Sistema de reconeixement d’objectes en una escena dinàmica en temps real

Jordi Vilaseca Corderroure

Director: Joan Aranda López (ESAII)
Codirector: Manuel Vinagre Ruiz (ESAII)

June 29th, 2016
Computer Engineering - Computation

Barcelona School of Informatics (FIB)
Polytechnic University of Catalonia (UPC BarcelonaTech)
Resum

Aquest projecte està orientat al camp de la visió per computador i aprenentatge automàtic. Es tracta de un sistema encarregat del reconeixement d’objectes en una escena dinàmica en temps real. L’escena en qüestion és una réplica d’una real i els objectes tractats són envasos.

El sistema al complet utilitza el framework *Robot Operation System* per facilitar la feina d’integració amb altres elements.

Les dades s’obtenen mitjançant una càmera *Kinect 2* que proporciona informació RGBD, aquesta s’emmagatzema utilitzant un núvol de punts on s’apliquen la majoria dels algorismes utilitzats.

El sistema comença amb una etapa de segmentació basada en plans. El primer pas consisteix en trobar el plans de l’escena i els seus límits, es consideren objectes els elements que estan sobre aquests plans i dins els límits establerts.

El següent pas consisteix en extreure la informació de forma (*CVFH*) i de color (*Histograma de Color*), i finalment, seguint una estratègia del tipus *Bag of Features*, obtenir el descriptor final. Aquesta informació es passa a una *Support Vector Machine* per aconseguir l’identificador del objecte. Els resultats finals es mostren en un visualitzador *rviz* i es fan accessibles per tal de que altres programes hi puguin treballar.

S’han proporcionat diferents eines per fer la vida més fàcil al usuari. Per exemple per capturar les imatges dels objectes i el posterior entrenament de la *Support Vector Machine*, a més a més de l’extracció de resultats per un conjunt d’objectes determinat. També s’han proporcionat altres eines per a la integració del sistema amb un robot.
Resumen

Este proyecto está orientado en el campo de la visión por computador i apren-
dizaje automático. Se trata de un sistema de reconocimiento de objetos en una
escena dinámica en tiempo real. La escena consiste en la representación de una
cocina real y los objetos a reconocer envases.

El sistema al completo está construido utilizando el framework *Robot Oper-
ating System* para facilitar la tarea de integración con otros sistemas.

Los datos se obtienen mediante una cámara *Kinect 2* que proporciona infor-
mación RBGD, esta se almacena una nube de puntos donde se aplican la mayoría
de algoritmos utilizados.

El sistema empieza con una fase de segmentación basada en planos. El primer
paso consiste en encontrar los planos existentes en la escena y sus límites, se con-
sideran envases los elementos que están encima de estos planos y dentro de los
límites definidos.

El siguiente paso consiste en extraer la información de forma (*CVFH*) y color
(*Histograma de Color*) de los objetos antes mencionados, utilizando una estrate-
gia basada en un *Bag of Features* se obtiene el descriptor final. Esta información
se pasa a una *Support Vector Machine* para obtener el identificador del objeto.
Los resultados finales se muestran en el visualizador *rviz* y se hacen accesibles
para que otros programas puedan trabajar con ellos.

Se proporcionan diferentes herramientas que permiten hacer la vida más fácil
al usuario. Por ejemplo, para capturar imágenes de objetos y el posterior entre-
namiento de la *Support Vector Machine*, a parte de la extracción de resultados
para un conjunto de objetos determinado. Se incluyen otras herramientas para
la integración del sistema con un robot.
Abstract

Project orientated to the field of the computer vision and machine learning. System in charge of recognising objects of a dynamic scene in real time. The scene is a representation of a real kitchen and the objects kitchen containers.

The entire system is built upon Robot Operating System to make it easily integrable to other systems.

The data is acquired using a Kinect 2 camera that provides with RGBD data that is stored in a point cloud structure in which is applied most part of the algorithms that the project uses.

The system has different steps. The first one is the segmentation that is based on plane segmentation. It starts by finding the available planes in the scene and their limits, these planes are defined as the background and the elements above them the objects to be extracted.

Shape (CVFH) and colour (Colour Histogram) features are extracted from the objects, and using a Bag of Features scheme the object descriptor is computed. Then, this information is fed to a Support Vector Machine to obtain the identifier of the object. The resultant data is displayed into the rviz visualizer and made available for other programs to work with.

Tools for capturing data and the posterior training of the Support vector machine are provided to ease the work to the user, including extraction the results for a determined set of objects point clouds. It also has some tools to integrate the system to a robot.
Acknowledgements

I wish to express my sincere thanks to Joan Aranda López, the director of this project to give me the opportunity of doing it in this great workspace and being always available.

I would like to thank the co-director Manuel Vinagre Ruíz for answering my endless questions and together with the other people of the laboratory to easing the workday and making the stressful days something easy to go by.

I take this opportunity to express also gratitude to my family, for the support they always showed, the discussions about this project and for the useful ideas they give me keeping in mind they nearly know nothing about the themes involved in it, but they make the effort to understand them, and a greater effort to try to help me.

Finally, I would like to thank to my friends, university colleagues and everyone involved directly or indirectly with the development of the project.
## Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Problem’s formulation</td>
<td>2</td>
</tr>
<tr>
<td>3 Background</td>
<td>3</td>
</tr>
<tr>
<td>3.1 Image and point cloud</td>
<td>3</td>
</tr>
<tr>
<td>3.2 Why use a Kinect?</td>
<td>4</td>
</tr>
<tr>
<td>3.3 Used libraries</td>
<td>4</td>
</tr>
<tr>
<td>4 Temporal planning</td>
<td>6</td>
</tr>
<tr>
<td>4.1 Project planning</td>
<td>6</td>
</tr>
<tr>
<td>4.2 Design and implementation stage</td>
<td>6</td>
</tr>
<tr>
<td>4.3 Final task</td>
<td>7</td>
</tr>
<tr>
<td>4.4 Stages duration</td>
<td>7</td>
</tr>
<tr>
<td>4.5 Planning deviation</td>
<td>8</td>
</tr>
<tr>
<td>5 Scope</td>
<td>9</td>
</tr>
<tr>
<td>6 Budget</td>
<td>10</td>
</tr>
<tr>
<td>6.1 Human resources</td>
<td>10</td>
</tr>
<tr>
<td>6.2 Direct costs</td>
<td>10</td>
</tr>
<tr>
<td>6.3 Indirect costs</td>
<td>10</td>
</tr>
<tr>
<td>6.4 Total budget</td>
<td>11</td>
</tr>
<tr>
<td>6.5 Deviation</td>
<td>11</td>
</tr>
<tr>
<td>7 Methodology</td>
<td>12</td>
</tr>
<tr>
<td>8 Social commitment</td>
<td>13</td>
</tr>
<tr>
<td>9 Sustainability</td>
<td>14</td>
</tr>
<tr>
<td>10 Implemented nodes</td>
<td>15</td>
</tr>
<tr>
<td>11 Camera calibration</td>
<td>16</td>
</tr>
<tr>
<td>11.1 Intrinsic calibration</td>
<td>16</td>
</tr>
<tr>
<td>11.2 Extrinsic calibration</td>
<td>16</td>
</tr>
<tr>
<td>12 Changing coordinate frame</td>
<td>17</td>
</tr>
<tr>
<td>13 Point cloud creation</td>
<td>18</td>
</tr>
<tr>
<td>14 Plane detection</td>
<td>19</td>
</tr>
<tr>
<td>15 Plane limitation</td>
<td>21</td>
</tr>
<tr>
<td>15.1 Normal filtering</td>
<td>21</td>
</tr>
<tr>
<td>15.2 Project points to the plane</td>
<td>22</td>
</tr>
<tr>
<td>15.3 Discarding unwanted clusters</td>
<td>22</td>
</tr>
<tr>
<td>15.4 Finding plane limits</td>
<td>22</td>
</tr>
<tr>
<td>16 Segmentation</td>
<td>25</td>
</tr>
</tbody>
</table>
1 Introduction

It was some time ago that the first robots were introduced to the industry because it was a necessity; currently this urge is translated to other environments where robots could be of great use. I am talking about the house environment, where robots will be introduced developing multiple tasks, firstly with the idea of assisting people with their housework and afterwards trying to release their owners of doing these tasks. This future could be near that is expected, and its introduction will provoke a great change in the way the housework is done.

My bachelor thesis is included in this context. Where the recognition of objects is highly used to provide the robots of eyes to understand what is surrounding them, knowing with what objects it can interact and possible obstacles to avoid while an object is moved from one place to another. To do this, the initial position, final position and bounding box of the object is needed, with the system suggested in this project these data could be obtained.

In this case the scene involves a kitchen equipped with different cupboards, a cook-top, a sink and so on. A true scenario where sometimes the recognition of objects and its movement could be a truly nightmare.
2 Problem’s formulation

The principal goal of this project is to do a real time system that could be able to detect and recognise objects that are placed on a table or any other rectangular surface. Its output will be the recognised objects identifier besides with its position and an oriented bounding box.

The system will be capable to start with a scene that has objects already placed in it. Another important feature is to recognise objects partially hidden, either by other ones or by someone holding them. The system will continue recognising the visible elements although somebody is moving the objects from one place to another.

The used objects will be kitchen containers, that is because they easy manipulable and with enough variability and characteristics to be able to recognise them.

A Kinect 2 camera will be used to acquire the images, this camera sends RGBD information, that apart from colour it also provides with the depth of the pixels (not from all of them, but using an interpolation method a depth for each pixel could be computed). This is a valuable information, because it will allow the system the know the position of an object and find the table planes.

It must be easy for the user to use the system, for that reason different nodes are provided to ease the training stage and to know what is going on in the scene, one of them is a visualizer that shows the points of the segmented objects together with their bounding box and identifier.
3 Background

3.1 Image and point cloud

In the computer vision field the input data was historically in format of images. In the last years a new tendency has appeared, the use of point clouds. A point cloud is a set of data points in a specific coordinate system, each one of them usually defined with X, Y and Z coordinates and a colour. This additional data is very valuable in this field of research because it allows us to use geometrical properties.

For example in the segmentation step, where the background must be separated from the foreground one of the most common approaches using colour data is the background subtraction. This technique makes use of a reference frame that contains the empty scene (without elements that are wanted to be recognised), when a new frame must be segmented, the difference between them in each pixel is computed and the pixels with a difference bigger than a threshold are marked as part of an object. This method only works when the scene is fixed in relation to the camera and the level of illumination effects the obtained segmentation.

![Figure 1: Example of the background subtraction method.](image)

But there is a thing that is present in most object recognition contexts. Objects are usually placed on tables, cupboards, the floor, etc. And all these surfaces have a property, they are flat. One of the things that could be done with a point cloud is to use an algorithm to find planes based on the position of the points. Then the segmentation step consists on only extracting the points that are above this plane. This method of segmentation has different advantages in relation to previous one, the illumination does not effect the segmentation, and the environment could change and the system will continue to work properly unless there is a movement of the plane in relation of the camera, in this case a recalibration of the system will be needed.

In the recognition step, in case of using an image the used descriptors are based only on colour information. In the point cloud case, in addition to colour, shape information could be used making the recognition step more powerful. Two objects with similar colour and very different shape could be misclassified if only the colour information is used, the same happens with objects with very different colour and similar shape. Using the two types of data at the same time could improve the robustness of the recognition.
3.2 Why use a Kinect?

There are different methods to obtain the necessary information to create a point cloud. The most common is stereoscopic vision, this method is based in the way the human being percepts the profundity. It makes use of two conventional RGB cameras displaced horizontally one from another by a known distance, then the depth is calculated by dividing the two images in small blocks and making the correspondence between them in the two images, this correspondence must be calculated for each pair of received images, making the acquisition slower.

Another way is to use a combination between a Time-of-Flight and an RGB camera. A Time-of-Flight (TOF) camera works by illuminating the scene with a modulated light source and observing the reflected light. The phase shift between the illumination and the reflection is measured and translated to distance. The only problem here is that you need to know the correspondence between points of the two cameras to know the depth of a pixel, this correspondence is calculated using a calibration step, more information about this is explained in section 11. Kinect 2 makes use of this technology to provide RGBD data.

The principal disadvantages of stereo vision are that it could have difficulties with uniformly coloured elements or bad illuminated scenes because the correspondence step could give errors. Another disadvantage is that computing the correspondence between blocks involves complex and computationally intensive feature extraction and matching algorithms, resulting on a lower frame rate. Finally, the depth accuracy is worse than using TOF. On the other hand the hardware is cheaper and it is well-suited for capturing images for intuitive presentation to humans.

The second method has also some disadvantages as to have a smaller range of action and being very sensitive to other sources of infrared light.

Kinect 2 is used in this project for all the explained advantages, and because we have an indoor scene with small infrared interferences and the camera theoretical depth range (from 0.5 to 8 meters) is in the range of the objects that are wanted to be recognised.

3.3 Used libraries

**ROS** Robot Operating System is a robot management and distributed programming framework. The main reason of using it is because the system created during this project needed to be integrated in a server that is using ROS for most part of its functionalities, including serving the Kinect 2 images and controlling a robot arm to operate with objects. All the implemented code is running as ROS nodes.

**OpenCV** It is an open source library oriented to the treatment of images. It includes a massive quantity of functions including machine learning methods, feature descriptors, graphical tools, image filtering and so on. This library was used principally to implement the machine learning stage of the system and extracting colour descriptors from images.

**PCL** Point Cloud Library, as its name points out, it is a library destined to the management of point clouds. Most part of the used algorithms are implemented in this library. Plane detection with RANSAC, different clustering algorithms
and convex hull computation are some examples of functions that it contains. It also includes some shape feature descriptors implementations.

**iai-kinect2** It is a ROS package. Used for calibrating and acquiring the *Kinect 2* cameras.

**ar_track_alvar** It is a ROS package. It provides and easy interface to obtain the position and orientation of tags placed in the scene.

**rviz** It is a ROS package. It was used to visualize the point cloud data and the obtained results during the development of the system.
4 Temporal planning

This section will talk about the project temporal planning. It was started the 22 of January of 2016 and ended the 20 of June of the same year. It contains three different stages of the project development, each one of them are explained in one of the following sections.

4.1 Project planning

The first weeks of the project were orientated to its planning, in particular to the following points.

1. Define the goals.
2. Define scope of the project.
3. Temporal planning.
4. Economic management and sustainability.

4.2 Design and implementation stage

It was in this stage where most part of the effort and times was spent. It includes the definition of the necessary steps to accomplish the task, besides thinking how is the best way to implement it, and finally its implementation. Now, the detailed steps are defined.

5. Initial system set up. Having the software installed and working properly is the basic step to start developing the project with ease.

5.1. Software preparation. All the used software it was already installed, but some packages were updated to the last version to be able to use some of the last introduced features.

5.2. Prepare cameras. Kinect 2 cameras need to be calibrated in order to obtain more precise information and synchronize the depth image with the colour one.

6. Point cloud generation. Initially two images are received, one containing the colour and the other the depth information. The two must be merged in order to obtain a point cloud.

6.1. Data acquisition. The cameras are connected to a server that obtains the images using the ROS library iai-kinect2, a node that reads the data needs to be implemented.

6.2. Point cloud generation. The two images are joined to make a 3D representation of the scene.

7. Scene segmentation. Points that are part of the background of the scene (the ones that will be ignored), are differentiated from the important points (the ones placed on a plane).
7.1. **Plane detection and limitation.** The first step is to obtain the planes coefficients of a defined number of them, these coefficients define infinite planes that must be limited. The camera is fixed in relation to the scene, so these values can be calculated only at the very beginning of the system initialization or every time the camera is moved to another position.

7.1.1. **Find plane coefficients.** *RANSAC* is used to find the coefficients of the planes. As a result of being infinite planes they can contain unwanted elements, for example walls and objects from other tables.

7.1.2. **Find plane corners.** Based on the assumption that the planes are rectangular (for example a table), in this step the four corners are found.

7.2. **Relevant points extraction.** Based on the assumption that the objects are placed on a plane, relevant points are defined as the ones that are above it and inside its limits.

7.3. **Objects definition.** The set of relevant points are gathered into objects.

8. **Objects Recognition.** An identifier is obtained from a set of points representing an object.

8.1. **Define used features.** The points cannot be directly used to classify an object. Some features need to be extracted from these points to make easier the recognition step. In this stage the used features are defined.

8.2. **Choose machine learning method.** From a set of features, an object is identified. This step is achieved using a machine learning method.

8.3. **Training of the machine learning method.** The machine learning method is fed with examples of objects to be able to classify them afterwards.

9. **Objects extraction and visualization.** The objects and their positions are provided to other programs to make use of them, the option of having a visualizer to easily see the obtained results is also a must.

**4.3 Final task**

These was oriented to making the last minor changes in project, extracting results, writing the report and preparing the presentation. Some parts of the report were written just after their implementation.

**4.4 Stages duration**

The project contains two stages that need more time to develop than other ones, segmentation and recognition. On one hand the segmentation has the difficulty to know how do the thing to achieve the goal, there are different ways and some of them are not the most adequate. On the other hand, during the recognition a lot of choices have to be made, the descriptors and machine learning method in addition to the test to know what configuration is the best. Knowing that, the following number of days for each stage has been proposed.
<table>
<thead>
<tr>
<th>Stage</th>
<th>Duration (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

And they were developed following this order.

<table>
<thead>
<tr>
<th>January</th>
<th>February</th>
<th>March</th>
<th>April</th>
<th>May</th>
<th>June</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.5 Planning deviation

During the development of the project there were some temporal deviations that were fixed by simplifying non critical tasks.

For example, in the segmentation step the initial idea was use the colour of the plane to improve the results. By doing this a more intelligent way to remove planes from a scene is obtained because it helps to avoid removing points that are part of an object with a different colour than the plane. That is not in any case a fundamental part on the segmentation step but an extra, for that reason the decision to not implement it was made when we started to run out of time.

In general, the planning was followed with quite accuracy until the recognition step. The time needed for this part was underestimated because of the existence of multiple type of feature descriptors and machine learning methods that complicated the final choice. These complications were overcome because the report was started before it was planned, so the recognition stage could be extended a little bit and the two things done at once.
5 Scope

When project was started, all the hardware was already set up and working properly, the only problem was that the Kinect 2 cameras were not calibrated, so the first step was to calibrate them to guarantee that accurate images were acquired. This was done using the package iai-kinect2.

The next step was to acquire images to start the data processing step. Two images containing the colour and depth data in OpenCV format were acquired using iai-kinect2 package, they need to be merged to a Point Cloud Libarary structure for their posterior processing.

The initial idea was to make a small preprocessing of the data before starting manipulating it. It consisted in filtering the points to remove outliers, in order to have cleaner data, finally this step was removed because it was too slow and the results were not quite better.

The next step was to segment images to obtain objects that afterwards will be recognised. Some functions for clustering and plane detection from Point Cloud Library were used to obtain the segmentation.

Once the objects are obtained the recognition step starts. Feature descriptors from the objects must be extracted, It was used OpenCV for colour features and Point Cloud Library for shape ones. The machine learning algorithm used to make predictions were implemented in the OpenCV library as well, which has most possibilities and implemented methods that the machine learning library from Point Cloud Library.

After all this steps, all the necessary data to create our desired output build with the identifier, its oriented bounding box and position of an object is already available. Data needs to be sent using a Robot Operating System message, this way a robot could use this information to manipulate objects from the scene, but this last part is not inside the project’s scope.
6 Budget

6.1 Human resources

In this section it will be explained the people needed to develop the project. There are different roles that must be done, but in this case I will be in charge of all the positions because it is an individual project.

These positions are software engineer, project manager and beta tester. A total of 101 working days are estimated to develop the whole project. Estimating an average of 6 hours of work every day, it makes a total of 606 hours.

<table>
<thead>
<tr>
<th>Role</th>
<th>Hours</th>
<th>Price per hour</th>
<th>total price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project manager</td>
<td>125</td>
<td>50€/h</td>
<td>6250€</td>
</tr>
<tr>
<td>Software engineer</td>
<td>400</td>
<td>30€/h</td>
<td>12000€</td>
</tr>
<tr>
<td>Beta tester</td>
<td>81</td>
<td>25€/h</td>
<td>2025€</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>606</td>
<td></td>
<td><strong>20275€</strong></td>
</tr>
</tbody>
</table>

6.2 Direct costs

In the project only has direct costs in concept of hardware because all the used software is open source.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Units</th>
<th>total price</th>
<th>Service life</th>
<th>Amortization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>4000€</td>
<td>1</td>
<td>4000€</td>
<td>5 years</td>
<td>800€</td>
</tr>
<tr>
<td>-Graphic card</td>
<td>2000€</td>
<td>1</td>
<td>2000€</td>
<td>5 years</td>
<td>400€</td>
</tr>
<tr>
<td>-Processor</td>
<td>300€</td>
<td>4</td>
<td>1200€</td>
<td>5 years</td>
<td>240€</td>
</tr>
<tr>
<td>-Other components</td>
<td>800€</td>
<td>1</td>
<td>800€</td>
<td>5 years</td>
<td>160€</td>
</tr>
<tr>
<td>Kinect 2</td>
<td>150€</td>
<td>2</td>
<td>300€</td>
<td>5 years</td>
<td>60€</td>
</tr>
</tbody>
</table>

6.3 Indirect costs

In the project also appear some indirect costs deviated from the direct ones, a high-performance equipment with an important power consumption was provided. After looking at different companies, a price of 0.12€/kWh was decided to be a good approximation of the real price. To calculate the total consumption of the server it was used as a reference the consumption of a similar graphic card (300W), four Xeon processors (in total 200W) in addition to a approximation of 100W for all the other computer components. It makes a total of 600W (0.6 kW), that making the supposition that it will be always turned on while the project is being developed, it makes a total of 606 hours and a total energy consumption of 363.6 kWh.

On the other hand, a Kinect 2 has an approximated consumption of 20W each one, that it makes a total of 2.424 kWh between two of them.
<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Units</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>0.12€/KWh</td>
<td>363.6 KWh</td>
<td>43.63€</td>
</tr>
<tr>
<td>Kinect 2</td>
<td>0.12€/KWh</td>
<td>2.424 KWh</td>
<td>0.29€</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>366.024 KWh</td>
<td>43.92€</td>
</tr>
</tbody>
</table>

### 6.4 Total budget

The only thing that remains to do is to add up all the previous values. It could be concluded that most part of the expenses were destined to human resources and direct costs.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Resources</td>
<td>20275€</td>
</tr>
<tr>
<td>Direct Costs</td>
<td>4300€</td>
</tr>
<tr>
<td>Indirect costs</td>
<td>43.92€</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>24618.92€</td>
</tr>
</tbody>
</table>

### 6.5 Deviation

Things went smoothly in the chapter of expenses because there was not any unforeseen expense, the project was terminated on time and no hardware problems occurred, so there was no deviation in terms of costs.
7 Methodology

A working zone with all the necessary equipment already operative was proportioned, this was very helpful during the first steps of the project. My routine consisted in going there every working day to develop the project. Weekly goals were proposed to make sure the project was advancing properly.

The entire projects was developed using the already mentioned libraries and using C++ as a programming language. This last decision was made because the used libraries where natively written in this language and was thought that it was a good idea to use it because a lot of data is needed to be processed, around half a million points for point cloud in real time. In this context C++ is perfectly adapted, it provides speed, robustness and a lot of development tools to write code.

The system was deployed to a server where existed multiple projects related to computer vision and robot management, and all the components were interconnected using Robot Operating System.

One of the most important tools used in the project was Git to do the version control together with Github and Bitbucket to backup and share the work.

For the part of plane limitation and segmentation a visualizer was used as a validation system. Real time output data showed in the visualizer was analysed to make sure that the results were good enough. It was really difficult to extract numerical results to make a posterior analysis for this part of the project because there is not an obvious way to compare two planes that are part of a bigger scene, and a most arduous task it to know what is the best one when there are a lot of different variables that take part in the results. The same happens in the segmentation step.

On the other hand, in the recognition step was easier to obtain numerical results. The first step was to create a training and a testing set of objects. All the tests where made using the training set to train the machine learning method. Then, predictions using the testing set were made to extract conclusions using the accuracy, precision, recall and F-measure as metrics together with a confusion matrix that showed the principal prediction problems of the model.

The tests started with easy goals, and its difficulty incremented progressively. First objects trained and tested in the same position were used, when good results were achieved, different positions for training and testing were tested. Finally, some tests involving changes in the illumination were made to make sure that very different illuminations do not produce awful results.

When all the parameters were tuned, a test using a third set of data (the validation set) different from the previous ones was used to extract the final results.
8 Social commitment

The constructed system do not have any impact in the social commitment, but tools that can be easily developed using its outputs as a source of information could have a lot of impact in the society. The system is dealing with kitchen containers as objects, so for example this system with a little extension that could manage a robot can ease the life of people with reduced mobility. If an user wants an object from the scene, it has only to indicate to the system its identifier, then it only needs to find the object between all the other ones and send the position to a robot manager to pick the object and give it to the user. This utility could have an important impact in the society making people with reduced mobility more self-sufficient.

It could have other applications in the industry. Knowing the position of different elements is always a valuable information, for example it could be used to avoid issues in factories. You could have the case in which having two objects side by side is dangerous, using this system you could detect cases like that and warn someone or sending a robot to move one of the objects.

In the same case than before, knowing the position of some objects could also be used to send commands to a robot to manipulate them, avoiding workers doing very repetitive tasks and obtaining a more competitive industry.
9 Sustainability

The system makes use of a high-performance server and two Kinect 2 cameras that will be turned on every time someone wants to use it. In total it has a large power consumption, but this cannot be reduced because a massive quantity of data in real time must be processed. In the economic planning section the total energy used during the development of the project has been calculated, this amount of energy is equivalent to 0.295 tons of CO2. This system has a negative impact in the environment, and there are few things that could be done to reduce it.

There are two principal options to obtain a server of these characteristics, reuse an old server and make few extensions to it or buy a new one. In the construction of a component for a computer a lot of energy and materials are consumed, but as advantage new computers tend to consume less energy with the same potency of calculus than older ones. It is in the hands of the user to choose the best option depending on the time that the system will be working.

During the development of the project an old server was reused to make the tests, and it will be used afterwards in projects of other kinds. So the ecological impact in the project was only reduced to the energy consumption of the server and Kinect 2.
10 Implemented nodes

All the implemented code is in format of ROS nodes, every one represents a process and the communication between them is made using messages.

point_cloud It receives two images at the same time from iai-kinect2 package, one containing the colour and the other the depth information. It creates a point cloud as it is explained in section 13.

segment It extracts the objects from a point cloud following the steps of section 16. It also implements the plane detection and limitation (sections 14 and 15). The results contain the point cloud of the aisled object together with its position and oriented bounding box.

recognise Implements the recognition pipeline introduced in section 17. It complements the segmentation output by adding the object identifier.

objects_training It contains different tools for adding training and testing data, training machine learning methods using different parameters and descriptors and finally extracting results.

To capture new objects the only thing that it has to be done is to put the tag in a visible area, the node itself finds the tag position\textsuperscript{1} and removes everything from the scene that is not above the tag. This functionality eases the step of adding new training, testing and validation data because a new picture is taken every time the user press enter. To obtain data from another position, the only thing to be done is start the node again to find the new position of the tag.

It also provides a menu to change the parameters of the recognition step and a way to do online tests in the same way than the training. But the most useful possibility is to compute the metrics and confusion matrix for a specific testing set, most part of the results were extracted using this functionality of the tool.

objects\_to\_rviz rviz cannot understand the system output directly, so in this node it is translated to elements that rviz could understand. To do this the objects are joined to make a point cloud that contains them all, the oriented bounding boxes are translated to primitives and the identifier of an object it is sent (if it exists).

tf\_calibration As it is explained in section 12, a transformation between different frames is computed with the intention of integrating more elements in the system.

tf\_publisher It is in charge of sending the transform between frames during the execution of the system.

\textsuperscript{1}Using ar\_track\_alvar ROS node.
11 Camera calibration

A chessboard has been used to calibrate the two cameras of the *Kinect 2*. Using it allows the calibration program to find out the distortion of a camera comparing the expected result with the obtained one. The same it is applied for the synchronization of the two images, positions could be compared one to one to know the correspondence between pixels.

11.1 Intrinsic calibration

Digital cameras are not nearly perfect, the small lens that are used to acquire the images provoke distortions to them. This distortions are different from one camera to another, and as a results a calibration step is needed to quantify and counteract them.

![Barrel Distortion and Pincushion Distortion](image)

*Figure 2: Two cases of radial distortion.*

Normally, a lent is affected by two types of distortion, tangential (provoked by a bad lent alignment) and radial (the size of the lent). The *iai-kinect2* used 3 coefficients for the first one, and two for the second one. With them, the images could be rectified to obtain better accuracy.

This step is needed for the colour and infrared cameras separately.

11.2 Extrinsic calibration

The infrared and a colour camera are translated one from each other, and sometimes they could be rotated. As a consequence does not exist a direct mapping from a pixel of one camera to the other camera.

The extrinsic calibration defines a rotation and a translation between the two cameras to obtain a direct correspondence between pixels.
12 Changing coordinate frame

By default, received point clouds are in the camera coordinate frame. This is not the preferable frame to use if the system has to be integrated with other elements such as robots because they need to operate using the same coordinate frame. For this reason a ROS node was implemented\textsuperscript{2} to easily interconnect the recognition system with other elements from the system.

This uses the \textit{ar\_track\_alvar} ROS package to obtain the position and the orientation of a tag. These information is stored and automatically published\textsuperscript{3} when the system is used.

There exist only one condition on where to put the tag, it has to lay in one of the planes that is wanted to be detected. The reason of this condition will be explained in section 14.

\footnotesize\textsuperscript{2}tf\_calibration node in the source code
\textsuperscript{3}tf\_publisher implemented node
13 Point cloud creation

The received images are already synchronised and the depth information traduced to meters because *iai-kinect2* package with a calibration already performed is being used. The camera also sends the camera matrix (Equation 1). This contains the field of view of the camera (vertical and horizontal angles) and the centre of the projection.

\[
\begin{pmatrix}
  f_x & 1 & x_0 \\
  0 & f_y & y_0 \\
  0 & 0 & 1 \\
\end{pmatrix}
\]  

(Equation 1)

The point in the 3D space must be calculated and combined with the colour from the RGB camera. Knowing that \(d_{xy}\) is the depth of the point \((x, y)\) in meters, the new coordinates are calculated as follows:

\[
x' = \frac{d_{xy} \cdot (x - x_0)}{f_x}
\]

\[
y' = \frac{d_{xy} \cdot (y - y_0)}{f_y}
\]

\[
z' = d_{xy}
\]

\[
r' = r
\]

\[
g' = g
\]

\[
b' = b
\]
14 Plane detection

The plane detection is a trickier step than the previous one. The first problem is that there is a big amount of planes in a scene, so which ones of them are selected? RANSAC\textsuperscript{4} by default selects the biggest one. And what happens if you want to find more than one plane? The only solution is to remove the points that lay in the plane from the scene and using RANSAC again. This algorithm is not a general algorithm to find \( n \) planes of a scene because it was out of the scope of the project, this strategy will only work if there are enough points to represent a plane, so with a big number of planes it will probably fail.

It is easily seen that it is not fast enough to calculate the planes every time a new point cloud arrives, for these reason the planes are calculated only one time when the system starts. This is not a problem because the camera is fixed in relation to the scene and as a result the plane positions do not change as the time goes by.

![Image showing that the obtained point cloud is not perfect, some parts that should be plane are not, for example the wall and the table.](image)

This gave fairly good results, but sometimes the algorithm can select planes that are not interesting, for example a wall instead of a table. For this reason, the obtained data in the coordinate frame calibration step is used to filter the obtained planes. The tag provides the system with its position and orientation, that means that its normal could be computed using this information. Using one of the options of the RANSAC\textsuperscript{5} implementation in the PCL library a plane with a defined normal could be found.

The parameters of the RANSAC model were determined to include all the points up to a distance of 1.5cm from the plane. This could seem to be too much distance, but the tests that have been made showed very unstable places in the table (Figure 3), so using less than 1.5cm the model coefficients did not capture these imperfections and caused bad segmentations. It can also provoke that if there are objects on the plane

\textsuperscript{4}RANSAC stands for Random sample consensus, and it is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers.

\textsuperscript{5}Point Cloud Library. SACSegmentation class
part of them could be used to calculate the plane coefficients, but the normal filtering explained in the following section avoids it. Another parameter to set is the maximum angle between the reference normal defined with the tag and a valid plane, this was fixed to $10^\circ$ to give a small margin to RANSAC.

![Image of segmentation example](image)

Figure 4: Example looking for four planes, all the points that are inliers of one plane are coloured in the same way.

An example of segmentation can be found in Figure 4, this image also shows some problems that will be faced in the next steps. Sometimes not all the points of a plane are selected, this could give us a bad limitation of the plane (explained in section 15) that will cause a bad segmentation of the scene. On the other hand it could be seen that the blue plane has selected some points of the red one, this problem appears when two parallel planes are very close. This is a difficult problem to solve, specially in extreme cases as the one that is being faced, the only thing that could be done is to tune the parameters to try to minimize this effect, and that is what it has been done.

At this point the coefficients of an infinite plane are available, and as a result it can include other things that are not the plane itself, so what has to be done is to limit the boundaries of this plane. This is not a trivial step, so the entire process will be described in the next section.
15 Plane limitation

All the computations of this step were originally made using the points of a point cloud in a aleatory instant of the execution. This gave not very good results as a consequence of the instability of the received data. To sort out this problem a mask is used, the points of 50 consecutive received point clouds that are part of the plane are accumulated into it, making it a more steady representation of the plane. Comparing Figures 4 and 5a it can be seen the advantages of using a cumulative mask instead of the points of an individual execution. In the next steps the mask is used to do the calculations.

15.1 Normal filtering

![Images of obtained results using the mask with different levels of normal filtering.]

(a) No filtering  
(b) 10 degree filtering.

(c) 15 degree filtering.  
(d) 20 degree filtering.

Figure 5: Obtained results using the mask with different levels of normal filtering.

There are points that are inliers to the infinite plane but that are not part of the plane to be detected, for example walls. They could be removed by comparing the estimated normal of a point and the normal of the calculated plane, if its difference is bigger that a defined threshold the point is discard. The normal estimation\(^6\) of a point cloud is computed using all the points inside a range of 1.5 centimetres. The threshold angle was set to fifteen degree after doing some tests (Figure 5). With ten degree there are parts of the principal table where there is a small density of points and with 20 there are small parts of the wall in the yellow plane that are selected. Whatever angle for

\(^6\)Point Cloud Library. Normal Estimation class
filtering is used of the previously proposed, it can be seen that most points on walls are successfully removed. This is important for two things, firstly it removes potential clusters built with points in a wall that are separated from the plane, and secondly it deletes points from a wall directly connected to a plane and as a result avoids obtaining limits that include part of a wall.

15.2 Project points to the plane

It is important to notice that the system is dealing with a set of points that are positioned between ±1.5 centimetres of the plane, so they are not perfectly aligned. As it was explained before this is needed because of the imperfections of a point cloud. In the next steps, angles and distances between points are going to be calculated, and not using aligned points can be a source of errors. So, before doing the next calculations a projection of the points that are part of the plane to the plane itself is performed.

15.3 Discarding unwanted clusters

After the normal filtering step most part of unwanted points were removed, but as it can be seen in Figure 5c, the green plane contains not desired points (the ones placed in the red one), this could give problems if the algorithm to find limits is run using this data. To solve it, an Euclidean Clustering inside the points of one plane is done with 5000 as minimum cluster size (the used point clouds contain half a million points) and maximum distance between two points that are part of the same cluster fixed to 2 centimetres to avoid removing a lot of them in zones with small density. From the obtained clusters it is assumed that the bigger one contains the correct points that represent the plane, this could produce bad results in cases with two or more planes with similar plane coefficients because the bigger cluster could be different from the wanted plane.

Figure 6 shows the results after selecting only the biggest cluster for each plane.

15.4 Finding plane limits

Now the plane corners have to be found. Different approaches as selecting the extremes directly using the coordinates were tried, but it was not so easy how it seemed initially. Also it is not clear that this idea could work for most of the cases, so it was not even implemented and other options were thought.

Another option is to compute the convex hull of the plane, the results is between 20 and 35 points that are part of its boundary. They are to many points to use them, in part knowing that the system is dealing with rectangular tables that can be described using four points, another reason is that in the segmentation step will be needed to differentiate between the points that are inside the table and the ones that are outside, making this step linear in the number of points describing the boundary (one check for every two consecutive points), so it is important to do a small pre-process step to reduce the number of points one time, and speed up the extraction step that is executed every time the objects of a new point cloud have to be recognised.

---

7Point Cloud Library. Euclidean Cluster Extraction class.
The first approach was to use RANSAC to find lines, as in the planes case it is an easy task, but in order to find more than one line what it must be done is to remove all the points that are part of this line and then execute the algorithm another time. This is a problem because you are deleting information of the lines to be able to find other ones. More specifically, points that are part of the corners are removed, so how other lines that start in the corners will be found if the points have been already removed? It is impossible and the algorithm ends up creating new lines that hardly describe the correct ones, and not to mention that after some test it could never find more than three lines, so this solution was discarded.

A convex hull could be thought as a polygon instead of a set of points, in this case
an algorithm to simplify polygons could be applied. A variation of the Visvalingam-Whyatt algorithm was implemented to achieve it. Instead of using the area as the metric to decide what vertices are removed, the angle was used. Another difference is that the algorithm is forced to output a polygon with a certain number of vertices (fixed to 4), this can be done because of the naïve assumption that the system is dealing with rectangular planes. More information about the implementation and the decisions made are in the annex A. The obtained result can be seen in Figure 7.

After all this work, a good description of the plane has been obtained, build with its coefficients and its corners. This is very important because allows the algorithm avoid computations for regions that are not of interest, for example, usually there will be furniture near the table that there is no need to be processed.
16 Segmentation

In the segmentation step a point cloud is received and processed to extract the objects from it. The first step is to remove the points that are not inside a region of interest that is defined as follows:

- It is placed above a plane.
- It is inside the limits of the plane.
- It is not the plane itself.

16.1 Points on a plane

A way to know if a point is above or below a plane is to calculate the signed distance between the two and then discard the points that are below the plane (negative distance). A point is assumed to be in a plane if the absolute distance between them is less than 1.5 centimetres.

**Point to plane distance** Starting from the point that the coefficients are in the Hessian Normal Form. The distance is given for the following formula:

\[ D = \hat{n} \cdot x_0 + p \]

Having the general equation of a plane \( ax + by + cz + d = 0 \), \( \hat{n} \) is the unit normal vector, \( p = \frac{d}{\sqrt{a^2 + b^2 + c^2}} \) and \( x_0 \) is the query point.

16.2 Points inside the limits of a plane

It is more difficult to know if a point is inlier to the polygon built with the limits of a plane. An orientation test is used to achieve this.

**Orientation test** The used method is defined in [1]. First of all, the function \( \text{orien3d}(a,b,c,d) \) is defined, it returns a positive (or a negative) value if \( d \) lies below (or above) the oriented plane passing through \( a, b \) and \( c \).

In geometry, a parallelepiped is a three-dimensional figure formed by six parallelograms, and a parallelootope is called the generalization of a parallelepiped in higher dimensions. One way to compute the signed volume of a \( n \)-parallelootope \( P \) in \( \mathbb{R}^n \), whose \( n+1 \) vertices are \( V_0, V_1, ..., V_n \) is the following:

\[
Vol(P) = \det([V_0 \ 1] ; [V_1 \ 1] ; \ldots ; [V_n \ 1])
\]  

(2)

Where \([V_i \ 1]\) is the row vector formed by the concatenation of \( V_i \) and 1, and \( ; \) separates each one of the rows of the matrix.
This is the same case that the one found in this project because it exists four points (three defining a plane and the point to be tested) in a three dimensional space, so by applying the equation 2 the following matrix is obtained.

\[
\text{Vol}(a, b, c, d) = \begin{vmatrix}
  a_x & a_y & a_z & 1 \\
  b_x & b_y & b_z & 1 \\
  c_x & c_y & c_z & 1 \\
  d_x & d_y & d_z & 1 \\
\end{vmatrix} = \begin{vmatrix}
  a_x - d_x & a_y - d_y & a_z - d_z & 0 \\
  b_x - d_x & b_y - d_y & b_z - d_z & 0 \\
  c_x - d_x & c_y - d_y & c_z - d_z & 0 \\
  d_x & d_y & d_z & 1 \\
\end{vmatrix} = \begin{vmatrix}
  a_x - d_x & a_y - d_y & a_z - d_z \\
  b_x - d_x & b_y - d_y & b_z - d_z \\
  c_x - d_x & c_y - d_y & c_z - d_z \\
\end{vmatrix} \cdot 1
\]

(3)

The equation 3 is valid because adding a scalar multiple of one row to another row does not change the value of the determinant, and the equation 4 is expressed in terms of the Laplace expansion of a determinant.

The sign of the obtained value provides the orientation of the point respect to the plane, and if the volume is zero, it means that the point lays on the plane.

Using this method, an algorithm to determine if a point is inside a polygon can be implemented. A plane is defined with its coefficients \( ax + by + cz + d = 0 \) and normal \( \vec{n} = (a, b, c) \) together with the limits of a plane as a polygon \( P = (p_0, p_1, ..., p_m, p_0) \), having \( m \) borders built with the pair of points \( (p_i, p_{i+1}) \) with \( i \in [0, m] \). So, for each one of these borders the orientation of the point \( d \) must be calculated, and if it is always the same, then \( d \) is inside \( P \). Otherwise the point is in the boundary or outside, these two cases are considered as outliers.

But there is a small detail that must be solved. The function \( \text{orien3d} \) takes \( a, b \) and \( c \) as the points that form a plane, and \( d \) as a query point. A limit of the border is defined using two points, so it remains a third point to define the plane that is manually crafted with the formula \( c = \vec{n} + a \). This way, the plane that defines the border is perpendicular to the original plane.

### 16.3 Defining objects

These calculations provide a set of points that are above a table but that are not the table. By only doing this, as can be seen in Figure 8 the obtained set of points could be easily separated using a clustering algorithm. In this same figure it could be seen that with planes one above the other, if the plane below is a little bit translated or bigger than the above one, part of the surface of this last one is treated as object.
As before, an Euclidean Clustering\textsuperscript{8} is used to gather points in objects. After some tests, a limit of 3 centimetre is used to define a new cluster from another one. It has to be kept in mind that a very small value will produce several clusters for an object, fact that is not reasonable and that will produce difficulties in the next steps of the system. On the other hand, a bigger value can produce clusters that include more than one object, fact that is not reasonable either. A limit of 3 centimetres produce clusters containing an entire object, and an assumption of 3 centimetres between objects is good. The minimum number of points of the clustering was set two 200, smaller than before because now the system is dealing with separated objects with an average size of 2000 points approximately.

The implementation of the segmentation step makes possible to have multiple planes that can be positioned in diverse ways. For instance, two planes one above the other represents no inconvenient to extract the points of the objects without repetition (for example green and blue planes in Figure 8).

Another good feature proportioned by the use of the limits allows to have two planes side by side but with small deviation in the height (green and red planes in Figure 8), if the limits had not been used, the most probable thing is that the elements in the smaller height plane were partially removed because some points of them intersect with the plane coefficient of the bigger height plane.

The two cases are present in the scene where the tests have been made, and with the addition that one plane (green one in Figure 8) is affected for the two problems.

\textsuperscript{8}Point Cloud Library. Euclidean Cluster Extraction class.
17 Object recognition

17.1 Descriptors

At this point, the objects are described as a set of points with coordinates and a colour. They cannot be used directly to recognise an object because they represent an exact representation of it that hardly ever will be repeated (due to the imprecision in the depth of the point cloud and that they could be rotated), so it has a very small power of generalization. For these reason a descriptor for each object must be calculated.

Descriptors encode interesting information into a series of numbers and act as a sort of numerical "fingerprint" that can be used to differentiate one feature from another. They must have the following properties:

- Robust. It must be a good generalization of the object. As a result, point clouds that describe the same object but with small variation must have a very similar descriptor.
- Scale invariant. A very similar descriptor must be calculated for an object with two point clouds that have different size.
- Translation invariant. A change in the position of an object cannot effect the result of its descriptor.
- Illumination invariant. The descriptor has to hardly change regardless the illumination of the scene.

Selecting a good descriptor is one of the milestones of this stage of the project because this will be the information that will be fed to the machine learning algorithm, so a bad source of information will produce bad results of the recognition stage regardless the machine learning method used. There are two types of descriptors, local and global ones, after a segmentation step, using global descriptors is the natural choose.

The first descriptors appeared in computer vision made use of the colour information (for example the colour histogram, and more recently SIFT). But after the appearance of stereoscopic systems and the Kinect, point clouds become popular and descriptors extracting information about the shape of objects appeared, in this project the two types are used. In first place, the one capturing the colour information is explained.

Colour Histogram. As its name says, a colour histogram is a feature descriptor that only works with the colour of objects. It represents the distribution of colours of a image storing the number of pixels that have each specific colour.

It can be built in any kind of colour space but the most commons are RGB and HSV. For example, in HSV a histogram is a three dimensional array containing the hue $[0, 180)$, saturation $[0, 256)$ and value of an specific colour $(0, 256)$. As it was said before, the descriptor must have to be invariant to the illumination. Using HSV this can be achieved, because the V channel encodes the brightness of a colour (that is precisely the information that must be avoid encoding). Using a histogram only with channels H and S a descriptor robust to illumination is obtained, this does not mean that is totally invariant to it.
An histogram using all the spectrum of colours provides with a high dimensionality descriptor (more exactly of 46,080 values in a HS histogram). For these reason most times a colour is divided in bins, each one containing and interval of an specific colour. This way the size of the descriptor can be reduced down to the point that is wanted depending on the precision it is needed. Reducing the precision of the descriptor also improve its illumination invariance and generalization power.

It can be easily seen that the descriptor it is not invariant to the size of the object, the same image with different resolutions will have a different total number of points, and as a result the descriptor will be different. Using percentages instead of absolute values makes it scale invariant.

The total number bins will be determined in the next section after performing some tests.

Using only colour information when also positions are provided is a wasting of information. For that reason an additional descriptor is used to capture this type of information.

CVFH It comes from Clustered Viewpoint Feature Histogram [2], it is based on the VFH [3] global descriptor, but instead of only using one descriptor for an entire object, it is segmented using a region growing algorithm based on normals to extract the different parts of it, and finally an extended VFH descriptor is computed for each one of the clusters. The clustering step also remove points with a very large curvature because these have a inclination to be more imprecise.

Let $p_c$ and $n_c$ be the centroid point and its normal of a part of an object. For each point $p_i$ with normal $n_i$ the Darboux coordinate frame is defined as follows.

$$u_i = n_c$$

$$v_i = \frac{p_i - p_c}{\|p_i - p_c\|}$$

$$w_i = u_i \times v_i$$

All the next calculations are made using this frame to make the descriptor pose invariant. For each point of a part the following features are calculated.

$$\alpha = \arccos(v_i \cdot n_i)$$

$$\phi = \arccos(u_i \cdot \frac{(p_i - p_c)}{\|p_i - p_c\|})$$

$$\theta = \arctan2(w_i \cdot n_i, u_i \cdot n_i)$$

$$SDC = \frac{(p_c - p_i)^2}{\max((p_c - p_i)^2)}$$

The final descriptor is made with 45 bin histogram for each one of the previous features plus a viewpoint component that is computed by collecting a 128 bin histogram of
the angles that the viewpoint direction makes with each normal, all these histograms appended one after another.

The fact of dividing an object in parts makes the descriptor more robust to partially hidden object data (as could be seen in Figure 9).

### 17.2 Objects sets

As it was said before, in the project is used three disjoint sets of objects. In Figure 10 it could be seen the position where the data was captured for each one of the sets. It was decided to use the most external positions for the training data with the intention of capturing the most extreme available cases in the scene. For the testing and validation sets was decided to use two trained positions and a non trained one to know the generalization power of the system.

All the objects were acquired using the implemented ROS node `objects_training`. 
17.2.1 Training set

(a) High-angle shot  
(b) Frontal shot

Figure 11: Two examples of object 13 with different pitch angles

The final training set consisted on a total of 48 photos for each object distributed in three different places of the scene (Figure 10 colour green). The used descriptors are scale, position and illumination invariant, but not rotation invariant, that’s because an object viewed from two different rotations will have 2 different point cloud representations (Figure 11), and for instance some parts could disappear.

It is important to reflect all these things to the training set to be able to predict different outcomes. In each position a set of sixteen photos were taken, each one in a different rotation of an object. This is important because the object must be predicted regardless the orientation of it, so images of all the visible parts are needed.

Although having a colour descriptor invariant to illumination, It is not known up to each point it is invariant, for that reason a small test was made to find it out. Figure 12 show the results using two different training sets, one consisted on images with a lot of illumination (Figure 12a), and another one (Figure 12b) was built with images with little, normal and a lot of illumination.

The results shown in Figure 12 seem to indicate that using more than one type of illumination could be even counter-productive. For this reason there was no special emphasis in training the models with different type of illuminations.

17.2.2 Testing set

It consisted in a total of 30 images for each object divided in the three positions previously mentioned. The photos in each one of the positions were made with different levels of illumination.

17.2.3 Validation set

As in the testing set, the photos were also taken using different levels of illumination, but with 15 images for each object divided in the thee positions.
### Table 1

<table>
<thead>
<tr>
<th></th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>One illumination</td>
<td>0.972</td>
<td>0.851</td>
<td>0.81</td>
<td>0.813</td>
</tr>
<tr>
<td>Three illuminations</td>
<td>0.968</td>
<td>0.831</td>
<td>0.793</td>
<td>0.796</td>
</tr>
</tbody>
</table>

(a) Images with a lot of illumination.  
(b) Images with different illuminations.

Figure 12: Results obtained training only with the colour descriptor, with 6 and 13 bins for S and H channel respectively. The training set consisted on 48 images.

#### 17.3 Pipeline

The recognition pipeline starts with a set of objects, each of one build using a set of points containing position and colour. After doing some computation on this data a model will be obtained, and this will be fed to a machine learning method to know what object it is.

#### 17.3.1 Object decomposition

There are different ways to recognise an object, for instance the most straightforward way is to compute a descriptor for an entire object. This option will work properly in most cases, but exists other strategies based on the way that a human being differentiate objects that provide some advantages. For example human beings know that something is a bicycle because it has two wheels, a handlebar, a chain, etc. but a bicycle with
only one wheel could be also recognised, that is because they base the recognition in
the semantic analysis of an object based on the parts of it.

This idea could be extrapolated to object recognition in computer vision, the different
parts of an object will be defined using a Region Growing clustering based on normals.
The purpose of this algorithm is to merge the points that are close enough in terms of
a smoothness constraint and removing from the object points with a curvature bigger
than a threshold, that last step is necessary because this type of points usually have
imperfections. An example of this type of clustering could be seen in Figure 13.

Figure 13: Original figure (left) and its region growing segmentation (right)

Following the previous description, an object \( O_i \) is seen as a list of \( M_i \) parts \( (p_{i,0}, p_{i,1}, \ldots, p_{i,M_i}) \). Having \( CH_{i,j} \) and \( CVFH_{i,j} \) with \( j \in [0, M_i] \) as the colour histogram and the \( CVFH \) descriptor respectively of a part \( p_{i,j} \) of \( O_i \). The descriptor of a part is defined as follows:

\[
d_{ij} = < CVFH_{ij}, CH_{ij} >
\]

17.3.2 Colour descriptor

The first step is to study the behaviour of the colour histogram. Figure 14 shows the
obtained results changing the number of elements per bin for the separated channels S
and H.

After analysing the plots, it could be concluded than channel H gives more information
about an object than the S one. Two good possibilities of elements per bin in the H
case are around \( \log_{10}(22) \approx 1.34 \) and \( \log_{10}(13) \approx 1.11 \). The result of the first one is a
bit worse (around 1%) but it uses the around the half of the bins than the second, at
a first sight it could seem a good option to to have a little worse results in exchange
of a smaller dimensionality. For the S channel is easily seen that a good number of
elements per bin could be \( \log_{10}(42) \approx 1.62 \), in this case apart from being the best result
it has smaller dimensionality. One of the possible reasons of why using a lot of bins
give bad results is because a small training set with a small fixed number of clusters
are being used, for this reason if the granularity of the colour is too small it produces
an over-fitting of the model.

After analysing the behaviour of each separated channel it is time to find out the performance using the two channels at the same time with the parameters selected in

---

9 Point Cloud Library. RegionGrowing class
10 OpenCV. calcHist function
11 Point Cloud Library. CVFHEstimation class
Figure 14: Results obtained using only one channel of the colour histogram for the training and prediction, where the x axis represents the number of elements per bin. For each value of x the average of ten executions has been calculated. In the bag of features step, the number of clusters was fixed to ten per object.

The previous step. Table 1 shows the comparative using only S channel or using the two channels at once.

The best result is achieved using \(\left\lfloor \frac{256}{42} \right\rfloor = 6\) bins for saturation channel and \(\left\lfloor \frac{180}{13} \right\rfloor = 13\) for hue channel, so these are the selected parameters for the colour descriptor. The resulting colour histogram has 78 bins which is a small value taking into account the obtained results. The confusion matrix using the named parameters can be seen in
<table>
<thead>
<tr>
<th>S</th>
<th>H</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0.976</td>
<td>0.868</td>
<td>0.845</td>
<td>0.834</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0.979</td>
<td>0.887</td>
<td>0.861</td>
<td>0.865</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>0.978</td>
<td>0.881</td>
<td>0.857</td>
<td>0.859</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: It shows the comparison between the obtained results using the parameters defined as good ones. S and H is shown in elements per bin.

Figure 15, the most part of the objects have a good behaviour but there some of them that have very bad results. For example object 12 has almost the same probability of being correctly classified that being misclassified as object 4, the same happens with the object 7. The two cases have in common that the colour could be easily misclassified but the shape is different. The next step is to use the shape besides to the colour to try to improve the results. Annex B contains images about the objects.

![Confusion matrix](image)

Figure 15: Confusion matrix obtained using a colour descriptor with 13 bins for hue channel and 6 for saturation one.

The colour descriptor is normalized using the total number of pixels.

### 17.3.3 Shape descriptor

The used CVFH normalizes the histogram using the total number of points of object point cloud with the propriety that the sum of the 308 values of the histogram is equal to 500. The length of the descriptor cannot be changed because is defined by their creators and the second one for the used implementation.

The results using only the shape descriptor are shown in Figure 16. Compared with the colour descriptor, it gives more uniform results but with worse metrics.

### 17.3.4 Joining the two descriptors

Now that the shape and colour descriptors are defined individually it is time to find the best way to join them. The only parameter available for tuning in this step is the weight relation between the shape descriptor and the colour one. As a result of having a fixed shape descriptor, the only thing that could be done is to change the total sum of the colour descriptor. A test (Figure 17) using different weights for the colour descriptor has been made. After seeing that the weight it is not determinant
### 17.3.5 Bag of Features

The final descriptor is computed using a Bag of Features scheme. First of all the used vocabulary needs to be defined, to do this a K-Means clustering algorithm with all the parts of all the objects of the training set, then the vocabulary $V$ is defined as...
Accuracy | Precision | Recall | F-measure
--- | --- | --- | ---
0.985 | 0.906 | 0.901 | 0.9

Figure 18: Confusion matrix and result metrics obtained after joining shape and colour descriptors using the same weight.

(v₀, v₁, ..., vᵦ) where vᵢ is the centroid of the cluster i. The number of clusters is fixed to ten per each different object, this is the same value that was used in the previous test.

The equation 5 shows how the final descriptor is defined.

\[ D_i = \langle f_0, f_1, ..., f_K \rangle, \text{ where } f_k = \sum_{j=0}^{M_i} \text{same}(v_k, d_{ij}) \] (5)

The function same(vⱼ, dᵢⱼ) returns 1 if vⱼ is the most similar feature of all the vocabulary to dᵢⱼ and 0 otherwise. Dᵢ has the following property \( \sum_{j=0}^{K} f_j = M_i \).

17.3.6 Descriptor matching

The comparison between descriptors was made using a Brute Force matcher and Manhattan distance, that for a given descriptor it returns the vocabulary element similar to it. As its name suggest, it compares the given descriptor with all the features of the vocabulary. Taking into account that the size of the vocabulary increases linearly in the number of different objects, but not in the number of taken pictures for the training set, what a first sight could seem a bad election it is not in this case because the system is dealing with a limited number of 13 objects. In case of wanting to increase this number a lot, then another matcher must be used, for instance FLANN should be a good election.

By doing this, it is achieved the goal of describing an object as a set of parts. This makes the model more robust to occlusions or strange behaviours, because if one of the parts of an object is damaged or hidden, the object could be guessed correctly if the other parts are descriptive enough. This is a powerful feature, because in a normal scene sometimes one object can hid another one or a person can catch an object to move it from one place to another. So, being able to produce some good results in cases like these is a good thing.
17.3.7 Prediction

The final descriptor $d_i$ of an object $i$ is fed to a machine learning algorithm to obtain the name of the object, a Support Vector Machine together with RBF kernel were selected to achieve this. This method has several parameters that need to be tuned ($C$ and $\gamma$ in RBF case).

Intuitively, the $\gamma$ parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. The $\gamma$ parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.

The $C$ parameter trades off misclassification of training examples against simplicity of the decision surface. A low $C$ makes the decision surface smooth, while a high $C$ aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors.

The OpenCV function trainAuto was used to tune this two parameters. It creates a grid (Table 2) for each one them, and using k-Fold cross validation with the training data it selects the parameters that get better results. A total number of 5 folds was used.

<table>
<thead>
<tr>
<th>$C$</th>
<th>0.1</th>
<th>0.5</th>
<th>2.5</th>
<th>12.5</th>
<th>62.5</th>
<th>312.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>0.00001</td>
<td>0.0015</td>
<td>0.00225</td>
<td>0.03375</td>
<td>0.50625</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Used parameter values in RBF kernel.

In the end, $C$ was set to 62.5 and $\gamma$ to 0.03375.

Figure 19 shows an example of output after tuning all the parameters. It contains the oriented bounding boxes of the objects and the predicted identifier. In this example the unique error was made classifying the object 4.

![Figure 19: Results obtained after the recognition. The numbers on the objects are the predictions made by the system.](image-url)
18 Results

In this section some tests were made to quantify the final error of the model (validation test), but some other tests (generalization test, occlusions test and grasping test) were made to know the robustness of the system.

18.1 Validation test

The first test involves the validation set and the results are showed in Figure 20. The resultant metrics and the confusion matrix are similar to the ones obtained with the testing set. This means that the parameters were correctly tuned and without over-fitting.

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.986</td>
<td>0.931</td>
<td>0.908</td>
<td>0.912</td>
</tr>
</tbody>
</table>

![Figure 20: Results obtained with the validation set.](image)

18.2 Generalization test

![Figure 21: The left image shows the obtained results predicting a variation of the object 2 (right image).](image)

A test to show the power of generalization of the system it has been made. In this, the general idea was to predict an object that is a variation of a trained object. The object
in the Figure 21b was selected, it is a smaller and without label version of object 2. The object has two following principal problems.

In one hand, the shape is similar but with a different size, theoretically this should not make the prediction a lot more difficult because the descriptor that captures this type of information (the shape descriptor) is invariant to size. On the other hand, the colour descriptor will be slightly different because it will not contain the colours of the label.

It is important to notice that this object was not used in the training step. As it can be seen if figure 21a the object has been perfectly classified.

### 18.3 Occlusions test

During the introduction it was presented the idea of making the system robust to occlusions. The first step was to select the correct descriptors to make it possible, now it is time to try the robustness of the system with objects partially occluded. The test consisted in taking three images for every object, each one of them with a different orientation of the object. Another object was placed between the one that was tested and the camera, making the lower part of it to disappear, and resulting in an occlusion similar to a real one. As was expected the results (Figure 22) are worse than using non occluded objects but they are not bad at all, some of them are perfectly classified while other ones present more problems.

![Figure 22: Results obtained with objects partially occluded. It has been made three tests for object.](image)

### 18.4 Grasping test

It was decided to go further with the tests and know what happens if somebody is grasping and object while the photo is taken. Take into account that this test is even more difficult than the previous one, while in the first one a part of an object disappeared or was reduced, now it is introduced noise in form of new parts (the ones representing a hand) that were not previously trained. So let’s see how the system reacts to that type of data.

As before, a test using three images for object was made. This time the images were taken while the object was grasped by the lower, middle and higher part. One important fact is that the hand was classified as object 4 when it was alone. Figure 23 shows
that the system has the ability to predict some objects correctly, but has an obvious tendency to predict them as object 4. This could be explained because an extra part pointing out that it is object 4 (the hand) is added the descriptor, and sometimes this impact is big enough to produce a bad prediction.

Figure 23: Results obtained while the objects were grasped. It has been made three tests for object.
19 Future work

Until now it was explained all the things done in this project, but there are some extensions that could be done to improve it and new functionalities to add, here some of them will be explained.

**Invariance to drawers position.** The scene used for the project has different sliding drawers. During the project they were used as fixed ones, so after the plane detection and limitation step they could not be moved because it provoked problems to the system. An extension of this project is presented to solve this problem.

With the current version of the project, the planes coefficients and their limits are calculated only once. Instead of doing this, the mask used for the calculation of the planes could be updated every time a new image arrives and limits calculated periodically. This could be done because the planes coefficients do not change, only the limits change.

**Provide drawers positions.** An other extension based on the previously explained is to provide the system with the position of the drawers. A very valuable information for robot trajectory planning.

If the previous extension is implemented, the only thing that has to be done is to calculate the centroid of a plane using its limits. The position of this centroid indicates the position of the drawer.

**Dynamic number of limits in a plane.** The project has the naïve assumption that the number of limits of a plane is always four. This could not give good results in the real world. The idea is to use a number of limits depending on the shape of the plane. One idea is to define a maximum error between the original plane and the simplified, and use the minimum number of limits that provides with an polygon with an error less than the threshold. It has to be kept in mind that the number of points of a limit effects directly the total time of the segmentation step, for that reason a small number of them should be used. It is recommended to set a maximum number of points.

**Integration with a robot.** A robot could be provided with the information extracted from this system in order to manipulate objects from the scene.

**Ignoring not learned objects** The system is not capable of discarding objects obtained during the segmentation step. Using a learning model with the ability to provide a level of confidence of a result could be used to refine the outputs of the system, and only return the ones that are really likely to be correct.
20 Conclusions

Starting with the general conclusions about the field of computer vision and machine learning, they are fields with a wide range of possibilities and opportunities, thing that not always makes easy to know what methods to use. For example in the specific case of the descriptors, use only a colour one? a shape one? or both? and finally, how they are joined if more than one is used? All this decisions must be made to advance in the project, but not always is easy make up your mind, and neither clear why an option is better than another one. Another example is the path done during the segmentation step and the recognition one, the project was started with general idea of what and how to do the things, but it was impossible to predict the exact steps for these two processes. Some elements were added after seeing the weaknesses to end up with a robust system. Another thing related with this is that a lot of steps are needed to make a recognition system, some of them that are non trivial. It is also seen that a plenty of algorithms of different fields of study are needed to achieve this, including geometrical ones, algorithms based on treatment of images and point clouds and machine learning methods.

All these things made the problem to be really hard to solve, there is no perfect solution to solve the segmentation and recognition step, for these reason some assumptions were needed, and sometimes the only thing that could be done was minimizing the probabilities of error.

Talking about the project itself, the objectives of the project were achieved. The system is able to recognise objects placed on a table, the only thing that must be made is start the system and it alone is capable of detecting the available planes and segmenting the objects. Moreover, the initial idea of detecting the objects of only one plane was extended to more than one with the possibility of some of them being one above the other. Tools to ease the work of training and the visualization of the results were developed. The project was finished using the time initially planned, some delays were faced but could be overcome without major problems. The same has happened for the economical planning.

The conclusions about sustainability and social commitment are that this system does not contributes directly to the society, but it makes available a tool that could be used to easily make applications with a big impact in the society, so in the future it could have social impact. About the sustainability, it makes use of a big amount of resources without having direct impact to the sustainability. So in this case it could be said that it is not sustainable.
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example of the background subtraction method.</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Two cases of radial distortion.</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Image showing that the obtained point cloud is not perfect, some parts that should be plane are not, for example the wall and the table.</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>Example looking for four planes, all the points that are inliers of one plane are coloured in the same way.</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Obtained results using the mask with different levels of normal filtering.</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Points after using the bigger cluster as a representative for a plane.</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>The spheres represent the points of the convex hull, and the lines the limits of the plane after the polygon simplification step.</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>Example of how it is seen the point cloud after the segmentation step.</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>CVFH descriptor for two similar images.</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>It shows the positions where the sets were trained. Training set (green), testing and validation sets (blue).</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>Two examples of object 13 with different pitch angles.</td>
<td>31</td>
</tr>
<tr>
<td>12</td>
<td>Results obtained training only with the colour descriptor, with 6 and 13 bins for S and H channel respectively. The training set consisted on 48 images.</td>
<td>32</td>
</tr>
<tr>
<td>13</td>
<td>Original figure (left) and its region growing segmentation (right).</td>
<td>33</td>
</tr>
<tr>
<td>14</td>
<td>Results obtained using only one channel of the colour histogram for the training and prediction, where the x axis represents the number of elements per bin. For each value of x the average of ten executions has been calculated. In the bag of features step, the number of clusters was fixed to ten per object.</td>
<td>34</td>
</tr>
<tr>
<td>15</td>
<td>Confusion matrix obtained using a colour descriptor with 13 bins for hue channel and 6 for saturation one.</td>
<td>35</td>
</tr>
<tr>
<td>16</td>
<td>It contains the confusion matrix and the results using different metrics for the shape descriptor. As before the metrics are the result of a ten times execution and a value of cluster was fixed to ten per object.</td>
<td>36</td>
</tr>
<tr>
<td>17</td>
<td>Results obtained changing the total sum of the colour histogram.</td>
<td>36</td>
</tr>
<tr>
<td>18</td>
<td>Confusion matrix and result metrics obtained after joining shape and colour descriptors using the same weight.</td>
<td>37</td>
</tr>
<tr>
<td>19</td>
<td>Results obtained after the recognition. The numbers on the objects are the predictions made by the system.</td>
<td>38</td>
</tr>
<tr>
<td>20</td>
<td>pepe</td>
<td>39</td>
</tr>
<tr>
<td>21</td>
<td>The left image shows the obtained results predicting a variation of the object 2 (right image).</td>
<td>39</td>
</tr>
<tr>
<td>22</td>
<td>Results obtained with objects partially occluded. It has been made three tests for object.</td>
<td>40</td>
</tr>
<tr>
<td>23</td>
<td>Results obtained while the objects were grasped. It has been made three tests for object.</td>
<td>41</td>
</tr>
</tbody>
</table>
List of Tables

1. It shows the comparison between the obtained results using the parameters defined as good ones. S and H is shown in elements per bin. . . . 35
2. Used parameter values in RBF kernel. . . . . . . . . . . . . . . . . . . 38
3. It contains the time results using different versions of the code. The average of 50 segmentations have been made to extract the results . . . 47
Annex A: Implementations

Plane segmentation

Basic version

This is the implementation of the segmentation step of the section 16. cloud contains the point cloud structure, mask contains the accumulated mask of planes, where -1 means that a point is part of a plane, 0 that is background and 1 is part of an object. planeCoef contains the coefficients of the plane and planeLimits its limits.

The mask is initialized with all values to 0.

```cpp
1 // For each point in the pointcloud
2 for(size_t j = 0; j < cloud->points.size(); j++) {
3     if(isnan(cloud->points[j].x) continue;
4     if (isInlier(cloud, j , planeLimits, planeCoef)) {
5         Eigen::Vector4f pt(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z, 1);
6         float distance = distance(planeCoef, pt);
7         if (fabsf(distance) <= 0.015) {
8             mask[j] = -1;
9         } else if (mask[j] == 0 and distance < 0.0){
10             mask[j] = 1;
11         }
12     }
13 }
```

This code is sequentially executed for each one of the planes that must be segmented.

Line 4 avoids processing points that are invalid (coordinates set to NaN).

In line 6 it is checked if the the point is inside the limits of the plane, in this case if it is also part of the plane the mask is marked properly (lines 10 and 11). Otherwise it could be above or below the plane, if its above (distance < 0.0) and the mask is marked as a background, the mask is set to be a part of an object. By doing this planes that are above the currently treated are not selected as objects.

Optimizations

This is a critical part of the system, it is executed every time a new point cloud arrives, so it is important to optimize it.

An easily seen optimization is to reorder the operators to firstly evaluate the quicker ones. The evaluation of if a point was previously marked as a plane could be moved to the start of the code, this way it is avoided doing a lot of computations for sometimes do nothing when it is tested if the point is above the plane or not.
<table>
<thead>
<tr>
<th>Version</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>180</td>
</tr>
<tr>
<td>Optimized</td>
<td>173</td>
</tr>
<tr>
<td>Parallel (2 threads)</td>
<td>112</td>
</tr>
<tr>
<td>Parallel (4 threads)</td>
<td>68</td>
</tr>
<tr>
<td>Parallel (6 threads)</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 3: It contains the time results using different versions of the code. The average of 50 segmentations have been made to extract the results.

It is computationally less expensive to calculate a distance between a plane and a point that knowing if it is inside the limits, remember that for this last calculation an orientation test for each two points that define the boundary must be calculated, so this operation is linear in terms of the boundary points, while the distance is constant.

```cpp
for(size_t j = 0; j < cloud->points.size(); j++) {
    if(isnan(cloud->points[j].x) or mask[j] == -1) continue;
    Eigen::Vector4f pt(cloud->points[j].x, cloud->points[j].y, cloud->points[j].z, 1);
    float distance = distance(planeCoef, pt);
    if (distance >= -0.02) {
        if (isInlier(cloud, j, planeLimits, coef)) {
            if (distance <= 0.02) {
                mask[j] = -1;
            } else if (mask[j] == 0) {
                mask[j] = 1;
            }
        } else if (mask[j] == 0) {
            mask[j] = 1;
        }
    }
}
```

Making it parallel

The operations made are not dependent on other positions of the point cloud, so its parallelization is straightforward. OpenMP was used to achieve this, the only thing that has to be made is putting the following line of code just before the for to execute it using 4 threads.

```cpp
#pragma omp parallel for firstprivate(planeCoef, planeLimits) shared(cloud, mask) num_threads(4)
```

Results

Table 3 shows the times obtained. It must be taken into account that this segmentation step is executed one time for each plane to segment, that with the scene used in the project is fixed to four. Finally it was decided to use the parallel version with 4 threads because it gives a good performance with not a big number of threads.
Polygon Simplification

The general idea of the algorithm is to start with the complete polygon and at each step remove a vertex from it. The variable $c$ is defined as an aleatory point of the polygon, being $p$ and $n$ its previous and next neighbours in clockwise order, and the area of the point $c$ as the area between $p$, $c$ and $n$. The original algorithm removed the point with minor area of the polygon.

With the naïve assumption that the planes are rectangular, this did not give the expected results, the system is dealing with convex polygons and it is wanted to end up with points that are at the extremes (that have smaller angle between them and their neighbours), condition that could not be accomplished if the area is used because the distance between points obtains a lot of importance. So it was decided to remove the points with bigger angle instead of points with smaller area.

The algorithm is built in a iterative pattern. At each step removed the point $c$ with bigger angle, after this step the angles and neighbours of $p$ and $n$ needed to be updated and $c$ added to a stack to keep track the order of the extracted points. This was done until four points remain to the polygon, these are the simplification of the polygon.

The original algorithm does not define a number of vertices, instead it continues iterating until an error threshold is hit. Thanks the assumption of dealing with rectangular planes, the total number of returned vertices could be fixed.
Annex B: Used objects
References


