketokoun, Long Ngo, Ali Khamene, Fred Azar, and John V. Frangioni. The FLARE^{TM} intraoperative near-infrared fluorescence imaging system: A first-in-human clinical trial in breast cancer sentinel lymph node mapping. *Annals of Surgical Oncology*, 16(10):2943–2952, July 2009.

[43] N. Unno, M. Nishiyama, M. Suzuki, N. Yamamoto, K. Inuzuka, D. Sagara, H. Tanaka, and H. Konno. Quantitative lymph imaging for assessment of lymph function using indocyanine green fluorescence lymphography. *European Journal of Vascular and Endovascular Surgery*, 36(2):230–236, August 2008.

[44] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, page 666–673, 1999.