Systems Engineering and Automation Department

Bachelor's Thesis

OCULAR

In-hand object detection and tracking using 2D and 3D information

Author: Irene Sanz Nieto

Advisor: Víctor González Pacheco

Leganés, June 2014

Acknowledgements

First of all, I wish to acknowledge my parents because they have always supported and encouraged me. I would like to thank Alvaro as well for his support and comprehension during the whole project and for his help and patience in the difficult moments.

I am particularly grateful for the assistance given by my thesis supervisor, Víctor González Pacheco, for giving me some key pointers without which this project would not have been possible. I also would like to thank the teachers I had during my life. Those who were better for helping me to easily acquire the knowledge of their subjects, and those who were worse for aiding me in learning how to overcome difficulties and be self-sufficient.

And, finally, I would like to thank my family and friends, specially Rocío and Blanca, for always being there for me.

Agradecimientos

Lo primero querría dar las gracias a mis padres, por el apoyo y el ánimo que me dan siempre. También me gustaría agradecer a Álvaro su apoyo, comprensión y paciencia durante el proyecto y su ayuda en los momentos más difíciles.

La contribución de mi tutor, Víctor González Pacheco, ha sido muy importante, principalmente por algunas indicaciones que han hecho posible la realización de este proyecto. Me gustaría reconocer también la ayuda de todos los profesores que he tenido en mi vida de estudiante. Aquellos que eran mejores, por ayudarme a adquirir fácilmente los conocimientos y aquellos que eran peores por ayudarme a aprender cómo superar dificultades y llegar a ser capaz de aprender autónomamente.

Finalmente, me gustaría dar las gracias a mi familia y amigos, especialmente a Rocío y a Blanca, por estar siempre ahí cuando les necesito.

Abstract

As robots are introduced increasingly in human-inhabited areas, they would need a perception system able to detect the actions the humans around it are performing. This information is crucial in order to act accordingly in this changing environment. Humans utilize different objects and tools in various tasks and hence, one of the most useful informations that could be extracted to recognize the actions are the objects that the person is using. As an example, if a person is holding a book, he is probably reading. The information about the objects the humans are holding is useful to determine the activities they are undergoing.

This thesis presents a system that is able to track the user's hand and learn and recognize the object being held. When instructed to learn, the software extracts key information about the object and stores it with a unique identification number for later recognition. If the user triggers the recognition mode, the system compares the current object's information with the data previously stored and outputs the best match. The system uses both 2D and 3D descriptors to improve the recognition stage. In order to reduce the noise, there are two separate matching procedures for 2D and 3D that output a preliminary prediction at a rate of 30 predictions per second. Finally, a weighted average is performed with these 30 predictions for both 2D and 3D and the final prediction of the system is obtained.

The experiments carried out to validate the system reveal that it is capable of recognizing objects from a pool of 6 different objects with a F1 score value near 80% for each case. The experiments demonstrate that the system performs better when combines the information of 2D and 3D descriptors than when used 2D or 3D descriptors separately. The performance tests show that the system is able to run on real time with minimum computer requirements of roughly one physical core (at 2.4GHz) and less than 1 GB of RAM memory. Also, it is possible to implement the software in a distributed system since the bandwidth measurements carried out disclose a maximum bandwidth lower than 7 MB/s.

This system is, to the best of my knowledge, the first in the art to implement an in-hand object learning and recognition algorithm using 2D and 3D information. The introduction of both types of data and the inclusion of a posterior decision step improves the robustness and the accuracy of the system. The software developed in this thesis is to serve as a building block for further research on the topic in order to create a more natural human-robot interaction an understanding. This creation of a human-like interaction with the environment for robots is a crucial step towards their complete autonomy and acceptance in human areas.

Resumen

La tendencia a introducir robots asistenciales en nuestra vida cotidiana es cada vez mayor. Esto hace necesaria la incorporación de un sistema de percepción en los robots capaz de detectar las tareas que las personas están realizando. Para ello, el reconocimiento de los objetos que se utilizan es una de las informaciones más útiles que se pueden extraer. Por ejemplo, si una persona está sosteniendo un libro, probablemente esté leyendo. La información acerca de los objetos que las personas utilizan sirve para identificar lo que están haciendo.

Esta tesis presenta un sistema que es capaz de seguir la mano del usuario y aprender y reconocer el objeto que ésta sostiene. Durante el modo de aprendizaje, el programa extrae información importante sobre el objeto y la guarda con un número de identificación único. El modo de reconocimiento, por su parte, compara la información extraída del objeto actual con la guardada previamente. La salida del sistema es el número de identificación del objeto aprendido más parecido al actual.
El sistema utiliza descriptores 2D y 3D para mejorar la fase de reconocimiento. Para reducir el ruido, se compara la información 2D y 3D por separado y se extrae una predicción preliminar a una velocidad de 30 predicciones por segundo. Posteriormente, se realiza una media ponderada de esas 30 predicciones para obtener el resultado final.

Los experimentos realizados para validar el sistema revelan que es capaz de reconocer objetos de un conjunto total de 6 con un valor F cercano al 80% en todos los casos. Los resultados demuestran que el valor F obtenido por el sistema es mejor que aquel obtenido por las predicciones individuales en 2D y 3D. Los tests de rendimiento que se han realizado en el sistema indican que es capaz de operar en tiempo real. Para ello necesita un ordenador con unos requerimientos mínimos de un núcleo (a 2.4 GHz) y 1 GB de memoria RAM. También señalan que es posible implementar el programa en un sistema distribuído debido a que el máximo de ancho de banda obtenido es menor de 7 MB/s.

Este sistema es, según los datos de que dispongo, el primero en incorporar un reconocimiento y aprendizaje de objetos sostenidos por una mano utilizando información 2D y 3D. La introducción de ambos tipos de datos y de una posterior etapa de decisión mejora la robustez y la precisión del sistema. El programa desarrollado en esta tesis sirve como un primer paso para incentivar la investigación en este campo, con la intención de crear una interacción más natural entre humanos y robots. La introducción en los robots de una capacidad de relación con el entorno similar a la humana es un paso decisivo hacia su completa autonomía y su aceptación en áreas habitadas por humanos.

Contents

List of Figures

Chapter 1

Introduction

The robotic field has experimented an enormous increase in the past years. Different components and systems have improved. As an example, cameras have evolved and low-cost 3D sensors have appeared. This fact provides a higher and more reliable amount of information for the computer vision system of the robot. Also, the processing units have been upgraded, allowing a larger amount of computing that permits the inclusion of more complex algorithms in these robots. This leads to the increasing introduction of robots in human environments with assistive or social tasks.

The first automatisms performed their tasks without being aware of their environment. Their work area consisted mainly on closed areas in to which humans were denied entry. But the introduction of robots in the houses and other places frequented by humans increases the importance of perceiving the environment accurately. The robot must be able to recognize and interact with the objects and persons around it. This interaction needs many different sensors or input systems and as many output systems to retrieve the information and respond to it correctly.

The human environment has a lot of variables that contain information about the user's intentions or actions. One of this variables are the objects the human uses. For example, if a person is holding a toothbrush, he is probably going to brush his teeth. Also, if the user is holding the keys in his hand he might be going out of the house. It can be easily seen that identifying the objects the humans hold in their hands, key information about the actions they are undergoing is extracted. If the robots could recognize those objects, they could adapt their behaviour to the situation around them. As an example, if the user is holding a book, he is probably reading and he does not want to be disturbed except for an emergency. The robot could then change its behaviour to the new constrains of the environment, for example, making as less noise as possible.

Having a system that could track and identify in-hand objects could improve the robot's interaction with its environment. For this purpose I have developed a system that is able to learn and recognize hand-held objects in real time through a simple and intuitive gesture interface. This software is capable of recognizing the user's skeleton and, from it, extract the hands position. With this information the system studies the area around the hands and compares it with a previously acquired dataset. It is possible to use the software without input and output devices such a screen or a mouse, since it integrates a pose interface. This fact easies the transition from the dataset acquisition to the recognition of objects and allows the system to be implemented in a robot.

1.1 Socio-economical context

Europe's population is ageing. This phenomena occurs because of the augment in the life expectancy and the declining of birth rates. Recent studies provided by the World Health Organization (WHO) showed that the life expectancy in Spain is the highest in Europe, specially in women [1]. Spanish women live an average of 85.1 years in 2012, and the men around 79 years. In fact, the Spanish women life expectancy is only behind the Japanese women, that is, the second highest level in the world.

In the World Population Ageing Report of 2013 it is stated that "by 2050, Bosnia and Herzegovina, Germany, Malta, Portugal, Serbia and Spain are projected to attain median ages of 50 years or more"[2].

This means that soon enough there are going to be more elder people than young people. How are we going to be able to support and aid that major sector of the population? The solution might be to incorporate robots to help in this task [3].

Assistive robots can help elder people to interact with their environment. They can cooperate in the health control of the user testing the blood pressure or recognizing falls. Also, social robots may maintain a conversation and interact with people, for example, reminding them to take a specific medication. Japan is the country with the highest population ageing of the world. This country is the world leader in the assistive robotics field. Recently the he International Organization for Standardization (ISO) has adopted Japanese standards for assistive robotics technology [4][5]. The Japanese government is studying to introduce a system of assistive technologies and hence the funding in this field is high. As an example of the technology being developed in this country, the Pepper assistive robot was recently anounced by SoftBank Corp[6]. It is a robot that costs less than two thousand USD and that is capable of maintaining a conversation with the user. However, Pepper is not capable nowadays of performing assistive tasks. It has a humanoid form with arms, but they are only used to express and empathize the information that it is giving. Is, hence, the first step towards more environment-aware and interactive low-cost robotics.

In Spain the assistive field is mainly being evolved in universities and research groups. Projects such as the RobAlz between the Fundación Alzheimer España (FAE) and the Carlos III University [7] are developed to improve the interaction between robots and, in this particular case, Alzheimer patients. Another example of the Spanish research in the field is the work performed by the Ave María Foundation [8]. They have introduced various robots in the therapies of their patients such as the NAO or the REEM robots. NAO is a small robot that can recognize faces, sounds and is able to walk. It is used to instruct therapy seasons. The REEM robot, on the other hand is used to transport the necessary items such as clothes or food around the place, allowing the employees to focus on assistive tasks. Nevertheless, the research in Spain is severely restricted due to the dramatic reduction that has been inflicted in its budget over the past five years. Recently, the Spanish government presented the 2014's Research and Innovation budget. The numbers show that in 2014 the budget is of 6.148 millions of euros, almost a third less than the budgets of 2008, 2009 and 2010 [9].

The computer vision field has numerous applications in very different industries. Those range from assistive robotics and medicine to industrial and defense and security programs or even autonomous cars. It is a very active field in which many big companies such Bosch, Siemens or Sony are investing. In Spain the computer vision research is mainly performed in the universities in projects such as [7]. It is used as a tool for the construction of bigger robotic systems. Nevertheless, it is affected by the low budget available for I+D in this country as well, a fact that reduces the number of research projects being developed.

1.2 Motivation

Technology has evolved enormously in the past years. Nowadays most of the robots being developed are only able to recognize a small part of their environment. The decision algorithm of the robots is normally based on the instructions received from the user. Those commands are usually given by voice or inputting the desired task on a certain User Interface (UI).

The algorithm decision of the robots must be upgraded if they are to perform tasks in a human inhabited area. They must be able to respond to commands, but also be aware of their environment and respond autonomously to it. When this change has occurred, the robots may successfully perform the assistive tasks now reserved only to humans. But there is still a long way to research, mainly around the perception of the environment. The investment needed for its development is now being held due to the recession that appeared in the late 2000s decade, whose effects are nowadays still present. Nevertheless, this lack of funding might be mitigated using research, open data and open-source code.

I strongly believe in the ideals presented by the Open Source Initiative. This project is intended to be an open source code that can be a building block for other researchers that work on robot perception. There are various scientists that have already studied the importance of the objects around the human in the action recognition. For example, in [10], a study of the human-object interaction in still images was performed in order to relate those objects to actions. This relation between object and action may also be seen in [11], in which wearable cameras are used to retrieve the input. The objects being hand-held

are recognized and the action associated to them is learned. In this thesis I present a system that allows to easily learn and recognize hand-held objects in real time. It is intended to be the previous step to the association between the object and the action.

1.3 Proposed solution for in-hand object recognition: OCULAR

In this thesis a software has been developed that is intended to be used in assistive and social robots. The name and the logo (figure 1.1) of the project gives a first idea of the system's goal. The name of the project, OCULAR, is an acronym of "On-line objeCt Learning And Recognition". The first word, on-line, refers to the fact that the learning of new objects is done in real time. Figure 1.1 presents the logo of the software, which gives two additional pieces of information: the object learning and recognition is performed in-hand and the input of the system is a RGB-D (Red, Green, Blue and Depth) sensor, such as a Kinect. This type of sensor provides 2D and 3D information of the user located in front of it.

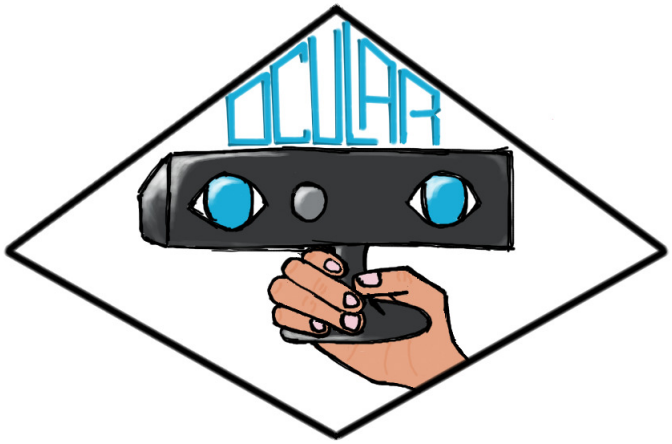
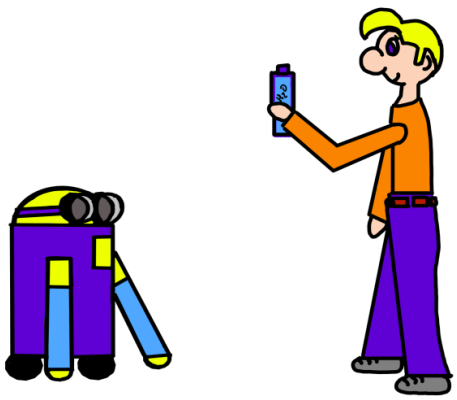


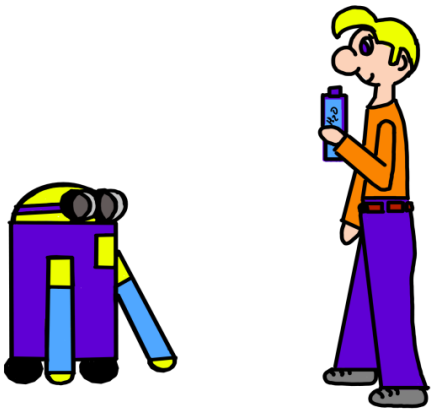
Figure 1.1: OCULAR logo

Since it is intended to be running on a robot, the software has to implement an intuitive human-machine interface. A gestural interface was designed that allows the passing of information from human to computer in a simple way. The software has two differentiated modes, the learning or data acquisition mode and the recognition or system exploitation modes. The system is able to detect the change of the mode using the pose of the user as can be seen in figures 1.2a and 1.2b. In order to trigger the learning mode, the user only needs to extend his arm towards the sensor, showing the item to it (figure 1.2a). This is a natural gesture usually performed between humans when introducing new objects to one another. Having the hand closer to the body the recognition mode is triggered and the program outputs a unique id for the learned object (figure 1.2b).

Apart from switching modes evaluating the user's pose, the system also detects the hand that is holding an object. The software is able to work with one hand at a time and it chooses the one that is higher. The present project has been designed to be as modular and reusable as possible making an easier task to develop complementary software such as handbags recognition or hats recognition with slightly modifications. Its structure consists in nodes that run on parallel and minimizes the lag due to the computing processes. It is an Open Source project, meaning that anyone can use and modify it.



(a) **Learning mode** triggering using the gestural interface. The learning phase starts when the user stretches the arm towards the RGB-D sensor. The system takes a definable number of views of each object, with a delay between views of 1 second. This allows the user to change the object's pose to learn a different view of the object.



(b) **Recognizing mode** triggering using the gestural interface. When the user pulls his hand closer to the body and the learning phase has finished, the recognizing mode starts. It is the default mode of the software. This means that if no event occurs the system keeps outputting the unique ID for the learned object that was recognized.

Figure 1.2: OCULAR working modes

In order to facilitate the usage of the code it has been developed as a Robotic Operating System (ROS) [12] package. The source code and further installation instructions and license details might be found in the following link to the software's repository[1].

1.4 Objectives

The objectives of this thesis are listed below.

- To develop a system capable of recognizing objects in real time. Recognition means the detection of new objects and the comparison with a previously obtained dataset outputting the ID number of the most similar template.
- To develop a system capable of learning objects in real time. This includes the storage of the dataset for further usages. The learning process is performed acquiring a definable number of views per object.
- The system must be able to detect the person that is in front of the robot.
- The software must detect the location of the hands of the user in order to extract from there the object to be recognized and learned.
- The system must be able to learn more than one view per object.
- The software must have a gestural interface that allows to trigger the recognizing and the learning modes.

[1] http://github.com/irenesanznieto/ocular

1.5 Thesis structure

The thesis begins with a perspective of the related art on the field and the different technologies that are commonly used in computer vision. This comparison of the previous work and the one proposed in the thesis can be found in chapter 3. For details regarding computer vision fundamentals, please refer to chapter 2.

Afterwards, in chapter 4 , the proposed solution is described and explained. The different parts are individually presented and the whole functioning of the software thoroughly explained. The alternatives to the proposed model are also shown in chapter 4 as well as the reasons behind the selection of the system that is presented in this thesis.

The experiments carried out to validate the system are presented in the chapter 5. Chapter 6 is devoted to the analytical explanation of those experiment's results. These results are then compared and analyzed in chapter 7.

Finally, the conclusions extracted from those experiments and the possible future lines of work are exposed in chapter 8.

Chapter 2

Computer Vision fundamentals

This chapter aims to give a global overview of the main computer vision vocabulary and methods related with object learning and recognition. Each section present a different processing step. The main processing phases of an object learning and recognition system are:

1. **Data input: Hardware**
   The data input of the system is performed using a vision acquisition device. In this thesis, an RGB-D (Red, Green, Blue and Depth) sensor is being used. Section 2.1 presents the most used vision sensors in robotic computer vision.

2. **Raw input data processing**
   Usually, the acquired information needs some preprocessing. This step transforms the input data coming from the sensor into a data structure that is easy to handle and useful for posterior processing. A more complete explanation can be found in section 2.2

3. **Segmentation**
   The next step is to segment the input data, that is, reduce it to the Region Of Interest. For further information please read section 2.3.

4. **Object description methods**
   The input data is not suitable to be matched as it is. In order to apply recognition and matching algorithms, the extraction of certain relevant information of each object is required. That extracted information is the one that describes the object and that can be used in matching algorithms. More information can be found in section 2.4.

The different phases appear in the same order in the software developed in this thesis. Further details about the processing steps may be found in section 4.4.

2.1 Hardware

The advance in the computer vision discipline is greatly linked with the development in the hardware. The hardware components present in a computer vision system are the following:

- Power Supply: Device needed by the other components in order to work.
- Acquisition device: Device that captures the world and represents it as an array of data. That data can be two or three dimensional.
- Processing unit: Receives the information from the acquisition device and processes it. It is usually programmable. Nowadays the most used processing units are PCs.
- I/O unit: Serves as a bridge between the acquisition device and the processing unit if needed.

In this chapter the state of the art of the different acquisition devices is going to be presented.

2.1.1 Acquisition devices used in object learning and recognition

The two most used devices in object learning and recognition are the RGB-D (Red, Green, Blue and Depth) sensors and the cameras. While RGB Cameras provide information in 2D, RGB-D provide information of the depth of the scene. This added information makes image processing easier because it reduces the ambiguities of the 2D data. Nevertheless, the computation time needed is also much higher.

The usage of one or another acquisition device depends on the application. For example, if a more precise information is needed, the RGB-D sensor is selected and if the computational cost must be low, a camera is used. The cameras are usually classified according to the technology of the camera, the dimensions of the sensor, the number of sensors and the output type. In object learning and recognition and in general in computer vision applied for robotics, the cameras used are Complementary Metal Oxide Semiconductor (CMOS), matricial cameras with 3 sensors and digital output. Each of the sensors is calibrated to detect one of the color components of the final image (RGB, Red, Green and Blue). In this system, a RGB-D (Red, Green, Blue and Depth) sensor has been used.

2.1.2 3D sensors

Computer vision is a field that needs specific hardware to retrieve a description of the world. This description has been done for a number of years in two dimensions. 3D sensors based on lasers devices existed for a long time but they are expensive. This fact diminished their use in the investigation field. But this changed when the first version of an affordable RGB-D sensor appeared in 2010: the Kinect. This project uses a Kinect RGB-D sensor as the input of the system.

This sensor was designed to be used in games, but developers soon realized the enormous potential of the hardware for Computer Vision. Now, instead of a two-dimensional information as an input, it was possible to have three-dimensional information. In November 2010 OpenNI was created. Openni is an open-source software framework that can read the data from RGB-D sensors [13]. That same year, PrimeSense released their open source drivers and motion tracking middleware called NITE[14]. PrimeSense is a company that manufactures RGB-D sensors and, in fact, the Kinect is based on their depth sensing reference. The software released by PrimeSense worked with the Kinect as well. From that time on, the OpenSource software related with the kinect has increased as well as the different models of RGB-D sensors available in the market. The Microsoft corporation finally released the SDK (Software Development Kit) on June, 2011 [15].

The 3D sensor consists in an infrared projector and an infrared camera. The projector creates an infrared pattern that is captured by the camera. Then, it is compared with the references stored in the device and the depth of each point is estimated. Afterwards, the depth data is correlated to a RGB camera. The output can be described then as a point cloud, a type of data that consists in 3-dimensional and color information for each point.

It can be easily seen that the use of the infrared spectrum provides a system that can be easily perturbed. As an example, the incandescent light irradiates infrared waves that distorts the output of this type of RGB-D sensors. Also, the sun's IR waves affects the measures. The RGB-D sensors may be used hence only in interiors.

2.2 Raw RGB-D sensor data processing

The Kinect is the sensor which acts as the input of the system. The output of this device cannot be used directly. A processing step performed by a driver is needed. In the case of this thesis the OpenNI library is being used. Two different computing phases are needed. The first is used to transform the raw Kinect's output to a formatted data. The second step transforms the formatted data into a more useful structure such as an image or a point cloud. These structures are easier to use and they contain a more condensed amount of data than the formatted information obtained in the first phase.

In this thesis these processing steps are being performed using third-party libraries. The formatted output of the first phase is converted into an image and a point cloud. An image is a matrix of values in which the RGB (Red, Green, Blue) components of each pixel or point is represented. A point cloud is also a matrix similar to the image, but with the difference that each point has also an associated position. The output of the kinect is processed to match the information contained by the image and the point cloud. This processing outputs a more useful data structure : a point cloud with RGB component, that is, a point cloud with color information in it. For further information of this process, please read section 4.4.4.

2.3 Segmentation

The segmentation consists in the extraction of the Region Of Interest (ROI), that is, separating the interesting parts of the image from the background of the input data. The segmentation is performed to remove all the useless information from the input data to the system. This results in a reduction of the information size. The fact of reducing the data size is advantageous as well because the time expended in later computations is reduced. Also, less noise is introduced in the system, increasing its performance.

The method applied to this step varies with the application. For example, for object recognition in a table the segmentation would be performed looking first for a flat surface and then extracting the region around it. In the case of this thesis, since the object learning and recognition is done in-hand, these extremities are being tracked. First, the hands' center position is obtained and later a fixed square is cropped around it. More information about this process can be found in section 4.4.5.

2.4 Object description methods

In order to be able to compare two objects, features or descriptors are extracted from them. The definition of feature changes depending on the application and context of the computer vision project. In this thesis, the definition that is going to be used is the following: a feature or descriptor is an interesting or important characteristic, point or region of an image.

he application determines which feature describes better the object of study. Descriptors may have different forms and characteristics. For example, if we want to recognize rectangles, its height could be a descriptor. The height is not scale invariant, but it is rotation invariant. If the application needs an scale independent descriptor, the ratio between the height and the width of the rectangle may be used. This simple example shows how a feature is defined when the sample object is known. But in this thesis I present a system that must be able to recognize all kind of common objects. There is no straight forward method to extract a feature as in the example, because there is no structure or characteristic that is repeated in every object a human can hold.

Descriptors may be local or general. A local descriptor represents a point using the information of the points around it. Local features from one object may be almost identical to a local feature from a very different object. An example of local feature could be the color map of a region composed of a certain number of pixels. The same pattern might be found in different highly-textured objects such as a book and a map. In order to overcome this problem, general or global descriptors are computed. A general descriptor is usually a set of local descriptors that undergo a transformation. For example, the local descriptor previously presented could be extracted from four points of the same object located in the borders. The information of the global descriptor would be a combination of the one given by the local descriptors and a parameter relating them, for example the relative distance between them. This creates a pattern that is more difficult to reproduce in different objects and hence it is more representative for it. Global features are used to obtain a more representative description of the studied object, but they are more computationally expensive.

The best feature or descriptor is the one that possess a higher repeatability, i.e the ability of obtaining the same output given different inputs. In this system, repeatability may be described as the ability of obtaining the same predicted object given different views of it. Recognition algorithms depend directly on the repeatability of the features they use. If the descriptors have a high repeatability, the system performs better than in the case of having a lower repeatability. Chapter 3 presents a description of the state-of-the-art feature extraction algorithms and the differences between them.

Chapter 3

State of the art

3.1 History

The evolution of computer vision might be seen as the evolution of the optimism of the researchers with respect to this field. In the 1950s the first computers were being developed. There were many science-fiction films and books that suggested a new robotic era approaching. The optimism and the confidence in the power of computers were enormous. Since seeing is easy for humans and the computers had a big capacity for computing, it was thought that the artificial vision field (as it was called in that period) was going to grow rapidly.

In the 1960s the researchers realized the complexity of the data processing made by the human brain. The fact that regular cameras received information in a 2D projection of the 3D real world required a processing step to interpret and make an abstraction of that data. It was not known by then how the brain processes the information received from the eyes and, therefore, it was not possible to reproduce it in a software. At this time, it was discovered the importance of the learning stage in the recognition of objects. Larry Roberts is accepted as the father of the Computer Vision field [16]. His Ph.D. thesis [17] was the model that later scientists followed. It studies the possibility of making a computer construct and display a three-dimensional array of objects given a single two-dimensional picture.

It was in the 1980s when the researchers changed the way of thinking about the field. They realized that the research must be centered in the low-level vision to process the input images and make them more easy to interpret and manage to the computer [16]. The evolution of the hardware (both computers and specialized hardware) also influenced the development of more complex projects.

3.2 Computer vision today

Nowadays the computer vision field is divided in many different branches. Some of these branches are object recognition, learning, indexing, video tracking, scene reconstruction, event detection, motion estimation and image restoration, among others. Nevertheless, they are not separate fields of study. Often the systems perform tasks that belong to different branches in order to achieve their goal.

In this thesis I present a system that allows to easily learn and recognize hand-held objects in real time. It could be classified inside the object recognition and learning, video tracking and event detection branches of study. This chapter is devoted to the discussion of the previous literature related to this topic. First the most similar systems to this thesis are presented and discussed. Afterwards the common items to these systems are presented as well as the most suitable alternatives for the application presented here.

3.3 Object learning and recognition

Object learning and recognition is an active field. In this section the most important works related to that field are presented. Also, the differences and improvements performed in this thesis with respect to the previous art are stated.

3.3.1 Object learning and recognition using 2D and 3D input data

This special branch in the object learning and recognition field has been active for many years. One of the first works in the field was the one performed by D.M.Gavrila and F.C.A. Groen. In 1992, they developed a system that performs 3D object recognition using 2D input data [18]. A 3D model was created and then the new 2D projection was matched to this model. A geometric hashing method of matching was applied to the input 2D projection. The image was only matched to a model if the error to a nearby 2D projection was within certain thresholds. The testing was performed under a controlled environment and using textured objects. The system would not give the same results in a real environment with noise and loosely defined objects. Also, the use of hashing limits the size of the dataset due to its high computing cost.

Later systems as [19] explore different descriptors for the objects that result less time-consuming. In this case, fuzzy logic is used to match the learned features to the new ones. Those descriptors are a set that include Affine [20], Zemike [21] and Hu [22] moments invariants among others. The results obtained a high percentage of true positives. But, again, the system uses a controlled setup: The input is collected from three cameras that has a known illumination and orientation. The objects are white with significantly different shapes and they rest in a black background during learning and recognition phases. The applications in which it could be used are more related to the industrial field than the robotics field.

An outdoor system is described in [23], which demonstrated the effectiveness of two methods when combined: local descriptors and 3D wireframes. The system was able to distinguish between cars and bicycles and to estimate their pose. The input of the recognition is a single 2D image. Nevertheless, the training was made off-line, using a high number of Computer-Aided Design (CAD) model's views.

These systems differ with the one presented in this thesis in one major aspect: the user-software interaction. This thesis is based on providing an easy and intuitive interface. The learning process is performed easily and on-line and the recognition is real-time. There is no setting needed, no previous processing and care about the environment. The combination of 2D and 3D information in my system is, to the best of my knowledge, unique. The fact of recognizing the objects using those two different methods and then apply a decision algorithm better the results, reducing the effect of the noise.

3.3.2 In-hand object learning and recognition

Most of the literature in object learning and recognition has been developed using wearable cameras as the input of the system. One of the most representative works is [24], which consists in a camera used to capture daily objects to construct a dataset more easily. Its main feature is that it eliminates the necessity of manually segmenting and labeling the learning datasets for object recognition libraries. It only learns new templates that are later fed to a off-line learning algorithm.

Another example is [25], in which a benchmarking of an egocentric object recognition system is performed. This in-hand object recognition uses as input an image that has the object used at the center. The background is not segmented in this system because the errors produced by it may be neglected. The object occupies most of the camera's frame and hence the processing of the input data to the system does not need to be as exhaustive as the one developed in this thesis.

The system presented here has a different approach: an in-hand object recognition and learning in which the user is located in front of the acquisition device. It is a solution to the same problem, the recognition of an object that a user holds, but the initial setting is different. In my thesis, the user is in front of the camera, because the software is running on a robot. The interaction between the software and the user is different from the previous art on the field. This change in the input creates an additional difficulty due to the hand's segmentation in the input image.

Furthermore, to the best of my knowledge there is no previous art in in-hand object learning and recognition using 2D and 3D information combined together. All of the algorithms described in this section use a camera and hence 2D data as input. The inclusion of 3D and the independence between the 2D and 3D recognition processes improve the noise resistance of my system with respect to the previous art.

3.4 Feature Extraction algorithms

In chapter 2 a brief introduction to the characteristics of the different feature extractor algorithms was performed. In this section the different state-of-the-art algorithms for 2D and 3D feature extraction are presented and compared.

3.4.1 Algorithms for 2D Feature Extraction

The most used algorithms for general object recognition are presented in this section. They appear in chronological order and the differences, similarities and improvements between them are explained. The selection of one algorithm or another depends on the application.

SIFT

SIFT (Scale Invariant Feature Transform) is a scale and rotation invariant feature descriptor [26]. There are various papers in which the performance of SIFT is compared with other descriptors and one of the most representative is [27]. In it, it can be seen that SIFT outperforms the previous algorithms, mainly due to its combination of local information and relative strengths and orientations of gradients. This combination makes it more robust to illumination and viewpoint changes and to noise.

The SIFT algorithm is based on four main phases: The first step is to detect the scale-space extrema, i.e. the location of potential interest points that are invariant to scale and rotation. The next step is to test those potential keypoints and select the more relevant based on measures of their stability. Then, one or more orientations are assigned to each of the previously obtained keypoints based on local image gradient directions. The following operations are performed on data that has been transformed relative to the location, scale and orientation of these features. This grants the invariantness to these transformations. The final phase consists on the description of the selected keypoint transforming the local image gradients into a representation that is descriptive enough to permit the information of various levels of shape distortion and illumination change.

In order to minimize the cost of extracting such a distinctive features, a cascade filtering approach is used in order to apply the most time-consuming operations only at locations that pass an initial test. It can be used for on-line applications but it still has a latency that was later improved. In order to reduce that lag, there were different approaches previous to the apparition of algorithms such as ORB, that reduced it drastically [28]. As an example, in [29] the SIFT algorithm was implemented on a FPGA (Field Programmable Gate Array), improving its speed by an order of magnitude and thus allowing it to run in real-time.

The main reason of the high computing time of SIFT features is the descriptor vector size. In the aim of creating a highly distinctive descriptor, the vector is over-dimensioned slowing the detection, description and matching processes. This over-dimension is patent in the duplication of information that could be removed without affecting in a high degree its descriptiveness.

SURF

SURF (Speeded Up Robust Features) is a scale and rotation invariant interest point or keypoint detector and descriptor [30]. It is a proprietary algorithm that simplifies the detection, extraction of the descriptors and matching steps thus obtaining them much faster than previous algorithms without losing repeatability, distinctiveness or robustness.

SURF algorithm is composed of three main steps: The first step of the algorithm is to identify the interest points such as corners, blobs or T-junctions, i.e. a junction were two lines meet forming a T. Therefore, this algorithm will be useful when evaluating a textured object. The next step is to represent the neighbourhood of the interest points as a feature vector. The final step is to match the descriptor vectors between different images, in order to stablish a recognition of a pattern. Usually the matching is performed using as a reference the distance between the vectors. In this part, it can be perceived that the size of the descriptor vectors affects directly the performance of the algorithm. SURF aims to reduce that size without losing distinctiveness in the features.

The SURF algorithm appeared after SIFT and hence it is interesting to see the similarities and differences between the two. In section 2.4 it was seen the good results obtained when combining the local information and relative data regarding gradients. This algorithm is based on similar characteristics: First, an orientation based on the information extracted from a circular region with the interest point as its center is obtained. Then, a square region aligned to that orientation is described and the descriptor is extracted from it.

The experiments in [30] show that the performance of these descriptors equals and in some cases improves the one of the SIFT descriptors. Also, the SURF descriptors are much faster computed and matched due to its smaller size.

ORB

ORB (Oriented FAST and Rotated BRIEF) is a fast rotation invariant, noise resistant binary descriptor based on BRIEF [28]. ORB authors claim that it is two orders of magnitude faster than SIFT while matching its performance in many situations. Nevertheless, since ORB is not scale invariant, if the scale difference is noticeable the SURF algorithm will outperform ORB.

The features used in ORB build on the Features from Accelerated Segment Test (FAST) [31] keypoint detector and the Binary Robust Independent Elementary Features (BRIEF) [32] descriptor. Both of this previous algorithms offer a good performance and computing time relation. Since neither of them had the orientation taken into account, the main improvement made by the ORB developers is to include this feature in the algorithm. Also, the computation of oriented BRIEF features was improved and an analysis of variance and correlation for this features created.

FAST is mainly used to find keypoints in real-time systems that match visual features. The orientation operator included in this algorithm is the centroid operator described in [33]. This technique is not computationally demanding and also, unlike SIFT, it returns a single dominant result.

BRIEF uses simple binary tests whose performance is similar to SIFT with regard to robustness to lighting, blur and perspective distortion, but it is sensitive to in-plane rotation. In order to eliminate this drawback, the lowest computing costing solution is to steer BRIEF accordingly with the orientation of the keypoints.

In the different tests in [28] can be seen that the percentage of inliers obtained with ORB are higher and do not variate as much as those obtained by SIFT or SURF. ORB is then a good alternative for the latter if the application does not need a scale invariant descriptor. Finally, it is noticeable that this algorithm is Open Source. Both SIFT and SURF are patent protected and the use for research is permitted, but for commercial uses the payment of a fee is needed. For all the above reasons, the ORB algorithm is the one being used in the system developed in this thesis.

3.4.2 Algorithms for 3D Feature Extraction

Applied to 3D data, a feature is a characteristic that describes a point inside the data. Features or descriptors can be compared to determine whether the point described is the same in two different inputs. They are used in object recognition and there are many different descriptors that can be used. Each of them has a certain speed in its computing and a reliability and robustness associated.

Depending on the application in which they appear the developer must select the features depending on the specifications. As an example, two geometric point features are the underlying surface's estimated curvature and the normal at a specific query point. Both are local features and they characterize the point using the information provided by its neighbors. There are two types of features: local and global. Local features are less time-expensive but they are less robust. This means that two different objects may obtain very similar local features. Global descriptors implement different techniques to generalize the information obtained for keypoints or important points in the mesh. They take more time to compute but are more robust than the previous ones. Since in our application the speed is key, local features are being used.

There exist a number of 3D local features that are computed using combinations of different geometric characteristics of the points. As an example, the 3D SIFT descriptor [34] is obtained performing the 3D gradient and magnitude for each pixel, which is directly derived from its computation in 2D. It is rotation and scale invariant but is very time-consuming. The 3D descriptors being used in this thesis are the Point Feature Histogram (PFH) features.

3.4.2.1 Point Feature Histogram (PFH)

The Point Feature Histogram [35] is a local 3D descriptor. It is fast to compute but its level of detail is limited. They are computed approximating the geometry of a point's k-neighborhood with a few values. This fact results in the possibility of obtaining a similar set of points in a very different object.

The PFH descriptors are invariant to rotation, position and point cloud density. They also perform well with noisy data. The features are created representing the mean curvature around the query point using a histogram of values. The normals' direction and magnitude of the points in the k-neighborhood of that point are studied.

The Fast Point Feature Histogram (FPFH) [36] descriptor is also a local descriptor based in the PFH. It is faster because it considers only the direct relations between the query point and its neighbors. This fact makes them less robust than the previous descriptors. This is the reason why the Point Feature Histogram descriptors are the ones being used in this thesis instead of the Fast Point Feature Histogram descriptors.

Chapter 4

System Description

4.1 Introduction

This chapter presents the description of the system developed. The proposed solution implements a software capable of tracking the user's hands and learning and recognizing the objects that are hand-held. The project's software is modular and the processing has been distributed among different processes. Those processes are called nodes.

The Robotic Operating System (ROS) framework has been used in order to improve the communication between nodes and the overall performance of the code. The Point Cloud Library (PCL) and the Open Source Computer Vision (OpenCV) C++ libraries have been used in the project to enable 3D and 2D data processing, respectively. The following sections are devoted to the design alternatives and the description of the final design used in the project.

4.2 Design alternatives

This section presents the steps that were taken to obtain the final system. The different intermediate designs are shown and the reasons behind the change from one to another are explained. The final design can be found in section 4.4.

First design

The first design was based in using a hardware with low cost and high accessibility, a camera. The decision of including that device forced the input information to be two-dimensional. The C++ programming language was selected for its high performance and the OpenCV library was picked to aid in the data processing. It was seen that the two dimensional information was not sufficient to obtain a reliable hand tracking. Also, this task created a high lag due to the high amount of computing it needed.

Hand tracking problem

In order to reduce the time spent in the hand tracking, it was decided to take advantage of the 3D capabilities of the RGB-D sensors. In this case, a Kinect is used, which is a low cost device. This type of sensors return a point cloud, a data matrix that presents the objects in front of it using three-dimensional information. This fact allowed to segment the scene using the depth parameter, reducing the amount of data that needed processing.

At this point, it was necessary to create a hand-tracker before starting the project itself, that is, the object learning and recognition nodes. Also, the introduction of the new coordinate created a problem. OpenCV is designed to deal only with 2D images and hence another library that implements state-of-the-art algorithms for 3D processing was needed : the Point Cloud Library (PCL).

Code structure

After coding some test cases it became evident that the lag created by the data processing was large enough to difficult the human-machine interaction. If the software was supposed to run on real time, it could be not sequential. It was then when, the decision of distributing the tasks between nodes was made. Nevertheless, this decision created a new problem: the communication between them. This could be solved using pipes, sockets and shared memory but it obviously complicated the code programming.

Programming Framework

To overcome this difficulty the Robotic Operating System (ROS) programming framework was introduced in the design. ROS integrates node communication, threading and many Open Source packages that could be useful. One of those packages is the pi_tracker, that implements a joint tracking software using the ROS tools. This software that was already present and available was useful because it solved the problem of the hand tracking. The output of the package is a ROS topic with the position of all the user's joints presented in a message. Thereupon the ROS framework was used and all the already created code was adapted properly. But the object learning phase was not yet defined.

Learning

The interaction between the software and the user is key. The system must work on real time and be able to learn new objects as quickly as possible. This fact was determinant in choosing the learning method of the software. It was decided to implement a template matching which do not compromise the efficiency of the software for small sized datasets and allowed to have an on-line learning.

The templates extracted in the code were initially to be only in 2D. The 3D information was only to be used in the segmentation of the Region Of Interest from the input raw data. But afterwards it was thought that extracting also the features of the point clouds could be helpful in the object recognition task. Since now the system has different nodes to perform the processing the lag added would be negligible.

Feature extraction

The object recognition is performed matching the descriptors extracted of the objects in the dataset and the new one obtained from the frame of the kinect. Since the descriptors are different for 2D and 3D data, two algorithms must be used, one for each information. In section 3 the different descriptor extraction methods for 2D and 3D are explained and the differences between them evaluated.

The feature extraction in 2D is done using the Oriented FAST and Rotated BRIEF (ORB) [28] algorithm. It was chosen because it is faster than the Scale-invariant feature transform (SIFT) [26] and the Speeded Up Robust Features (SURF) [30] algorithms extracting the descriptors of the image. The drawback it has is that it is not scale invariant. Nevertheless, in the program presented in this thesis the user must remain within one and two meters and a half from the Kinect. In this range, the size of the object seen from the sensor does not change significantly and hence a scale-invariant algorithm is not necessary.

On the other hand, the algorithm chosen for the 3D descriptor extraction is the Point Feature Histograms (PFH) [36] algorithm. Its relation between robustness and extraction speed is compliant with the project's requisites. It is fast and the features extracted are computed using just a few neighboring points. Due to this fact there might appear similar clusters in different input data that produce false positives and negatives.

The problem when using point clouds is the high amount of data that needs processing. Since the code is running in a personal computer, the computing capability is limited and hence the descriptor extraction must be as fast as possible. Nowadays the research on the field is centered in creating and improving the performance of the 3D information and hence probably in the following years this method could be substituted by another more efficient.

4.3 Description of the used libraries and technologies

This section covers the different libraries and technologies used in the development of this thesis. First, the programming framework present in the project's development is presented, as well as the different third-party libraries and packages that has been employed in the software. Afterwards, the tools used for profiling and testing the code are analyzed and, finally, the hardware being used is presented and its main characteristics explained.

4.3.1 Hardware

The hardware needed in order to retrieve three-dimensional information is an RGB-D sensor compatible with the openni_launch ROS package (see section 4.4.4.2), and a computer running Linux. The information provided by this kind of device is more complete and makes the segmentation of the Region Of Interest easier.

The sensor being used for the experiments presented in chapter 5 is a Kinect of the Xbox 360. The reason of choosing this specific sensor is that it is cheaper than the other devices that include depth information. The main drawback with respect to other sensors such as the Asus Xtion PRO LIVE [37] is that it needs a separate power plug to work, and also its size is bigger. Further information about the Kinect's functioning may be found in section 3.

4.3.2 Open Source Computer Vision (OpenCV)

OpenCV [38] is a library that implements state of the art real-time computer vision algorithms. It is cross-platform and it is being released under a BSD [39] license. This library implements an optimized version of various computer vision algorithms, a fact that makes this library an extremely useful tool for robotic vision projects. In this project the C++ interface of the OpenCV library has been used. The functionalities provided by this library are included in all the 2D data processing. The next paragraph details the operations in which the different methods of the Open Source Computer Vision Library appear.

- **Segmentation of the 2D ROI (Region Of Interest)**
  This library allows to crop the original raw image coming from the RGB-D sensor. In our application, the segmentation is done around the user's hand. For further details please read section 4.4.5.3.

- **Keypoints and descriptors extraction**
  The OpenCV library implements multiple optimized state of the art algorithms. Among them, there are a number of descriptor extractors that has been already presented in chapter 3. The Oriented FAST and Rotated BRIEF (ORB) [28] algorithm is being used in this project due to its open-sourceness, and robustness / speed relation. More information about the different algorithms implemented in OpenCV used to extract the descriptors can be found in section 4.2.

- **Descriptors matching**
  There are different matchers that might be used and that are implemented in the OpenCV library. The FlannBasedMatcher and the BruteForceMatcher are the two most used due to its speed. In this project, a FlannBasedMatcher (Fast Library for Approximate Nearest Neighbors Based Matcher) is utilized. It is faster than the BruteForceMatcher specially for larger datasets. It was selected to allow the use of more objects without losing efficiency.

4.3.3 Point Cloud Library (PCL)

PCL is a standalone library that is open source and implements state-of-the-art algorithms related to 2D and 3D image and point cloud processing. It is released under a BSD license, being free for commercial and research use. Is cross-platform and currently has been successfully compiled on Linux, MacOS, Windows, Android and iOS.

The library is divided in smaller code libraries that can be compiled separately. This modularity allows the PCL introduction on platforms with size constrains or that has a reduce computational size [40]. The Point Cloud Library is used within the software to perform transformations in the input point clouds and to perform descriptor extraction and matching in this data. More specifically, PCL is used in the following processes:

- **Segmentation of the ROI (Region Of Interest)**
  This library allows to crop the original raw point cloud coming from the RGB-D sensor. In this application, the region cropped is the one around the hand being used. More information about this operation can be found in the 4.4.5.2 section.
- **Keypoints and descriptors extraction**
  The PCL library implements multiple optimized state of the art algorithms. Among them, there are a number of descriptor extractors such as Point Feature Histograms (PFH), Fast Point Feature Histograms (FPFH) or multimodal-LINE (LINEMOD). In this project, the PFH descriptor is used due to its speed. More information about the different 3D descriptors implemented in PCL can be found in the section 4.2.
- **Descriptors matching**
  There are different algorithms that allow to match the information provided by the descriptors, such as knn search or radius search. For further information about the type of matcher used in the present thesis, please visit the section 4.4.5.7

4.3.4 ROS

ROS is the acronym of Robot Operating System [12]. It is an open source set of libraries and various tools that are aimed at robot applications. It is intended as well to improve the efficacy of collaborative robotics development, allowing easy interconnection of ROS packages. The tools, libraries and third-party packages available in the different ROS distributions are very useful when programming software for a robot. This is the main reason why this Operating System was selected as the framework in which develop the code of this Bachelor's Thesis.

The structure of the software is detailed in the chapter 4.4. It is a node structure in which different processes are executed in parallel in order to improve the efficiency of the code. Each node executes a different computation in the data-flow. Since each node is part of a chain of information between the input and output of the system, they must be connected. This information sharing is produced using the ROS topics.

The ROS nodes are simply executables that uses the ROS messaging system to communicate with other nodes. The ROS topics are messaging units in which the nodes can publish messages. Also, the nodes may subscribe to those topics in order to retrieve the messages already published on them. The messages are pieces of data that the nodes send to each other. In ROS, the messages might be customized combining the already defined standard and sensor data types to create the message with the information needed. Further information about the ROS packages being used in the system may be found in section 4.4.4.

4.4 System design

Computer vision is a field that involves high time-consuming algorithms. The processing time required is highly dependent on the type of data the system is managing. The input to a computer vision system may be 2D and 3D information. 2D data is structured as a matrix that has, for each point, 2 coordinates and the color information encoded in RGB (Red, Green, Blue). This means that it contains two numbers for the coordinates and three more integers for the RGB values per pixel. 3D information is a matrix that has 3 coordinates apart from the color of that point, which is described using 3 more integers. The handled data is enormous and the way to reduce the time lag due to those computations is modularizing the computation and executing in parallel those processes. The software was designed to be as modular as was possible so that each node performs an action that could be used in other applications. As an example, the software could be easily changed to recognize hats or shoes, just changing the initial node that extracts the information of the desired joint.

The Robotic Operating System (ROS) is designed to be used in robots. Since this project is intended to be running on a robot, the software has been developed as a ROS package. In order to manage 2D and 3D information (images and point clouds) two libraries have been used: OpenCV, and PCL. Both libraries implement basic and state-of-the-art algorithms that allow a easier and more time-efficient management of the data. Further information about these libraries might be found in section 4.3. The software is structured in nodes. These nodes have a relatively small functionality and run in parallel. The fact that they run simultaneously improves the efficiency of the code lowering the lag due to time-expensive operations. In the following sections, the nodes' functionalities are described and the communication between them presented. But first, the overall software work-flow and processing is explained.

4.4.1 Description of the proposed solution

The input of the system is the information coming from the RGB-D sensor. It produces two different data used as input: the 2D information, i.e. the raw image detected by the sensor's camera, and the 3D information, i.e. the raw point cloud. Also, there is another input to the system that shows the position of the different user's joints. This data is provided by a third-party package called pi_tracker that is explained in the following chapters.

The software was designed to be running on a robot as was previously explained. This implies that there can not be a GUI (Graphical User Interface) on a screen because the robot might not have it. Also, the usability and easiness to learn how to interact with the program was important to allow different people, not only investigators, to use it. In order to fulfill those requirements, a gestural interface was designed.

The recognition of the location of the hand with respect to the user's body shows how the arm is positioned. If it is stretched towards the sensor, the software enters in the dataset construction mode, i.e. the data acquisition and learning mode. If, on the contrary, the user's hand is located close to the body, the software starts the object recognition mode.

Figure 4.1 presents the generalized system's flowchart. The learning phase or dataset acquisition mode is started when the event recognized is *"learn"*. This is triggered when the user extends his hand towards the robot. On the other hand, if the event is *"recognize"*, the system exploitation phase is started. This event is triggered when the user pulls his hand to his body. This phase consists in the recognition of the objects that are shown to the system until a new "learn" event occurs. In the next sections each of those modes are detailed.

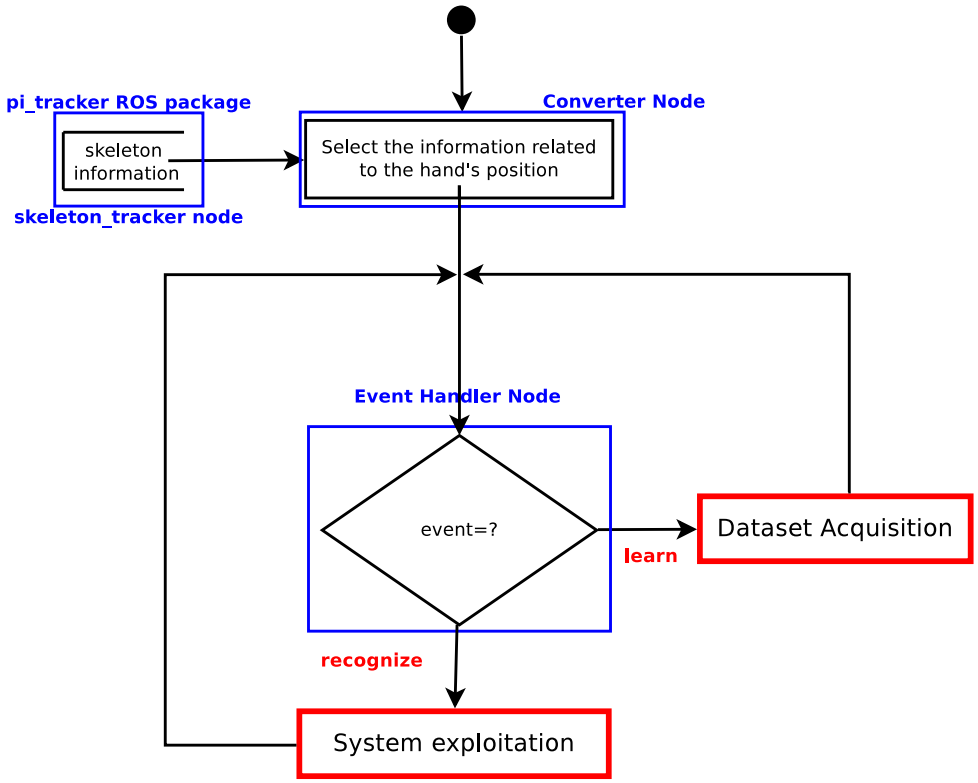


Figure 4.1: General system's flowchart showing the different modes depending on the event.

4.4.2 Dataset acquisition or learning mode

The first step is to segment or extract the ROI (Region of Interest) from the input raw data. This is a crucial step that allows to reduce noticeably the input data by cropping the image and discarding the parts that are not interesting. This implies that next phases have to operate with a much smaller image size, thus reducing the computation time. Figure 4.2 shows the process followed in the segmentation. First, the 3D ROI is extracted, and the limits of that region are transformed into pixel coordinates. Afterwards, the ROI segmenter 2D node crops the 2D region of interest.

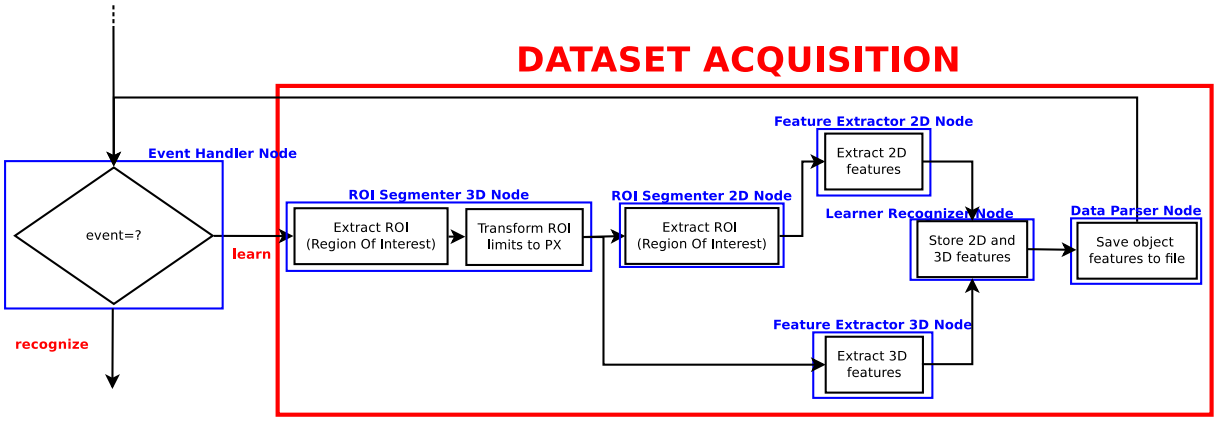


Figure 4.2: Flowchart detailing the dataset acquisition or learning mode of the system.

After the extraction, the feature extractor nodes obtain the 2D and 3D features out of the segmented data. As it was explained in section 2.4, the features or descriptors are characteristics that define and represent the data from where they were created. The 2D and 3D features that are being used in this system are the ORB and PFH descriptors, respectively. This cycle is repeated as many times as number of views has been defined in the system. As an example, if the software is configured to accept five views per object, the process is repeated five times. Between repetitions the program stops for a second to allow the user to rotate the object and hence obtain as much information as possible.

The number of views influence in the performance of the system. The larger the number of views is, the better the performance. Also, showing very different views of the same objects affects in the robustness of the system. The more different the views are, the more robust to rotation and pose the system is. That is the end of the data cycle of the learning process.

4.4.3 System exploitation or recognition mode

The recognition mode is triggered when the hand is located near the user's body. Figure 4.3 shows the detailed flowchart of the system exploitation. The steps that compose this part of the software are the following: first, the input information is converted to the custom message used within the code. Afterwards, as in the previous mode, the Region Of Interest is segmented from both 2D and 3D original information. Then, the descriptors are extracted exactly the same way as in the previous mode.

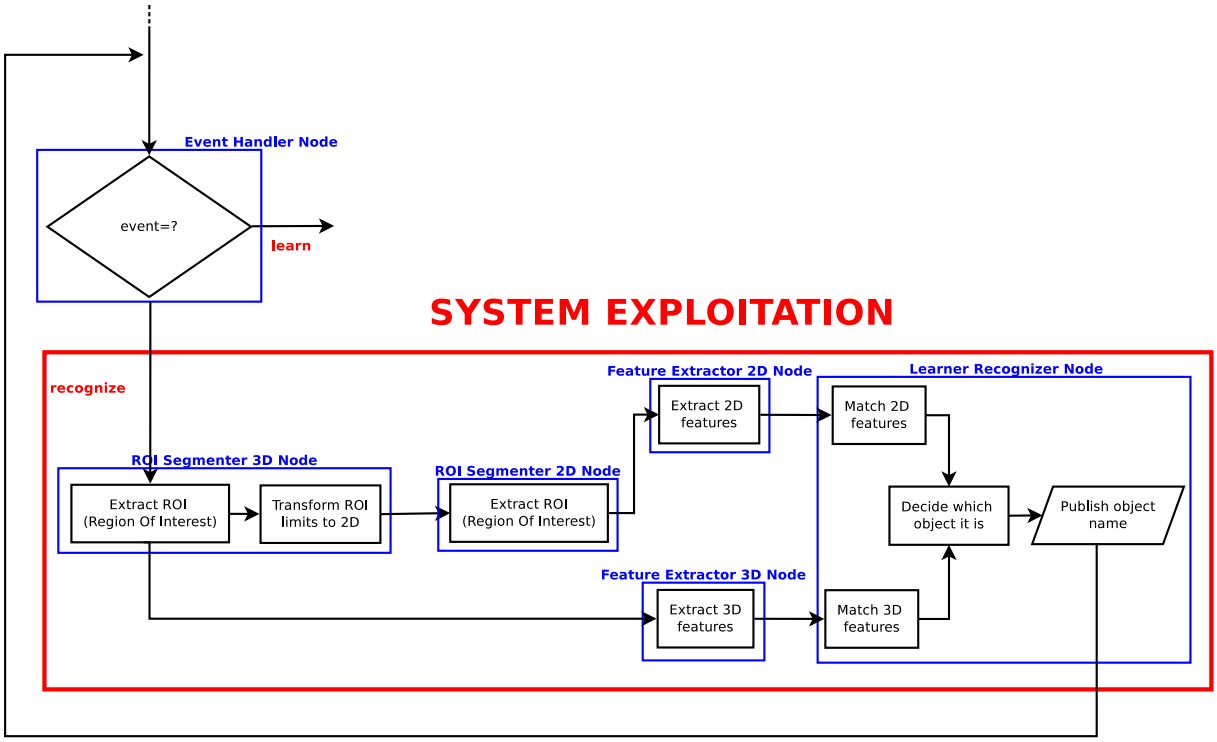


Figure 4.3: Flowchart detailing the system exploitation or recognition phase.

Finally, the learner recognizer node outputs the object identification number. This number is obtained comparing the previously learned descriptors with the ones extracted from the current frame. This data is the output of the system.

4.4.4 Third party libraries that process the input data to the system

In this section the ROS packages that are used in our system are presented. All of them are used for the initial processing of the data coming from the RGB-D sensor. The connection between them and the developed nodes are explained in this section and in section 4.4.5.

4.4.4.1 ROS package: openni_camera

This package implements the RGB-D sensors drivers. The package transforms the input raw data coming from the kinect into structured one. This information is prepared hence for further processing. Figure 4.4 shows the Connectivity graph of this package and its position in the RGB-D sensor data processing chain.

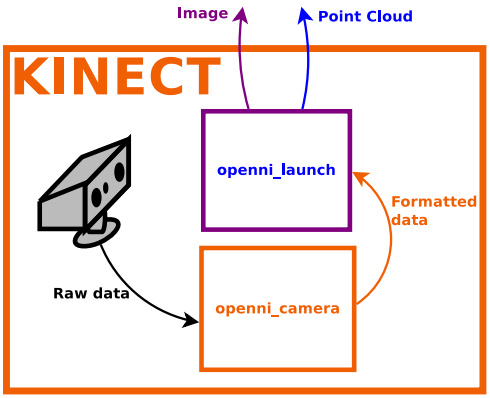


Figure 4.4: Kinect data processing using the openni ROS packages (openni_camera and openni_launch).

4.4.4.2 ROS package: openni_launch

This package provides useful transformations taking as input the openni_camera topics. It is composed of various nodes that can be executed using a launch file. Each node processes the raw input information from the driver into more useful data. This data may be a point cloud with color information or a disparity image for example. The output of the nodes is published into different topics, which are the input for the nodes that have been developed for this thesis.

4.4.4.3 ROS package: pi_tracker

This ROS package implements a joint tracker. It is used within the system to determine the position and orientation of the user's joints. This task is performed by the skeleton tracker node. Figure 4.5 shows the Connectivity graph of this node. The node takes as input the output data of the openni_launch ROS package. The node outputs the Skeleton message in which the positions and orientations of the joints are represented.

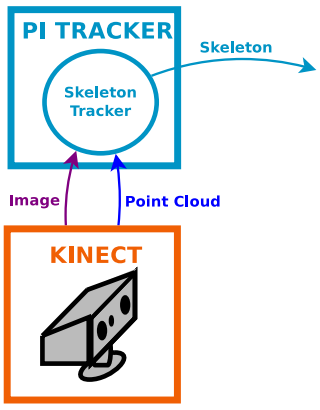


Figure 4.5: Connectivity graph of the Skeleton Tracker node.

4.4.5 Description of the developed nodes

The processing of the system is divided in nodes. Figure 4.6 shows the graph of the nodes that have been developed for the project. The circles represent the nodes. The arrows show the communication between nodes. The names next to the nodes' interconnections are the messages interchanged.

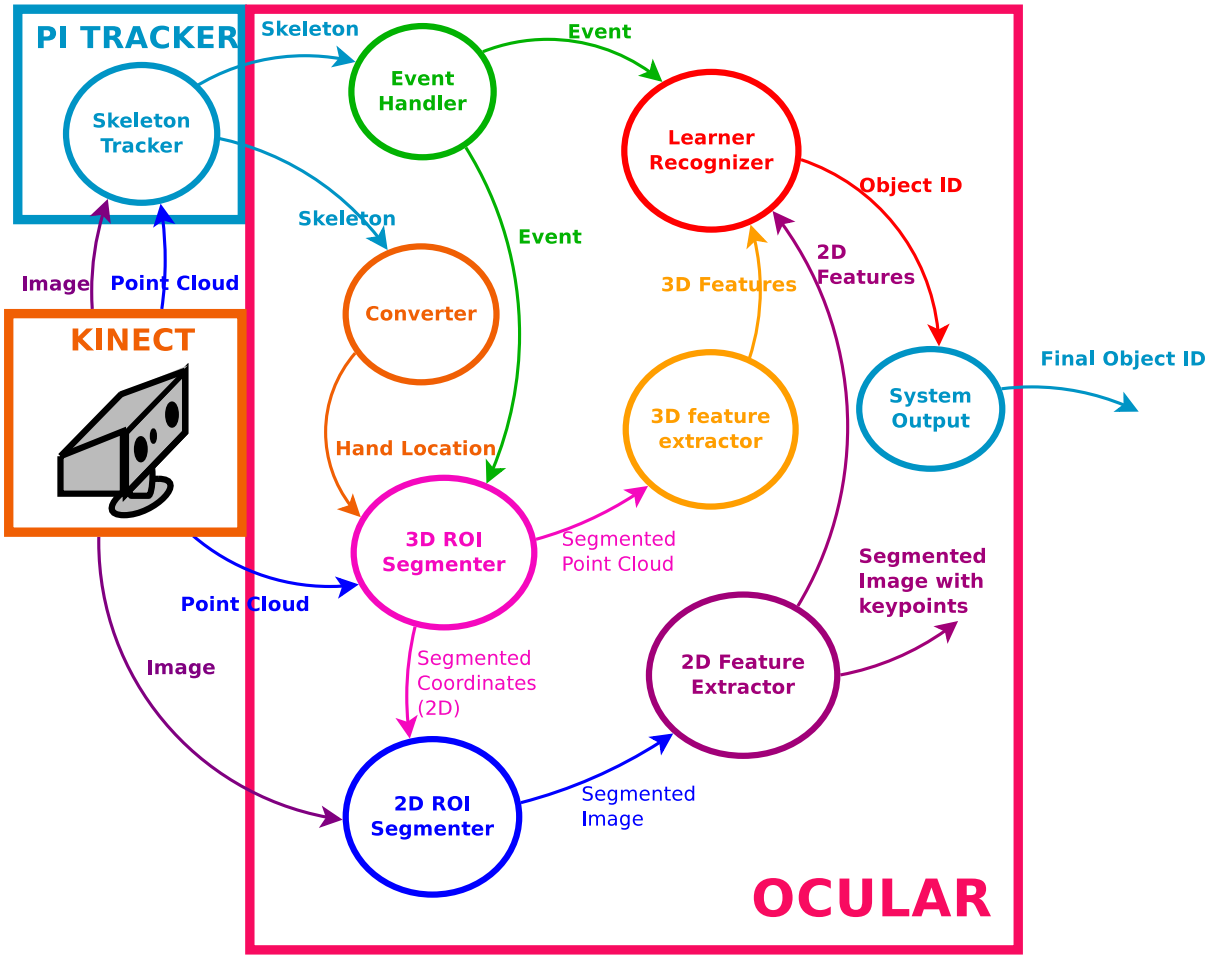


Figure 4.6: System nodes and their interaction.

The square areas define the different ROS packages that have been used. The pink square with the label "OCULAR" separates the nodes developed in this thesis from third-party nodes. The nodes are represented in the order that follows the data-flow. Sections 4.4.5.1 to 4.4.5.8 present the processing performed by each node.

4.4.5.1 Converter node

This node is the first step of the developed software, a translator to isolate the OCULAR package from third party packages. It converts the input data containing the skeleton position to a custom message used through the rest of the code. It allows to easily change the package from which the skeleton is obtained without affecting the whole system.

The converter node transforms the input data from the pi_tracker package into the custom message used within the software. It was only implemented a converter for the pi_tracker package, but the software is designed to easily develop and add more converters in case it is necessary. s Figure 4.7 represents the Connectivity graph of the node. The skeleton message enters the node and the custom message containing the hands location is the output.

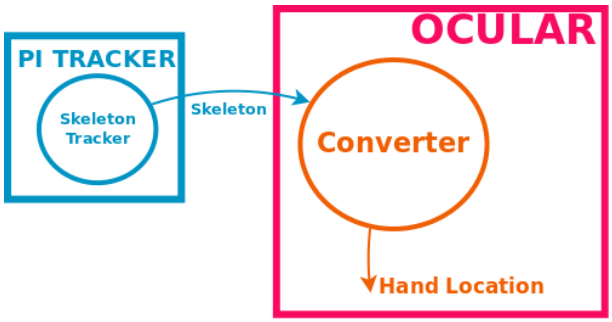


Figure 4.7: Connectivity graph of the Converter node.

The use case diagram of the node can be seen in figure 4.8. Since it is a simple translator node, it has only three cases. Receive the skeleton message is the first one. The skeleton message is obtained from the third party package that tracks the user's skeleton. In this system, the package that tracks the skeleton is pi_tracker (see section 4.4.4.3 for further information). The second case is to select the hand's location from the input skeleton message. This is the conversion step of the node, in which the information from the third-party package is parsed and the data relevant to the system's nodes is extracted. Finally, the last case is to publish the location of the user's hands. This is the output of the system, the custom message used within the system in which the hands location is specified.

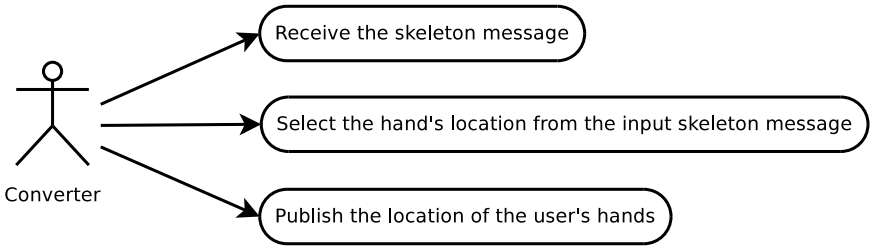


Figure 4.8: Use Case diagram of the converter node

4.4.5.2 3D ROI Segmenter node

This node extracts the 3D Region Of Interest from the input point cloud. This process is convenient to eliminate useless information and hence reduce the size of the 3D data used by the system. Having a smaller 3D data size also lowers the computing time of the following nodes that use this information. Figure 4.9 presents the connectivity graph of the node. The input of this node is the raw 3D information from the sensor and the hand's locations from the third-party package pi_tracker, as well as the hand in which the user is holding the object.

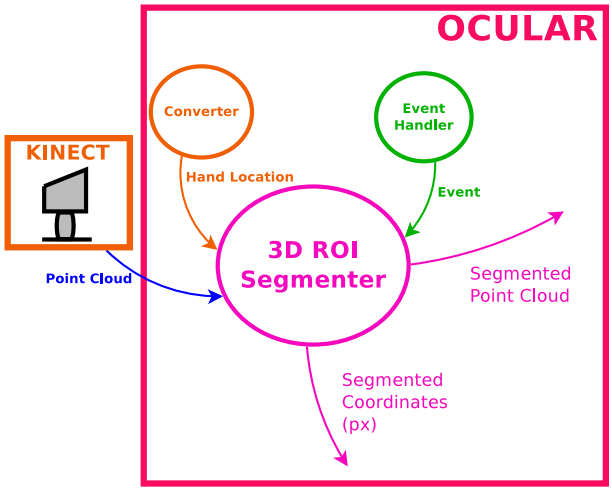


Figure 4.9: Connectivity graph of the ROI segmenter 3D node.

The system is used to detect in-hand objects. Hence, the hand and the area around it conform the Region Of Interest of the software. This is the reason why this node segments a prism around the selected hand's center from the input point cloud.

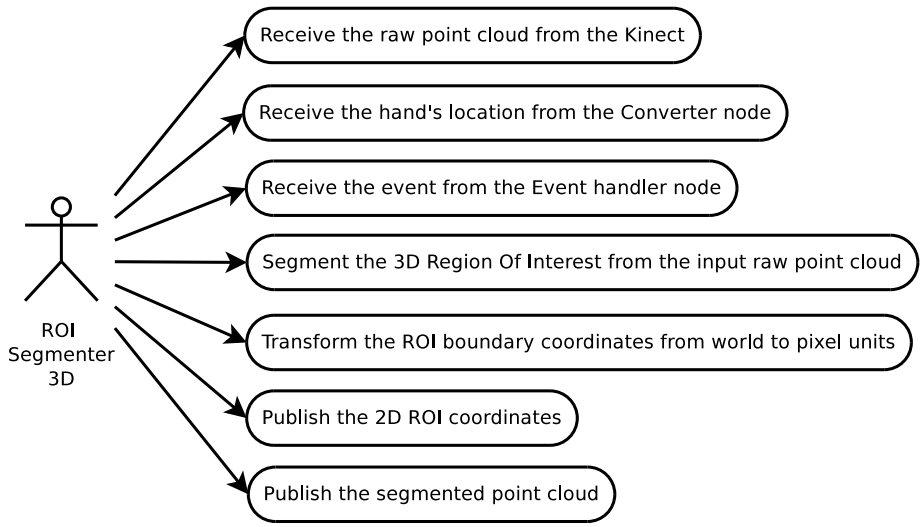


Figure 4.10: Use Case diagram of the ROI segmenter 3D node

The 3D ROI Segmenter node provides information to two different nodes: the 3D Feature Extractor (see section 4.4.5.5) and the 2D ROI Segmenter node (see section 4.4.5.3). The segmented point cloud is passed to the 3D Feature Extractor and the segmented coordinates are shared with the 2D ROI Segmenter node. This latter needs the values of the prism vertex coordinates in order to crop the input image. Nevertheless, the coordinates in the 3D ROI Segmenter node are in world units and hence they need to be converted into pixels. This step can be observed in Figure 4.10, which shows the use case diagram of the node.

In Figure 4.10 the different cases of this node are shown. Since this node is more complex than the previous Converter node (section 4.4.5.1), there are more cases in which the program may be. The first three present the reception of information: the raw point cloud from the kinect, the hand's location in the custom message used within the system from the converter node and the event from the event handler node. This last piece of information is used to determine which hand is being used in the software. Afterwards, the processing cases is presented: the segmentation of the 3D Region Of Interest from the input point cloud and the transformation of the boundary coordinates from world to pixel units. Finally, the node publishes the segmented point cloud and the 2D ROI coordinates.

4.4.5.3 2D ROI Segmenter node

Similarly to the previous node described in section 4.4.5.2, this executable is used to extract the Region Of Interest with the difference that this is segmented from the input 2D data, an image. The reasons behind this operation are the same: eliminating the non-important part of the information allows to obtain a smaller data that reduces the computation times and also lowers the noise introduced into the system. Figure 4.11 depicts the connectivity graph of this node. The raw 2D information and the hand location in pixels are inputs to the ROI Segmenter 2D node.

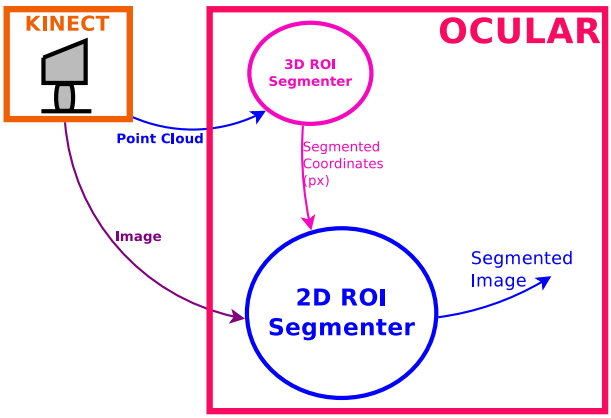


Figure 4.11: Connectivity graph of the ROI segmenter 2D node.

This node processes the information as follows: first, the ROI (Region Of Interest) is cropped taking a square section around the center of the hand. The size of the square is fixed because the difference in the scale in negligible in the operating range of the system (1.5m to 2.5m). This range is determined by the skeleton tracker node and also by the low resolution of the RGB-D sensor. The skeleton tracker node operates correctly in that range. If the person is located closer to the kinect, the node does not recognize the skeleton. If the user is outside the range, it is considered a background object and hence the node does not track his skeleton.

In figure 4.12 the use case diagram of the node can be observed. The first two cases show the reception of information from other nodes such as the 2D ROI location from the 3D ROI Segmenter node, the raw image from the Kinect. Then, the processing case is presented which is the segmentation or extraction of the 2D Region Of Interest. Finally, the last case involves the publishing of that segmented information.

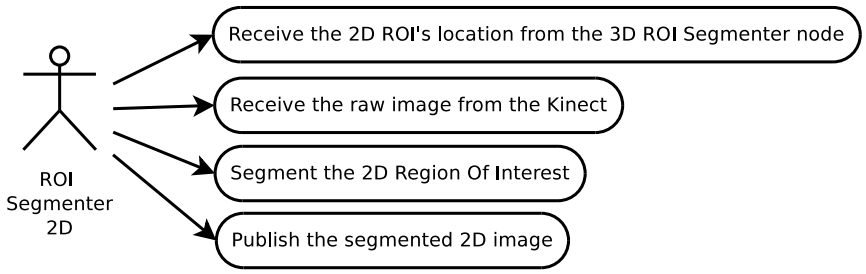


Figure 4.12: Use Case diagram of the ROI segmenter 2D node

4.4.5.4 2D Feature Extractor node

This node is used to obtain the descriptors or characteristics from the segmented 2D information. These features are used in later nodes to match two objects and determine whether they are the same one or not. The 2D Feature Extractor node is hence the last part of the 2D data processing in the system. Figure 4.13 shows the connectivity graph of the node. There are two output messages of this node, the segmented images with keypoints and the 2D ORB descriptors matrix. This matrix is used in the rest of the system. The segmented image with the keypoints drawn on it is outputted for debugging and development reasons.

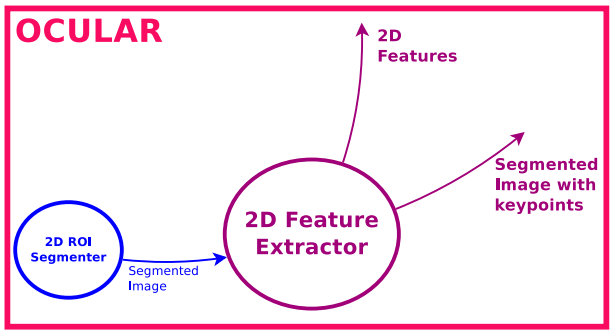


Figure 4.13: Connectivity graph of the Feature Extractor 2D node.

Figure 4.14 represents the use case diagram of this node. The first case shows the reception of the segmented image. The two cases in the middle reflect the extraction of 2D features and the saving of those features to a file. This last operation is only performed when the system is shut down to avoid a lag in the software in runtime. Finally, the last case contains the publishing of the 2D features, which is used by the learner-recognizer node (see section 4.4.5.7) and the segmented image with keypoints, which is only used for debugging and developing.

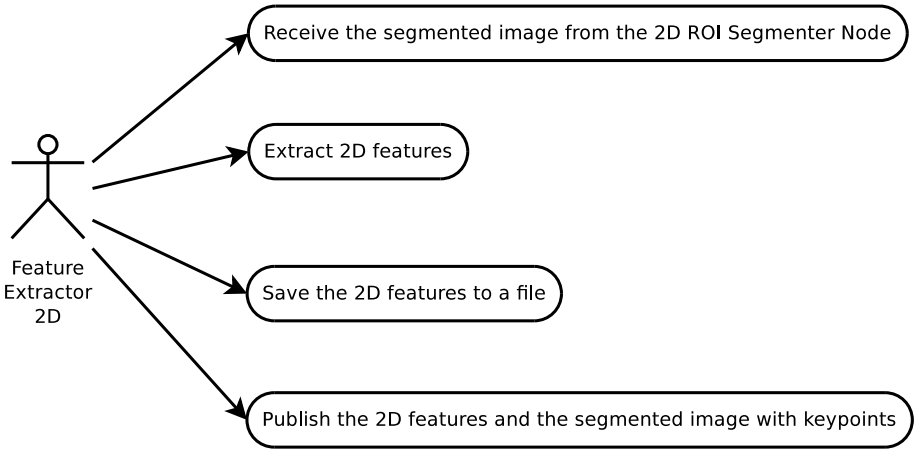


Figure 4.14: Use Case diagram of the Feature Extractor 2D node

4.4.5.5 3D Feature Extractor node

This node is used to extract the 3D features or descriptors from the segmented point cloud. Those features are stored in a matrix, which is used to compare the data from different objects. This comparison is performed in the Learner-Recognizer node (see section 4.4.5.7). Figure 4.15 shows that the input of this node is the segmented point cloud from the ROI Segmenter 3D node (see section 4.4.5.2). The descriptors are extracted from this information and are published in the output topic.

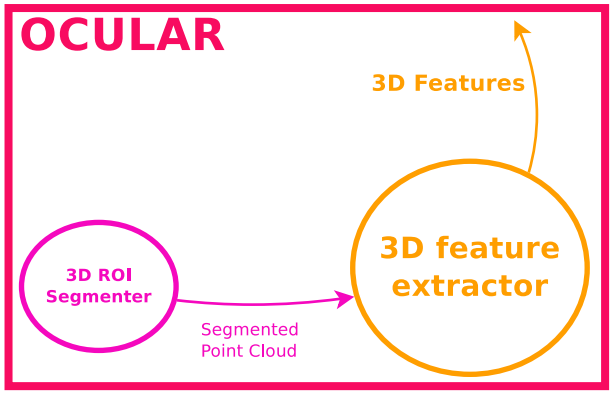


Figure 4.15: Connectivity graph of the Feature Extractor 3D node.

Figure 4.16 shows the use case diagram of the node, which is very similar to figure 4.14 from the 2D Feature Extractor node (section 4.4.5.4). The first case presents the acquisition of the segmented point cloud. The two following cases are the processing of the node: the extraction of 3D features and the saving of this features to files in the system's shut down. The final case involves the publishing of these 3D features.

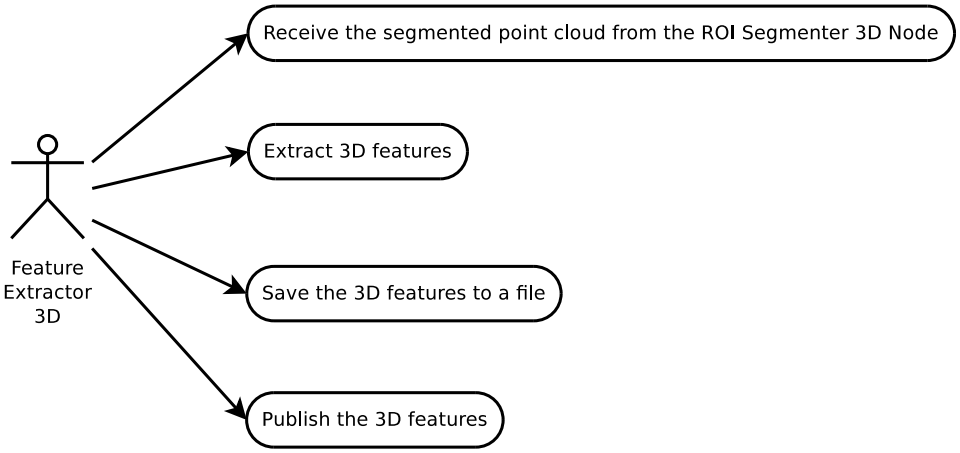


Figure 4.16: Use Case diagram of the Feature Extractor 3D node

4.4.5.6 Event Handler node

The system is intended to be running in a robot. This implies that there are no devices such as a mouse or a keyboard to interact with it. Hence, an Event Handler node is needed to allow the human-machine interaction using gestures. There are two gestures or poses that the system detects. If the user has the arm stretched towards the RGB-D (Red, Green, Blue and Depth) sensor the event recognized is *"Learn"*. If the arm is located near the user's body, the event is *"Recognize"*. The first event triggers the learning cycle described in section 4.4.2. The second event is the default of the system and enables the recognition of the in-hand objects.

This node also selects the hand that is being used in the software. The system is able to work with one hand at a time. The user's pose to use the software is having the hand that has no object near the body and the one holding the object shigher. Figure 4.17 shows the connectivity graph of the node. Event Handler receives its input from the pi_tracker package and outputs the event message to the rest of the system.

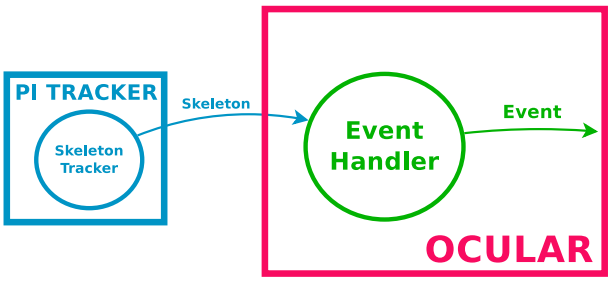


Figure 4.17: Connectivity graph of the Event Handler node.

The input of the system is the skeleton message that is obtained from the third-party package pi_tracker. This message contains the information of all the joints of the user. The information is screened to detect the height at which each hand is located. The one that is the highest is the one being used in the software. Afterwards, the distance between the body and the chosen hand is computed. When that distance is similar to the distance of the user's arm, the event triggered is "learn". If, otherwise, the hand is located close to the body, the event that is published to the output topic is "recognize". The distance that triggers the modes is proportional to the distance between the user and the RGB-D sensor in order to obtain a range of use of the software higher.

Figure 4.18 is a diagram with the use case of the node. The first case shows the input of the information about the relative hand's position from the pi_tracker node. Afterwards, the event being triggered is decided and finally published. The information of the events is used in the Learner-Recognizer node (section 4.4.5.7).

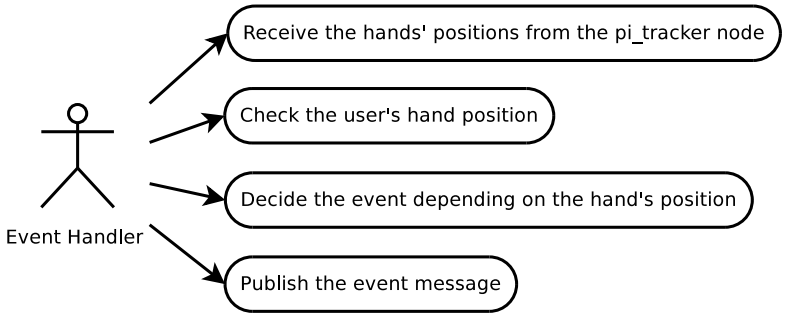


Figure 4.18: Use Case diagram of the Event Handler node

4.4.5.7 Learner-Recognizer node

This node implements the state machine of the system. It is needed to switch the mode of the software accordingly with the event being triggered by the user. If the event received is "Learn", the learning sequence starts. If the event is "Recognize", the recognize sequence is triggered. These events are detected by the Event Handler node, described in section 4.4.5.6

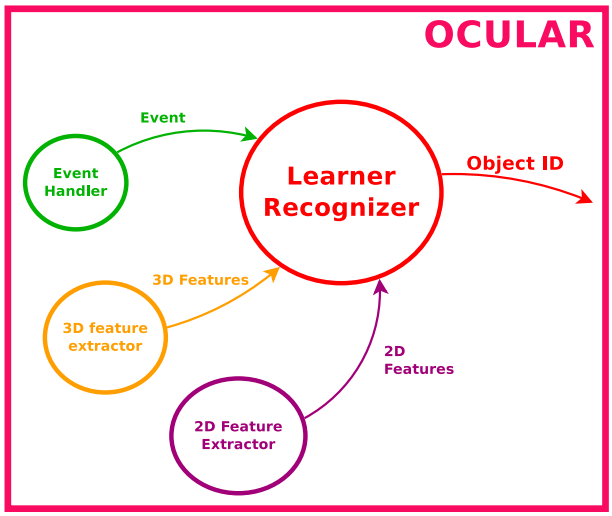


Figure 4.19: Connectivity graph of the Learner-Recognizer node.

The learn sequence consists in obtaining and storing the features both 2D and 3D and waiting a second, allowing the user to move the object to capture a new view of it. The dataset extracted is saved to a folder when the software is closed, to prevent possible lags in the runtime of the program. Each view is saved separately. This node loads the files that are still in the saving folder when the program is restarted. The recognition sequence compares the newly obtained features both 2D and 3D with the ones that are stored in the dataset. Afterwards, the result of the recognition for both types of descriptors are published in the output topic. Figure 4.20 presents the use case diagram of the node.

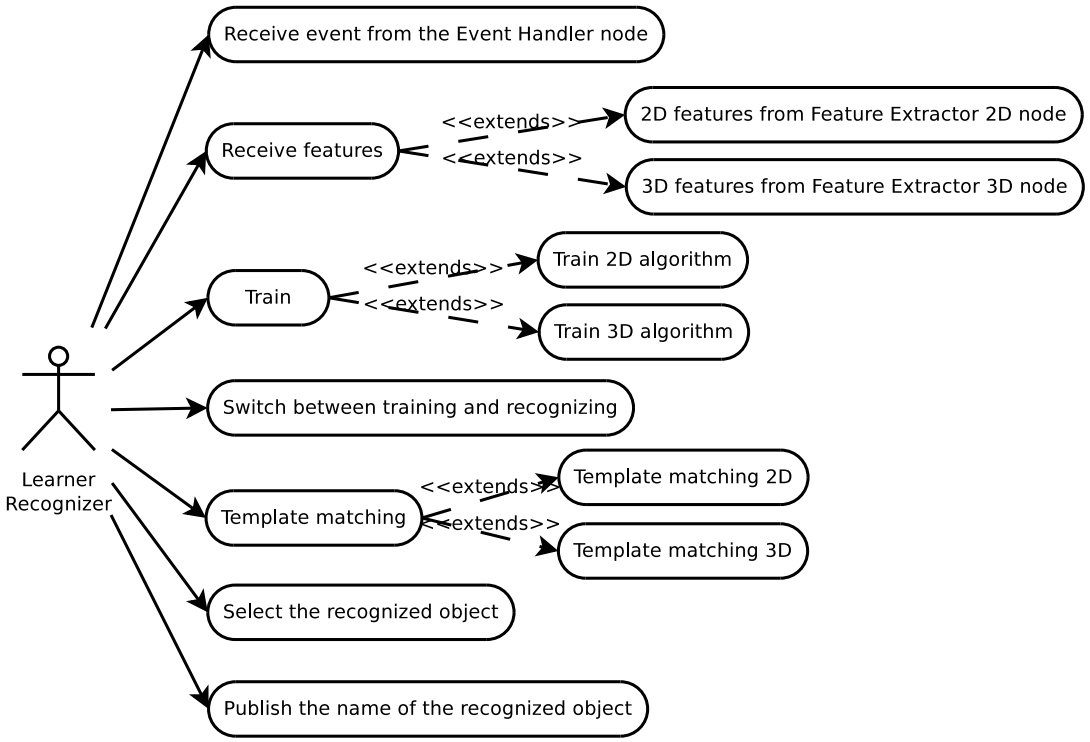


Figure 4.20: Use Case diagram of the Learner-Recognizer node

The first two cases are the data acquisition. That data is the event that the user has triggered and the 2D and 3D features from both Feature Extractor nodes (sections 4.4.5.4 and 4.4.5.5). Afterwards, the train case is shown which is performed for both 2D and 3D descriptors. In the middle, the case that implements the state machine is presented: the switching between the training and the recognizing modes. Then, the template matching case appears that is also performed on 2D and 3D. In it, the descriptors extracted from the current frame are compared to those previously obtained. The next case is the selection of the recognized object. The information is obtained for the 2D and 3D information separately. This information is then published at a rate of approximately thirty messages per second.

4.4.5.8 System Output node

The output of the Learner-Recognizer node described in section 4.4.5.7 is the predicted object. This prediction is very noisy due to different factors such as sudden changes in the illumination of the room. In order to reduce the noise effect in the system, it was decided to introduce the System Output node. This nodes is a low-pass filter, which means that it eliminates those results that has a high frequency. This task is performed using a buffer to store a certain amount of predictions and then managing the stored data.

The input of the node is the object ID message from the Learner-Recognizer process as can be seen in figure 4.21. The node stores thirty values of instantaneous object estimations. Since the Kinect runs at 30 frames per second, approximately each second a new final object estimation is outputted.

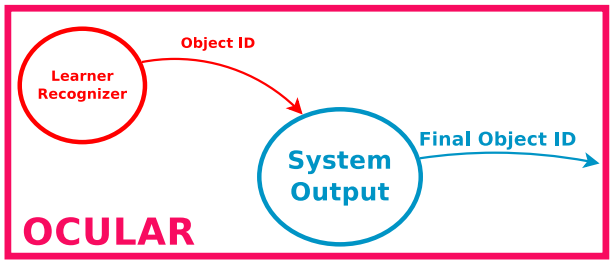


Figure 4.21: Connectivity graph of the System Output node.

The decision is performed as follows. The thirty 2D and 3D instantaneous object estimations are stored in two vectors. The frequency is the amount of times each object appears in those vectors. This frequency is obtained for 2D and 3D separately. Let us represent as y'_{2D} and y'_{3D} the vectors containing in each element the frequency of the object with $object\_id = element$. Both vectors are combined in one, which is called y' using formula 4.1.

$$y' = 0.6 * y'_{2D} + 0.4 * y'_{3D} \tag{4.1}$$

The reason of applying the 0.6 and 0.4 constants for 2D and 3D respectively, instead of calculating a simple mean is because we found empirically that the 2D descriptors produced better predictions than the 3D ones. The estimated final object id (Y) is obtained as the vector element that has the highest frequency. That is, the final prediction is the object_id whose frequency is higher among the latest 30 predictions. This number is Y, which is the output of the whole system.

$$Y' = argmax(y') \tag{4.2}$$

Figure 4.22 shows the use case diagram of this node. Since this node is very simple, there are only three cases. The first one involves the reception of the object ID message. The case in the middle represents the buffering and later decision processing of the node. The last one is the publishing of the resulting object ID, which is the output of the whole OCULAR system.

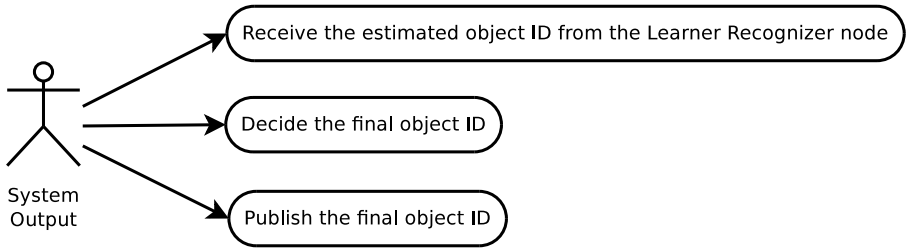


Figure 4.22: Use Case diagram of the System Output node

Chapter 5

Evaluation of the system: Methods

In this chapter the experiments that are performed in the system are detailed in order to allow others to replicate them. The software has been built in a modular way that allows an easy location of the problems and bottlenecks. This modularity also permits the change of the different processing steps without affecting the whole system. Different aspects of the code are evaluated such as the CPU and RAM consumption. Also, since the computing has been distributed over the different nodes, the test describes the response of each one separately. The topics that communicate the nodes of the system are also studied. The characteristics being measured in the topics are the publishing rate and the bandwidth. All these are being measured in order to ensure that the system is able to work on real time.

Apart from the system's performance its accuracy is also being tested. In order to do so, three experiments have been performed using a different number of views per object. In each of them, the confusion matrix has been constructed. From the confusion matrix, the success ratio and the precision and recall of the system is obtained. The success ratio is the ratio of true positives returned by the system, that is, the percentage of times the system estimates the object being shown to it correctly. The precision and recall allow to calculate the F1-score of the software for each object. More information about this calculations and the definitions of precision, recall and F1-score may be found in section 5.3. The results obtained in the experiments are presented in chapter 6 and later discussed in chapter 7.

5.1 Experimental setup: hardware and software components

There are only two hardware devices being used in the system: a computing unit and a sensor. It was previously explained in section 4 that the sensor being used as input of the software is an RGB-D (Red, Green, Blue and Depth) sensor. The computing unit used in the experiments is a laptop, whose technical details are presented in section 5.1.1. Those details such as the processor speed and the RAM memory determine the results of the computing performance evaluation experiments. In this section the hardware specifications of the hardware used in the experiments is presented. It is shown as well the software that is needed to permit the correct functioning of the RGB-D sensor.

5.1.1 Computer hardware specifications

The computer used for testing the software is a Mountain f-11 Ivy. This laptop has a Intel Core i7-3630QM CPU that operates at a speed of 2.4 GHz and has the capability of enhancing this speed up to 2.4 GHz. It features a 8 GB Kingston HyperX at 1.600 MHz RAM memory and a Kingston HyperX 3K of 120 GB Solid State Drive (SSD).

5.1.2 RGB-D sensor

The RGB-D sensor used in the testing process is a Kinect 360. This sensor is able to retrieve color and depth information using its color and depth-sensing lenses. These lenses determine the field of view of the sensor as well as its working depth range. The horizontal field of view is 57 degrees and the vertical field of view, 43 degrees. The kinect is able to work correctly between 1.2 m and 3.5 m, which is the maximum depth sensor range.

The sensor outputs two data streams, one containing the depth information and the other containing the color data. The first one is a matrix of 320x240 of 16-bit information that is outputted at a rate of 30 frames per second. The color information is a matrix of 640x480 composed of 32-bit data that is published at a rate of 30 frames per second as well.

5.1.3 Software

In this section the software that was installed on the computer used in the experiments is detailed. The laptop had the Ubuntu 13.04 (64 bits) Operating System installed. The Robotic Operating System (ROS9 distribution being used is the Groovy (v 1.9.50), and the ROS packages versions are the following: openni_camera (v1.8.9), openni_launch (v1.8.3) and pi_tracker, which is unversioned. This last pi_tracker package was downloaded from their github repository[1] (branch groovy-devel). The Open Source Computer Vision Library (OpenCV) version installed on the computer is v2.3.1, and the Point Cloud Library's (PCL) is v.1.7.1.

Apart from these software, three more complementary libraries are needed: the OpenNI SDK, the Avin2 driver and the NITE library. The Open Natural Interface (OpenNI) is a set of libraries needed by the openni_camera and openni_launch packages. The version being used is the OpenNI SDK v1.5.4.0, 64bits. The Avin2 drivers are a modified Kinect drivers released because the official ones did not work with the NITE skeleton tracking library. The version installed on the computer is the Avin2-v0.93-5.1.2.1. NITE is a set of libraries that allows to track the skeleton, and the version that was used for the experiments is the NITE v1.5.2.21, 64 bits.

5.2 Computing performance evaluation

This section presents the tests that I performed to evaluate the computing performance of the system. The main objective of these tests is to demonstrate that the system is able to operate in real-time, so it can be used while interacting with a human. The topics and the nodes of the system are being evaluated measuring their main characteristics. The nodes computing performance is obtained from their CPU and RAM consumption. The topics are the means of communication of the nodes. In the topics the messages are published and more than one node can subscribe to each topic and hence access to those messages. In this evaluation, the total bandwidth and publishing rate of the topics are measured.

5.2.1 Nodes' CPU and RAM usage

The CPU and RAM consumption of each node was measured. The total usage of the whole system can be calculated summing the individual consumptions. In section 6, the results of this evaluation are presented. The CPU and RAM usage are presented in percentage.

[1] https://github.com/pirobot/pi$_t$*racker*/*tree*/*groovy* − *devel*

5.2.2 Topic network usage

The computing performance tests also evaluate the topic bandwidth (BW) and frequency requirements. The motivation of this evaluation is the following. ROS is a distributed system that allows spreading the computation across multiple machines. This allows to create distributed computing systems where the robots carry low-power consuming computers and delegate the CPU intensive tasks to remote servers. The drawback of this approach is that communicating machines consume network resources. This is specially relevant for image or video tasks, which do consume great amounts of BW. The need to evaluate the BW and frequency requirements of the system comes from this issue. Therefore, the purpose of the topic network usage measurement is to evaluate if the system could be distributed across several machines.

The publishing frequency of a ROS topic is the number of messages published on it over time. It is measured in Hz. ROS provides a built-in command that is similar to the one above that returns the maximum, minimum and the standard deviation of the time between messages in seconds. Also, it returns the average publishing frequency.

5.3 Evaluation of the object recognition accuracy

This experiment is designed to obtain data related to the accuracy of the system. Sections 5.3.1, 5.3.2, 5.3.3 and 5.3.4 explain the conditions and parameters that affect the experiment and also the testing procedure that has been followed.

5.3.1 Experimental setup

The experiment was performed in a highly illuminated room with white walls. The light was a mixture of natural light and artificial light coming from halogens. The experiment was performed at a distance of approximately 1.7 m from the RGB-D (Red, Green, Blue and Depth) sensor. The background was composed of a white wall with a painting in the right side, a window in the left side and shelves in the middle-left part. Figure 5.1 presents an sketch of the experimental setup.

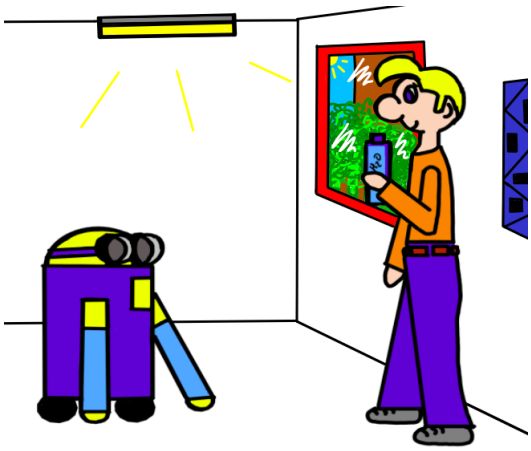


Figure 5.1: Experimental setup sketch.

5.3.2 Experimental set of objects

The objects being used in the experiment are common objects. They can be seen in Figure 5.2. Using common objects allow to obtain more reliable experiment results, since they were obtained under working conditions. Common objects are usually featureless. That characteristic complicates the object recognition, since objects with rich textures usually generate more robust descriptors. In order to overcome this difficulty, different views of each object are obtained and compared when doing the matching. Also, the introduction of both 2D and 3D features diminish the possibilities of false positives and false negatives.

Figure 5.2: Experimental dataset. From left to right: ball, skull, cup, mobile and calculator.

This particular dataset is conformed of objects that could be easily confused by a computer vision system using descriptors. As an example, the 3D shape of the ball and the skull is very similar, a fact that would lead to akin 3D descriptors. Also, the cup, the bottle and the mobile have comparable 2D texture.

5.3.3 Experimental procedure

The testing is performed following this sequence. First, the objects are stored in the program's dataset. In order to do so, each object is recorded using the data acquisition mode of the software. It is done turning the the object in the process in order to obtain as many different views as possible. This improves the performance of the recognition phase when the object is presented in different positions. Afterwards, the recognition mode is used. Each of the objects are presented to the system the same amount of time, 30 seconds. The objects are positioned still during 10 seconds in the most natural grasping position. Then, they are rotated during the remaining 20 seconds. This is done to measure the robustness of the system to rotation changes.

The information about the 2D and 3D separate recognitions and the final software's output is recorded to a file for further processing. That data is presented in chapter 6. The system allows to specify the number of views being taken per object in the data acquisition phase. Taking advantage of this feature, the whole procedure described above is repeated for one, five and ten object views. Theoretically, decreasing the number of views should worse the effectiveness of the recognition. This experiment is designed to confirm this intuition.

5.3.4 Accuracy measurement

The accuracy of the system is measured using the F-score or F-measure. The general formula for a positive real β is described in formula 5.1.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{5.1}$$

The β is a weight whose value depends on the applications of the system being tested. For example, in critical systems such as a police face recognition software it is preferable to have false positives than false negatives. Since in this system the weight being given to the precision is the same that the one being given to the recall, $\beta = 1$. This special case is called the $F_1 score$, and its formula may be found in equation 5.2. The precision and recall are parameters that are described in formulas 5.3 and 5.4, respectively.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.2}$$

Apart from the $F_1 score$, a confusion matrix is constructed for each experiment. This matrix shows the percentage each different object was predicted when each of the real objects were presented to the system. This percentage is shown transformed to a zero to one scale. The precision is the fraction of retrieved results that are relevant. Using a confusion matrix as the one above, the precision is computed from it using formula 5.3.

$$precision_{ij} = \frac{M_{ij}}{\sum M_j} \tag{5.3}$$

M_{ij} are the elements of the confusion matrix. The i represent the rows and the j the columns of the matrix. The recall is the fraction of the results that are relevant and that have succeeded. It may be obtained using the confusion matrix with formula 5.4. The precision and recall are computed for each of the six objects conforming the dataset.

$$recall_{ij} = \frac{M_{ij}}{\sum M_i} \tag{5.4}$$

Chapter 6

Evaluation of the system: Results

6.1 Introduction

This chapter presents the results obtained in the tests and experiments performed on the software. First, the nodes' CPU and RAM usage and topics' network usage are presented. Afterwards the information obtained with the accuracy testing is shown. The results of the experiments are merely reported in this chapter, the discussion of these results may be seen in chapter 7.

6.2 Computing performance evaluation

This section presents the results of the computing performance experiment. First, in section 6.2.1, the CPU and RAM usage of each of the nodes of the system is presented. Afterwards, in section 6.2.2, the bandwidth and the publishing rate of each of the nodes is shown.

6.2.1 Nodes' CPU and RAM usage

As it was explained in chapter 5, the CPU and RAM usage is measured in the different nodes that compose the software. There is a high difference in the CPU and RAM usage between nodes. The nodes that only perform a transformation of the data such as the converter node have a lower CPU and RAM consumption. The nodes that process the input images and point clouds have much higher values. Figure 6.1 shows the results of the experiment.

The nodes with a higher computing consumption are the ROI segmenters and the feature extractors both 2D and 3D. Since the 3D data has a higher size, the both the CPU and RAM usage of the nodes using 3D information is much higher than those processing 2D data. The learner recognizer node also has a higher consumption than the converter, event handler or system output nodes. This is because these perform simple conversions and computations of integers and floating data. On the other hand, the learner recognizer node implements the state machine of the software. This means it has to deal with a higher amount of data than the previous nodes.

The total CPU usage is lower than the 23%, and the RAM usage of the whole software is of less than the 5%. Since the processing unit has 4 physical cores with two threads per core, the system needs less than one core to operate in real time. The computer used has 8 GB of RAM. Using that number, the total RAM usage of the system is of 0.4 GB. Figure 6.1 shows that the CPU and RAM consumption of the third-party packages is very high. The total CPU and RAM usage including the third-party packages is almost the double of the total of the OCULAR nodes. This means that it is needed a computer featuring less than two cores (at 2.4 GHz) and less than 1 GB of RAM to be able to run the whole system on real time.

NODES	CPU USAGE(%)	RAM USAGE(%)
Converter	0.20	0.50
ROI segmenter 2D	0.84	0.50
ROI segmenter 3D	13.34	1.50
Feature extractor 2D	2.44	0.10
Feature extractor 3D	4.74	0.50
Event handler	0.16	0.50
Learner recognizer	1.04	0.80
System output	0.13	0.30
TOTAL OCULAR NODES:	**22.88**	**4.70**
pi_tracker node: skeleton_tracker	4.86	3.2
openni_launch nodelet	16.19	1.00
TOTAL WITH THIRD-PARTY PACKAGES:	**43.84**	**8.90**

Table 6.1: Nodes' CPU and RAM usage

6.2.2 Topic network usage

The following figures 6.2 and 6.3 show the publishing rate and bandwidth of the different topics. In figure 6.2, the difference in the average rate between topics can be appreciated. The publishing rate varies significantly between topics. The first column shows the average number of messages published in each topic. This is due to the reduced processing time and hence delay between message publishes. All topics but one have a more or less similar average publishing rate. The bottle neck of the process that establishes the publishing rate is the kinect. Since the information provided by this sensor is the input to most of the nodes of the system, their publishing rate is similar to the one of the kinect, 30 frames per second.

The only topic that publishes at a lower rate than the others is the final object ID topic. This node transmits the output of the system, as was previously explained in section 4. The node that publishes it is buffering the object ID topic messages. This creates a delay and the node publishes approximately one message per second in the final object ID topic.

Also it is noticeable the minimum rate column. Most of the topics have a minimum publishing rate of 0 seconds. The only one that is different is the final object ID topic. Another singular fact can be found in the maximum column. All topics but two have less than 1 second as maximum publishing rate. Those two are again final object ID and the object ID topic. The reasons behind this particular behaviour are discussed in chapter 7.

The maximum column shows that the time between messages increases with the complexity of the processing performed by the node that publishes in that topic. As an example, the segmented and descriptors topics have similar time, around half a second. It is remarkable the value of the maximum time obtained by the object ID topic, which is the highest of the column. The node that publishes in the object ID topic is the Learner-Recognizer node described in section 4.4.5.7. This node only publishes a message when the system is recognizing the objects. If the system is learning new objects, there is no message published. This could be the cause of the high result in the maximum time between messages column.

TOPIC	AVERAGE [Hz]	MIN [s]	MAX [s]	STD DEV [s]
Hand location	27.56	0.00	0.04	0.01
Segmented image	26.70	0.00	0.50	0.04
Segmented image with keypoints	25.91	0.00	0.50	0.03
Segmented coordinates (px)	11.56	0.05	0.13	0.03
Segmented point cloud	18.18	0.05	0.26	0.01
Descriptors 2D	25.98	0.00	0.50	0.03
Descriptors 3D	15.29	0.00	0.42	0.05
Event	27.72	0.00	0.04	0.01
Object ID	26.40	0.00	4.23	0.21
Final object ID	0.75	1.02	1.63	0.19

Table 6.2: Topic network usage - Publishing rate

TOPIC	AVERAGE [kB/s]	MIN [kB]	MAX [kB]
Hand location	1.38	0.12	0.12
Segmented image	$1.79{\cdot}10^3$	60	60
Segmented image with keypoints	1.64	60	60
Segmented coordinates (px)	212	57	58
Segmented point cloud	$1.48 \cdot 10^3$	0	830
Descriptors 2D	29.48	0.81	1.26
Descriptors 3D	$2.60 \cdot 10^3$	140	170
Event	1.06	0.05	0.06
Object ID	1.39	0.03	0.03
Final object ID	$1.15{\cdot}10^3$	$2{\cdot}10^3$	$2{\cdot}10^3$
TOTAL:	$\mathbf{6.65{\cdot}10^3}$		

Table 6.3: Topic network usage - Bandwidth

Figure 6.3 show the bandwidth measured in each of the system's topics. There is a huge different in the results for the different topics, the numbers range from the bytes to the megabytes. Those topics using custom messages use a higher bandwidth than those using a standard message such as a number.

This fact can be illustrated with the final object ID topic, which publishes integer messages and uses an average bandwidth of approximately 1 B/s. The segmented coordinates in pixels topic is the one that has the next lowest bandwidth and its messages are vectors that contain two integers. Its average bandwidth is around 200B/s.

In the kilobytes range, the hand location, descriptors 2D, event and object ID are located. All but descriptors 2D are filled with custom made messages. The data consists in integers and floats mixed with strings. It might be noted that the descriptors 2D topic is an order of magnitude higher than the other ones. This is due to the fact that its messages are images and hence have a higher size than the other ones.

Finally, the megabytes range houses the segmented image topics as well as the segmented point cloud and the descriptors 3D. All are filled with heavy messages that store two-dimensional and three-dimensional data.

6.3 Evaluation of the object recognition accuracy

This sections presents the results obtained in the object recognition accuracy experiment. It was previously explained that the different objects may be learned using one or more views per template. The accuracy is analyzed depending on the number of views learned. The experiment was repeated using one, five and ten views in order to evaluate the influence of the number of views in the accuracy of the system. The data in the confusion matrices shown below is normalized to a zero to one range to make them comparable. The discussion of the results can be found in section 7.

6.3.1 Template using 1 view

In this experiment, the system was trained using one view per object. Figure 6.4 shows the confusion matrix obtained with the experiment. The diagonal of the matrix are the success rates obtained for the different objects. The success rate is defined as the number of true positives over the total number of estimations returned by the system. From table 6.4 it can be seen that software has around a 50% of success rate in all objects. The F1-score obtained per object as well as their precision and recall values may be found in figure 6.5.

Real \Predicted	**ball**	**skull**	**cup**	**bottle**	**mobile**	**calculator**
ball	**0.55**	0.00	0.03	0.14	0.28	0.00
skull	0.03	**0.45**	0.28	0.24	0.00	0.00
cup	0.00	0.03	**0.52**	0.14	0.28	0.03
bottle	0.21	0.00	0.03	**0.59**	0.17	0.00
mobile	0.14	0.03	0.14	0.10	**0.55**	0.03
calculator	0.07	0.00	0.07	0.14	0.21	**0.52**

Table 6.4: Confusion matrix using a template that stores one view per object. The results are given in a 0 to 1 range.

The bottle is the object with a higher success rate. Nevertheless, the system recognized this object a high number of times when all the other objects of the dataset were presented to it. Hence, the precision in the bottle recognition is low, as can be seen in figure 6.5. Also, its recall is low because different objects such as the ball or the mobile were detected when the bottle was being presented to the software.

The ball has a precision and recall of exactly the same value. It is a low value because there were false positives when the ball was being used. The system recognized the bottle and the mobile a large number of times. Also, the output of the system showed a ball when the bottle and the mobile were shown with high rates. It also appeared when the calculator and the skull were presented but a lower and almost negligible amount of times.

The skull presents a very good precision. It is almost negligible the amount of times it was detected when other objects were being held in front of the system. Nevertheless, the recall is worse since the cup and the bottle were outputted a high amount of times when the skull was being used. It could be because all three objects have very similar 2D texture, leading to matchable 2D descriptors.

The cup has a success rate of 0.52. Its precision, recall and F1 score values are around that number as well. This is due to the fact that the system outputted the mobile's and the bottle's identification number a high number of times when the cup was being used. Also, the software detected the cup when all the other objects were being presented to it.

The mobile has even a lower precision than the cup and the results are comparable to the previous ones. The system detected the ball, the cup and the bottle when the mobile was being used a noticeable amount of times. Apart from that, the software recognized the mobile when all the other objects but the skull were presented a very high amount of times.

The calculator is the object with the best precision and F1 score of the dataset. It can be seen that it was detected when other objects were shown to the system a negligible amount of times. Nevertheless, when the calculator was being used the system outputted the mobile, the bottle with a high ratio and the cup and the ball with a lower one. Since the calculator is a heavily 2D textured object, is possible that a descriptor is very similar to other obtained in the mobile or the object. Also, the 3D shape of calculator, mobile and bottle is similar and could lead to 3D confusions.

Object	**Precision**	**Recall**	**F1 score**
ball	0.55	0.55	0.55
skull	0.87	0.45	0.59
cup	0.48	0.52	0.50
bottle	0.44	0.59	0.50
mobile	0.37	0.55	0.44
calculator	0.88	0.52	0.65

Table 6.5: F1-score calculation using the precision and recall parameters. Results for templates using one view per object.

6.3.2 Template using 5 views

In this experiment, five views were learned per object. The confusion matrix may be seen in figure 6.6. Figure 6.7 shows the F1-score obtained in this measurement, as well as the precision and recall per object.

Real \Predicted	**ball**	**skull**	**cup**	**bottle**	**mobile**	**calculator**
ball	**0.83**	0.14	0.00	0.00	0.03	0.00
skull	0.34	**0.62**	0.00	0.00	0.00	0.03
cup	0.03	0.21	**0.66**	0.07	0.00	0.03
bottle	0.14	0.14	0.00	**0.69**	0.00	0.03
mobile	0.10	0.13	0.00	0.07	**0.67**	0.03
calculator	0.10	0.21	0.00	0.07	0.00	**0.62**

Table 6.6: Confusion matrix using a template that stores five views per object. The results are given in a 0 to 1 range.

The success rate represented in the diagonal of figure 6.6 is higher than the ones obtained in the previous experiment. In this test, the object that obtained the highest success rate is the ball. It can be observed that the system only recognized two different objects when the ball was presented to it: the skull and the mobile. Nevertheless, the amount of times the software outputted the mobile can be neglected.

The cup has a higher rate than the skull, a 0.66. It is noticeable that the system outputted a cup only when it was shown to it. This leads to the maximum precision possible, 1. Nevertheless, the system recognized the skull, the bottle and the ball and calculator when the cup was shown to it. The two last objects were detected a negligible amount of times.

The bottle has a ratio of 0.69. Both the ball and the skull were outputted when the bottle was shown to the system. The most probable reason is that all three objects have curved sides that could be almost identical in terms of PFH 3D descriptors. Also, the calculator was recognized a negligible amount of times. The bottle was detected when the cup, the mobile and the calculator were presented to the system. Nevertheless, the ratio for all three of them is low, a 0.07.

The mobile has a success of 67%. It is remarkable that the precision obtained is very high, a 95 %. This is due to the fact that the mobile was only detected wrongly when the ball was being used in the experiment. The recall of the system with this object is much lower, around the 70%, due to the false positives returned by the software. The system outputted the ball, the skull, the bottle and the calculator when the mobile was being presented to it. .

Finally, the calculator has a score of 0.62. It was recognized in all previous measurements except when the ball was shown to the system. Nonetheless, the ratio of false positives is negligible (a 3%). The objects that were detected when the calculator was presented are the skull, the ball and the bottle. Since the calculator has a very cluttered 2D texture it is possible that some descriptors are similar to the ones extracted in other objects.

Figure 6.7 shows that the object with a higher F1 score is the cup, thanks mainly to its extremely high precision. All testing objects obtained similar scores, around the 70 % except the skull and the ball. This is because the high 3D similarity between both.

Object	**Precision**	**Recall**	**F1 score**
ball	0.53	0.83	0.65
skull	0.43	0.62	0.51
cup	1.00	0.66	0.79
bottle	0.77	0.69	0.73
mobile	0.95	0.67	0.78
calculator	0.82	0.62	0.71

Table 6.7: F1-score calculation using the precision and recall parameters. Results for templates using five views per object.

6.3.3 Template using 10 views

The last experiment was performed introducing ten views per object in the dataset. Figure 6.8 represents the confusion matrix obtained. The F1-score and the precision and recall per object may be found in figure 6.9.

Real \Predicted	**ball**	**skull**	**cup**	**bottle**	**mobile**	**calculator**
ball	**0.93**	0.00	0.07	0.00	0.00	0.00
skull	0.31	**0.69**	0.00	0.00	0.00	0.00
cup	0.03	0.14	**0.76**	0.00	0.03	0.03
bottle	0.03	0.14	0.00	**0.83**	0.00	0.00
mobile	0.07	0.03	0.14	0.00	**0.72**	0.03
calculator	0.21	0.00	0.00	0.03	0.00	**0.76**

Table 6.8: Confusion matrix using a template that stores ten views per object. The results are given in a 0 to 1 range.

The ball is the object with the highest success rate, a 93%. This time it was only confused with the cup a 7%. When the skull, the calculator and the mobile were presented to the system, the ball was detected a 0.31, 0.21 and 0.07. Also, the ball was recognized a negligible ratio when the cup and the bottle were shown. These results are expressed as well in table 6.9, the precision is much lower than the recall and the final F1 score is around a 70%.

The skull was detected correctly a 69% of times, it was only confused with the ball. The skull was also recognized when the cup, the bottle and the mobile were being used.

The cup had a success rate of 76%. The system recognized wrongly the skull 14 % and the ball, the mobile and calculator a negligible amount of times. The skull has a similar 2D texture, a white background with highly defined drawings on it. This could be the reason of the system's error. The cup was also detected when the ball and the mobile were shown to the system.

The bottle obtained a 89% success rate. The system only made a false recognition of the bottle when the calculator was being shown to it, and the ratio is negligigle. The bottle was confused with the skull and the ball, for the sames reasons as the cup. Those objects, the skull, the cup and the bottle have similar 2D features that may be matched wrongly.

The mobile obtained a 0.72 rate. The cup, ball, skull and calculator were recognized badly. The mobile was wrongly recognized only when the cup was being shown to the system and the ratio is negligible (0.3).

Finally, the calculator has a 76% success rate. The system detected the ball a 21% of times in this measurement. It is possible that the dense 2D textures of both the ball and the calculator affected the system in the matching process. The calculator was recognized a negligible amount of times when the mobile and the cup were being used.

Object	Precision	Recall	F1 score
ball	0.59	0.93	0.72
skull	0.69	0.69	0.69
cup	0.79	0.76	0.77
bottle	0.96	0.83	0.89
mobile	0.95	0.72	0.82
calculator	0.92	0.76	0.83

Table 6.9: F1-score calculation using the precision and recall parameters. Results for templates using ten views per object.

6.3.4 Comparison of the experiments results using different number of views

In this subsection the results obtained in the three object recognition accuracy experiments are summarized and presented together. This is performed to offer a more global view of the system's performance as a function of the number of templates being used. Figures 6.1 and 6.2 present a compilation of all the data extracted from the experiments.

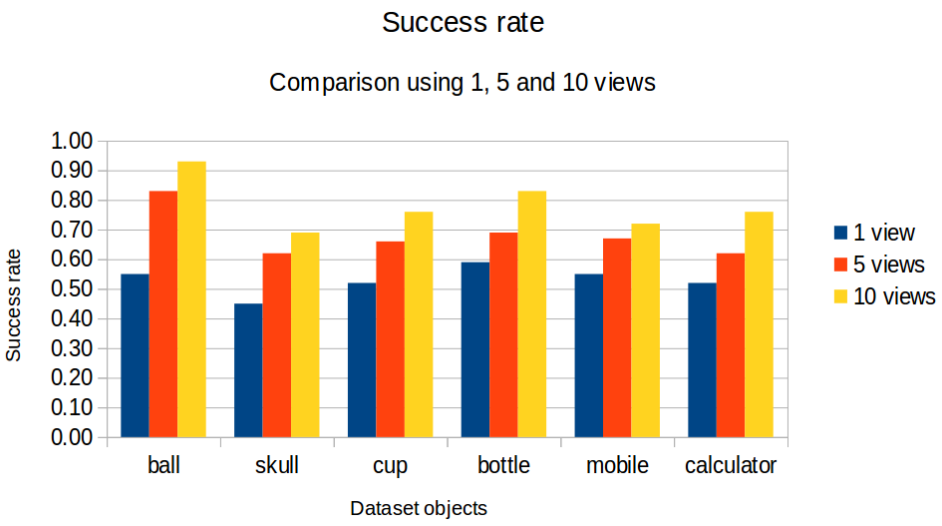


Figure 6.1: Comparison of the success rate when using templates with 1, 5 and 10 views per object.

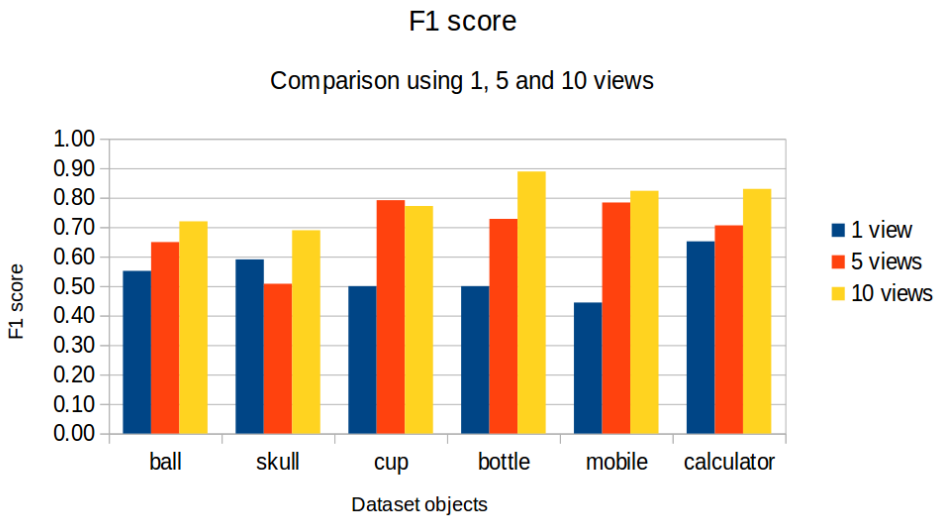


Figure 6.2: Comparison of the F1 score when using templates with 1, 5 and 10 views per object.

In 6.1 it can be seen that through the experiments the success rate for the different objects has increased. The differences in this rate between objects are almost identical from one experiment to the other. This demonstrates that the errors of the system due to 2D or 3D similarity between objects may be improved using a higher number of views. On the other hand, figure 6.2 shows that the improvement of the F1 score with the number of views is not as clear as it was in the case of the success rate. It is particular the case of the skull. When the views were increased from one to five, the system outputted the skull more when other objects were being used. This reduced the skull's precision and even though the recall was improved, the final F1 score was affected and resulted lower than the experiment using 1 view. The same phenomena occurred with the cup but when the number of views was increased from five to ten. All the other objects increased their F1 score with the augment of the number of views.

6.3.5 Comparison of the 2D and 3D independent recognition results and the system's output

It was previously described in chapter 4 that the system perform a 2D and 3D independent match. The output of the whole software is a combination of these instantaneous 2D and 3D predictions. In order to confirm that the final system's response improves the 2D and 3D independent recognitions, this section presents a comparison of the precision, recall and F1 score values of each type of recognition.

Object	Precision	Recall	F1 score
ball	0.49	0.84	0.62
skull	0.65	0.69	0.67
cup	0.52	0.73	0.61
bottle	0.61	0.87	0.72
mobile	0.73	0.79	0.76
calculator	0.89	0.82	0.85

Table 6.10: F1-score calculation of the independent 2D recognition using the precision and recall parameters. Results for templates using 10 views per object.

Object	Precision	Recall	F1 score
ball	0.62	0.79	0.69
skull	0.53	0.42	0.47
cup	0.43	0.64	0.51
bottle	0.56	0.72	0.63
mobile	0.45	0.63	0.53
calculator	0.66	0.82	0.73

Table 6.11: F1-score calculation of the independent 3D recognition using the precision and recall parameters. Results for templates using 10 views per object.

The experiment used for the comparison is the one in which ten views per object are learned. It is the one that obtained better F1 score values and hence the most representative of the system. Figure 6.10 shows the results obtained in the independent 2D object recognition, and Figure 6.11 the ones of the 3D object recognition. Since the most important parameter in the accuracy evaluation of the system is the F1 score, Figure 6.12 was created to summarize the results.

Object \ F1 score	2D	3D	System's output (2D+3D)
ball	0.62	0.69	0.72
skull	0.67	0.47	0.69
cup	0.61	0.51	0.77
bottle	0.72	0.63	0.89
mobile	0.76	0.53	0.82
calculator	0.85	0.73	0.83

Table 6.12: F1-score comparison of the independent 2D and 3D recognitions and the final output of the system combining them. Results for templates using 10 views per object.

The results in 6.12 show that the system's output F1 score is in all cases but one higher than the ones obtained in the independent 2D and 3D recognitions. Only in the last object, the calculator, the 2D matching process obtained a higher F1 score, but even in this case the difference is only of a 2%.

Chapter 7

Evaluation of the system: Discussion

This chapter covers the discussion of the tests results presented in section 6. It follows the same structure than chapters 5 and 6. First, the computing performance evaluation of both the nodes and the topics is discussed in section 7.1. Afterwards, the results obtained in the accuracy experiment are explained and justified in section 7.2. Chapter 8 is devoted to the improvement of the system based on the observations performed on the experiments.

7.1 Computing performance evaluation

In this section the results obtained when evaluating the computing performance of the system are discussed. First, the nodes' CPU and RAM usage is evaluated and afterwards the topic network usage, making a separate reference to the publishing rate and the bandwidth measurements.

7.1.1 Nodes' CPU and RAM usage

The total CPU usage of the system is of a 25%, which is roughly a single core of the CPU, and the RAM utilization of less than 1 GB of memory. This information suggests that the software is able to run on real time with the hardware used in the experiments. The percentage of the CPU and RAM usage (23% and 5% respectively) show that the system would able to work on real time in a computer with lower RAM and slower or smaller CPU than the one used. That is, the minimum requirements for a computer to be able to run this system on real time would be to have at least 1GB of RAM memory and a 1 core processor with a minimum speed of 2,40 GHz or equivalent processing unit.

7.1.2 Topic network usage

Chapter 6 presented the results of the topic network usage experiment. Two different characteristics were measured: the publishing rate and the bandwidth used. That data can be found in the figures 6.2 and 6.3 respectively.

- **Publishing rate**

  The bottle neck of the publishing rate of all the nodes is the kinect. Since this sensor is the one that provides the input information to the system, the maximum publishing rate of its nodes will be the one of the kinect, 30 messages per second. This figure is high enough to allow the system to perform on real time. In fact, the publishing rate of the output of the system was lowered up to approximately one result per second. This reduction of the publishing rate was performed to filter the instantaneous results and hence reduce the noise of the system without damaging its performance. Outputting one result per second is fast enough to permit a natural information exchange between the system and its users.

- **Bandwidth**

  The ROS framework allows the distribution of the processing among different devices that are interconnected. The total bandwidth measured in the system is below 7 MB/s, a figure that allows the implementation of this system using a distributed architecture. Nevertheless, the bandwidth of the different network types and technologies must be checked previously to ensure that their bandwidth is high enough.

  As an example, a LAN, WAN or MAN network might be introduced to communicate the system since their maximum bandwidth is in the range of the hundreds of megabytes per second. Nevertheless, the communication could not be performed on a Internet network using the 802.11b protocol, whose bandwidth has a maximum of 2 MB/s. This does not mean that the system is not able to communicate through the Internet. Other protocols created later such as the 802.11n have a much higher maximum bandwidth, in the range of the hundreds of megabits per second.

7.2 Evaluation of the object recognition accuracy

This section discusses the results obtained in the accuracy measurement experiment. To consult them please refer to section 6.3. Each different experiment is discussed in a different subsection.

7.2.1 Template using 1 view

The tables that show the confusion matrix of the experiment and the precision, recall and F1 score obtained are 6.4 and 6.5 respectively. The results show that the system outputted the skull, cup, bottle and mobile a noticeable amount of times when each of them were being used. This could be due to the similarity in the 2D texture in the skull, cup and bottle and and 3D similarity of the cup, when presented occluding the holder, the bottle and the mobile.

Figure 7.1: Detail of the skull, cup, bottle and mobile objects to illustrate their similarities.

As it can be seen in figure 7.1, the 2D details of the highly textured cup and bottle and also those of the skull present round lines that could have created similar 2D descriptors. Also, the 3D form of the cup, the bottle and the mobile is very similar which might lead as well to the creation of matchable 3D descriptors in the borders.

The cup's results are similar to the ball's. The precision and recall are similar in number and low. The features extracted from this object were not descriptive enough. The most descriptive features were obtained from the calculator, an object that presents an F1 score of 65% in this experiment.

7.2.2 Template using 5 views

The skull obtained approximately a 0.6 success rate, and most of the ball almost 0.4 recognition ratio when the skull was being used in the system. This results supports the theory that the 3D similarity between those objects, which may be seen in figure 7.2 may lead to very similar 3D descriptors.

Figure 7.2: Detail of the ball and the skull using different views to illustrate their 3D similarities.

The skull was recognized a ratio of 0.2 when the cup was being shown to the system, which is a high value. The reason behind it could be the similarity in the 2D texture of skull and cup. Both are white objects with drawings that contrast with that background, as can be seen in figure 7.1.

Figure 6.6 shows that when the mobile was shown to the system, the skull, the ball and the bottle were detected. The mobile has rounded corners and sides that could make 3D descriptors on those locations be similar to the previous objects. Also, the mobile is white with details represented in the 2D texture. This feature is common with the bottle and the cup. For certain positions of the cup and the mobile the 2D projection could be similar and lead to obtain less representative descriptors of the object. These similarities between the skull and the bottle may be seen in figure 7.1.

7.2.3 Template using 10 views

The results of this experiment may be found in figures 6.8 and 6.9. It can be observed an improvement with respect to the previous measurements. Now, all success rates and F1 scores of the objects are above the 69%. Also it can be seen that the number of false recognitions was reduced noticeably mainly in the bottle, mobile and calculator objects.

The bottle is the object with a higher F1 score of the whole experiment. This implies that the augment of the number of learned views improve the accuracy of the system limiting the high amount of false positives obtained in the experiments using 1 and 5 views. It is noticeable that the bad precision presented in the ball recognition limited its F1 score result. This was caused by the outputting of the ball when all the other objects were being used. The ball was detected wrongly a high amount of times when the skull and the calculator were being shown. The first case was already explained in section 7.2.2: the ball and the skull have very similar 3D form. The second case might be because of the fact that both objects have high 2D texture composed of straight lines and the 2D descriptors extracted might have been very similar.

The results of the experiment using ten views are very promising. Since there is an almost proportional relation between the F1 score and the number of views learned per object, further measurements could be performed to determine the optimum balance between F1 score and processing requirements.

7.2.4 Comparison of the 2D and 3D independent recognition results and the system's output

The comparison performed of the F1 score results for the 2D and 3D independent recognition processes and the final system's output, which combines both of them showed that the combination of results improves the F1 score. It is noticeable that the 3D algorithm obtained lower results in all the cases but one than the 2D. This fact corroborates the usefulness and correctness of the decision step performed in the system, in which a higher importance was given to the 2D predictions (see section 4.4.5.8).

It is also important to remark that further experiments may be performed to adapt better the importance given to each algorithm in the decision procedure. It is possible to even adapt it to the different situations and objects that could appear. Further details are explained in section 8.2.7.

Chapter 8

Conclusions and future work

This last chapter is devoted to present the conclusions drawn from the system. Also, the improvements that could be applied to enhance the software are enumerated and explained.

8.1 Conclusions

This system is, to the best of my knowledge, the first in the art to implement an in-hand object learning and recognition algorithm using 2D and 3D information. The fact of using both types of informations improve the robustness of the system to illumination changes or noise in the input data stream. The goal of the system is to serve as the base for further research on the topic to improve the interaction between humans and robots. More specifically, it might be used for social or assistive robots as an input. Evaluating the objects that the humans are holding may help to analyze the context around the robot. This is a crucial step in the path to the complete autonomy of social and assistive robots.

Two different sets of experiments were carried out to validate the system. The first goal was to ensure that the system was capable of operating in real time and that it would be possible to implement it using a distributed structure. The second one was performed to evaluate the accuracy of the system and its relation with the number of views per object learned. The results demonstrated that the software is able to run on real time and that the bandwidth consumption of its nodes, whose total value is less than 7 MB/s, was low enough to allow the implementation in a distributed architecture.

The experiments carried out to validate the accuracy of the system demonstrated a recognition rate near 80% when the dataset was created with ten views per object. The F1 score in these experiment was between the 70% and 80%, which is also a high value for this low number of views. Most object recognition systems obtain their best performance when a large dataset is used, containing training sets per object whose size range from fifty templates per object to hundreds of thousands.

Nevertheless, further experiments may be created to determine the optimum number of views for this system. The optimum number would be the one that gives a better accuracy and that permits the software to run on real time. Further studies may also be performed to evaluate the response of the system to perturbations introduced by the users. Each person holds the different objects and interacts with the robot in a different way. An study evaluating the performance of the system when different persons use it could be useful to determine future improvements of the software.

8.2 Future work

The present section contains the upgrades that could be implemented in the software to improve different aspects of it. The improvements could be done in some of the system's nodes. The introduction of this enhancements could be performed easily due to the modular structure of the code. This section is divided in subsections to observe the particular upgrades that could be done in each node of the system.

8.2.1 Hardware

The hardware used in this thesis is a kinect. It is a type of RGB-D sensor that was launched in 2010. It has a lower resolution than the recent kinect 2 launched the past year. When using small objects at the operating distance range of about 1.5m the images and point clouds that produces have a low resolution This fact leads to lower quality descriptors.

Another drawback of these devices is the noise. The sunlight and in summary all light sources that emit infrared radiation affect the output of the sensor. This noise creates a variation in the depth measures that is translated in erroneous event recognition or even hand recognition. The improvement in the noise resistance would improve the software's performance.

8.2.2 Hand location

The ROS package used for this task implements a robust solution to this problem for users that are not holding external objects. Nevertheless, when a person catches an object it is recognized as an extension of the arm. This means that the returned hand's position is not accurate if there is an object in the hand. This affects the ROI segmentation processing of the system because it causes that certain objects are not segmented correctly due to its size.

The solution would be to implement a new hand location taking into account for example the skin color. This way, other objects that are held and are not skin-colored are rejected for the hand's location computation. Another solution would be to create a wrapper to the existing package that creates an offset to the given position using the same principle. Alternatively, the center of mass could be computed in the hand. Assuming the user is holding a medium-sized object, the center of mass of the hand and object cluster is a good approximation to the center of the hand.

8.2.3 ROI segmentation

It was seen in previous chapters that the ROI segmentation was performed by two separate nodes. One node computed the 2D segmentation and the other the 3D. Both segmentations do not take into account the hand, they just crop a squared region around the center of the hand. Neither of them mask afterwards the hand, a fact that is a source of errors in the system. 2D and 3D descriptors are hence being extracted from the hand as well as from the object it holds. This means that if two objects are similar and therefore they are grabbed in a similar way the system could match them producing a false positive.

This problem may be solved more easily using 3D segmentation than using 2D segmentation, because the first has a higher amount of information than the latter. The hand segmentation may be done through a color segmentation or modeling the hand's form, searching for that pattern in the point cloud and subtracting it. The first option could be performed in 2D and 3D in the same way but since the skin's color range is wide and colors depend heavily on the illumination of the scene a correct segmentation is not guaranteed. The second option would be the most robust to illumination and pose changes. It consists on modeling the hand as an union of cylinders and conical frustums and search for these geometrical figures in the input point cloud. Nevertheless, this option would need a higher amount of processing time than the first one.

An even more robust and complete solution would be to combine both options, but an evaluation of the CPU and RAM cost of using both methods must be performed. After locating the hand, its contour may be obtained, transformed from world to pixel coordinates and finally published in a topic. The 2D ROI segmenter would then use the contour to apply a mask to eliminate the hand from the 2D information as well. The introduction of noise to the system would be highly reduced. The hand's position and location would only have an effect in the keypoints occlusion in the image.

Regarding 2D segmentation, a mask could be performed from the 3D hand segmentation. This way the hand would be removed obtaining the same improvements than in the 3D segmentation. In 2D segmentation the background is kept in the image. It was not removed due to the problems derived from using a background subtractor. Those problems are that the foreground could be considered background after a certain time. If that time is increased to avoid that problem, certain parts of the background would then become foreground. Those algorithms depend heavily on the lighting conditions. The 3D depth segmentation could be used to perform a better background subtractor for the 2D data. The contour of the object might be obtained in world coordinates and then transformed to pixels. Then, passing that information to the 2D ROI segmenter a new mask could be obtained to eliminate the background.

8.2.4 Feature extraction

As in the previous point, the feature extraction in the system is performed in 2D and 3D data. The 2D features are the state-of-the-art best solution for this application. They are faster and less time-consuming than the alternatives with comparable experimental results, as stated by [41]. The 3D features are computed using many approximations in order to reduce the huge computing time they require. Hence, they are less representative and is easier to obtain matches of features from different objects.

The main upgrade in this field would be then to improve the 3D features. It is a field that is still being an open question for object recognition. Some new algorithms such as LINEMOD [42] are appearing. In this particular one, new descriptors for texture-less objects are defined. The inclusion of a feature extractor similar to this one would improve the system's output, since most common objects do not have a marked texture. Many of the common objects do not posses a marked texture.

8.2.5 Learning and recognizing methods

This system implements a recognizing procedure using template matching. This methodology does not contemplate a learning of the algorithm. The descriptors extracted in each frame are compared with the previously stored and the identification number of the best match is the output of the system. There exists other machine learning algorithms that could be used for the learning and recognizing phase. As an example, random forests [43] or Support Vector Machines (SVMs) [44] might be implemented in order to upgrade the system's performance over time.

8.2.6 Interaction with the system

The interaction of the user's with the system could be improved using a voice interface that complements the gestural interface. Through voice commands the human could be able to label the newly learned objects and the system could output directly the name of the recognized object instead of an identification number.

Also, the interaction could improve the recognizing procedure of the system in the following way. If two objects have obtained a similar matching ratio and either of them could be the real object the user is presenting, the system may ask the human to show another view of the object to allow a more accurate recognition. This simple confirmation step could improve the F1 score obtained in the different experiments since the software would keep matching characteristics of different views until a more distinctive one is found and the best match is obtained.

8.2.7 Combination of 2D and 3D matching information

The results obtained (section 6.3.5) demonstrated that the weights being applied to each of the algorithms (2D and 3D) improved the system's output. Nevertheless, it is possible to apply a different weighting that varies with the user's pose or the object's size or 3D form to improve this important decision step.

The user's pose is important because each object is usually held in a different manner. This gives information about the size and form of the object the human is presenting. This data could be obtained as well from the point cloud, and mixing both decide whether the 3D figure is large and hence the 3D matching has a higher importance or if the object is small, the 2D texture becomes more relevant.

Another improvement that could be performed in this step is to calculate the precision and recall of the 2D and 3D predicted objects and from them obtain the F1 score for each algorithm. Then, the weightings for 2D and 3D could be directly the F1 score or a mixture of the F1 score and other factors such as the size or the amount of 2D texture of the object. This approasch would probably obtain better results, but the computational cost of these calculations may be studied to ensure that the system still works on real time.

Bibliography

[1] "Spanish women behind only the Japanese in life-expectancy." http://www.fabpropertysales.com/spanish-women-behind-japanese-life-expectancy/. Accessed: 2014-6-8.

[2] D. o. E. United Nations and P. D. Social Affairs, "World population ageing 2013 [st/esa/ser.a/348]," 2013.

[3] J. Broekens, M. Heerink, and H. Rosendal, "Assistive social robots in elderly care : a review Assistive social robots," vol. 8, no. 2, 2009.

[4] "Japanese robots: Iso assistive robotics tech standards to be based on japan's." http://akihabaranews.com/2013/08/07/article-en/japanese-robots-iso-assistive-robotics-tech-standards-be-based-japans-11157743. Accessed: 2014-6-8.

[5] "Iso 13482:2014. robots and robotic devices – safety requirements for personal care robots." http://www.iso.org/iso/catalogue_detail.htm?csnumber=53820. Accessed: 2014-6-8.

[6] "Softbank mobile and aldebaran unveil "pepper" – the world's first personal robot that reads emotions." http://www.softbank.jp/en/corp/group/sbm/news/press/2014/20140605_01/. Accessed: 2014-6-8.

[7] "Investigadores españoles desarrollarán robots asistenciales para ayudar a los cuidadores de EA." http://www.alzheimeruniversal.eu/2012/03/12/investigadores-espanoles-desarrollaran-robots-asistenciales-para-ayudar-a-los-cuidadores-de-ea/. Accessed: 2014-6-8.

[8] "La fundación ave maría integra robots asistenciales." http://sitgesvida.com/la-fundacion-ave-maria-integra-robots-asistenciales-y-de-apoyo-para-la-mejora-en-la-atencion-a-personas-discapacitadas/#. Accessed: 2014-6-8.

[9] "La ciencia española retrocede una década, hasta niveles de 2005." http://www.elmundo.es/ciencia/2014/03/05/5317517e268e3e34238b4580.html. Accessed: 2014-6-11.

[10] V. Delaitre, "Learning person-object interactions for action recognition in still images," pp. 1–9.

[11] A. Fathi, Y. Li, and J. M. Rehg, "Learning to Recognize Daily Actions using Gaze," *European Conference on Computer Vision*, 2012.

[12] "ROS." http://www.ros.org/. Accessed: 2014-3-27.

[13] "Openni. The standard framework for 3D sensing.." http://www.openni.org/. Accessed: 2014-3-30.

[14] R. Mitchell, "PrimeSense releases open source drivers, middleware for Kinect." http://www.joystiq.com/2010/12/10/primesense-releases-open-source-drivers-middleware-for-kinect/, 2010. Accessed: 2014-3-30.

[15] R. Knies, "Academics, Enthusiasts to Get Kinect SDK." http://research.microsoft.com/en-us/news/features/kinectforwindowssdk-022111.aspx, 2011. Accessed: 2014-3-30.

[16] T. S. Huang, "Computer Vision : Evolution and Promise," *19th CERN School of Computing*.

[17] L. Roberts, *Machine perception of three-dimensional solids*. Ph.d. thesis, Massachusetts Institute of Technology, 1963.

[18] D.M.Gavrila and F. Groen, "3D object recognition from 2D images using geometric hashing," 1991.

[19] A. F. Sheta, A. Baareh, and M. Al-Batah, "3D object recognition using fuzzy mathematical modeling of 2D images," *2012 International Conference on Multimedia Computing and Systems*, pp. 278–283, May 2012.

[20] T. H. Reiss, "The revised fundamental theorem of moment invariants," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, pp. 830–834, Aug. 1991.

[21] M. Teague, "Image analysis via the general theory of moments," *J. Optical Soc. Am.*, no. vol. 70, no. 8, p. 920–930, 1980.

[22] M.-K. Hu, "Visual Pattern Recognition by Moment Invariants," pp. 66–70, 1962.

[23] M. Z. Zia, M. Stark, B. Schiele, and K. Schindler, "Detailed 3D representations for object recognition and modeling.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, pp. 2608–23, Nov. 2013.

[24] P. Roth, M. Donoser, and H. Bischof, "On-line learning of unknown hand held objects via tracking," *Int. Conf. on Computer Vision*, 2006.

[25] M. Philipose, "Egocentric recognition of handled objects: Benchmark and analysis," *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8, June 2009.

[26] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.

[27] K. Mikolajczyk and C. Schmid, "Performance evaluation of local descriptors.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, pp. 1615–30, Oct. 2005.

[28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*, pp. 2564–2571, IEEE, Nov. 2011.

[29] S. Se, H.-k. Ng, P. Jasiobedzki, and T.-j. Moyung, "VISION BASED MODELING AND LOCALIZATION FOR PLANETARY EXPLORATION ROVERS," 2004.

[30] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," *Computer Vision–ECCV 2006*, 2006.

[31] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision–ECCV 2006*, pp. 1–14, 2006.

[32] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, 2010.

[33] P. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, 1999.

[34] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," *Proceedings of the 15th international conference on Multimedia - MULTIMEDIA '07*, no. c, p. 357, 2007.

[35] R. Rusu, N. Blodow, Z. Marton, and M. Beetz, "Aligning point cloud views using persistent feature histograms," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3384–3391, Sept. 2008.

[36] R. B. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D Registration,"

[37] "ASUS Xtion Pro Live RGB-D Sensor." http://www.asus.com/Multimedia/Xtion_PRO_LIVE/. Accessed: 2014-3-27.

[38] "OpenCV." http://opencv.org/. Accessed: 2014-3-27.

[39] "BSD." http://www.linfo.org/bsdlicense.html. Accessed: 2014-3-27.

[40] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[41] O. Miksik and K. Mikolajczyk, "Evaluation of local detectors and descriptors for fast feature matching," *Pattern Recognition (ICPR), 2012 21st*, 2012.

[42] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit, "Gradient response maps for real-time detection of textureless objects.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, pp. 876–88, May 2012.

[43] J. Gall, N. Razavi, and L. V. Gool, "An introduction to random forests for multi-class object detection," *Outdoor and Large-Scale Real-World*, pp. 1–21, 2012.

[44] M. Pontil and a. Verri, "Support vector machines for 3D object recognition," June 1998.

[45] "Sloccount." http://www.dwheeler.com/sloccount/. Accessed: 2014-6-11.

[46] B. e. a. Boehm, *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, January 2000.

Appendices

9.1 Appendix: Regulatory compliance

The present section covers the regulatory compliance that affects directly to the system presented. Nowadays, the author of the software has the right of sharing it using a contract. In it he determines which of the author rights he is going to yield and under what conditions. This type of contract is called a software license. Sections 9.1.2 to 9.1.4 show the details of the licenses of this thesis and the third-party libraries being used.

9.1.1 OpenCV

The Open Source Computer Vision library is released under a BSD license. Hence it is free for both academic and commercial use. Nevertheless, there are certain algorithms implemented whose license is different from the whole library. The SIFT or SURF descriptors are two examples of this fact: they have a software patent. This legal figure allows the use of the algorithms for investigation purposes, but imposes the payment of a fee for commercial uses. Since OCULAR is intended to be an Open Source system, neither of these algorithms are used. Instead, the ORB algorithm is utilized in the system.ORB has no patent and hence can be incorporated in software designed for both commercial and research uses.

9.1.2 PCL

The Point Cloud Library has as well a BSD license. It is then free for commercial and research use. Further information about the library and its license may be found in this webpage [1].

9.1.3 ROS

All ROS core code is distributed under a BSD license, more specifically a BSD 3-Clause license. It is very similar to the OCULAR license, the redistribution is permitted under certain conditions. More information may be found in this webpage[2]. The different ROS packages that are used in this thesis (openni_camera, openni_launch and pi_tracker) are distributed under a BSD license, according to their web pages (openni_camera[3], openni_launch[4], pi_tracker [5]).

9.1.4 OCULAR

The license of this system must be coherent with the ones of the third-party libraries it uses. In fact, Open Source packages and libraries were selected to ensure that the whole system could be free for commercial and research use. This is the reason why this project is being distributed with a MIT License (MIT). The MIT license may be found in this link[6], and states the following:

[1] http://pointclouds.org
[2] http://opensource.org/licenses/BSD-3-Clause
[3] http://wiki.ros.org/openni$_camera$
[4] http://wiki.ros.org/openni$_launch$
[5] http://wiki.ros.org/pi_tracker
[6] https://raw.githubusercontent.com/irenesanznieto/ocular/master/LICENSE.md

The last paragraph is published in uppercase letters, but it was converted to lowercase to avoid a disturbance in the structure of the thesis. The license claims that software is available for redistribution and use, but no warranty is provided with it. The different algorithms and third-party packages were selected taking into account that their licenses must be compatible with the one being provided by this system. In the next sections the licenses under which the different packages are distributed are presented.

Apart from the licensing of the software, it was taken into account in the development of the system that this project needs to store the information learned to be able to use it in later sessions. In order to observe the data protection Spanish law (Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal), the stored data does not contain any personal information. Instead of storing, for example, an image of the user, the information retrieved is a matrix of numbers containing the descriptors extracted from the objects. This ensures that the system could be used in commercial software or that further investigations could be performed accordingly with the law.

9.2 Appendix: Project management

The present section describes the project management followed in this thesis. The Gantt diagram created for this purpose can be seen in the following page.

The thesis was started on February 7th 2013 and was finished on July 11th 2014. The project management has suffered modifications throughout its development. The final project management chart has three differentiated parts: the learning phase, the project programming and the documentation phase.

PHASE	BEGIN DATE	END DATE	DAYS
Planning of the Thesis	07/02/13	25/02/13	18
State of the art	28/04/13	11/07/14	439
Learning	26/02/13	29/10/13	185 **
OpenCV	26/02/13	24/04/13	57
OpenCV Demo	25/04/13	25/04/13	MILESTONE
PCL + ROS	25/04/13	29/10/13	127 **
PCL + ROS Demo	30/10/13	30/10/13	MILESTONE
Project Objectives Definition	30/10/13	07/11/13	8
Design	08/11/13	21/11/13	13
Project Programming	22/11/13	20/05/14	179
pi_tracker integration : converter	22/11/13	20/01/14	59
ROI segmentation implementation	21/01/14	20/02/14	30
Feature extraction implementation	21/02/14	15/04/14	53
State machine implementation	16/04/14	14/05/14	28
Decision algorithm development	15/05/14	20/05/14	5
Tests	22/11/13	20/05/14	179
Documentation	21/05/14	11/07/14	51
Thesis writing	21/05/14	22/06/14	32
Thesis hand-in	22/06/14	22/06/14	MILESTONE
Presentation preparation	23/06/14	11/07/14	18
Thesis presentation	11/07/14	11/07/14	MILESTONE
TOTAL DAYS:			**454**
TOTAL HOURS (3h per day) :			**1362**

Table 9.1: Summary of the days and hours spent in each of the project's phases.
** 60 days of holidays are not taken into account in the calculations.

Figure 9.1 presents a summary of the hours dedicated to each of the phases of the project. These thesis parts are the following:

- **Planning of the thesis**
  In this phase a first research on computer vision and the different methods and algorithms used in object learning and recognition was performed. The outline of the thesis planning based on the acquired knowledge was created.

- **State of the art**
  This part is the most time-consuming. It consisted on the profound research and understanding of the different methodologies and algorithms devoted to computer vision in general and in-hand object recognition in particular. Also, it includes the learning and understanding of third-party packages in order to use them later on in the project's development. Figure 9.1 shows that this phase was developed in parallel with the other ones.

- **Learning phase**
  This part consisted on a exploration of all the different technologies that could be used and the different state of the art techniques available. A thorough research on the object recognition and human tracking fields was performed. This research continued until the finishing of the thesis, but the most important part of it was made in this period of time. The methods found were tested through demonstrations and there the main problems and possible solutions were obtained. This phase was crucial for the project, since in it the requisites of the software were defined as well as the technologies used and the general skeleton of the project's design that would later be implemented.

- **Project objectives definition**
  After the learning and first learning of computer vision algorithms, the objectives of the project were defined. They can be consulted in section 1.4.

- **Design**
  This phase consisted on the design of the software that has been developed in this thesis. In this part the project's requirements were created. This document can be found in appendix 9.4. The requirements document content was followed during the programming phase.

- **Project programming**
  In this phase the project was coded and tested. The state of the art algorithms research continued to allow the overcoming of the different difficulties that appear when implementing theoretical concepts. This modified the project's design as well as some of the technologies applied in the thesis.

- **Documentation**
  This final part of the project consisted on creating the documentation of the project and the present thesis. Also, the presentation documentation was prepared.

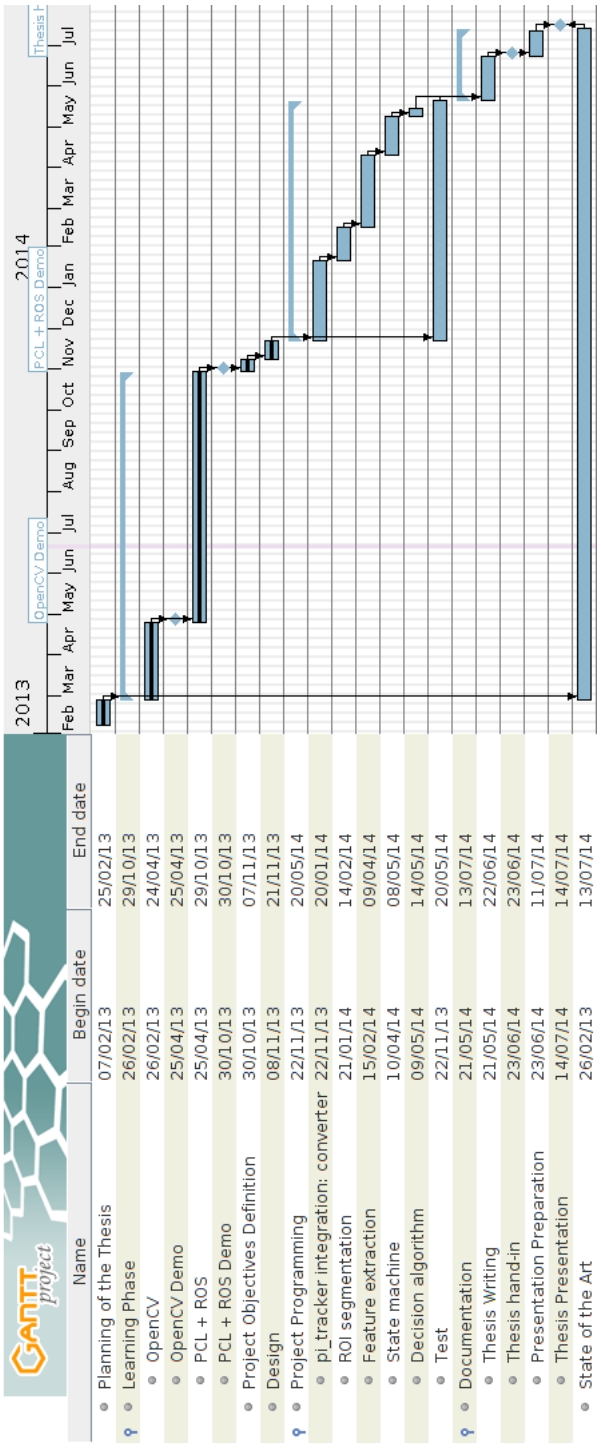


Figure 9.1: Gantt Diagram that specifies the different phases of the project and the time devoted to each of them.

9.3 Appendix: Budget

This appendix presents the estimated budget of the project. Appendix 9.2 explained the time management followed in this project and the different phases it underwent. Figure 9.1 shows the summary of the days spent in each of the phases. The phase that occupied most of the time was the learning phase. The state of the art learning occupies most of the project since it was performed in parallel with most of the other phases, as can be seen in figure 9.1. The next phase in size is the project programming and then the documentation of the project.

It can be seen from figure 9.1 that the total days spent in the development of this project were 454. It was assumed that around 60 days between the begin and end dates were not devoted to this thesis. Also, an average of 3 hours per day were reserved to implementing this system and hence the total hours used were 1362. Figure 9.2 shows the final budget of the project. The first two items are the hardware required, then the man hours cost and finally the software cost. It can be noted that the software total cost was of 0 €. This is because all software used in this system is Open Source, hence no licenses were paid.

ITEM	COST (€)	UNITS	QUANTITY	TOTAL COST (€)
Hardware				
Kinect 360	120	€/unit	1	120
Mountain Ivy 11 Laptop	1000	€/unit	1	1000
Man hours				
Undegraduate Engineer	8	€/h	1362	10896
Software				
Linux	0	€/license	1	0
ROS	0	€/license	1	0
openni_camera	0	€/license	1	0
openni_launch	0	€/license	1	0
pi_tracker	0	€/license	1	0
PCL	0	€/license	1	0
OpenCV	0	€/license	1	0
TOTAL PROJECT'S COST:				**12016**

Table 9.2: Project's budget

Since all the software used is Open Source, I have performed an estimation of the cost of the development of each of the libraries I have used in this project. Figure 9.3 summarizes the results.

SOFTWARE	TOTAL COST (k€)
Linux (Ubuntu 13.04, 64 bits)	10680
ROS (Groovy distribution, v1.9.50)	2932
openni_camera (v1.8.9)	99
openni_launch (v1.8.3)	6
pi_tracker (Groovy release)	74
PCL (v1.7.1)	13101
OpenCV (v2.3.1)	36264
OCULAR	59
TOTAL ESTIMATED COST:	**63216**

Table 9.3: Open source software cost estimation

The estimations are created using the SLOCCount Open Source program [45]. This software measures the number of lines and automatically estimates the effort, time and money needed to create the software. The figures in table 9.3 are obtained using the basic COnstructive COst MOdel (COCOMO model) [46].

All defaults were used and the final cost of developing each of the tools are the ones presented in figure 9.3. It is important to note that in the case that these projects would not have been Open Source the cost of using them would not be, of course, the cost of the complete development. Instead, a payment of a license is needed in order to obtain the right to use of the software.

The cost estimation of the system developed in this thesis was also performed. It can be seen the high difference between the real cost and the estimated one. This is mainly due to the difference in the man-hours cost. By default, the software assumes a yearly cost per man of around 56 k USD which means an hourly rate of around 20 USD per hour, or 15 €. The final cost of the developed thesis is less than 13000 €, including all the required hardware.

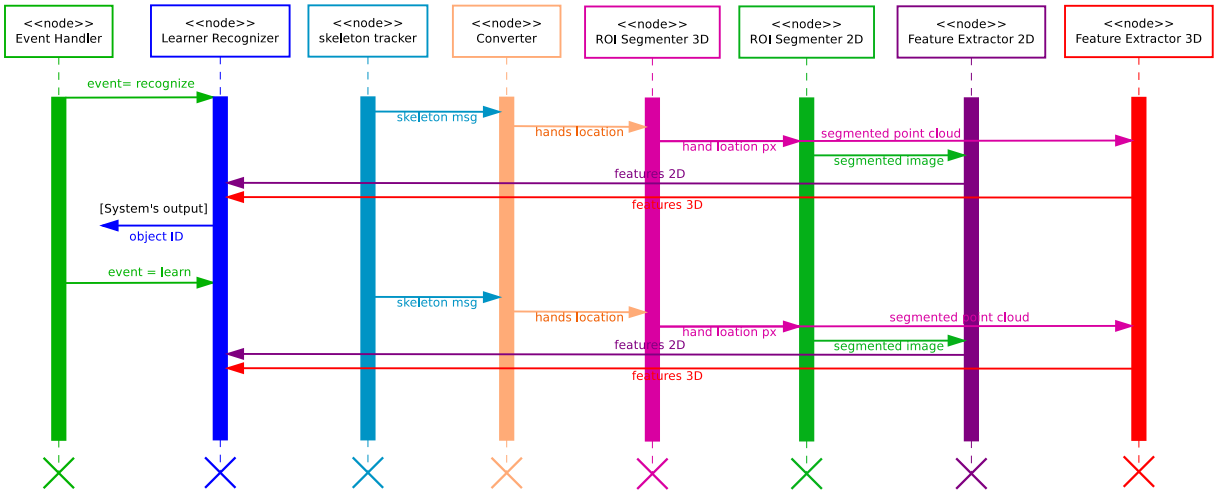
9.4 Appendix: Software requirements specification

9.4.1 Node Interaction

In the following diagram it can be seen the interaction and the information exchanged between nodes.

9.4.2 Sequence Diagram

In the following diagram can be seen the interaction between nodes.
The Event Handler node triggers the execution of the Recognizer and the Learner nodes. As it was previously mentioned, the triggering event is defined to create a more natural human-machine interaction. The converter, ROI Segmenters and Feature Extractors nodes are continuously working and publishing the information on their topics. Hence, the information needed for the Event Handler, Recognizer and Learner nodes is obtained subscribing to those topics.



9.4.3 Functional requirements

9.4.3.1 General requirements

FR001 The software must be developed under ROS (Robotic Operating System), using the Groovy distribution and rosbuild.

FR002 A version control system should be used (GIT) and all the code must be periodically updated in the following github repository: repository[7].

FR003 The software must accept as inputs the following RGB-D sensors: Microsoft Kinect, ASUS Xtion PRO, ASUS Xtion PRO Live, PrimeSense PSDK 5.0.

FR004 The output of the system must be a topic specifying the name of the detected object.

9.4.3.2 Converter

FR005 The information must be received subscribing to the topics of the different third-party packages.

FR006 This node must convert different message formats to the custom message used within this project, serving as an interface.

[7] http://github.com/irenesanznieto/ocular

9.4.3.3 ROI Segmenters

ROI Segmenter 3D

FR007 This node must segment the Region Of Interest in 3D (point cloud).

FR008 The ROI must be a box around the hand's position.

FR009 The dimensions of the ROI box must be fixed and must have a value suitable for this application.

ROI Segmenter 2D

FR010 This node must segment the Region Of Interest in 2D (image).

FR011 The original input image must be the one obtained directly from the RGB-D sensor being used.

FR012 The size of the 2D ROI must be extracted from the 3D ROI x and y dimensions. Those measures must be in pixels.

FR013 The 2D ROI must be cropped from the original image using the points obtained from the 3D ROI.

9.4.3.4 Feature Extractors

Feature Extractor 2D

FR014 This node must extract the 2D features.

FR015 The 2D features will be expressed as a vector of descriptors.

FR016 The 2D features of all the views of each object will be stored in a file, using the data parser class.

Feature Extractor 3D

FR017 This node must extract the 3D features.

FR018 The 3D features will be expressed as a vector of features.

FR019 The 3D features of all the views of each object will be stored in a file, using the data parser class.

9.4.3.5 Data Parser

FR020 This class compresses the features obtained to have one data file per object with all the information.

FR021 It has to be able to create a new file for storing the features of a new object.

FR022 It has to be able to delete certain views or a complete file of the dataset.

FR023 It has to be able to add another view to a previously stored object.

FR024 This class will also be able to read the files to extract the 2D and 3D features of each object of the dataset.

FR025 The name of the file will be the one given to the object.

9.4.3.6 Event Handler

FR026 This node must have access to the hand's position information.

FR027 This node must launch the object learner node or the object recognizer node depending on a triggering event.

FR028 The triggering event occurs when the distance between the hand and the torso is equal or higher than 0.5m.

9.4.3.7 Learner-Recotnizer

Learner mode

FR029 The input to this node will be the features (both 2D and 3D) extracted from the feature extractor node.

FR030 This node must have two algorithms, one for 2D and other for 3D.

Recognizer mode

FR031 A weighting must be made depending on the amount of texture possessed by each object to decide which matching should have more importance, the 2D or the 3D one.

FR032 The output of this node must be a topic in which the object's name is published.

9.4.4 Performance requirements

PR001 The recognition must run on the lowest amount of time possible, to allow a fluid human-robot interaction. This time must be lower than 500 ms.

PR002 The learning must run on the lowest amount of time possible, to allow a fluid human-robot interaction. This time must be lower than 5s.

9.4.5 Documentation requirements

DR001 The code must be completely documented, i.e. each class, class function , class member and piece of code should have comments explaining all the aspects.

DR002 The documentation will be made using Doxygen notation, through the rosdoc_ lite ROS package.

DR003 All the documentation must be available at the git repository.

9.4.6 Maintainability requirements

MR001 The code must be as modular as possible.

MR002 All the classes' variables that are susceptible of being modified through a inheritance will be declared as protected.

MR003 Gtests will be built to demonstrate each functional requirement.