FCTUC

University of Coimbra
Faculty of Sciences and Technology
Department of Physics

Integration of optical tracking system for determination of the position and orientation of instruments during the surgery

Michel Antunes

Coimbra, 2008

Integration of optical tracking system for determination of the position and orientation of instruments during the surgery

Michel Antunes

A Thesis submitted for the degree of Integrated Master in Biomedical Engineering

Department of Physics

Faculty of Science and Technology, University of Coimbra

August 2008

Contents

List of Figures

List of Tables

Glossary

Anterior Cruciate Ligament (ACL)

One of the four major ligaments of the knee. Connects the femur to the tibia, providing stability to the knee and minimizing stress across the knee joint.

Application Programmer Interface (API)

Set of function declarations to support requests made by a computer

Augmented Reality (AR)

Field of computer research which deals with the combination of real-world and computer-generated data.

Computer-Aided Design (CAD)

Computer technology to aid the engineers and architects in the design of a variety of products.

Computer Assisted Surgery (CAS)

Surgery performed with the help of a computer system that aims to assist the practitioner and improve the accuracy

Charge Coupled Device (CCD)

Sensor technology, commonly used in conventional digital cameras, that measures the incident light intensity.

Computed tomography (CT)

Medical imaging method.

Dynamic-link library (DLL)

Microsoft's implementation of the shared library concept.

Digital Video (DV)

Video using digital signals.

Fundação para a Ciência e a Tecnologia (FCT)

Portuguese governmental institution responsible for financing the *ArthroNav* project.

Ground (GND)

Reference voltage in which other voltages are measured

General Purpose Input/Output (GPIO)

Device providing a set of ports that can be configure for either input or output. The communication with the Flea2 camera is achieved with a *GPIO* port.

Global Positioning System (GPS)

Navigation System

Input/Output (I/O)

Communication between a device and the outside.

Infrared Light emitting diodes (IRED)

Diodes emitting *IR* light when actvated.

Institute of Systems and Robotics (ISR)

Institution where the *ArthroNav* project is in development.

Minimalist GNU for Windows (MinGW)

MinGW provides a complete Open Source programming tool set which is suitable for the development of native Windows programs that do not depend on any 3rd-party C runtime DLLs.

Northern Digital Inc. (NDI)

Company where the optical tracking system was acquisted.

Optotrak Application Programmer Interface (OAPI)

Set of functions to support the interface with the opto-tracker

Object-Oriented Programming (OOP)

Programming which is oriented around objects, taking advantage of *Encapsulation*, *Polymorphism*, and *Inheritance* to increase code reuse and decrease code maintenance

Open Source Computer Vision Library (OpenCV)

Library with programming functions principally aimed at real time computer vision.

Open Graphics Library (OpenGL)

Cross-platform API to produce 2D and 3D computer graphics.

Printed Circuit Board (PCB)

Provides place to mount components,and means of electrical connection between the components, for electronic devices

RGB

Additive color model, adding red, green and blue light together.

System Control Unit (SCU)

Processing device that controls the operations of the Position Sensor and the Strober

Simple DirectMedia Layer (SDL)

Cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer

Total Knee Arthroplasty (TKA)

Replacement of the arthritic knee surface

Total Knee Replacement (TKR)

Refer to *Total Knee Arthroplasty*

Transistor-Transistor-Logic (TTL)

Class of digital circuits. Logic gating and amplification are both performed by transistors.

Vtable

C++ implements dynamic dispatch using virtual function tables. The dispatch process consists of loading the receiverŠs dispatch table, loading the function address by indexing into the table with the selector number, and jumping to that function.

Extensible Markup Language (XML)

A custom markup language.

Acknowledgments

During this year a lot of persons were very important to me. Some of them helped me to become a better engineer. Others helped by standing at my side during the pains of the process.

I want to thank my Advisors to trust me from the beginning and give me the chance to continue my work. My thanks to Prof. Joao Barreto for his constant motivation and help, and to Prof. Paulo Menezes for his numerous tips and suggestions.

The biggest acknowledgment goes to my girlfriend. This year was a hard one where I worked a lot. But she was always near when I needed, and gave me strength to keep going.

I want to thank Mom and Dad. Without them I could never be here. Thank you for all your efforts to provide me with the tools to build a better future.

I want to thank my Lab. mates Joao, Pedro, Hugo and Cristovao for being my project companions. Thanks also to Jose Pascoal for the PCB help.

I want to thank my sister and my friends. Thank you friends for being my friends. Sorry for the less attention this year.

Thanks to all that in one way or another were close to me during this year.

Abstract

The purpose of this project was the design and implementation of a complete system for acquisition and fusion of data provided by an optical tracking system and a set of cameras (including an arthroscope).

Synchronous data acquisition was a major requirement. For this purpose we designed a Printed Circuit Board (PCB), that assures perfect synchrony between equipments, and can be configured to use different trigger sources. In addition, we developed a complete software architecture in C++, that runs in a work-station and makes the communication with the opto-tracker and cameras. The software enables the insertion of different modules for data processing. We developed a Visualization Module using OpenGL and Qt, and a recording module that enables data acquisition for off-line processing in Matlab.

In future work, the navigation system will be extended with *3D* reconstruction/registration computer vision algorithms. These algorithms will use the intra-operative endoscopic video, enhanced with the pose information, for estimating the 3D pose of organs and anatomical structures in the knee. The final goal is to build a complete navigation system to assist surgeons during arthroscopic interventions.

Chapter 1

Introduction

The project described in this thesis is part of a bigger project, funded by Fundação para a Ciência e a Tecnologia (FCT), and named *ArthroNav - Computer Assisted Navigation in Orthopedic Surgery using Endoscopic Images*. The goal of the *ArthroNav* project is to enhance the surgeon's perception and enable precise navigation in the knee joint during the Anterior Cruciate Ligament (ACL) reconstruction.

The ACL connects the femur to the tibia, providing stability to the knee and minimizing stress across the knee joint. It is a common injury, that usually occurs by sudden change of direction (e.g. sport practice). The treatment of the damaged ligament involves minimal invasive surgical reconstruction. The injured ligament is removed, and drill guides are used to open holes in the tibia and the femur at the attachment locations of the original ligament. A ligament graft is passed trough the bone tunnel into the knee joint, and it is attached using screws. It is a difficult surgical procedure, requiring the surgeon to be able to navigate inside the knee joint using only endoscopic image feedback. Small errors in the graft placement lead to abnormal tensions during the movement of the knee, that can cause complete clinical failure. Statistics show that, in 50% of the cases, there are non negligible errors. The objectives of the *ArthroNav* project is to develop a computer navigation system to assist the practitioner, and improve the accuracy in the graft placement [3] [4].

1.1 Surgical Navigation

Surgical navigation stems from the demand of higher accuracy and reproducibility in a wide variety of medical interventions, allowing the surgeon to track the patient anatomy and surgical instruments in a three dimensional space. Real-time manipulation of the obtained positions, with fusion of image registration techniques, allow surgeons to localize surgical positions on preoperative images.

To better understand the need for a navigation system in the surgery room, it is possible to make an analogy with a car trip. Suppose we are at *city A* and want to arrive at *city B*. Being the first time that we go trough this path, we first look and analyze some maps of the region, and then start the trip. Without road signalization and no Global Positioning System (GPS), we know more ore less when to turn right, or in which city we will pass, but easily fall in disorientation. The principal problem of the trip is the doubt if we will arrive at the desired location. Using an updated GPS, we arrive at the exact same location where we wanted initially while looking the road maps.

So, the trip is the surgery, and *city B* is a surgical position where the medical intervention must be done. First, the surgeon analysis the anatomy maps, like Computed tomography (CT) images, and during the surgery tries to arrive at the planed *position B* (e.g. to open the hole for the graft introduction). The surgeon knows more or less, due to his experience, the location of *B*. However and since the incision must be made with sub-milimetrical precision, it is a very difficult task to accomplish using just the human eye without external support. This is the scenario where the surgical navigation has impact. The surgical navigation is like a GPS in the surgery room, tracking real-time positions of instruments and tools with respect to a target organ. The system will *tell* the surgeon how far he is from the desired location, and will supply all the necessary measurements to arrive at *point B*. It will give instructions to the practitioner (e.g. $1.1mm$ more right and $0.63mm$ more down). *Ironically*, it is possible to say that $1mm$ error in the surgery room is $1km$ mistake during the road trip. So, $1mm$ for the patient is a considerably distance, and has a strong repercussion in the success of the operation.

Today, Computer Assisted Surgery (CAS) is broadly used for navigation purposes. In orthopedics, it has been proven that a surgery with the assistance of a navigation system is much more accurate than a surgery using traditional techniques. CAS, beside improving the accuracy of the surgery, it also improves the reproducibility and allows surgeons with less experience to do the interventions without fear of making mistakes [5].

An example of a surgery using a navigation system is shown in Figure

1.1. The figure concerns the Total Knee Arthroplasty (TKA) procedure. The goal is to replace the knee joint by a prothesis. From a clinical perspective, it is fundamental a correct alignment between the prothesis and the femur head that is hidden in the hip. The alignment is achieved using optical tracking. The surgeon starts by determining landmarks in the knee joint using a digitizing touch probe over the surface of the femoral head. The acquired positions from the tracked probe are used to calculate the center of the femoral head via registration with a pre-operative model. Rigid attached bodies are also tracked, providing a reference frame for the calculation of the pose of the objects in displacement [6].

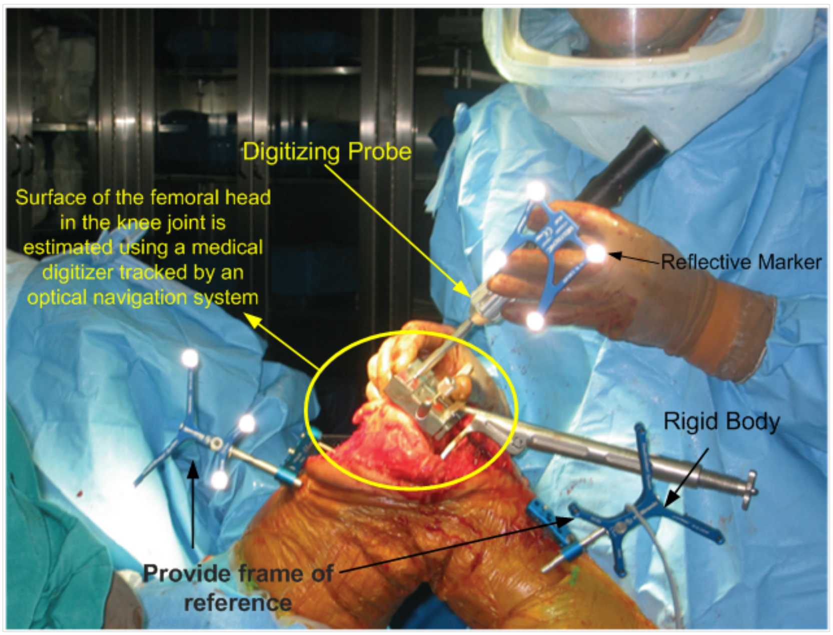

Digitizing Probe
Surface of the femoral head in the knee joint is estimated using a medical digitizer tracked by an optical navigation system
Reflective Marker
Rigid Body
Provide frame of reference

Figure 1.1: Total knee replacement surgery using navigation based on optical tracking. The opto-tracker is a stereo head able to estimate the positions of Infra Red *Markers* using triangulation. This enables the estimation of the 3D position and orientation of tools and organs [6].

1.2 Surgical Navigation in minimal invasive procedures (Arthoscopy)

Arthroscopy is a minimally invasive procedure that aims decreasing the incision size and minimize the time for surgery recovery. Large incisions are replaced by a couple of small incisions (surgery ports). One port gives access to an arthroscope, providing the surgeon with intra-operative endoscopic images. This enables the visualization of the interior of the knee joint on a video screen. Other ports are used to insert small surgical instruments, allowing the diagnosis of injuries, as well as the repair, reconstruction, or removal of damage tissue and organs.

The Arthroscopy is a difficult technique to perform. Visibility is very reduced throughout the procedure, and anatomic landmarks are difficult to localize. The surgeon has difficulty in perceiving depth, and can easily loose the orientation. As the success of the surgery depends on the accuracy and precision of the surgeon, the final clinical outcome can be considerably influenced by these problems. Computer systems for enhancement of surgeon's perception and precise navigation in the knee joint can improve the clinical success rate and decrease the practitioner training requirements. The development of such systems can bring great benefits in improving the life quality of patients [5].

The development of navigation systems for minimal invasive surgery is a difficult task. The main problem is the fact that tissues and organs can not be directly accessed. The Total Knee Replacement (TKR) surgery, described in the previous section, is an open procedure, where it is possible to attach markers to the bones. While in the TKR procedure the pose of the femur/tibia can be estimated by optical tracking, this is no longer possible for the case of the arthroscopic ACL reconstruction. The surgeon observes the cavity in an indirect manner using the endoscopic camera, and it is virtually impossible to attach optical markers that can be tracked from the outside.

The *ArthroNav* project aims to overcome this problem by employing computer vision techniques over the endoscopic video. The key idea is to employ optical tracking for the pose of the arthroscope and instruments (observable from the exterior), and estimate the rotation/translation of the organs with respect to the camera by registering pre-operative *3D* models to intra-operative images.

Our work concerns the hardware and software infrastructure that will support this research. The computer vision algorithms for reconstruction/ registration will require endoscopic video augmented with pose information provided by the opto-tracker. During the project we designed, developed and implemented a complete system, composed by a work-station connected to cameras and an optical tracker. The system was complemented by suitable hardware and software to enable synchronous acquisition of video and optical data.

1.3 The Project

The objective of this project was the development and validation of a complete infrastructure for acquisition and fusion of data provided by the opto-tracker and a set of N cameras (including the arthroscope). The final outcome was a system providing real-time video, enhanced with pose information about cameras and other tools.

In order to achieve the objective, the main project steps were:

- Familiarization with the equipment, including the study of the manuals for the Arthroscopic system, opto-tracker, and corresponding Application Programmer Interface (API).
- Specification, acquisition and installation of equipments (work-station, arthroscope, etc).
- Decision about operating system, development environment, and additional software tools (libraries).
- Integration of the opto-tracker and arthroscopic video system for synchronous data acquisition.
- Development of a supporting software infrastructure, enabling the future integration of different processing modules.
- Development of a software module for Data Recording
- Construction of a marker tool to be attached to camera and instruments, that minimize surgeon disturbance while keeping good line of sight.

These steps were all accomplished, except the construction of the marker tool. This is explained by the fact that several objectives took more time than initially expected, and also because new features were added during the project. The tool was not built, but the parameters involving the utilization of rigid bodies in surgical environment were studied, and an initial design was proposed. We hope to finish this part in a near future.

The accomplishment of the objectives must keep in mind the following important requirements :

- Synchronous Data Acquisition: This requirement is essential to combine the tracking system with the endoscope. The *ArthroNav* project will use the information from the arthroscope, enhanced with the pose information, to construct a complete surgical navigation system. If the data from different sources would not correspond to the exact same time instant, then this would be a major source of error, with direct impact in the accuracy and reliability of the final navigator.

- Real-Time: The final infrastructure will be used intra-operatively to assist the surgeons. This implies real-time information retrieval to the medical practitioner.

- Flexibility and Modularity: Future research within the *ArthroNav* project will rely on this infrastructure. Different approaches will need the acquisition of different data sets (e.g. variation of the frequency, different devices, different rigid bodies). Also, various processing algorithms will be produced and tested along the project. Therefore, the software infrastructure must be flexible, allowing re-configuration of the data acquisition, and easy insertion of user defined modules.

- The software should be portable between Windows and Linux operating systems.

Table 1.1 briefly overviews some of the solutions used to meet the requirements.

The created system makes the synchronous fusion of opto-tracker information and arthroscopic video, and allows its visualization and processing. The additional features added to the system during the project were:

- Implementation of time analysis mechanisms enabling the control of the acquisition rate. This is enables the double-checking of synchrony, as well as alerts for information loss or when information captured in different instances is shuffled.

- During the project, we realize the need of a third camera. This camera will allow video of the acquisition environment, which can be helpful for ground truth analysis for the algorithms that will be created.

Table 1.1: Solutions to meet the specified requirements

Requirements	**Solution Overview**
Synchronism	Design and implementation of PCB that synchronizes the different equipments using as trigger source the arthroscope, the opto-tracker, the *FireWire* camera, or an external push button.
Real-time	Careful programming in order to minimize latency.
Easy Re-Configuration	Achieved by using Extensible Markup Language (XML) parsing.
Easy addition of modules	Abstract classes were produced as input port for external modules.
Portability	The development environment was *Cygwin*, which should ensure easy portability.

- Development of a software module for visualization. An interface was designed, allowing an easy and user-friendly utilization of the navigation system. A *3D* animation of rigid bodies using the rotational and translational data from the optical tracking was also implemented. This animation allows the visualization of kinematics applied on rigid bodies during the real-time tracking.

Chapter 2

Equipment Description

The developed surgical system employs an optical tracking system, an arthroscope, a *FireWire* camera, and a work-station (PC computer). Table 2.1 gives an account of the relevant specifications of the opto-tracker, and the cameras.

At the beginning of the project, the Institute of Systems and Robotics (ISR) had already purchased an optical tracking system. We were the first that used this system in order to capture pose information. The opto-tracker is a stereo head able to estimate the positions of Infrared Light emitting diodes (IRED) by triangulation. The utilization of this equipment involved the study of its manuals, and its Application Programmer Interface (API).

An old endoscope was used for the first contact with medical equipment. To improve the capabilities of our research, a new endoscopic with surgical lens was acquired. The new endoscope has a camera with three *CCDs*, allowing the capture of high quality imagery.

A third camera was included in the setup. This camera will allow external processing, and will provide ground truth information for the evaluation of algorithms created during the future research.

In order to merge the information in real-time, a PC capable of a good performance was acquired. Table 2.2 provides the principal characteristics of the computer.

Table 2.1: Technical specifications of the equipment used in the navigation system.

Endoscope (3CCD) (Smith & Nephew 460H)		**Opto-tracker** (NDI CERTUS)		**PointGrey Camera** (Flea2-08S2C)	
Size (pixel)	576×720	Resolution(mm)	0.01	Size [a](pixel)	1032×776
Sensor	3 CCD 1/2	Accuracy (mm)	0.15	Sensor	ICX204 1/3
Output	DV-25	Operation Range(m)	$2.3 - 6$	Output	Digital Data [b]

[a]This is the maximal image size.

[b] 8, 16 and 24-bit digital data. The image data formats are : Y8, Y16, RGB, YUV411, YUV422 and YUV444

Table 2.2: PC Specifications

Processor	Intel Core 2 Quad
Processor Speed	2.4GHz
Memory	4Gb
Graphic Card	GeForce 8800 GTS 320Mb 500MHZ
Hard Disk	500Gb 16Mb 7200rpm

2.1 Opto-tracker

There are many surgical navigation systems based in electromagnetical tracking. The main advantage of this technology, when compared with opto-tracking, is that it does not suffer from line-of-sight problems. The main drawback is its susceptibility of interference to electromagnetical sources, resulting in measurement errors. [7]. To achieve the best accuracy for the estimation of surgical positions, an optical tracking system has been used. Our optical navigator is an *Optotrak Certus System*. However, and despite of the differences, the problem aimed by the *ArthroNav project*, which is image-based navigation, is in a large extent independent of the chosen of technology to track camera and instruments.

The opto-tracker is a position sensor, tracking motions of markers consisting of IRED. [1] The typical operation of the opto-tracker is shown in Figure

[1]The manufacturer supplies two different Optotrak Certus Systems. We chose the *E-Type* because it was designed with electromagnetical shielding, as specified by medical electrical equipment standards [1].

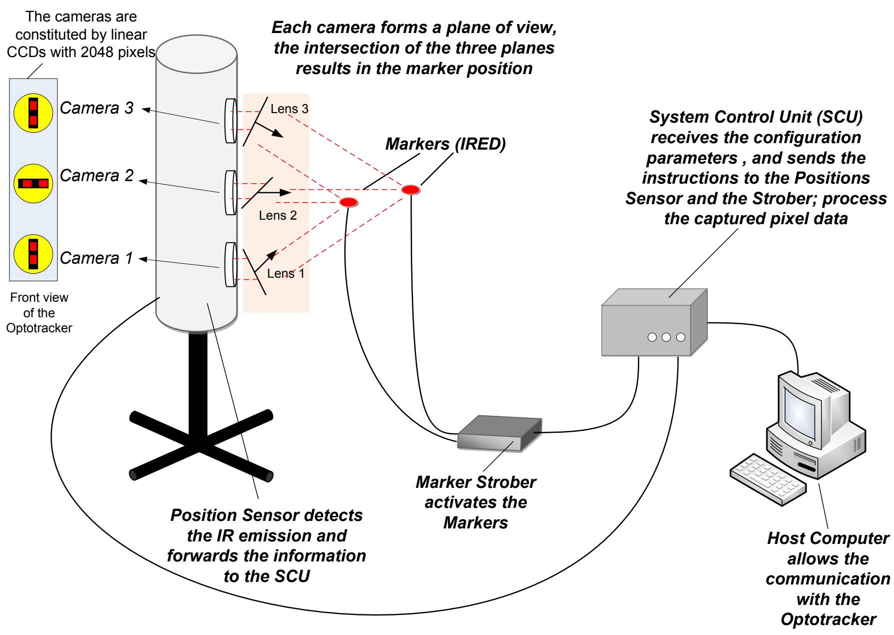

The cameras are constituted by linear CCDs with 2048 pixels
Camera 3
Camera 2
Camera 1
Front view of the Optotracker
Each camera forms a plane of view, the intersection of the three planes results in the marker position
Lens 3
Lens 2
Lens 1
Markers (IRED)
System Control Unit (SCU) receives the configuration parameters , and sends the instructions to the Positions Sensor and the Strober; process the captured pixel data
Marker Strober activates the Markers
Position Sensor detects the IR emission and forwards the information to the SCU
Host Computer allows the communication with the Optotracker

Figure 2.1: Optotrak Certus System

2.1. First, the Position Sensor is placed so that the detection region[2][3] incorporates all the field of interest, from where we want to obtain the marker positions. Then, the communication with the *Optotrak Certus system* is initialized. The communication can be made using the supplied software[4], or with an our own application developed using the Optotrak Application Programmer Interface (OAPI). The System Control Unit (SCU)[5] receives the configuration parameters from the host computer, processes the values, and sends the appropriate instructions to the Positions Sensor and the *Marker Strober*[6]. Once started the collection, the *Strober* activates the markers, and the Position Sensor detects the infrared light emission, forwarding the intensity information to the SCU. It is the *Optotrak System* that controls the activation order of the markers, being able to link the captured pixel data to the corresponding marker. The SCU processes the received raw pixel data[7]

[2]Volume in which the position sensor can detect a marker, defining the field of view of the Sensor

[3]The acquired Optotrak System is far focus, *seeing* from 2.2m until 6m

[4]*NDI First Principles* allows to capture,record and store data captured by the *Optotrak Certus System*.

[5]Processes the data received by the Position Sensor and forward the information to the host computer.

[6]Turn the markers on and off in response of the instructions supplied by the user.

[7]Subpixel positions from the Charge Coupled Device (CCD)'s

in their corresponding *3D* or *6D* positions if requested, and transmits the information to the host computer.

2.1.1 Position Determination

The operation of the Optotracker is based on three cameras, each composed by a linear Charge Coupled Device (CCD) with 2048 pixels. In front of the CCD, a cylindrical lens focuses all incoming light rays onto the linear matrix of the CCD, forming a *plane of view*. The intersection of the three *plane of views* results in the position of the marker.

A cylindrical lens has two perpendicular meridians, but just one has optical power [8]. The meridian parallel to the axis of the cylinder has zero refractive power, and the meridian perpendicular to the axis is the power meridian. Imagine the lens as if they were composed by thin horizontal sections. To each section corresponds a point in the *image line* that is right behind. The lens makes incident light rays, lying on the horizontal planes defined by the sections, to converge into the corresponding point of the *image line* (see Figure 2.2). From the coordinates of the point in the *image line*, it is possible the estimate from which horizontal segment it was formed, and so the vertical coordinates of the light source can be calculated.

Figure 2.1 shows that the CCD *one* and *three* of the opto-tracker are perpendicular to the CCD *two*. The cylindrical lens of the cameras *one* and *three* are oblique. As for each camera it is possible to form a *plane of view*, the intersection of the three planes gives the position of the light source, in this case, of the IR Marker [9]. Using the centroid [8] positions of the three linear pixel matrices, instead of the pixel with maximal intensity, it is possible to achieve sub pixel precision, and estimate more accurately the position of the IRED.

2.1.2 Pose Estimation

The opto-tracker captures light from the markers, and can transform the CCD readings of multiple markers into 3D (or pose) data[9]. In order to calculate the rotation and translation of an object with respect to a reference frame, it is necessary to use the concept of rigid body. Generically, a rigid body is an object in which the distance between any two points, and the

[8]The centroid position is the gaussian mean position of the intensities captured by the CCD

[9] The pose data refers to the rotation and translation.

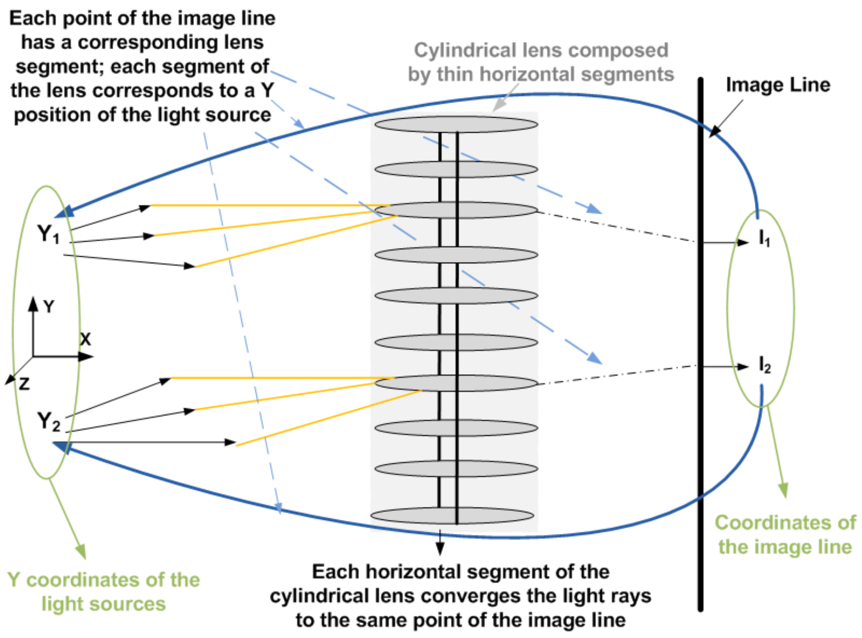

Each point of the image line has a corresponding lens segment; each segment of the lens corresponds to a Y position of the light source
Cylindrical lens composed by thin horizontal segments
Image Line
Y1
Y
X
Z
Y2
I1
I2
Y coordinates of the light sources
Each horizontal segment of the cylindrical lens converges the light rays to the same point of the image line
Coordinates of the image line

Figure 2.2: Cylindrical lens

angles between any two vectors do not vary in time (it is rigid). By tracking a certain number of points is is possible to compute the rigid motion undergone by the object. Or in an alternative manner, to compute the translation and rotation of a local reference frame, attached to the rigid body, with respect to a global fixed reference frame [10]. To calculate the transformations of a rigid body with the Optotracker, we need to observe a minimum of 3 markers at each frame instant.

The OAPI can return various formats for real-time rigid body motion. It's possible to work with Euler parameters[10], rotation matrices[11] and quaternions.[12]

A rigid body is characterized using the *NDI 6D Architect* Software supplied by Northern Digital Inc. (NDI). This software enables the creation of a file that describes each feature of a rigid body. This file contains the in-

[10]The Euler angles describe completely any rotation, the parameters are roll (R_z), pitch (R_y) and yaw(R_x).

[11]The rotation matrix is a 3×3 matrix $\mathbf{R}$, satisfying $\mathbf{R^T} = \mathbf{R^{-1}}$ and $det(\mathbf{R}) = 1$.

[12]Quaternion is a mathematical efficient notation used to represent the rotation of an object. Quaternions do not suffer from singularities and ambiguities, because they use four parameters to represent the three degrees of freedom of the rotation. A quaternion is represented by $Q = q_0 + q_x i + q_y j + q_z k$. The unit vector of the rotation is given by $v = (\frac{qx}{\sqrt{1-q_0^2}}, \frac{qy}{\sqrt{1-q_0^2}}, \frac{qz}{\sqrt{1-q_0^2}})$ and the angle of rotation is $\theta = 2 * \arccos(q_0)$.

formation that the system needs to determine the transformations. Such information includes the origin and orientation of a coordinate system local to the rigid body, the local coordinates of the attached markers, the normals to the markers and the tracking tolerances. Setting the tracking tolerances allows to specify the allowed maximum errors during the tracking of the rotational and translational movements.

2.1.3 API Description

The Optotrak System has an API with every necessary functions and routines needed to write custom application software. The OAPI was written in C, but it can be easily integrated under C++ encapsulation. This enables the creation of user defined wrapper classes that simplify its utilization. With the OAPI, it is possible to obtain the system status, and obtain either real-time data or buffered data in a blocked or non-blocked manner[13]. The OAPI comes with useful sample programs, with which is possible to learn how to use the opto-tracker.

Rigid bodies can be defined either by the above described rigid body files, or by an array of 3D coordinates with the locations of the IRED markers in a local reference frame. There are also two useful routines for mapping coordinates between reference frames: *OptotrakChangeCameraFor* and *RigidBodyChangeFor*. The former allows to change the global coordinate system, the latter allows to set a new global reference frame, so that the transformations are expressed in a coordinate system based on the position and orientation of another rigid body.

There are different methods to receive the rigid body data: *DataGetLatesttransforms* retrieves the latest frame of the *6D* transformations. The problem is that the same data can be read more than once, in the case of the request being faster than the acquisition frequency. This can be however be controlled by checking the number automatically associated to each of the frame; *RequestNextTransforms*, *DataIsReady*, and *DataReceiveLatestTransformations* allow request and receive of non-blocking data. The routine used in the designed application was *DataGetNextTransforms*. This function has a blocking behaviour and retrieves every time a new frame of data. We chose this mode, mainly because the capture routines of the other devices also acquire in this manner, allowing an uniform main loop design.

[13]Blocking means that the system waits for a response which can lead to a stall

2.2 Arthroscope

An arthroscopic camera is a conventional CCD camera to which is coupled a special lens system (the rigid scope or arthroscope). The scope is inserted through a small incision into an anatomic cavity (in our case the knee joint), enabling the inspection and visualization of its interior. The arthroscopic system allows the surgeon to diagnose and treat injuries in a minimal invasive manner.

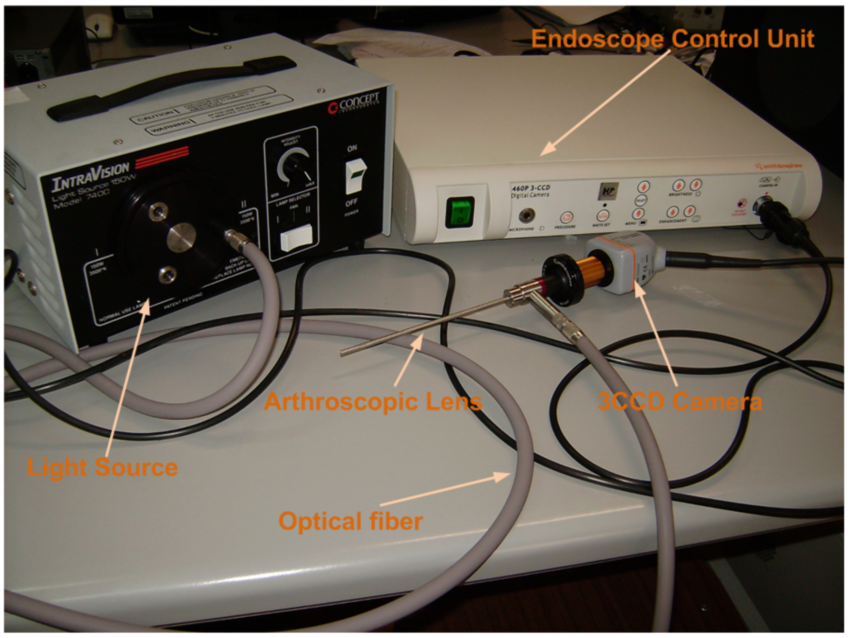


Figure 2.3: The complete endoscopic system

The complete arthroscopic system is shown in Figure 2.3. The camera coupled to the arthroscope contains three CCDs [14], allowing high quality image acquisition [11]. The Camera Control Unit used is a *Smith Nephew 460P 3-CCD Camera Control Unit*. The Control Unit provides different video outputs: composite video, analogic RGB, digital DV-25 (compressed), and S-Video.

During of our development we made use of the DV-25 and composite video. The image acquisition was done using the Digital Video (DV) Output, which provides compressed digital video through a *IEEE1394* interface (the

[14]Normally, digital color cameras use a *Bayer filter* over a single CCD. Each square of four pixels has a filtered red pixel, a filtered blue pixel, and two filtered green pixels. Using a three CCD camera and a beam splitter, each CCD responds to a particular color, achieving better color separation and higher quantum efficiency.

same interface used by the *FireWire*, standard). The composite video, as we will see ahead, was used as synchronization source because unfortunately commercial endoscopic systems do not have External Synchrony input.

2.3 FireWire Camera

To support the development of the navigation system, a second camera was included in the project. This camera will be used for ground truth analysis along further project steps (evaluation of created algorithms). The support camera used in the project is a *Point Grey Flea 2*. This camera contains a powerful API [2], allowing the design of user defined applications. The communication is along a *IEEE1394* b connector, that with its 800Mb/s, enables full frame rate RGB image transmission. Another advantage for using this camera is the availability of software drivers for Microsoft Windows, Macintosh and Linux (portability).

Chapter 3

Hardware for the Synchronization

In this chapter, the implementation of the synchronism in the navigation system will be described. The circuit will be analyzed, and the resulting construction of the board will be discussed.

3.1 Synchronization's Problem

An essential requirement of the navigation system is the precise synchronism of their components. We cannot use the information from the devices together, if the capture is not done in the same time instant. It is like the trip by car. We see the exit toward *city B*, but the GPS informs us to take the exit when it has already passed. In this situation, the GPS becomes useless. The navigation system can show the surgeon the correct localization for the incision during the surgery. However, if the surgeon is informed after the operation, that the frames shown on the monitor do not correspond to the navigation information, he will certainly be afraid about the patient.

The SCU of the opto-tracker has an external *Synchronization Port*. This port can be used to either trigger the opto-tracker for frame capture, or to extract the acquisition rate signal and use it to trigger other devices. The *Synchronization Port* uses Transistor-Transistor-Logic (TTL) Input/Output (I/O) signal logic. Figure 3.1 shows the timing diagram of the output signal during an optical tracking using two markers. The *Frame Clock Output(FR OUT)* signal contains the pulses generated every time a frame is captured by the opto-tracker, and can be directly used as synchronism source. In addition there is an input to for external triggering under high to low transitions. The voltage levels for this purpose should be :

$$2.0V \leq V_{IH} \leq 5.0V \tag{3.1}$$
$$0V \leq V_{IL} \leq 0.8V \tag{3.2}$$

The Table 3.3 in section 3.3 provides a full account of the signals involving the opto-tracker.

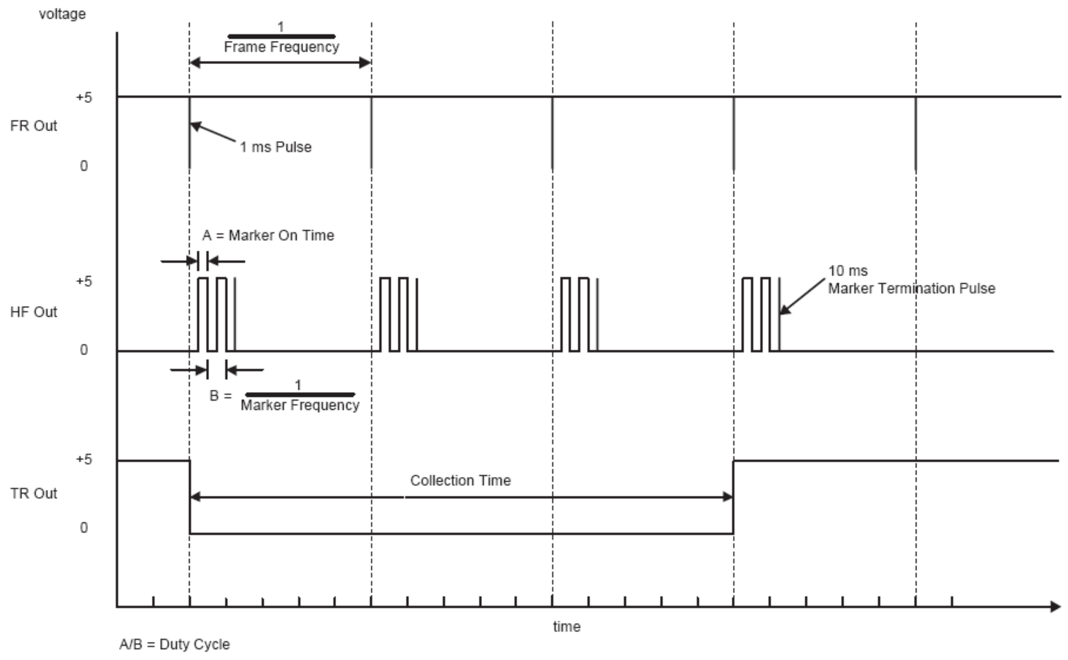


Figure 3.1: Output signals of the Opto-tracker Synchronization Port [1]. (FR OUT) - Frame Clock Output, (HF OUT) - Marker Activation Signal, and (TR OUT) - Start/Stop Collection [1]. Consult Table 3.3 for a complete description.

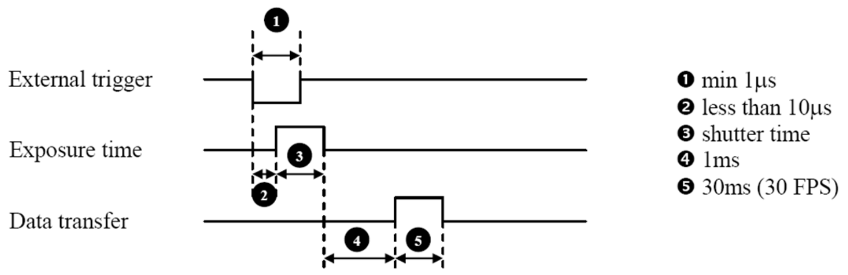


Figure 3.2: External signals from the Flea2 Synchronization Port [2]

The Flea 2 camera used for environment observation also has an input/output pin, allowing bidirectional synchronism using TTL signals. Figure 3.2 shows the trigger timing of the camera. The connector pin accept $3.3V$ TTL signals for the capture of frames. The API of the camera allows to set various trigger modes, depending on the response that should be obtained. We used *Trigger Mode 14*, allowing the overlapping of the exposure with the data transfer. This mode enables the triggering of the camera at faster frame rates than the standard trigger response.

The main challenge to achieve full synchronization was the arthroscopic system. It has a lot of output ports, capable of supplying multiple video formats. However, it does not accept external synchronism signals, which means that it is impossible to trigger this device for the frame capture.. [1]. Thus, to achieve synchronous acquisition between opto-tracker, Flea 2 and arthroscope, the unique possibility is to use one of the arthroscope video outputs as triggering source. The signals that the Camera Control Unit can supply were all analyzed. The solution found, was using the composite video signal as explained in the next section.

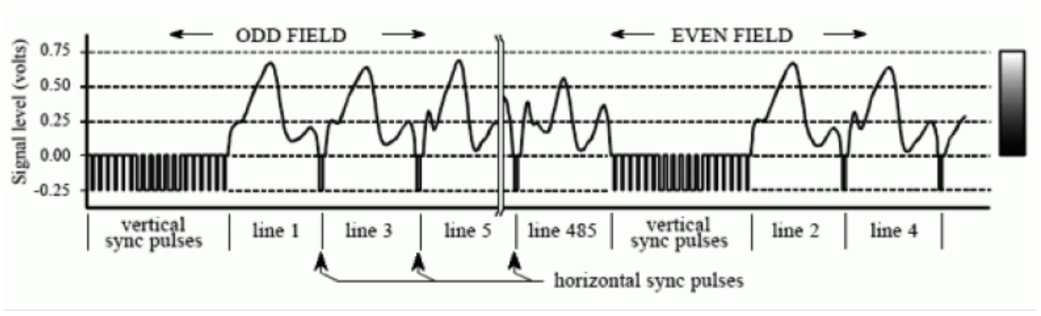
3.2 Synchronization's Solution

The solution found to synchronize the system, was the extraction of the timing signals from the composite video output of the arthroscope (a pulse for each start of frame).

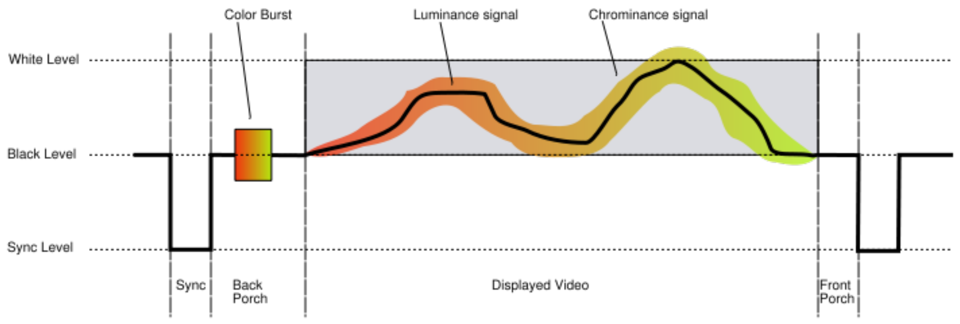
Composite video is an analog video interface that carries all the information necessary to display a 2D picture. The frames from a composite video

[1]The Arthroscope has just a bidirectional port that enables to call camera unit functions for the voice control.

signal are composed by two fields, as shown in Figure 3.3(a). [12]. The composite video signal transports the luminance [2], the chrominance [3], and the synchronization information [4] modulated in a single signal [13]. Figure 3.3(b) shows a video line, in which the *color burst* is the reference signal to decode the color information. The synchronization part of the signal is composed by the horizontal and vertical pulses, the beginning of a horizontal line scan, and the beginning of an even/odd field scan, respectively 3.3(a).



(a) Composite Video Signal [14]



(b) Composite Video Line [15]

Figure 3.3: Composite Video

In order to extract the timing information, the *LM1881* Video Sync Separator was used. As input the *LM1881* receives the composite video signal, and as output it extracts the signals shown in Figure 3.5. Each *Vertical*

[2]Intensity information of the video image, brightness or darkness.
[3]Color information of the video image.
[4]Horizontal and vertical scan signals.

Output Pulse corresponds to the beginning of a field.To obtain the beginning of each frame, it is necessary to separate the odd fields from the even fields. The separation is achieved indexing each vertical pulse with the corresponding *odd/even* signal.

We must capture the vertical signal that coincides with the start of an odd field. Fur this purpose, the logic circuit contains a *NOR* gate, as shown in Figure 3.4. First, the *odd/even field* signal is inverted. Then both signals, the inverted signal and the *vertical* output signal, are given as input to a NOR Gate [5]. The output is the frame signal, that can now be used to trigger the cameras, and synchronize the complete navigation system.

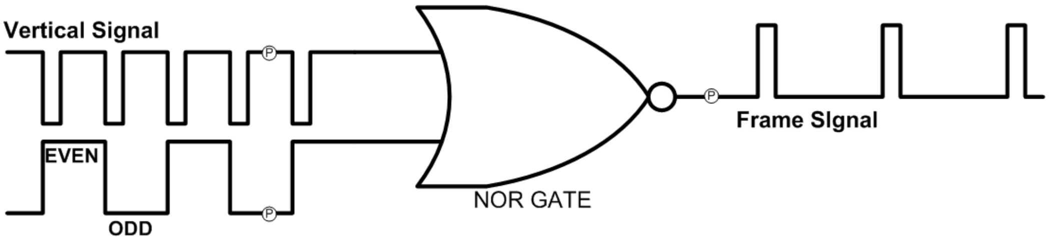


Figure 3.4: Logic circuit for frame signal extraction

3.3 Hardware specification

In order to achieve the synchronism of the devices, we decided to construct Printed Circuit Board (PCB) with all the necessary electronics. As discussed above, in normal circumstances the arthroscope has to be the source of synchronism. However, and in order to maximize the flexibility of the created system, we decided to build a PCB that allows the user to choose between different trigger sources.

The final design of the PCB is shown in Figure 3.6. The various synchronization modes are represented in Table 3.2, where the triggering source can alternate between the arthroscope, the opto-tracker, the Flea 2 or an external push button (single frame acquisition). A *multiplexer* selects the trigger signal based on the *DIP* switch combination configured by the user. The power supply is done through *USB* connector plugged to the work station. In Table 3.2, the arthroscope appears two times as trigger source. This done because we were not sure that for all cameras the same field comes first.

[5]The output of NOR GATES with two inputs is *true* if both inputs are *false*; else, the output is *false*.

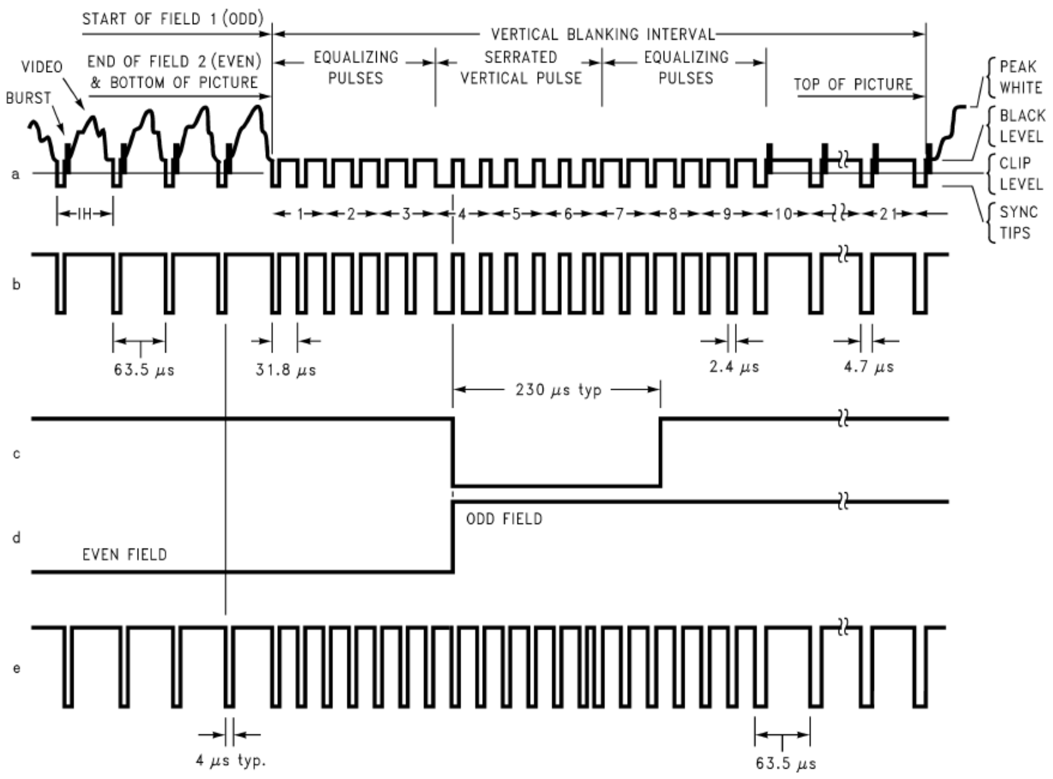


Figure 3.5: (a) Composite Video; (b) Composite Sync; (c) Vertical Output Pulse; (d) Odd/Even Field Index; (e) Burs Gate/Back Porch [16]

Thus we made the two modes available. The user can choose which is the first field of the resulting vertical frame signal, the odd or the even field. In Table 3.1 are the various components used in the board represented.

The *opto-couplers* between the *multiplexer*, the opto-tracker and Point-Grey Camera, were use as a preventive measure. They assure isolation between circuits and protect expensive equipment. As we do not know the internal circuits of the different equipments, the *opto-couplers* were used to overcome possible overvoltages, separating the camera grounds from the ground of the remaining board [6].

In this section, the synchronization ports of the devices, and the different synchronization modes of the board will be briefly analyzed.

[6]We had a problem with the Synchronization Port of the Optotracker, this because incorrect voltages were put in wrong pins.

Table 3.1: The components of the SyncBoard

Component	**Description**
NE555N	Timer for stable and controlled pulses
SN74LS00N	Quad two input *NAND* Gates[a]
SW DIP-4	DIP switch containing four switches
MHDR2X2	Simple connector with four inputs
LM1881N	Video Sync Separator
SN74LS02N	Quad two input positive *NOR* Gates[b]
SN74HC7266D	Quad two input *Exclusive NOR* Gates[c]
1-353576-1	*USB* connector
LM7805CT	Positive Voltage Regulator ($5V$)
D Connector 9	*DB-9* Connector
SN74LS253N	Multiplexer with three state outputs
Optoisolator	*PC817* Photocoupler

[a]The output of NAND GATES with two inputs is *false* if both inputs are *true*; else, the output is *true*.

[b]The output of NOR GATES with two inputs is *true* if both inputs are *false*; else, the output is *false*.

[c]The output of Exclusive NOR GATES with two inputs is *true* if both inputs have the same logic value; else, the output is *false*.

Table 3.2: The Synchronization Modes of the SyncBoard. The mode is selected using the correct *DIP* switch combination.

DIP Switch				SN74LS00N	SN74LS253N			Trigger
SW1	SW2	SW3	SW4		A	B	Output	Source
On	OFF	OFF	X	High	Low	High	1C2/2C2	Optotracker
OFF	ON	OFF	X	High	High	Low	1C1/2C1	*PointGrey*
OFF	OFF	ON	X	Low	Low	Low	1C0/2C0	PushButton
OFF	OFF	OFF	High	High	High	High	1C3/2C3	Arthroscope[a]
OFF	OFF	OFF	Low	High	High	High	1C3/2C3	Arthroscope[b]

[a]The first field of the complete frame is the *even field*.

[b]The first field of the complete frame is the *odd field*.

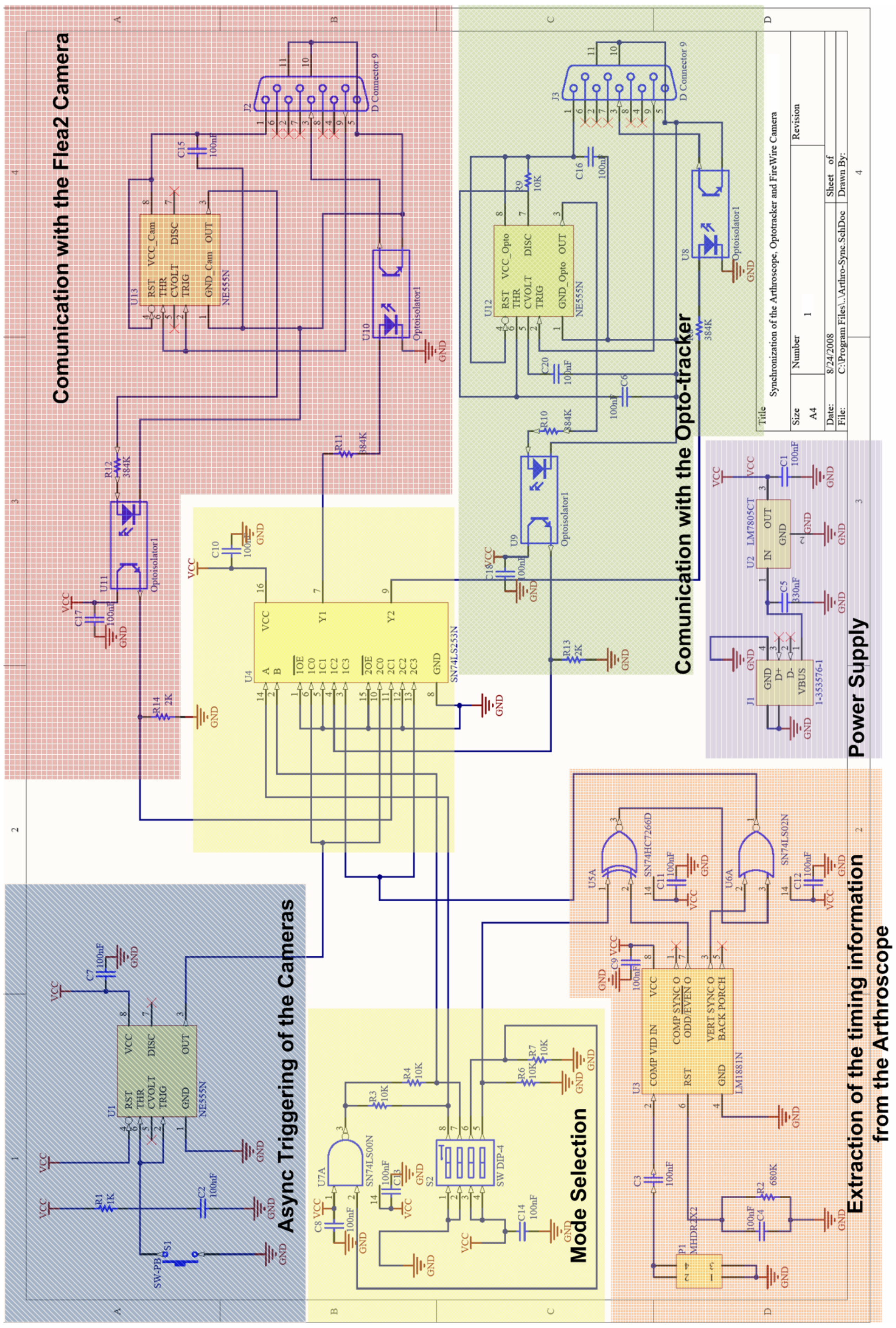

Comunication with the Flea2 Camera
Comunication with the Opto-tracker
Power Supply
Async Triggering of the Cameras
Mode Selection
Extraction of the timing information from the Arthroscope
Synchronization of the Arthroscope, Optotracker and FireWire Camera
NE555N
SN74LS253N
LM1881N
SN74HC7266D
SN74LS02N
SN74LS00N
LM7805CT
Optoisolator1
D Connector 9
SW DIP-4
SW-PB
8/24/2008
C:\Program Files\..\Arthro-Sync.SchDoc

Figure 3.6: The final circuit for the SyncBoard

3.3.1 Opto-tracker Synchronization

The Synchronization Port of the opto-tracker is a *DB-9* connector located at the back of the SCU. The description of the allowed pins is in Table 3.3, and the corresponding pin numbers are shown in Figure 3.7(a). The output and input signals have the same TTL logic. The diagram in Figure 3.1 shows the form of the output pulses[7]. The pins used for the synchronism were *pin 1* and pin *5*, allowing the power supply of the components in contact with the *Sync Port*, and pins *3* and *9*, allowing the bidirectional triggering.

The *FR OUT* is an active low, open collector output. It can provide until $150mA$ through an external pull-up resistor, connected to V_{CC}. V_{CC} cannot exceed $15V$. The pull-up resistor should be chosen so that:

$$R_{pull_{up}} > \frac{V_{CC}}{0.15}. \tag{3.3}$$

Before the *FR Out* signal is received by the *optocoupler* and transmitted to the multiplexer, it passes trough a pulse modulation component. The configuration of the *NE555* set, allows the output of a fixed pulse whenever the trigger voltage falls below $V_{CC}/3$. The pulse duration will be $t_{pulse} = 1.1 \times C_6 \times R_9$ [8].

The *FR IN* pin is pulled internally high by an $1k\Omega$ pull-up resistor connected to $5V$. This input is triggered by a high to low signal. The TTL levels are represented in Equation 3.1 and 3.2. The frequency supplied as input should not exceed the threshold of:

$$Freq_{max} = \frac{MarkerFrequency}{Number\ of\ Markers + 2}, \tag{3.4}$$

the maximal frame frequency that the opto-tracker can produce [9] [1].

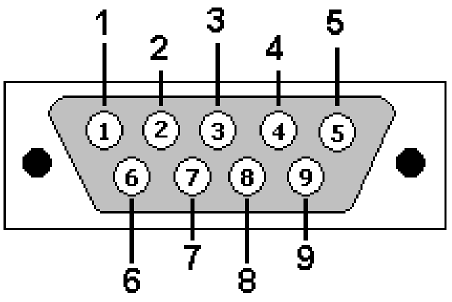
3.3.2 PointGrey Camera Synchronization

The *Flea2 PointGrey* Camera has an General Purpose Input/Output (GPIO) connector on the back, that can be configured to receive or send signals. The pin definitions are shown in Table 3.4, and the corresponding

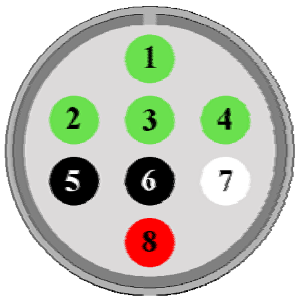
[7]The input signals are equivalent.

[8]$C_6 = 100nF$, $R_9 = 10K$; $t_{pulse} = 1.1ms$

[9]The maximum Marker Frequency of the *Optotrak Certus System* is $4600Hz$.



(a) DB-9 Optotracker connector



(b) GPIO Flea2 connector

Figure 3.7: The Trigger connectors of the cameras

Table 3.3: Opto-tracker Sync Port Pin Description

PinNumber [a]	Value	Description
1	$+5V$	SCU alimentation : $200mA$, $5V$
3	FR IN	Frame trigger input (high to low)
5	GND	Ground
6	HF OUT	Marker activation signal(active high)
7	TR IN	Trigger starts collection (high to low)
8	TR OUT	Start/stop collection(active low)
9	FR OUT	Frame clock output(active low)

[a]The missing pins are unknow and reserved by NDI

connector is shown in Figure 3.7(b). The operation of the CCD in response of the trigger signal is represented in Figure 3.2. The pins used were *pin 5* and *pin 8* to power the components in contact with, and *pin 1* and *pin 2* for the communication.

The I/O pins are internally pulled high using a weak pull-up resistor, allowing easy triggering by shorting the pin to Ground (GND). The configured output pin sink until $150mA$. The trigger signal of the camera passes trough a *Schmitt Trigger*, allowing the signal to be increased from a $3.3V$ TTL signal to a $5V$ TTL signal, maintaining a clean trigger pulse.

3.3.3 Arthroscope as Trigger Source

As explained before, to extract the timing information from the Arthroscope, the principal component is the *LM1881*. The final design allows to

Table 3.4: *GPIO* Flea2 Pin Description

PinNumber	Value	Description
1	IO_0	*IO* default trigger source
2	IO_1	*IO*
3	IO_2	*IO* RS232 Transmit (TX)
4	IO_3	*IO* RS232 Receive (RX)
5	GND	Ground
6	GND	Ground
7	V_{EXT}	Power camera externally
8	$+3.3V$	Power external circuit up to $150mA$

set if the signal who carries the *odd/even* index should be inverted or not. If selecting the arthroscope as trigger source, and putting the fourth switch to *OFF*, the index signal will be inverted in the *Exclusive NOR* Gate. This implies that the output of the *NOR* Gate contains the timing information of the start of a frame, in which the first field is *odd*. If the fourth switch is set to *ON*, the signal passes the *Exclusive NOR* Gate without being altered. The final synchronism signal correspond to a frame with an even field as start frame. The output signal of the logic comparison, in which the first field is an odd field, is shown in Figure 3.4.

3.3.4 PushButton Trigger

The *PushButton* on the board allows asynchronous triggering of the Optotracker and the PointGrey Camera (a frame is acquired each time the button is pressed). As explained before, it is not possible to do external trigger the arthroscope. However, this functionality can be useful in situations where the camera is kept static (e.g. calibration data). This because, as the camera does not move during the capture, the information will be the same in every instant, allowing a correct fusion of the acquired data.

The pulse from the PushButton passes first trough an *Schmitt trigger* before it reaches the devices. The *Schmitt trigger* outputs a clean and square inverted signal from the input pulse, and acts as a *debouncing* circuit for the trigger signal. The resulting output signal of the *Schmitt trigger* is :

$$V_{CC}\ (Output\ High)\ if\ V_{input} < \frac{1}{3}V_{CC}\ (Input\ Low) \tag{3.5}$$

$$0\ (Output\ Low)\ ifV_{input}\ > \frac{2}{3}V_{CC}\ (Input\ High) \tag{3.6}$$

3.4 PCB design

Initially, the schematic design and the layout for the Printed Circuit Board (PCB) of the synchronization circuit, were made using the *OrCAD Capture and Layout* software, respectively. After trying a few tutorial examples using this software, and exchanging ideas with experienced PCB designers, we decided to switch for*Altium Designer* as the Computer-Aided Design (CAD) software for the board design.

3.4.1 OrCAD for the board design

The first attempts in PCB design were made using *OrCAD*. With *OrCAD Capture* the schematic of the board is created, and the *parts* and *netlists* are imported to *OrCAD Layout*, where the final PCB is designed. The procedure is as follows:

1. Make the circuit schematic using *OrCAD Capture*
2. Generate a *Layout* netlist
3. Start *Layout* and select a PCB technology template
4. Import the generated netlist to *Layout*
5. Position the parts within the board outline
6. Route the board
7. Generate the files to manufacture the PCB

This software involves the use of to different applications for the board design. It is not easy to maintain the schematic design and the PCB layout synchronized. The second drawback is related with the footprints. It is a difficult task to find the necessary footprints in the library, as they are sorted by inadequate names. This made us switch to *Altium Designer*.

3.4.2 Altium Designer for the board design

Altium Designer is much more intuitive than *OrCAD*. It contains a very helpful documentation to start learning PCB design. The characteristic that we appreciated more, was the possibility of designing the complete project in the same application. This allows maintaining the schematic and the PCB layout perfectly synchronized, enabling comparisons and matches between them at each design stage. The footprint library is very complete, and the required components are easy to find.

Since we do not have experience in soldering, the design was done in order to simplify the subsequent construction and assembly. The principal characteristics of the PCB are:

- The initial PCB has just one layer. To achieve the design of a smaller board, two layers were used.
- Ground planes were set in order to decrease the necessary number of tracks and vias.
- Initially, auto-routing was used. This feature does not lead to an adequate space occupation, and the tracks are not designed parallel to each other. The final board was routed manually, using interactive routing, to overcome the auto-routing limitations.
- 90 Degrees angles of tracks were avoided. The circuit does not contain high frequency signals, but is a good practice to keep a fluid current.
- Tracks were not designed under the components. This makes the solder of the components much easier.
- The principal design rules are shown in Table 3.5 [10]. The rules were strictly followed, in order to simplify the process of construction and assembly.

Table 3.5: Principal design rules of the *PCB*

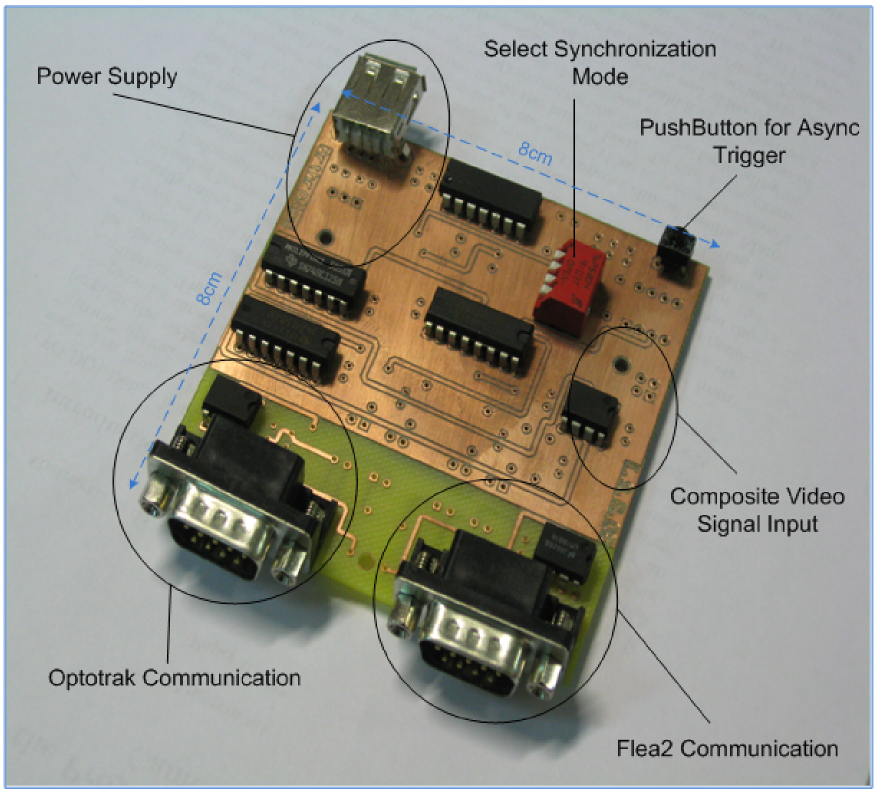
Design Rules	**Constraints**
V_{CC} and GND track width	$40mil$
Track width	$24mil < Width < 30mil$
Minimum Clearance	$26mil$

The manufacturing of the PCB was also done in the ISR. The board was printed using mechanical techniques.

[10] $1mil = 0.001inches = 0.0254mm$

3.5 Evaluation of the Final SyncCard

The construction of the board is now in progress. Figure 3.8 shows the board with the first components soldered. The construction is achieved using a normal solder station, and the board is continually tested against short circuit.



Power Supply
Select Synchronization Mode
PushButton for Async Trigger
8cm
8cm
Composite Video Signal Input
Optotrak Communication
Flea2 Communication

Figure 3.8: The first components soldered on the *SyncBoard*

As the board is under construction, the operation of the circuit was tested using a *BreadBoard*. The principal mode of the board is the triggering of the opto-tracker and the Flea2 camera with the arthroscopic frame signal. It is this configuration that will be used in the surgical environment. The results of the synchronism, using the vertical pulses from the composite video, are shown in Figure 3.9. The extracted frame pulse initiates at the first serration pulse [11] of the vertical synchronization signal. These signal is given as frame capture pulse for the Optotracker and the Flea2 camera, and their response

[11]The serration pulses of the video signal are *equalization pulses* to facilitate the monitor to enter and to leave the vertical sync signal [17].

is observed in order to evaluate the synchronism. The two bottom signals of Figure 3.9 are the output frame signals of the devices. They are contained in the duration of the vertical pulse. So, the hardware implementation of the synchronism with the arthroscope as trigger source can be considered enough accurate for our application needs, and can be used without to worry concerning hardware delays.

The other synchronization modes follow the same efficiency as the arthroscope as trigger source. This enables the use of the board for the navigation system, proving the synchronism of the involving devices using different trigger sources.

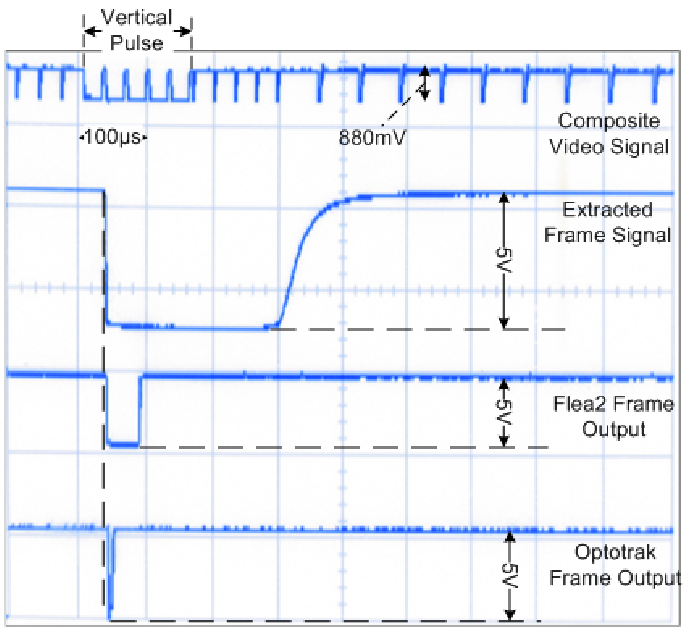


Figure 3.9: Arthroscope as Trigger Source. The response of the opto-tracker and the Flea2 camera is contained in the duration of the vertical pulse.

Chapter 4

Software

In this chapter, the complete design of the software for the navigation system will be explained. It allows the acquisition of frames following the hardware timing explained in the last chapter. Th captured information can be processed in real-time, visualized, and stored for offline investigations.

4.1 General Objective

The navigation system that will be created, should allow the following features:

- Integration of the information acquired from the cameras during an arthroscopy
- Process the information in real-time, and allow the surgeons to be guided based on it
- The acquisition should follow the hardware timing, in order to create an Augmented Reality (AR) for the surgeon

It was necessary the design and implementation of a software infrastructure, that accomplishes this requirements in order to be used in the surgical environment. This application will allow to use a normal computer in the surgery room, that captures the information from the complete acquisition system, processes the information intra operatively, and shows the result on a monitor to assist the practitioner during the surgery.

These system will be used in investigations based on computer vision, besides the design for an adequate surgical visualizator and GPS, it also should allow to be expanded with necessary modules for the research purpose.

4.2 Development environment

We start to work with the Optotracker under *Linux (Suse 10.1)*, but encounter a lot of errors while compiling our samples using the OAPI. Contacting the NDI, we were informed that the Optotrak drivers just work under a few older versions of Linux, and the new drivers for recent versions will be available in a few months . In this situation, the two options to solve the problem were:

- Downgrading to a older recommended version of *Linux*. Not a good idea if we want to have our application updated.
- Choose a windows platform.

The OAPI works fine under Windows and Linux. Also do the API of the *Flea 2*. To use the arthroscope, and after a long search on the internet, the PortVideo Framework [1] was found. It is a open-source cross-platform framework that provides access to cameras for video processing and display. It contains libraries for Windows, Linux and MacOS X. The technique used by this framework, is based on a simple wrapper class that communicates with various platform dependent subclasses. The communication with the devices is so achieved in a platform independent manner.

The final decision for the development environment is to work under Windows using Minimalist GNU for Windows (MinGW) [2], that contains a complete Open Source programming tool set, including the *GCC* compiler. In order to achieve portability requirement, the *GNU build system*, also know as *autotools*, was used. The main goal of using the *autotools*, is that it allows the developer on writing the programs, while he solves the problems of portability across Unix and Windows systems. To use this build system, we incorporate *Cygwin* [3] in our development environment. *Cygwin* consist of two parts, a Dynamic-link library (DLL) which acts as a Linux API emulation layer providing substantial Linux API functionality, and a collection of tools which provide Linux look and feel, containing the required *autotools*. So, as

[1]For further reading consult www.reactable.iua.upf.edu/?portwideo
[2]For further reading consult the main page www.mingw.org
[3]For further reading consult the main page www.cygwin.com

we just need the *autotools* for the development of our application, we will compile our applications without the *Cygwin's* DLL [4].

The development languages chosen are *C/C++*. The *C* part gives the necessary low-level capability for the interface with the devices. The *C++* part instead, allows to *encapsulate* data and functionalities using classes, and to use features like *Inheritance* and *Polymorphism* [5], besides other features, in order to allow *Reusability* and *Extendibility* of the code.

4.3 Requirements

For the validation of the navigation system in the surgery room, and efficient support in theoretical analysis based on computer vision algorithms, there are various important requirements that must be accomplished.The Software requirements of the navigation system are:

- It is important that the application used for the navigation system is of easy configuration. The user should be allowed to choose between different device combinations, synchronization sources, and acquisition parameters. In base on this configuration, the application must do the capture in conformity with the hardware timing.

- The acquisition of the navigation system should be implemented in real-time. It is essential that all the time latencies are minimized. The objective is not to acquire at the maximum rate at which the devices can work. Important is that the frames are processed intra-operatively, in order to support the surgical intervention. For the surgeon, real-time means that the processed frame shown on the monitor corresponds to image seen by the arthroscope.

- The navigation system must allow the user to store the information in a required format, modalities like continuous recording and storing on demand of frames should be allowed. This feature is important for analysis of intra operative situations. It allows to test the system in a real environment, that can later be evaluated using the stored data.

[4]This is achieved using the flag *-mno-cygwin*

[5]*Inheritance* is a form of Software reusability, where new classes are created from existing classes by owning their attributes and behaviors, and overriding or improving these with capabilities that the new classes require.*Polymorphism* enables to write programs in a general fashion to handle a wide variety of situations [18] [19].

- The system will be used by a few investigators, with various objectives, in the future. It should not be a closed application. It must have input gates to introduce processing and visualization capabilities, without the need to recompile and change the base system every time.

- Timing information should be analyzed by the system during the acquisition, in order to inform the user if the frames correspond to the same capture time, or if errors may have happened.

4.4 Application's Architecture

A simplified sequence of the final application's architecture is shown in Figure 4.1. The user writes the *configure.xml* file, and introduces processing, storing and/or visualization modules into the *ArthroNavMain* component. The application configures the acquisition according the *configure.xml* file, and call's the virtual methods contained in the introduced modules.

For the complete description of the Software created during the project, consult B. This appendix contains the documentation of the code produced durign the project [6].

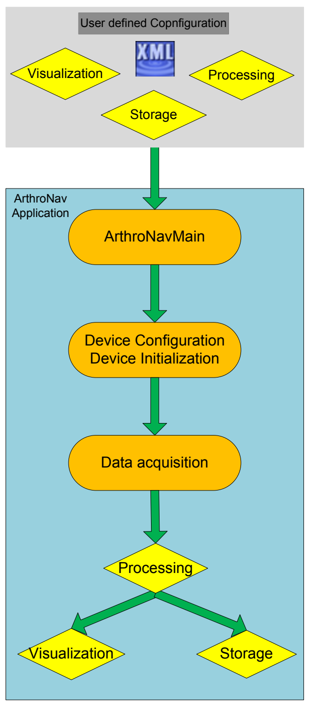
4.4.1 Configuration using a XML file

To solve the requirement of easy configuration, a parsing component was put at the start of the application. This component allows the user to write a configuration file [7] that will be parsed at the start of the application. The parsing component is composed by two classes, the *ArthroParser* class and the *OptoTrackerParser* class. Both use the *expatpp* library [8] to parse the configuration file written by the user. The operation of the parsing is performed as a *tree* configuration. When the parser recognizes an element name, it will call the appropriate handler for that element. The handler reads the possible attributes from the file until it encounters a final element tag. The *OptoTrackerParser* class is a support for the *ArthroParser* while parsing the opto-tracker attributes. The parsed information is stored in the

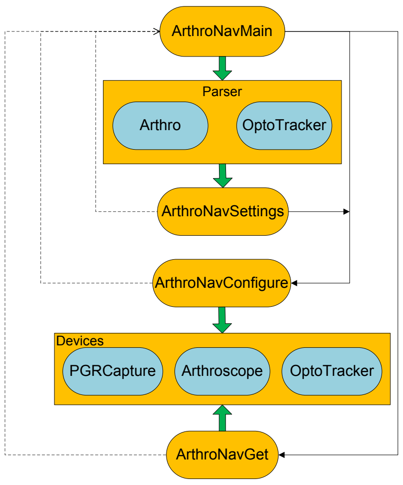
[6]The documentation was generated using *doxygen*.

[7]The application contains an initial template, so the user just changes it according his needs.

[8]Expatpp is a C++ wrapper for expat, a Extensible Markup Language (XML) Parser Toolkit. For further reading and for download, consult www.oofile.com.au/xml/expatpp.html



(a) Base flow of the application



(b) ArthroNavMain communication

Figure 4.1: An overview of the Software Architecture

ArthroNavSettings class, that will be later used for the configuration of the devices, and for the acquisition loop.

The element names that the application recognizes are shown in Table 4.1. The opto-tracker attributes possible to set, can be consulted in Table 4.2. For attributes and/or element options not set in the initial configuration file, the application will use default values for the initialization of the devices.

4.4.2 The devices and their configurations

Each device used in the navigation system has a respective communication component in the software architecture. It allows the initialization and configuration of the correspondent device, and captures the frames when requested.

The component for the communication with the opto-tracker is the *OptoTracker* class, a wrapper class of the OAPI. These class facilities the usage of the Optotracker and lowers the necessary code dimension. Each method

Table 4.1: Elements that the application can parse

Element	**Description**
Arthronav	Root element. Needed to initiate the application. No input.
Optotracker	Activates the Opto-tracker. Consult the attributes on Table 4.2.
Arthroscope	Activates the Arthroscope. No input.
PTGCam	Activates the PointGrey Camera. As input can receive the frame frequency.
Trigger	Set the trigger source : *(a)* for Arthroscope, *(o)* for Optotracker, *(p)* for the PointGrey Camera
RecordTime	Time to record the capture
RecordRate	Record frequency will be TriggerFrequency/(RecordRate*AcquisitionRate)
AcquisitionRate	Acquisition frequency will be TriggerFrequency/AcquisitionRate

Table 4.2: Opto-tracker attributes that can be parsed

Atributes	**Description**
Frequencia	Rate at which a frame of data is generated by the Optotracker
MarkerNr	Number of markers in the collection
RigidNr	Number of rigid bodies in the collection
Port1	Number of markers connected to strober port 1
Port2	Number of markers connected to strober port 2
Port3	Number of markers connected to strober port 3
Format	Return format for real-time rigid bodies
ChangeFor	Rigid body ID which will be the new coordinate system

encapsulates the necessary OAPI routines for a specific action on the optotracker. Besides other functionalities, this class allows the capture of marker positions, and to track until three rigid bodies.

To communicate with the *Flea2* camera, also a wrapper class was constructed. The *PGRCapture* class contains the necessary routines to capture at internal rate, and set the camera to listen the external trigger port during the capture. These components uses the API supplied by the manufacturer.

The arthroscope is activated by the *Arthroscope* class, inherited from the *cameraEngine* class. The *cameraEngine* class is the base wrapper class that allows communication with platform specific classes for video capture and display . This class and the platform dependent classes associated, belong to the PortVideo framework.

The *ArthroNavConfiguration* component achieves the communication between the application and the devices. When initialized, it starts the selected devices, and configures them according an *ArthroNavSettings* object given as input. These component is constituted by the *ArthroNavConfigure* class, and contains an object for each device that composes the navigation system.

4.4.3 Insertion of of user defined modules

The application should be a closed box, allowing as input the insertion of modules, and as output, supplies the navigation information according the inserted modules. The way to do this in Object-Oriented Programming (OOP), is using *polymorphic classes* containing *virtual member functions*, that can be redefined in its derived class.

To better understand this part of the application, we will briefly make an overview of two features of the used OOP. The first feature is the *Inheritance*. This propriety allows one class to inherit, or derivate, the functionalities of an existing class, being the existing class not a data member of the new created class. The existing class is the base class, and the new created class will be the derived class. An object of a derived class is at the same time an object of the base class, and can redefine the member functions of the base class. In this situation, when pointer assignments are used, the *type of the pointer* determines which member function will be called, and not the type of the object.

The second feature is the *Polymorphism*, the ability of a certain entity to behave differently in different contexts. As said before, the standard behavior is the *type of pointer* determines the method invoked. This can be changed using the concept of *Polymorphism* and *virtual member functions*. Declaring a function *virtual* in the base class and redefining this method in the derived class, when pointer assignment is used, the type of object pointed-at will now determine the method called. The caller of the *virtual* method just needs to know the base class, in order to call the restant derived classes. It will be at run-time that the correct function will be called using the Vtable, depending on the object refered by the caller.

The implementation of the *Polymorphism* for the introduction of modules is shown in Figure 4.2. The application *three classes* abstract classes, called when the frames are captured, in order to process, visualize, and/or store the acquired information. These classes are the *ArthroProcessor* class, called with the information of the arthroscope. The *OptoProcessor* class, related to the opto-tracker processing. Finally, the *PGRProcessor* component, that filters the frames grabbed from the PointGrey camera. The user can derivate from these classes, and reimplement the virtual functions that they are composed by. The *ArthroNavMain* can receive at the start of the application pointers to the user defined abstract classes. These objects will be called during the acquisition loop, even if the application do not know them.

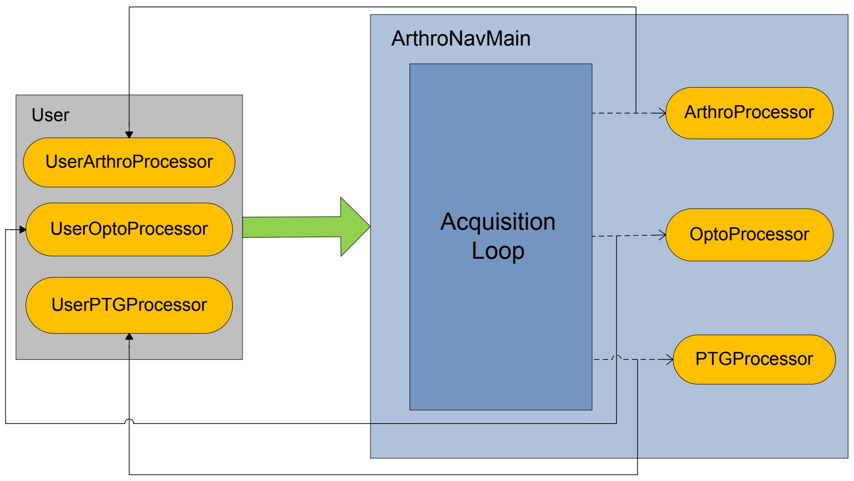


Figure 4.2: Virtual call of user defined methods

4.4.4 ArthroNavMain

The *ArthroNavMain* component forms the core of the application. It is constituted by the ArthroNavMain class. This class contains all necessary objects to read a Extensible Markup Language (XML) file, initialize and communicate with the devices, and call user defined objects/modules. This is the single class of the system that the user should know in order to use the application. It has seven public methods, three methods to add processors, one method that initiates the system, and three methods to remove the processors from the application.

Each added processor is pushed into the corresponding list (e.g. all *ArthroProcessors* are put in to the *ArthoList*). During the initialization and the acquisition loop, the application will call all the objects in the list, allowing more than one filter for each device to be inserted.

4.4.5 Acquisition Loop

The acquisition loop captures the frames from the devices in a controlled manner. The devices are configured to retrieve a new frame on every software request. If the acquisition loop runs without any control of the request time, frames can be lost whenever the duration of the processing is longer than the frame rate acquisition (the loop does not close on time). For constant acquisition and processing, the solution is to control the acquisition period based on the frame rate of the device. The acquisition period [9] should be chosen to accomplish the Equation 4.1, where n is the smallest real integer that verifies 4.2.

$$T_{Acquisition} = n \times T_{TriggerDevice} \tag{4.1}$$
$$n \times T_{TriggerDevice} > T_{Processing} > (n-1) \times T_{TriggerDevice} \tag{4.2}$$

To better illustrate this problem, Figure 4.3 shows two different acquisition situations. Since the first capture is not controlled, it is not possible to calculate the frequency of the acquisition, and frames can be lost without realizing (which can lead to nasty consequences). In the second situation, the duration of the acquisition is two times the duration of the frame period.

[9] Sum of the time to request the frame and processing.

The acquisition loop reads every time a frame after the processing period ends, allowing perfectly synchronous acquisition and processing.

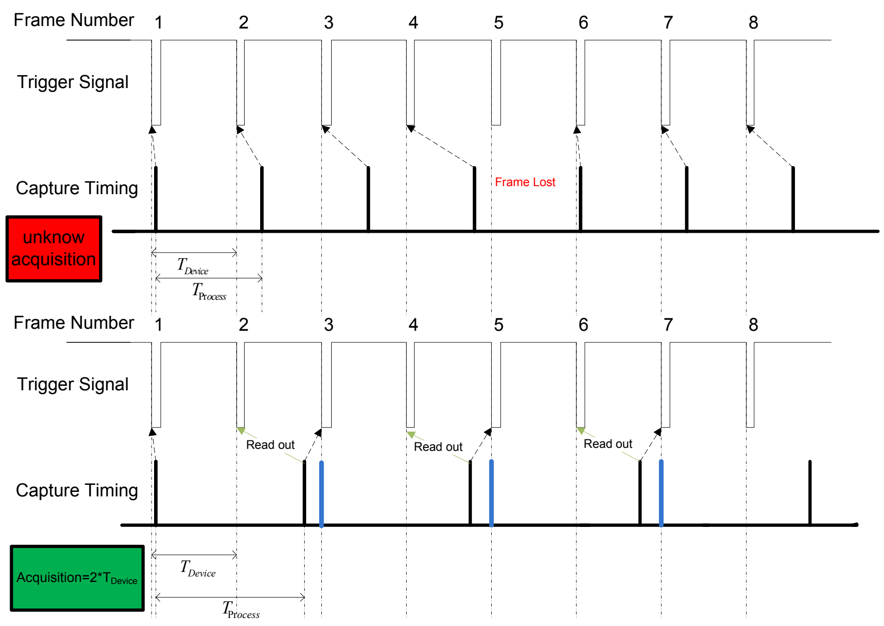


Figure 4.3: Two examples of an acquisition loop. In the first example, frames can be lost without to realizing. In the second example, one frame is discarded per iteration, synchronous acquisition is achieved.

The need of the time control is clearly shown in the example of Figure 4.4. Each frame is composed by three components, one for each device (arthroscope, opto-tracker and Flea 2). If we do not control the time of the frame request, the acquired information can be shuffled. Information received in the same frame iteration can not correspond to the same time instant (loss of synchronism). The situation in which a set of frames are read out, allows the constant processing of $T_{Device}/2$. This also ensures that each iteration works with information incoming from the same time instant.

The initial *configuration* file allows to set the *Acquisition Rate* and the *Record rate*, for the purpose of controlling the acquisition loop and storing loop, respectively. It is the responsibility of the user to verify if this is done correctly, since the application sends an alert message when frames are lost, or frames are shuffled. If the user realizes that frames are shuffled, he just needs to increase the *Acquisition Rate* attribute, in order to lower the acquisition rate, achieving a constant and synchronized capture.

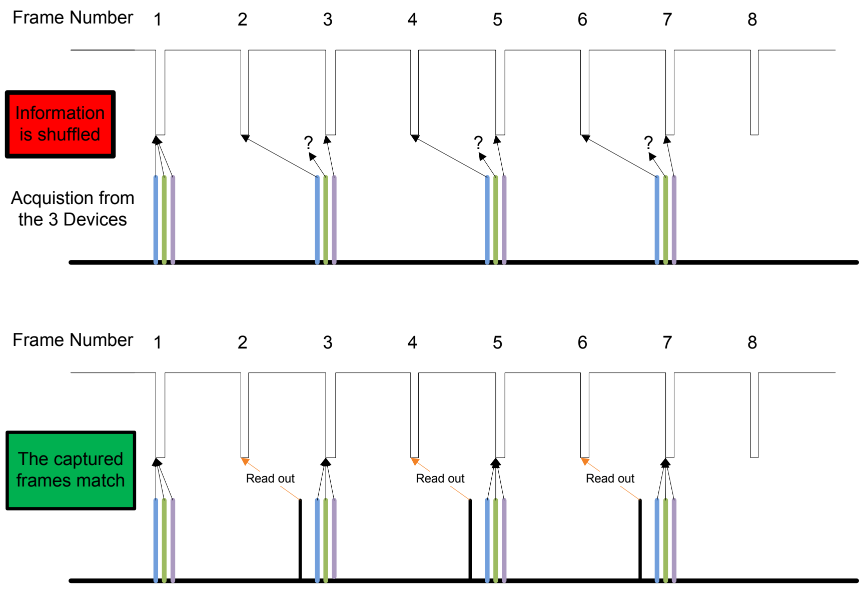


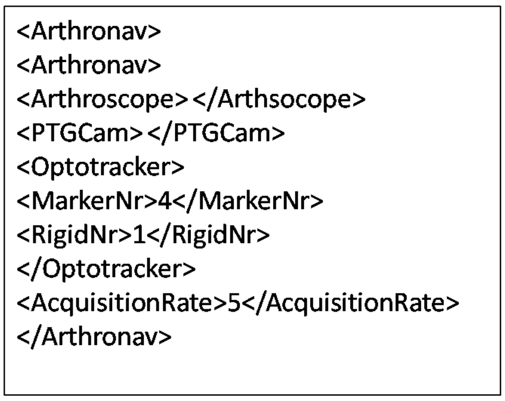
Figure 4.4: Acquired frames can be shuffled if the capture time is not controlled. Reading a set of frames out, ensures that each iteration works with information from the same time instant.

4.4.6 How to use the application

In this section, the necessary steps to use the application, as well as the corresponding system flow are described.

The first step in order to use this application is to write the *configuration.xml* file, that will configure the devices and the acquisition loop. The element names and the opto-tracker attributes possible to be parsed are shown in Table 4.1 and Table 4.2, respectively. Suppose we want to use the an arthroscope, the PointGrey Camera, and the opto-tracker, tracking a rigid body with four markers. The acquisition rate should be $5Hz$. As the default trigger source is the arthroscope, the fame rate will be $25fps$. The final *configure.xml* file will look like the configuratio shown in Figure 4.5. For the attributes not set, a default value is used.

The *ArthroProcessor* and the *PTGProcessor* base classes contain the necessary code to visualize the acquired frames in a *OpenCV* window. The methods from the base class of the *OptoProcessor* print the marker or rigid body positions on the command line. If the user wants a different visualization and/or a different processing, he should inherit from this classes, and re-define the virtual methods according his needs. When writing the *main*



```
<Arthronav>
<Arthronav>
<Arthroscope></Arthsocope>
<PTGCam></PTGCam>
<Optotracker>
<MarkerNr>4</MarkerNr>
<RigidNr>1</RigidNr>
</Optotracker>
<AcquisitionRate>5</AcquisitionRate>
</Arthronav>
```

Figure 4.5: The *configuration.xml* file

program, he just need to add the *pointers* of the objects created, and start the capture.

In Figure 4.10 the flow of this configuration is shown. In this acquisition sequence, all devices were used, and for each device a processor has been added before the initialization.

4.4.7 Processors created

Various processors that can be added to the *ArthroNavMain* component were developed during the project, a few of them will briefly described in this section.

The *StoringProcessor* was created to allow the storing of acquired information. A folder is created for each capture loop, storing the equivalent frames and their timing information. It uses the *OpenCV* library for the storing purpose. This class allows to write frames in various formats, the only requirement is to change the frame name termination to the corresponding format.

An interface for the application was created using the *Qt Cross-Platform Application Framework*[10]. The interface of the *ArthroNavTracker* is shown in Figure 4.6. The interface is composed by the *ArthroNavTracker* class, and a *QWidget* [11] class for each device. The *ArthroNavTracker* class is a *QMain-*

[10]Qt is a cross-platform application framework for desktop and embedded development.For further reading consult http://trolltech.com/products/qt/.

[11]The *QWidget* is the atom of the user interface: it receives mouse, keyboard and other events from the window system, and paints a representation of itself on the screen.

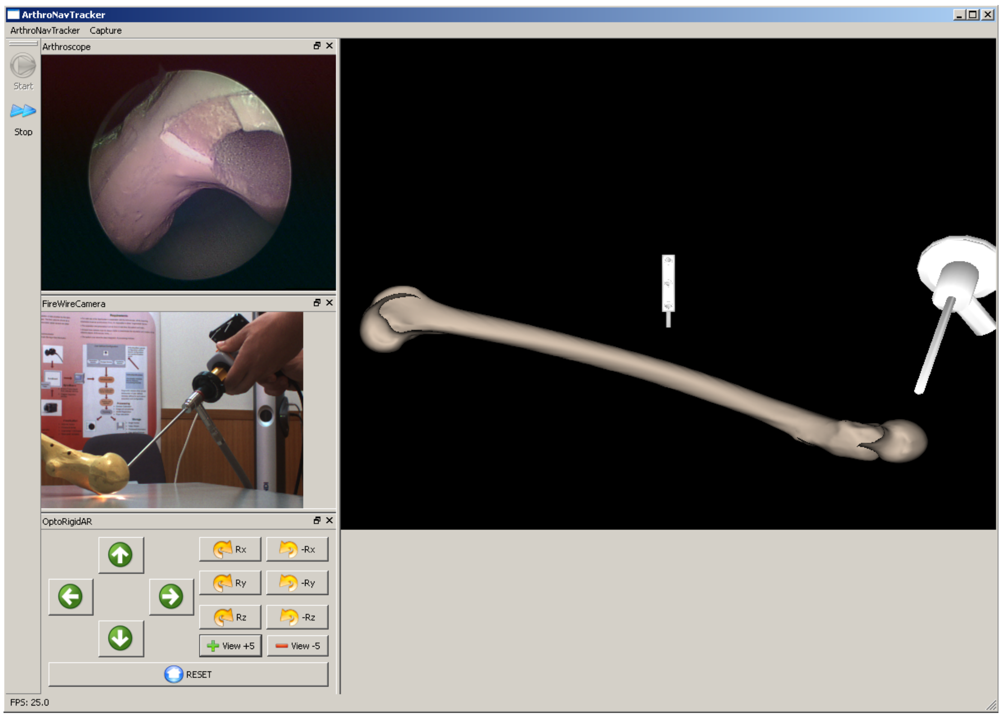


Figure 4.6: ArthtroNavTracker

Window [12] that docks the *QWidgets* inherited from the device processors. The communication is schematically shown in Figure 4.7.

The *ArthroNavTracker* contains an *OptoRigidAr* processor that provides an animation of virtual rigid bodies using data acquired with the Optotracker. It is a class derived from the *OptoProcessor* class, that uses the Augmented Reality (AR) library created in the ISR. For this purpose, it was necessary the *3D* design of the objects used in order to create an *OBJ* [13] file. The *3D* designs were made using *Wings3D* [14]. The *3D* objects are introduced in a *glEngine* [15] object. The *glEngine* class reads the *OBJ* files, draws them using the Open Graphics Library (OpenGL) library, and renders them in a Simple DirectMedia Layer (SDL) window. The objects can than be positioned according *3D* data given as input.

Suppose we want to do the animation of two rigid bodies, as shown in Figure 4.8. A *femur* and an *arthroscope* should be animated in front of

[12]The *QMainWindow* class provides a main application window, with a menu bar, dock windows, and a status bar.

[13]Geometry definition file format.

[14]For further reading consult www.wings3d.com.

[15]A base class for the *OpenGl* engine; is contained in the AR library of the ISR.

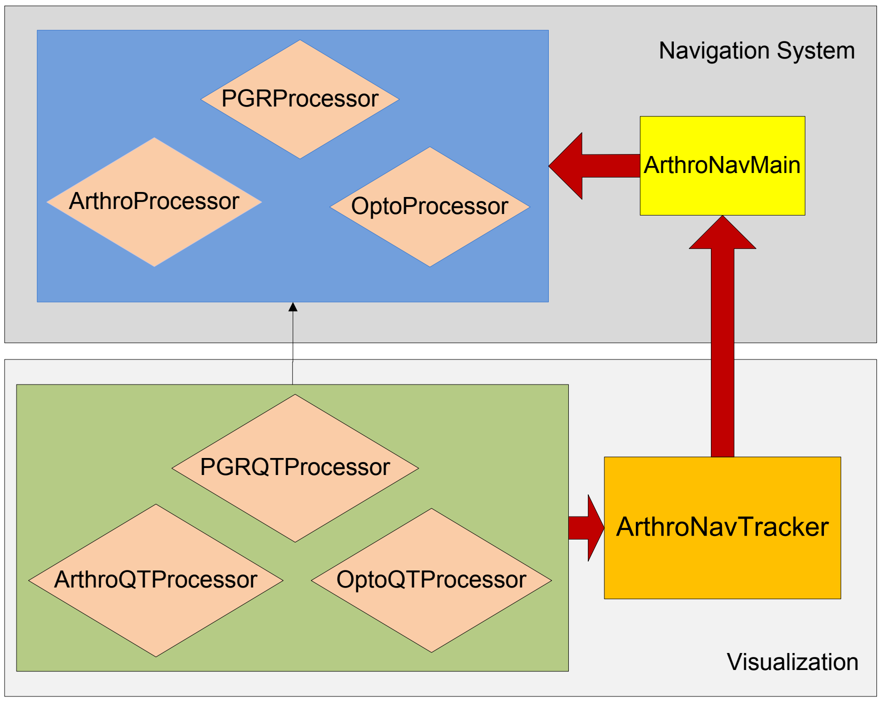


Figure 4.7: ArthroNavTracker communication

the opto-tracker. First, it is necessary the design of the objects in order to create the corresponding *OBJ* files. After the introduction of the *virtual rigid bodies* in a *glengine* object, the movements must be supplied. The opto-tracker retrieves the transformations of the local frame of reference of the rigid body relative to its own fixed coordinate system [16]. The *virtual rigid bodies* are positioned according the data received by the opto-tracker, transformed relative to the frame of reference of the *virtual opto-tracker*. If the opto-tracker has a transformation $T_{Optotracker}$ relative to the *OpenGL*, the *virtual rigid bodies* have $T_{Optotracker} \times T_{rigidbody}$ transformation relative to the *OpenGl*, where $T_{rigidbody}$ is the data received from the opto-tracker, rotated by 90° in R_y. This transformation is necessary because the frame of reference of the opto-tracker, an the frame of reference of the *OpenGL* are transformed by this rotation. The calculations were performed using the *Quaternion* representation of the transformations, as support the *quaternion* class of the AR library was also used.

[16]The opto-tracker frame of reference can also be altered using alignment routines.

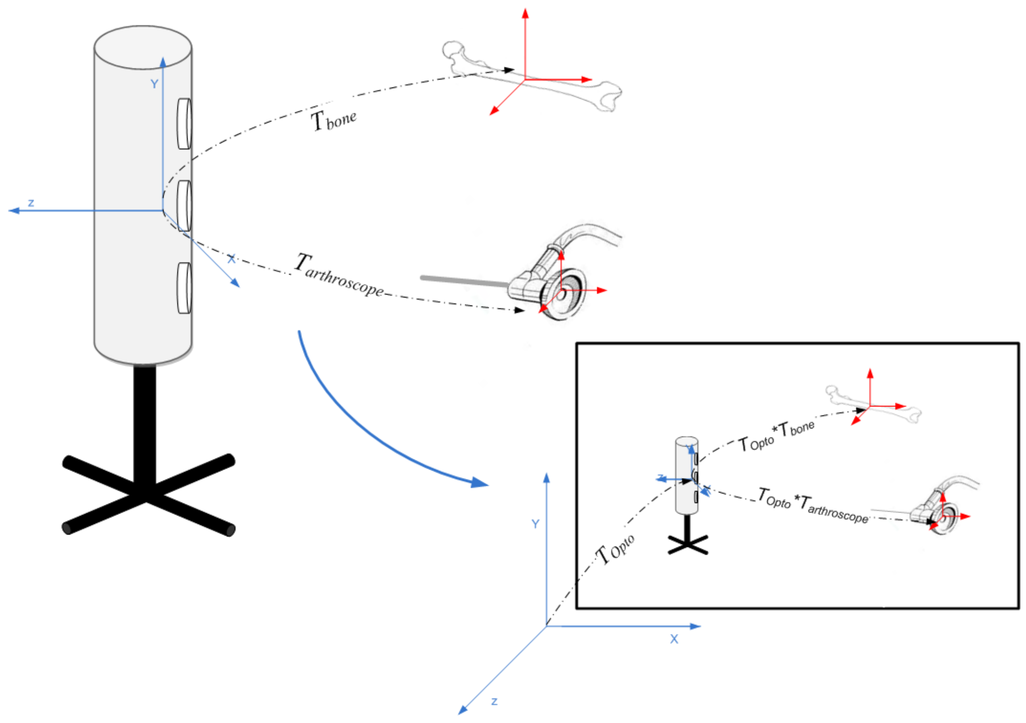


Figure 4.8: OptoRigidAR animation

4.4.8 Software results

In this section we discuss the results of real experiments in running our software. The analysis is done based on data obtained with the Storing module described above. We use the PC clock to measure for each iteration the time instants of entrance in the loop and of finishing processing. By subtracting consecutive entrance instants we can infer the loop period, and by subtracting the finishing time from the entrance time we can infer the processing time.

Let the trigger source be the Arthroscopic video, which means that the frame acquisition rate is 25 Hz (the PAL video frequency). The application was ran with and without time control in entering the acquisition loop. In the former case the data requests are performed at constant time intervals. In the latter the request instant depends on the time that each processing iteration takes (closes the loop as fast as it can). For each case the application was tested under two different circumstances: (i) with the visualization module, and (ii) with visualization and image storing modules.

Tab 4.3 shows the measured loop period, and Figure 4.9 shows a diagram of the situation (ii) with and without the control of the acquisition loop.

Table 4.3: Processing and acquisition duration of three different capture situations[a]. (i) with visualization module, (ii) with visualization and image storing module (StoringProcessor), and (ii + Control) Discarding one frame in (ii).

Acquisition	**Processing Time**	**Loop Period**
(i)	$15ms$	$40ms$
(ii)	$108ms$	$109ms$
(ii+Control)	$108ms$	$120ms$

[a]These are mean measurements of 30 seconds of acquisition. σ was : $2ms$ for (i) Processing Time, $3ms$ for (i) Loop Period, $6ms$ for (ii) Processing Time, $5ms$ (ii) Loop Period, $7ms$ (ii+Control) Processing Time, and $3ms$ (ii+Control) Loop Period.

In situation (i), it is possible the visualization of 25 frames per second, as the duration of the processing is less than the trigger period. However, in (ii) this is not the case. Using modules with a processing duration longer than the frame period, involves the *jumping* of triggered information.

In situation (ii), the application was executed without time control. At the 9^{th} trigger, it was possible that information was shuffled, as shown in Figure 4.9. This occurs because the end of the processing loop almost matches the instant of a trigger signal (The processing ends $319ms$ after the start of the application, and the 9_{th} trigger occurs at $320ms$ after the initialization.). In this case, the system cannot be sure if the data corresponds to the same time instant.

Controlling the situation (ii), we achieved the synchronism of the acquisition, and a constant loop period of $120ms$, shown in the the lower situation of Figure 4.9. This occurs, because on frame was discarded in each iteration.

The software of the navigation system created, enables the user to control the data acquisition following the hardware mode selected. If correct configuration parameters are set, the acquisition is synchronous, overcoming the principal requirements of the project.

4.4.9 Problems during the development

The initial problem was the little knowledge about programming and about the *C/C++* languages. It was necessary to learn these languages and their features, and to try and retry to use them, until it *clicked*.

At the beginning of the project, i did not have experience in Linux-like environments and *GNU* tools. It was difficult to assemble all the device's

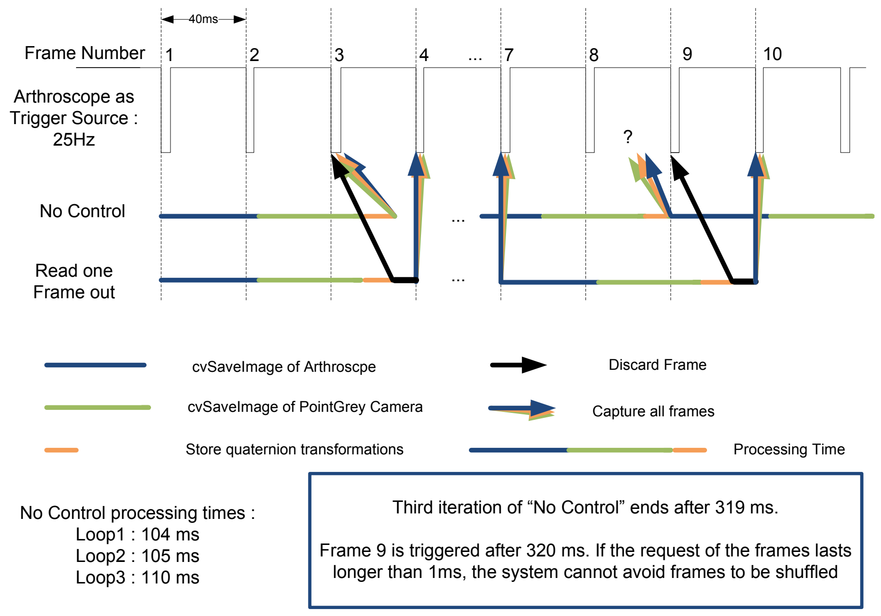

40ms
Frame Number
1
2
3
4
...
7
8
9
10
Arthroscope as Trigger Source : 25Hz
?
No Control
...
Read one Frame out
...
cvSaveImage of Arthroscpe
cvSaveImage of PointGrey Camera
Store quaternion transformations
Discard Frame
Capture all frames
Processing Time
No Control processing times :
Loop1 : 104 ms
Loop2 : 105 ms
Loop3 : 110 ms
Third iteration of "No Control" ends after 319 ms.
Frame 9 is triggered after 320 ms. If the request of the frames lasts longer than 1ms, the system cannot avoid frames to be shuffled

Figure 4.9: Acquisition using a storing module. The frame of the storing rate can not follow the hardware triggering. In order to capture synchronous information, frames must be discarded.

capabilities in *Cygwin*, and to produce code using the compiler and debugger from *GNU*. After a while however, it was easy to understand how useful these tools are for the programming purpose.

With the Arthoscope, we had difficulties to use the open-source available PortVideo Framework. This because the libraries to use with Windows, containing the required object files, were compiled using the *Microsoft Visual Studio* development environment. As the code is written in *C++*, and to give unique names to overloaded functions, the compilers must use name mangling[17] [20]. The problem is that the *Microsoft Visual C++* compiler and the *g++* have different name mangling conventions, so the linker of *g++* could not find the required functions from the libraries.

The solution was to create wrapper functions inside the DLL, that export the necessary methods using the keyword *extern*[18] with *C*. These functions

[17]Name mangling is used by the *C++* compilers to add additional information to the names of functions and objects, allowing function overloading, comparison by the linker of them in different modules, and linkers to give information about unresolved references in error messages [20].

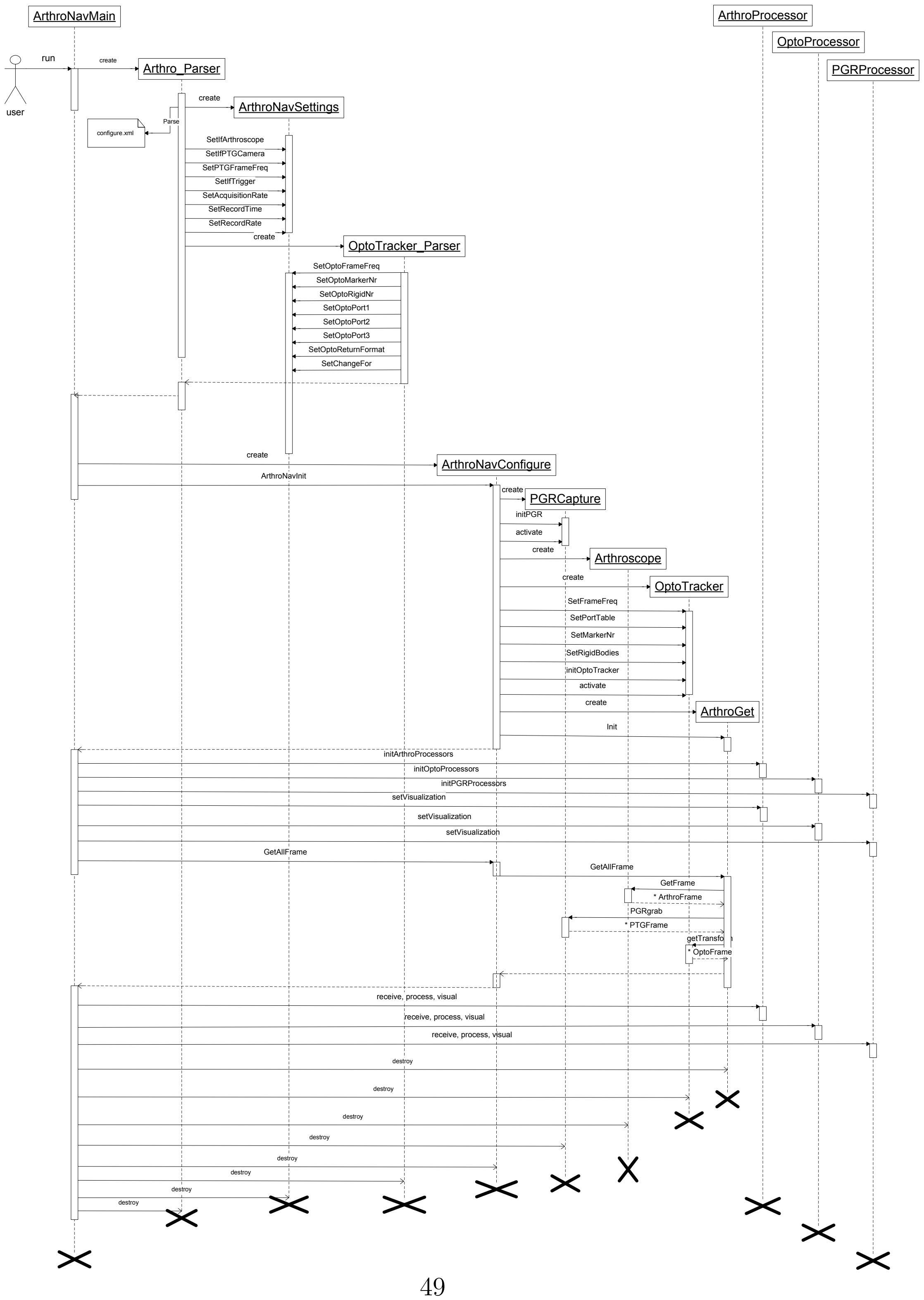
[18]The *extern* keyword declares a variable or function and specifies that it has external linkage [21].

are now compiled as *C* code, allowing the *g++* compiler to recognize them. These functions are called using *callback functions* outside the DLL [22]. When called, they forward the message to the *brother* class methods inside the library, and return their result to the caller. With this technique, it is possible to *Microsoft Visual Studio C++* code with the *GNU* compiler.

The interface for the application was designed using *Qt*. To use the SDL window from the *glengine* class, for the bone and arthroscope animation, an environment variable must be set every time the application starts. SDL will create the window for the rendering purpose in an address that the *SDL_WINDOWID* environment variable contains. To integrate the SDL window in the *ArthroNavTracker*, the solution is to initially create a *QWidget*, and to get its window ID. At the start of the application, the *SDL_WINDOWID* variable is altered with the identity of the *QWidget* window created [23]. So, any *SDL* window can be rendered inside *Qt* applications, without to copy the images from one window to the another.

It was not possible to acquire from the arthroscope and from the PointGrey Camera simultaneously *RGB* data at $25Hz$. This was not possible as we used just on FireWire controller for the acquisition. TO overcome this limitation, the pixel data of the PointGrey camera is captured using the *YUV* format, and is after transformated to the corresponding RGB data.

Figure 4.10: The flow of the application



Chapter 5

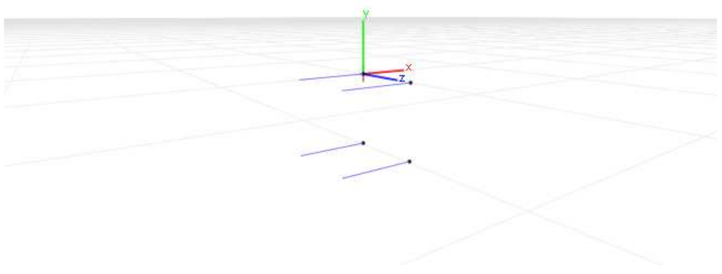
Construction of a Tool with LEDs

5.1 Rigid Body characterization

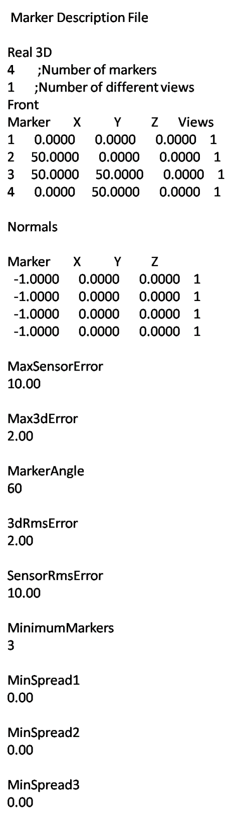
As explained in 2.1.2, a rigid body is a group of markers where the positions relative to each other is fixed. The minimum number of markers to calculate the transformation of the object is three. For the opto-tracker calculate the pose of a rigid body, the positions of the markers in its local coordinate system must be supplied to the system, besides the information of Tracking Tolerances if necessary. The best way to do this is creating a rigid body file for the object using the *NDI 6D Architect Software*. An example of a constructed rigid body using *NDI 6D*, an its equivalent rigid body file is shown in Figure 5.1.

The steps to create a rigid body using the *NDI 6D Software* are [24]:

1. Select the Collection settings (marker power, frame frequency,...)
2. Position the rigid body in the measurement volume
3. Assign a local coordinate system for the rigid body
4. Add Marker Normals
5. Set Tracking Tolerances



(a) Rigid Body (*NDI 6D Architect*)



```
Marker Description File

Real 3D
4      ;Number of markers
1      ;Number of different views
Front
Marker     X        Y        Z     Views
1    0.0000    0.0000    0.0000   1
2   50.0000    0.0000    0.0000   1
3   50.0000   50.0000    0.0000   1
4    0.0000   50.0000    0.0000   1

Normals

Marker     X        Y        Z
  -1.0000    0.0000    0.0000   1
  -1.0000    0.0000    0.0000   1
  -1.0000    0.0000    0.0000   1
  -1.0000    0.0000    0.0000   1

MaxSensorError
10.00

Max3dError
2.00

MarkerAngle
60

3dRmsError
2.00

SensorRmsError
10.00

MinimumMarkers
3

MinSpread1
0.00

MinSpread2
0.00

MinSpread3
0.00
```

(b) Rigid body file

Figure 5.1: Rigid Body

The Tracking Tolerances are the maximum allowed error tolerances when fitting marker coordinates to a rigid body and calculation the rotational and translational data. If a marker fails one of these tolerances during a capture, depending on the tolerance constrain, the data for the marker, the frame, or the transformation is considered as invalid and the transformation is computed again without this marker.

One Tracking parameter that can be set is the Maximum Marker Angle, the maximum angle that a marker can be pointed away from the Position Sensor. Any marker turned away more than this value will not be used to compute a rigid body transformation. In order to calculate the marker angle, the Rigid body file should also contain the marker normals. A marker normal of a marker is a vector that defines which way the marker is facing. The greater the angle between the marker normal and the Position Sensor, the less accurate will be the tracked position. The Minimum Number of

Markers is also a Tracking parameter, then the more number of markers to calculate the pose of a Rigid Body are used, the more accurate will be the measurement. The 3D error parameters are related to the errors between the positions where the markers are observed by the system, and the positions that are expected to be in the Rigid Body file. These parameters are set to overcome errors like markers partially obscured, and external infrared light sources misinterpreted. The last rigid body settings are the three minimum spread parameters, the minimum distance, area, or volume that markers used to calculate the transformations of a rigid body must cover. The minimum spread improves the accuracy of the pose calculation, this because the accuracy improves by increasing the 2D or 3D distribution of markers on the rigid body [25].

5.2 Tool Constraints

As during a surgery the doctors must have enough freedom to operate, there is a need for the construction of a tool to fix on the arthroscope and on surgical tools, in order to allow its tracking. The tool to attach the markers should not perturb the surgeons during the medical intervention, and at the same time allow the best possible pose calculation accuracy. The principal constrains for the tool will be :

- To solve the line-of-sight problem, use low 3D tolerances to overcome errors caused by markers partial obscured.
- For good accuracies, the maximum allowed marker angle should be 60° or less.
- Three visible markers is the minimum. A good practice would be using four markers for every calculation, ensuring accurate results.
- The tool should be great enough to have good spread distances between the markers, but small to not perturb the surgeon during the medical intervention. It is important to found a compromise between this requirements.

At the beginning of the project, prototypes were constructed to study this constrains. However, as the problem of the synchronism was difficult to solve, its construction has been left for a further step of the *ArthroNav* project.

Chapter 6

Conclusion

The outcome of the project was a system, capable of the synchronous acquisition of optical tracking data, and image data from the arthroscope and FireWire camera. In addition we implemented modules for visualization and storing that can be easily inserted in the software application. Test were performed to confirm the synchronism and real-time processing.

This project was a very enriching experience, that gave me the opportunity to acquired different skills, both in engineering (hardware and software) and in science. I learn about the following topics:

- To program medium dimension applications in *C/C++*. This was necessary in order to use the API of the opto-tracker and the API of the cameras. A lot of features of the OOP were used during the project that initially were unknow to me (e.g. Inheritance, Polymorphism, Encapsulation). I also tried to follow good practices of software development.

- To understand an electrical problem and design a circuit to solve it. I studied the opto-tracker manuals, the video formats of the arthroscope, and the manuals of the PointGrey Camera in order to find a solution for their synchronism. I extensively used digital electronics principles.

- PCB design, necessary to create a board with the electronics for the synchronism of the system.

- Kinematics theory, in order to understand the operation of rigid bodies, and to manipulate the data supplied by the opto-tracker.

- *3D* modeling and design, required to create the models for the rigid body animation.
- Different programming libraries (e.g. *Qt*, Open Source Computer Vision Library (OpenCV), SDL, OpebGL).

Despite of being happy with the outcome, I believe there are improvements that can be done:

- Create generic device classes, containing the communication and the processing.
- Use of non-blocking grabbing routines to improve the time control of the acquisition.
- Use various threads in order to increase the velocity the application.
- Implement an adaptive acquisition timing, in order to overcome non-constant processing durations.

However, it can already be used by the *ArthroNav* project in order to employ computer vision algorithms over the endoscopic video enhanced with the pose information.

The future step of the project is to expand the navigation system with *3D* reconstruction/registration techniques, to enable the migration of the complete application to the surgical environment, and to provide the surgeon with an accurate and precise visualization/navigation of the knee joint.

Appendix A

Software Documentation

The software produced during the project was documented using the code generator *Doxygen*. The code and the documentation can be consulted on http://orion.isr.uc.pt/∼michel/.

The documentation is avaliable online, in order to be simpler to consult and find the required class/funtion. This because it is a large quantity of code, and its management is difficult on a *PDF*.

Appendix B

Schematic Design and PCB Layout

The schematic circuit and the final PCB were designed usign *Alium Designer 6*.

The final schematic design is shown in Figure B.1. First the circuit's response was evaluated using a *BreadBoard*. As the output was suitable for our application, a PCB was designed in order to construct a board, making it easier to synchronize the navigation system.

The Bottom Layer of the PCB is represented in Figure B.2, and the Top Layer is shown in Figure B.3. A separate GND plane enables less number of tracks and vias, and separates the user circuit from the camera and opto-tracker circuit.

Figure B.1: Schematic of the SyncBoard

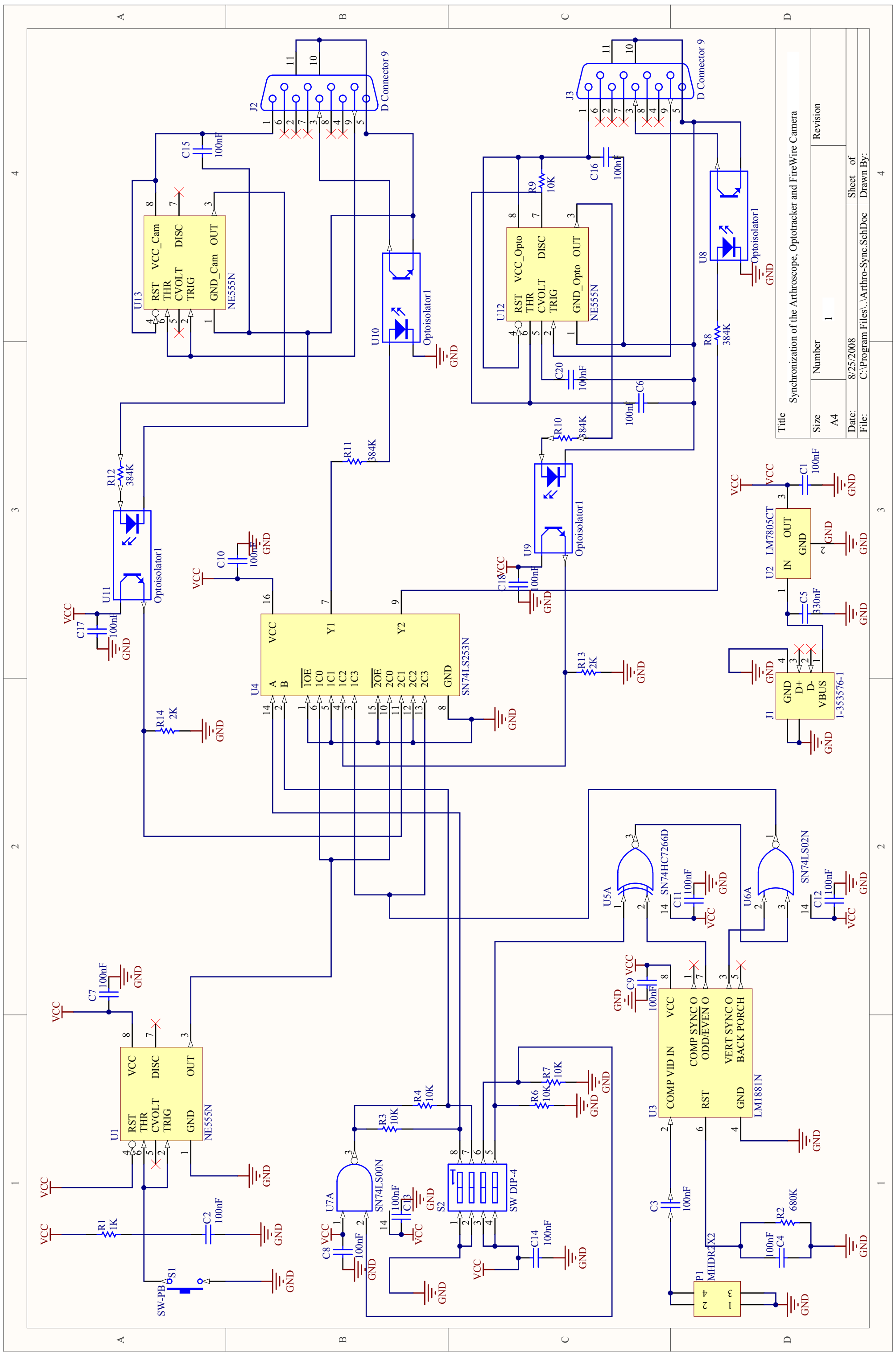


Figure B.2: Bottom Layer of the PCB Layout. In *yellow* are the top layer tracks/planes represented, and in *red* the bottom layer tracks/planes.

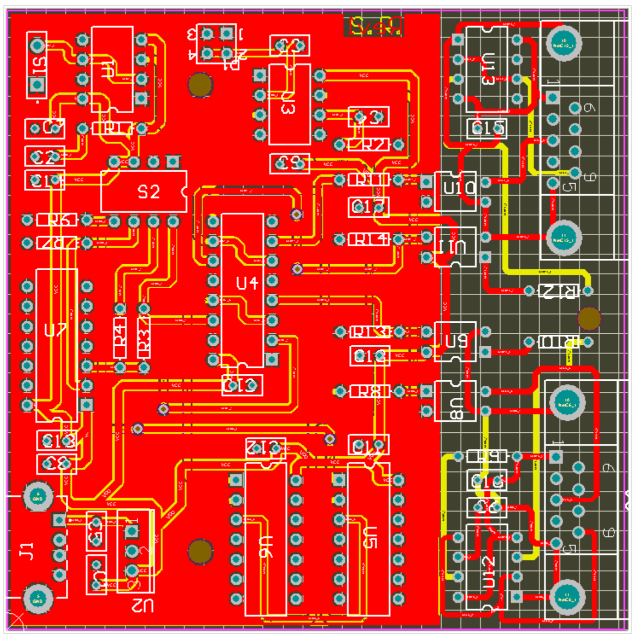
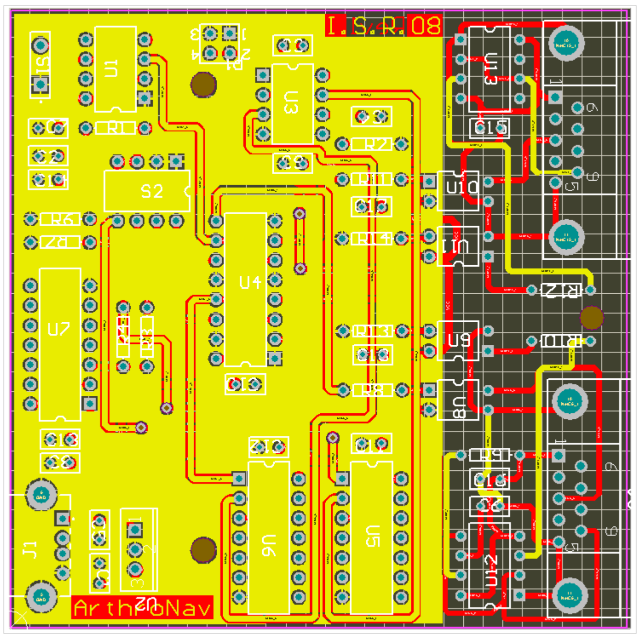


Figure B.3: Top Layer of the PCB Layout. In *yellow* are the top layer tracks/planes represented, and in *red* the bottom layer tracks/planes.



Bibliography

[1] (2007, September 20th) Arthronav - navegação assistida por computador em cirurgia ortopédica a partir de imagens endoscópicas. Fundação para a Ciência e a Tecnologia. [Online]. Available: http://www.fct.mctes.pt/projectos/pub/2006/Painel_Result/vglobal_projecto.asp?idProjecto=68887&idElemConcurso=891

[2] (2008, August 21th) Arthroscopic acl (surgery) reconstruction. [Online]. Available: http://www.arthroscopy.com/sp05018.htm

[3] M. Hafez, B. Jaramaz, D. Gioia, and M. A. M, "Computer-assisted knee surgery: An overview," in *Insall and Scott - Surgery of the Knee, 4th Edition*, F. W. Norman Scott, MD, Ed. Elsevier, 2006, vol. 2, pp. 1655 – 1674.

[4] J. B. Stiehl, "Computer-assisted surgery in adult reconstruction," Tech. Rep., August 8th 2008. [Online]. Available: http://www.touchbriefings.com/pdf/1680/Steihl%5B1%5D.pdf

[5] A. J. Chung, P. J. Edwards, F. Deligianni, and G.-Z. Yang, "Freehand cocalibration of optical and electromagnetic trackers for navigated bronchoscopy," in *MIAR*, 2004, pp. 320–328.

[6] *Optotrak Certus User Guide (e-Type)*, Northern Digital Inc., October 2007, Rev 2.

[7] K. K. Sharma, *Optics: Principles and Applications.* Academic Press / Elsevier, London, UK, 2006.

[8] R. L. Galloway, W. A. Bass, and C. E. Hockey, "Task-oriented asymmetric multiprocessing for interactive image-guided surgery," *Parallel Comput.*, vol. 24, no. 9-10, pp. 1323–1343, 1998.

[9] M. R. M., L. Zexiang, and S. S. S., *A Mathematical Introduction to Robotic Manipulation.* CRC, March 1994. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20\&amp;path=ASIN/0849379814

[10] *Smith and Nephew 450P and 460P 3-CCD Camera Control Units Operations/Service Manual*, Smith amd Nephew, Rev 2.

[11] *Flea2 Technical Reference Manual*, Point Grey Research, September 2006, Version 1.1.

[12] L. V. Guide, "Interlacing," May 2002. [Online]. Available: http://neuron2.net/LVG/default.htm

[13] (2007, November 11th) Video signal measurement and generation fundamentals - intermediate analog concepts. National Instruments. [Online]. Available: http://zone.ni.com/devzone/cda/tut/p/id/5364

[14] S. W. Smith, *The scientist and engineer's guide to digital signal processing.* San Diego, CA, USA: California Technical Publishing, 1997.

[15] (2008, July 20th) How television works. Wikipedia. [Online]. Available: http://en.wikipedia.org/wiki/How_Television_Works

[16] N. Semicondutor, "Data sheet lm1881 video sync separator," April 2001.

[17] (2008, July 30th) Building blocks of a video format. sgi. [Online]. Available: http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650\&db=bks\&fname=/SGI_Developer/VFC_PG/ch03.html

[18] F. B.Brokken, *C++ Annotations.* University of Groningen, March 1994-2008, version 7.2.0.

[19] H. M. Deitel and P. J. Deitel, *C How to Program.* Prentice Hall, March 2006, 3rd edition.

[20] A. Fog. (2008, August 9th) Calling conventions for different c++ compilers and operating systems. Copenhagen University College of Engineering. [Online]. Available: http://www.agner.org/optimize/calling_conventions.pdf

[21] (2008, August 9th) Using extern to Specify Linkage. MSDN. [Online]. Available: http://msdn.microsoft.com/en-us/library/0603949d(VS.80).aspx

[22] P. Jakubik, “Callback implementations in c++,” in *TOOLS ’97: Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems*. Washington, DC, USA: IEEE Computer Society, 1997, p. 377.

[23] (2008, June 12th) Intégration de sdl dans qt. Developpez.com. [Online]. Available: http://irmatden.developpez.com/tutoriels/sdl/

[24] *NDI 6D Architect User Guide*, Northern Digital Inc., March 2004, Rev 4.

[25] *Optotrak Certus Rigid Body and Tool Design Guide)*, Northern Digital Inc., March 2006, Rev 2.