Universitat Politècnica de Catalunya

Enginyeria Informàtica:

Computació

Treball final de grau

Objects recognition from their volumetric data for an autonomous kitchen

Lara Solà Gallego

Director:
Joan Aranda López

Curs Acadèmic 2015/16

Juny 2016
Contents

List of Figures .......................... 5
List of Tables ............................ 7
Frequently Used Acronyms ............... 8
Resumen .................................. 11
Summary .................................. 13

1 Introduction ............................ 15

2 Scope of the project ..................... 17
  2.1 Objectives ................................ 17
  2.2 Scope .................................. 18
  2.3 Stakeholders ........................... 18
      2.3.1 Project manager, software analyst, designer, programmer, tester
            and intern ................................ 19
      2.3.2 Project director ...................... 19
      2.3.3 Users ................................ 19

3 Methodology ............................. 21
  3.1 Methodology and rigor ............... 21
      3.1.1 Development methods .................. 21
      3.1.2 Tracking tools ........................ 22
      3.1.3 Validation methodology ............... 22
  3.2 Obstacles and risks ................... 22
3.2.1 Unable to access the real set up of the automated kitchen . . . . . . 23
3.2.2 Limited processing capabilities . . . . . . . . . . . . . . . . . . . . 23
3.2.3 Carrying all the roles of a project . . . . . . . . . . . . . . . . . . . 23
3.2.4 Use of third party libraries: OpenCV . . . . . . . . . . . . . . . . . 23
3.2.5 Use of third party services for version control . . . . . . . . . . . . 24

4 State of the art 25
4.1 Object recognition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
4.2 OpenCV library . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
4.3 Collecting data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
4.4 Contribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

5 Temporal planning 29
5.1 Tasks overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
5.2 Description of tasks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
5.2.1 Project planning . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
5.2.2 Project execution . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
5.3 Effort distribution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
5.4 Resources used . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
5.4.1 Hardware resources . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
5.4.2 Software resources . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
5.5 Gantt chart . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
5.6 Action plan . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

6 Design and implementation 39
6.1 Learning the feature responses dictionary . . . . . . . . . . . . . . . . . . 41
6.1.1 Pre-processing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
6.1.2 Unsupervised learning . . . . . . . . . . . . . . . . . . . . . . . . . . 43
6.2 Feature extraction and classification . . . . . . . . . . . . . . . . . . . . . 44
6.2.1 Interest point detection . . . . . . . . . . . . . . . . . . . . . . . . . . 44
6.2.2 Convolutional k-means descriptor extraction . . . . . . . . . . . . . 45
6.2.3 Classification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

7 Tests and results 47
7.1 Experiments set-up . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
7.2 Choosing the number of $k$ centroids to learn .......................... 49
  7.2.1 Category classification ........................................ 49
  7.2.2 Instance classification ...................................... 50
7.3 Influence of PCA whitening and bootstrapping ..................... 52

8 Budget and sustainability ........................................ 57
  8.1 Budget estimation .................................................... 57
    8.1.1 Human resources costs ....................................... 57
    8.1.2 Material resources costs ..................................... 58
    8.1.3 Total budget .................................................. 59
  8.2 Budget control ...................................................... 60
  8.3 Sustainability ....................................................... 60
    8.3.1 Economic sustainability ....................................... 60
    8.3.2 Social sustainability .......................................... 61
    8.3.3 Environmental sustainability ................................ 61
    8.3.4 Sustainability score ........................................... 62

9 Conclusions ................................................................ 63
  9.1 Future work ........................................................... 64

Bibliography .................................................................. 65
List of Figures

5.1 Gantt diagram ............................................................ 36
6.1 Steps to learn a feature representation .............................. 40
6.2 Steps for feature extraction and classification .................... 40
6.3 Unsupervised learning patches acquisition ......................... 42
7.1 Computation time .......................................................... 50
7.2 Category confusion matrices for 600 and 1200 centroids .......... 51
7.3 Category precision-recall curves for 600 and 1200 centroids ..... 51
7.4 Category evolution of metrics ........................................... 52
7.5 Instance confusion matrices for 600 and 1200 centroids .......... 52
7.6 Instance precision-recall curves for 600 and 1200 centroids ..... 53
7.7 Instance evolution of metrics ............................................ 53
7.8 Confusion matrices for k-means, PCA Whitening and bootstrapping .... 54
7.9 Precision recall curves for k-means, PCA Whitening and bootstrapping .... 55
List of Tables

5.1 Time totals ................................................................. 33
5.2 Times for the task: Project execution ................................. 34
5.3 Times for the task: Learn features responses ......................... 34
5.4 Times for the task: Recognition evaluation .............................. 35
7.1 Pipelines results .......................................................... 55
8.1 Total human resources costs ............................................. 58
8.2 Hardware resources costs ............................................... 58
8.3 Indirect resources costs ............................................... 59
8.4 Total cost .................................................................. 59
8.5 Sustainability score ....................................................... 62
Frequently Used Acronyms

**OpenCV**  Open Source Computer Vision Library

**BoW**  Bag of Words

**API**  Application Programming Interface

**PCA**  Principal Component Analysis

**SVM**  Support Vector Machines

**OCR**  Optical Character Recognition

**ROS**  Robot Operating System

**ROI**  Region of Interest
Resumen

El problema en torno al reconocimiento de objetos, cuyas formas y colores difieren entre sí, todavía hoy permanece abierto. Conseguir que dichos objetos sean correctamente reconocidos, a pesar de las múltiples perspectivas, iluminaciones, posiciones y de la ocultación parcial por interposición de otros elementos en su contexto inmediato, es una ardua tarea.

El presente trabajo se inscribe en un proyecto en curso que pretende automatizar una cocina robótica otorgándole la capacidad tanto de tomar decisiones como de realizar acciones de forma autónoma. Para la culminación de este objetivo, el reconocimiento de objetos juega un papel clave, ya que permite al sistema analizar aquello que pueda estar ocurriendo en una habitación determinada y planear una respuesta en consonancia.

Hasta hace muy poco, llevar a cabo esta empresa requería unos medios tan costosos que su investigación quedaba circunscrita a aquellos pocos que podían permitírsela. Afortunadamente, con la llegada de los sensores y artefactos RGB-D (por ejemplo, Microsoft Kinect), ahora podemos recolectar, a un precio asequible, datos 3D que incluyen, además, un estrato extra de información con datos de profundidad para cada píxel.

En el decurso de estas páginas vamos a centrarnos en resolver el problema relacionado con el reconocimiento de objetos en un entorno de cocina, describiendo cada objeto a partir de su color y volumetría.
Summary

The problem of natural objects recognition with different shapes and colors is still an open one, objects can appear in different viewpoints, lighting, rotated, cluttered, occluded,... making their recognition a difficult task.

This work is part of an ongoing project to automate a robotic kitchen by providing it with the capability to autonomously make decisions and take actions. To that end, object recognition plays a key role by allowing the system to analyze what might be happening in the room and plan its response.

Until recently, attempting this endeavor would have required costly equipment which restricted research to the few who could afford it. Thankfully, with the arrival of cost reduced RGB-D sensors and devices (e.g., Microsoft Kinect) we are now able to inexpensively gather 3D data with an extra layer of information containing depth data for each pixel.

In this work we will focus on solving the object recognition problem in a kitchen environment by using depth and color to describe an object.
Chapter 1

Introduction

Object recognition is a problem which appeals to researchers from machine learning, computer vision and robotics. It has been widely studied resulting in a variety of methods, applications and standardized benchmark problems. Although the performance in these benchmarks has improved over the years, object recognition of common items in images of real-world scenes is still an open research problem.

The problem lies mainly in the fact that objects can appear in variable view points, sizes and scales, different lighting conditions and shadows, translated or even rotated. It aggravates when they are also partially occluded or in cluttered scenes.

Existing object recognition methods can be divided into two main categories: feature-matching and learning-based approaches.

Commonly in a feature-matching approach, feature descriptors, which can be defined as transformations of raw vision data to a representation, are extracted from the training models and the test scene. Next, a descriptor matching technique is used to compute feature correspondences between the model and the test scene. These correspondences are then fed to a correspondence grouping algorithm to filter outliers and generate a consensus for a specific object proposal in the scene. These methods perform well in the presence of partial occlusions and clutter but they are computationally expensive and their recognition time and complexity scales linearly with respect to the number of training models in the database. They are also limited to the recognition of objects whose models are already available in a database and are unable to generalize to unseen objects.
Learning methods are a set of techniques that learn features that can be effectively exploited by machine learning techniques (e.g., Support Vector Machines (SVM)) to perform the object recognition. Typical methods to obtain such transformations are: deep convolutional neural networks [25], k-means based Bag of Words (BoW) [16] and hierarchical matching kernels [14]. While the learning itself takes a long time, once the model has been computed, object recognition is fast. It should be noted that this technique also allows the inclusion of objects whose instances were not present in the learning process, as long as they belong to one of the trained categories.

Object recognition can be broadly used in factory and office automation through the creation of Optical Character Recognition (OCR) systems, assembly-line industrial inspection systems, as well as chip defect identification systems. In our project the context for the recognition system will be a robotic kitchen which has been built to help disabled people. Some of the tasks to perform would be storing food in the correct shelves and cupboards, bringing items to people which might be of use to them, helping them with mobility disabilities... Therefore, knowing which objects are in the scene and where they are is of high importance to achieve such tasks. To help in these tasks the kitchen is equipped with a mobile robot and two static color and depth (RGB-D) cameras.

Recently, cheap RGB-D sensors, like the Microsoft Kinect, have been developed, which are capable of providing depth information for each pixel. This has led to the availability of richer image data that might result in an increase of recognition performance.

This project deals with the object recognition problem at the classification level. In other words, we will look for good feature representation of objects so we can later label them. To accomplish that we will make use of the depth information layer paired with the color.

It should also be noted that this work is designed to be extended and embedded into more complex systems, for instance, it could be integrated into Robot Operating System (ROS) [9], a framework designed to empower robotics researchers and developers throughout the world.
Chapter 2

Scope of the project

2.1 Objectives

In this project we aim to develop a system which is able to correctly classify objects under similar lighting conditions; it is designed to be used in a controlled environment with almost no light variations. In order to achieve our goal the system needs to fulfill the following requirements.

**Time**  The system has to be efficient: it needs to analyze its environment and provide results almost instantly so that decisions can be made in real time. Consider the following example: the system recognizes the user preparing a salad, it notices she is missing one of the ingredients and it hands it to her - ideally while she is still preparing it.

**Limited processing**  The system must be able to classify objects in a timely fashion while using low resources. It should be able to recognize the objects in real time on an average laptop.

**Precision**  There should be a lower ratio of false positives than false negatives (for further information refer to chapter [7]). It is of paramount importance to have the certainty that what we think is a given object really is that object. Mistaking an object for another could lead to further mistakes in tasks like identifying the activity being performed by
a person. For instance, going back to our salad example, the system should be able to recognize a tomato and hand it to the user, recognizing properly the tomato is paramount to being able to hold it.

2.2 Scope

We want to be able to identify objects in a controlled environment, provided that the system has previously been trained on recognizing them. In order to solve this problem we need to pay special attention to the fact that the system which is going to use our solution will be working in real time, meaning efficiency is of utmost importance.

The first step in solving the identification problem is gathering images from Microsoft Kinect sensors to conform our data set. These cameras are also placed in our controlled environment kitchen and allow the robot to see while it moves around its surrounding space.

The next step is to load these 3D images on the system so that we can classify them and extract their features.

The images are going to be in color-depth format: each object will be represented by 4-channel images (RGB-D) showing its different perspectives.

We have chosen Open Source Computer Vision Library (OpenCV) which is a computer vision and machine learning library to help us load and process all the image data. For further details on OpenCV refer to section 4.2.

2.3 Stakeholders

The development of our project involves several actors which are listed and described below.
2.3.1 Project manager, software analyst, designer, programmer, tester and intern

The author will assume the tasks performed by the project manager, software analyst, designer, programmer, tester and intern. She will create the project plan, write the documentation, perform the research, code and test.

2.3.2 Project director

The director of this project is Joan Aranda López. His role is to supervise the project in order to comply with the schedule and achieve its goals. He can also guide and help the developer.

2.3.3 Users

We regard as users the scientific community of computer vision and robotics since, as previously stated in section 1, our project itself is not a final product but a tool designed for further development and to be integrated in more complex systems, e.g., as an element of context sensitive systems.
Chapter 3

Methodology

3.1 Methodology and rigor

3.1.1 Development methods

The development of the project is going to use the C++ Open Source Computer Vision Library (OpenCV), which is compatible with multiple platforms (Windows, Linux and Mac). However, we have chosen Linux as the OS for development since the object recognition is to be embedded in a Linux environment.

We are going to use an iterative software delivery model. We will start with a basic version of the program which loads a set of images for training and performs a preprocessing on them. Afterwards, in every iteration new functionality will be added or improved.

There will be a meeting with the director of the project on each iteration. These meetings will be scheduled periodically, probably with a time lapse of a week, two at most. They will involve, for example, checking that the current goals are being reached and, if so, deciding which new functionality or improvement deliver next. In a nutshell, to set milestones that lead towards the final solution.
3.1.2 Tracking tools

The iterative process described in the previous section allows us to track in each meeting on which part of the production process we are, if we are behind or ahead of schedule and to rearrange goals at an early stage if need be.

Git [3] is going to be the version control tool used for the development of the project. In this way we will be able to document all the changes that are done to the code.

GitHub [4] is going to be the third party service that will host our source code. It uses Git for version control and it is widely used throughout the open source community.

3.1.3 Validation methodology

To verify if our solution is working as designed, we need to validate if an object has been correctly recognized. To that end, we will prepare a labeled data set, where every element has already been manually classified, and compare the output of our system with the expected solution.

On every iteration we will check the solution’s correctness by performing several tests to ascertain whether the current state of the software matches the expectations of the current state of the project. This validation will be done as well with the director’s supervision on every project status meeting. If several goals have been set on an iteration, each one of them has to be validated by itself to obtain a robust solution and move to the next round.

3.2 Obstacles and risks

In the development of the project we might face different problems. The main ones have been listed and explained in the following sections together with their corresponding mitigation strategy.
3.2.1 Unable to access the real set up of the automated kitchen

Because of unmatching timetables, access to the automated kitchen is not possible, impeding the acquisition of data from the actual environment.

Mitigation strategy: use a dataset from a publicly available online repository (see section 5.2.2).

3.2.2 Limited processing capabilities

Computer vision processes usually require a large amount of processing power (they rely on machine learning). We just need to consider the fact that only the dataset that is fed to our algorithms has a size of 4.6GB.

Mitigation strategy: use Amazon cloud computing \[1\] if we do not have enough time to perform the required simulations.

3.2.3 Carrying all the roles of a project

Simultaneously being in the role of project manager and developer can be difficult, if goals are not met the project manager can show leniency towards the developer and re-schedule goals for later stages of the project thus affecting the overall schedule.

Mitigation strategy: count on the project director to supervise scheduling and goal management.

3.2.4 Use of third party libraries: OpenCV

We will use OpenCV to load and process the image data-set because it is a well known and widely recognized computer vision library. Nonetheless, there always exists the risk of facing a steep learning curve to learn what is required and even encountering a software bug that could prevent us from finishing the project on time.

Mitigation strategy: be prepared to use one of the other open source alternatives.
3.2.5 Use of third party services for version control

We will use Git as our source control tool and GitHub as our source control service. Relying on a third party to manage our source code always introduces the possibility of downtime and of irrecoverable data loss (as slim as that possibility may be).

Mitigation strategy: perform backups of the repository in a separate server.
Chapter 4

State of the art

In the sections below we have listed the areas that our project is going to explore, giving details of each one of them and explaining how they relate to our solution.

4.1 Object recognition

As previously described, object recognition is particularly difficult because objects can appear in variable view points, sizes and scales, different lighting conditions and shadows, translated or even rotated. Moreover, it aggravates when they are also partially occluded or in cluttered scenes.

The state of the art algorithms for object recognition try to avoid those problems by extracting local feature descriptors around interest points (points that contain relevant information for the algorithm being applied).

This method is commonly used in a Bag of Words (BoW) setting and has been used in many different recognition problems. In computer vision, a bag of visual words is a vector of occurrence counts of a vocabulary of local image features.

To represent an image using BoW model, an image can be treated as a document. Similarly, 'words' in images need to be defined too. To achieve this, it usually includes the following three steps: feature detection, feature description, and dictionary generation. A general definition of the BoW model can be the 'histogram representation based on...
independent features’.

Most methods within this class make use of hand-designed descriptors based on orientation histograms, such as SIFT [23] and SURF [11]. These methods though, discard available information like color or depth. That is why different researchers have proposed the addition of other information like color histograms [10].

Because of that we have looked for alternatives that treat all available information in a unified way. That can be achieved using a machine learning approach which uses kernel descriptors [13].

Several different machine learning methods have been designed to learn low level feature responses from data: deep belief networks [20], deep auto-encoders [15] [24], sparse coding [17] and local coordinate coding [26] among others. Another interesting development on unsupervised feature learning is using standard unsupervised learning techniques like k-means if image pre-processing and feature encoding is used [16].

The aim of this project is to implement a new approach presented in Learned Feature Descriptor for Object Recognition in RGB-D data [12]. It considers a specific recognition setting in which the objects are represented using high resolution RGB-D data and propose to extract a feature histogram descriptor combining the 4 channels. To make the approach scalable to high resolution images the authors chose to extract the learned feature responses around interest points. The descriptor is built from features, which are learned via k-means clustering. This technique is called convolutional k-means descriptor.

### 4.2 OpenCV library

There are several libraries oriented to computer vision (ie. SimpleCV, Accord.NET framework,...). Amongst them the most commonly used because of their robustness and the fact that they are open source are:

- **Open Source Computer Vision Library (OpenCV)** to work with multi-channeled images RGB-D.

- **Point Cloud Library** to work with 3D point clouds.
Since we will work mainly with multi-channeled RGB-D images we have chosen the OpenCV library. It provides a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

It is used extensively in the industry, research groups and by governmental bodies [7].

4.3 Collecting data

When hearing the word Kinect, most people think of gaming. It is partially true for it has become popular thanks to the Microsoft Xbox gaming console, but it is also true that given its relative low cost it has gained popularity in the computer vision field as well [18].

Microsoft provides a SDK [6] to let developers design programs with the Windows version of Kinect, yet this SDK is of no use to us since we are working on a Linux OS.

Fortunately, OpenCV supplies other ways to obtain RGB-D images from Kinect. The OpenNI Grabber Framework is a generic interface that provides access to different devices and their drivers, file formats and other sources of data [8].

4.4 Contribution

We are going to revisit the algorithm used to obtain the convolutional k-means descriptor [12] to see if it would be a good addition to a recognition pipeline for our kitchen scenario.

Therefore, we are going to implement the convolutional k-means descriptor, tune its parameters and adjust them for our own needs. Verify that it can meet our requirements, integrate it within an object recognition pipeline and train it for further use in context sensitive systems such as the one in our kitchen scenario.
Chapter 5

Temporal planning

This section is about temporal planning and it aims to describe the tasks that are going to be executed in order to do the project, providing an action plan that summarizes the actions that have to be taken to finish the project in the desired time frame. However, we have to take into account that the planning described in this project is subject to modifications depending on the development of the project.

The project started on February 16th and its deadline is July 26th, 2016, giving an approximate duration of 4 months and a half.

5.1 Tasks overview

- Project planning
  - Scope of the project
  - Temporal planning
  - Economic management and sustainability
  - First presentation
  - State of the art
  - Specialization documentation
5.2 Description of tasks

This section describes the planned tasks required to reach the milestones and goals of our project.
5.2.1 Project planning

It is essentially about everything that is covered by the GEP course and it defines what
is going to be done and how to reach the goals of the project.

5.2.2 Project execution

The project execution covers all the tasks related with the implementation and testing of
the program that solves the recognition problem of kitchen objects by their volumetric
data.

Although in an ideal situation the project execution would be done after the planning,
some of the tasks will be performed in parallel with the project planning ones.

Each task is described in the following sections.

Initial system set up

Before starting the development of the project we need to set up the tools required to
work on it.

To develop the project we need the Open Source Computer Vision Library (OpenCV) as
the pillar of our object recognition algorithms, all the dependencies needed to use OpenCV
and Latex to generate documentation. We will install Linux and test our solution on it
since one of the application’s further uses will be to be integrated in the Robot Operating
System (ROS) [9], a flexible framework for writing robot software which only runs in
Linux.

After installing all the software, configuring it and checking that everything works as
expected, we can start learning feature responses.

Data Acquisition

Our data will come from the RGB-D Object Dataset [22], but we will need to classify and
label it properly to meet our needs.
Recognition implementation

In this stage we will have the bulk of the workforce. For each task we will have an analysis, design, implementation, tests an validation stage. Therefore we will ensure that everything is working as required.

- **Learn feature responses** A set of feature responses must be learned capturing the image structure around interest points.

- **Interest point detection** We must evaluate which algorithm is going to capture the most interesting points.

- **Descriptor extraction** Descriptor responses will be extracted around each detected interest point.

- **Train classifier** Decide which is the best classifier for our needs and train it with the dataset.

All theses tasks must be performed in cascade, the only thing that can be done after a first recognition cycle implementation are the [PCA] whitening transformations and bootstrapping.

Recognition evaluation

Once we have implemented the recognition and tuned its parameters we have to check the limitations of the method. We will perform the following tests to determine its viability:

- **Scale test** It must work with little scale variations between the object and the scene.

- **Test required image quality** It is paramount to validate the quality of the model we are developing. In order to test it several questions will be put forth, such as if high resolution is a strict requirement or if the presence of noise is greatly affecting the results of the model.

- **Test false positives vs. false negatives** We need to know if the rate of false positives is higher than the rate for false negatives.
Final result analysis

When the recognition has been validated we can analyze the results versus the ones in "Learned feature descriptor for object recognition in RGB-Data" [12].

Final report

When we have the analysis results we will prepare the delivery of the project and ensure that the documentation is correct.

Oral presentation

Once the final report is reviewed and definitive, we will prepare the final presentation.

5.3 Effort distribution

Table 5.1 shows the time spent for each main block.

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project planning</td>
<td>84</td>
</tr>
<tr>
<td>Project execution</td>
<td>505</td>
</tr>
<tr>
<td>Total</td>
<td>589</td>
</tr>
</tbody>
</table>

Table 5.1: Time totals

Table 5.2 summarizes the time spent in each of the tasks described in the previous section.

We can see in more detail the subtasks for learning features responses and making the recognition evaluation in tables 5.3 and 5.4.
<table>
<thead>
<tr>
<th>Project execution</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Time(h)</td>
</tr>
<tr>
<td>Initial system set up</td>
<td>35</td>
</tr>
<tr>
<td>Data acquisition</td>
<td>20</td>
</tr>
<tr>
<td>Learn features responses</td>
<td>174</td>
</tr>
<tr>
<td>Interest point detection</td>
<td>28</td>
</tr>
<tr>
<td>Descriptor extraction</td>
<td>52</td>
</tr>
<tr>
<td>Train classifier</td>
<td>28</td>
</tr>
<tr>
<td>Recognition evaluation</td>
<td>60</td>
</tr>
<tr>
<td>Final results analysis</td>
<td>28</td>
</tr>
<tr>
<td>Final report</td>
<td>60</td>
</tr>
<tr>
<td>Oral presentation</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>505</td>
</tr>
</tbody>
</table>

Table 5.2: Times for the task: Project execution

<table>
<thead>
<tr>
<th>Learn features responses</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Time(h)</td>
</tr>
<tr>
<td>Pre-processing</td>
<td></td>
</tr>
<tr>
<td>Standardize data</td>
<td>28</td>
</tr>
<tr>
<td><strong>PCA whitening</strong></td>
<td><strong>52</strong></td>
</tr>
<tr>
<td>Unsupervised learning</td>
<td></td>
</tr>
<tr>
<td>K-means</td>
<td>28</td>
</tr>
<tr>
<td><strong>PCA whitening</strong></td>
<td><strong>28</strong></td>
</tr>
<tr>
<td>Bootstrapping</td>
<td>38</td>
</tr>
<tr>
<td>Total</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 5.3: Times for the task: Learn features responses

5.4 Resources used

We can divide the resources that we are going to use between hardware and software.

5.4.1 Hardware resources

- PC (Intel core 2 Duo at 2.53GHz, 6GB RAM, nVidia GeForce GTX 260M): used in all tasks of the project.
- 2 Kinect units. Used in data acquisition.
### Recognition evaluation

<table>
<thead>
<tr>
<th>Task</th>
<th>Time(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale test</td>
<td>20</td>
</tr>
<tr>
<td>Test required image quality</td>
<td>20</td>
</tr>
<tr>
<td>Test false positives vs. false negatives</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 5.4: Times for the task: Recognition evaluation

#### 5.4.2 Software resources

- Ubuntu 14.04 LTS: used in all tasks.
- Open Source Computer Vision library 3.1: used in all tasks.
- Latex: to document the project.
- Git: for version control.
5.5 Gantt chart

Figure 5.1: Gantt diagram

Blue lines: start and end of the project.
Thin red line: first deadline (26/05/2016).
Thick line: final report deadline (20/06/2016)
5.6 Action plan

There has been an 11% incremental deviation respect the original planning.

Descriptor extraction, PCA whitening and bootstrapping have been more challenging than expected. Descriptor extraction took almost three times the planned time to implement, especially because while implementing, new issues arose that suggested the initial analysis was wrong. For the PCA whitening and bootstrapping the most difficult part was understanding the algorithms.

Because of this issues a re-planning has been made. Evaluation of the recognition and the final result analysis have been done while writing the final report, having to work overtime for a little more than 3 weeks.

The oral presentation has been moved after the final report deadline, and therefore the project schedule has been extended 5 days, finishing on 25/06/2016 instead of 20/06/216.
Chapter 6

Design and implementation

In this section, we describe the framework used for feature learning with the convolutional k-means descriptor.

At a high-level, the system performs the following steps to learn a feature representation (see figure 6.1):

1. Extract random patches from unlabeled training images.
2. Apply a pre-processing to the patches.
3. Learn a feature dictionary using an unsupervised learning algorithm.

Given the learned dictionary and a set of labeled training images we can then perform feature extraction and classification (see figure 6.2):

1. Detect interest points in the input image.
2. Convolutional k-means descriptor responses are extracted around each interest point.
3. Pool features together over regions of the input image to reduce the number of feature values (further explained in 6.2.3).
4. Train a linear classifier to predict the labels given the feature vectors.

We will now describe the components of this pipeline in more detail. We will mention as well whenever we make use of Open Source Computer Vision Library (OpenCV).

To implement the pipeline described in the following sections we have developed an
Application Programming Interface (API), as an extension of OpenCV, that allows to easily obtain a feature representation of an image to use alongside the desired classification algorithm.

---

**Figure 6.1:** Steps to learn a feature representation

**Figure 6.2:** Steps for feature extraction and classification
6.1 Learning the feature responses dictionary

The approach is to learn good features representations for RGB-D images using unsupervised learning [12]. As mentioned above we want to obtain a dictionary $D \in \mathbb{R}^{N \times k}$ given a set of input vectors $X = \{x^{(1)}, \ldots, x^{(p)}\}$ with $x^{(i)} \in \mathbb{R}^{N}$.

The input vectors are patches of $w \cdot w$ pixels extracted from a training set represented as column vectors.

Each pixel is represented using $d$ channels, in our case $d = 4$ since we are using RGB-D data, therefore the dimension of the input vectors is $x \in \mathbb{R}^{N}$, where $N = w \cdot w \cdot d$.

To generate the set of patches/input vectors $X$ given a set of training images we first detect interest points and extract a Region of Interest (ROI) around each interest point of $M \times M$ pixels.

For every ROI we extract $r$ random patches of size $w \cdot w$ to build the training set $X$. Figure 6.3 shows the extraction process.

Assuming we obtain a fixed number keypoints of interest points for every image, we would obtain $p = r \cdot \text{keypoints} \cdot \# \text{training images}$ patches and the size of $X$ would be $X \in \mathbb{R}^{N \times p}$.

Once the training patches $X$ are obtained we can apply a pre-processing step followed by the unsupervised learning algorithm.

6.1.1 Pre-processing

We first rescale all image patches contained in $X$ so that the final patches vectors lie in the range $[0, 1]$.

Afterwards we standardize the patches by subtracting their mean and dividing by the standard deviation to obtain zero-mean and unit-variance. This is done along each dimension of the patch.

Given a pixel $x_j^{(i)}$ where $i \in [0 \ldots p)$ and $j \in [0 \ldots N)$ the standardization of that pixel
would be

$$standardization \left( x_j^{(i)} \right) = \frac{x_j^{(i)} - \bar{x}_j}{\sigma(x_j)}$$  \hspace{1cm} (6.1)$$

In this way we obtain standardization $(X)$ and the vectors $\mathcal{M}_D$ and $\mathcal{S}_D$ which will represent the mean and standard deviation of the dictionary $D$: 
\[
X = \begin{bmatrix}
x_0^{(0)} & \ldots & x_0^{(p-1)} \\
\vdots & \ldots & \vdots \\
x_{N-1}^{(0)} & \ldots & x_{N-1}^{(p-1)} 
\end{bmatrix},
\mathcal{M}_D = \begin{bmatrix}
x_0 \\
\vdots \\
x_{N-1}
\end{bmatrix},
\mathcal{S}_D = \begin{bmatrix}
\sigma(x_0) \\
\vdots \\
\sigma(x_{N-1})
\end{bmatrix},
\]

\[
\text{standardization}(X) = \frac{X - \mathcal{M}_D}{\mathcal{S}_D} = \begin{bmatrix}
\frac{x_0^{(0)} - \bar{x}_0}{\sigma(x_0)} & \ldots & \frac{x_0^{(p-1)} - \bar{x}_0}{\sigma(x_0)} \\
\vdots & \ldots & \vdots \\
\frac{x_{N-1}^{(0)} - \bar{x}_{N-1}}{\sigma(x_{N-1})} & \ldots & \frac{x_{N-1}^{(p-1)} - \bar{x}_{N-1}}{\sigma(x_{N-1})}
\end{bmatrix}
\]

(6.2)

Afterwards, a Principal Component Analysis (PCA) whitening transformation \cite{21} is applied to the image patches to ensure that values are decorrelated and have unit variance. According to the results shown in \cite{16}, this is an important step to ensure the quality of the learned dictionary.

The PCA class in OpenCV computes the eigenvalues and eigenvectors for us but we have to extend it to be able to perform the whitening.

The transformation is computed as follows:

given \( S \), a column vector containing the eigenvalues and \( \epsilon \), a regularization parameter to avoid dividing by 0 and removing noise, the whitening vector \( w \) can be defined as:

\[
w = \frac{1}{\sqrt{S + \epsilon}}
\]

(6.3)

given \( U \), a matrix of eigenvectors and \( W \), a diagonal matrix containing the whitening values of \( w \), the whitening transformation is:

\[
\text{whiten}(X) = WU'X
\]

(6.4)

### 6.1.2 Unsupervised learning

We learn the dictionary \( \mathcal{D} \in \mathbb{R}^{N \times k} \) by clustering the extracted patches \( X \) with \( k \)-means, obtaining \( k \) centroids which will represent our feature responses.
We also try a bootstrapping learning approach to train the $k$ centroids \cite{12}. Since the data is clustered in PCA whitened space, we first cluster in the subspace spanned by the first $p$ principal components and fill the learned centroids with zeros for all other $n - p$ dimensions. These centroids are used to start the clustering procedure in the $n$ dimensional PCA whitened space.

OpenCV provides a $k$-means function but for the bootstrapping process its parameters do not allow us to start the algorithm with a given set of centroids, therefore we have modified the code to allow us to do so.

6.2 Feature extraction and classification

Once we have learned a dictionary we can start extracting the features in our labeled training images for classification.

6.2.1 Interest point detection

At first we chose SURF corners as interest points since they are reasonably fast to compute and detect interest points of high quality \cite{11}. But after running the algorithm over the entire dataset we found out that for many images it was unable to find any interest point, even after setting a very low threshold. So it rendered the algorithm useless for our setting.

On a second try, we looked for Harris corners \cite{19}. Which rendered a good result but too many interest points to process. Because of this we discard points by:

- Point quality: a threshold is applied.
- Point proximity: since we will take a patch of $M \times M$ around each interest point we believe that keeping only the best quality points that are within a minimum distance of $M/2$ would suffice.

Both Harris and SURF corners implementations are available in OpenCV.
6.2.2 Convolutional k-means descriptor extraction

For each region of interest $\text{ROI} \in \mathbb{R}^{M \times M}$ we obtain every possible sub-patch of size $w \times w$ within the $\text{ROI}$. The total number of sub-patches for a $\text{ROI}$ being $(M - w + 1)^2$. We then apply the same pre-processing steps explained in section 6.1.1 by standardizing and whitening the patches using the same $\mathcal{M}_D$, $\mathcal{S}_D$, $U$ and $W$ calculated there.

Then the feature response $f(x) \in \mathbb{R}^k$ of each $w \times w$ sub-patch $x \in \mathbb{R}^N$ in the region of interest is computed by calculating:

- The mean of the distance from the patch to all the centroids $c^t \in \mathcal{D}$, as
  \[
  \bar{z} = \frac{1}{k} \sum_{t=1}^{k} z_t, \quad \text{where } z \in \mathbb{R}^k \text{ with } z_t = \| c^t - x \|_{L_2} \quad (6.5)
  \]

- For each centroid we save the information of the mean minus the distance to the centroid, meaning that there will be a 0 for all the distances bigger than the mean value (the furthest ones).
  \[
  f_t(x) = \max (0, \bar{z} - z_t) \quad (6.6)
  \]

Therefore we obtain a vector $v = f(x) \in \mathbb{R}^k$, where in each position there is a 0 if the distance of the patch to a centroid $c^t$ is over the mean $\bar{z}$, or a value between $(0, \bar{z}]$ if the distance is inferior, note that this function maximizes the closeness to the centroid.

Finally, these vectors are pooled over 4 quadrants of the ROI to form a feature vector for classification $F \in \mathbb{R}^{4k}$. This vector will contain in every position the sum of each $v^{(t)}$ with sub-patch in the quadrant $t/k$.

6.2.3 Classification

The convolutional k-means feature mapping produces a $\# \text{keypoints} \cdot 4k$ representation image representation. Before classification, we reduce its dimensionality by pooling. We split the $F_p \in \mathbb{R}^{4k}$ where $p \in [0, \ldots, \# \text{keypoints})$ into four equal-sized quadrants, and compute the sum of the $F_p$ in each. Obtaining a reduced $4 \cdot 4k$ representation that we use for classification.
Given the pooled \((4 \cdot 4k\)-dimensional) feature vectors for each training image and label, we apply standard linear classification with Support Vector Machines (SVM), available in OpenCV.
Chapter 7

Tests and results

In this chapter, we evaluate the performance of the pipeline explained in the previous chapter on the \textit{RGB-D Object Dataset} \cite{22}.

First, we explain the set up used in the tests and some of the parameter values that will not change during the experiments.

Second, we try to determine the best number of centroids to use given the objectives in section \ref{sec:k-means}. Because we do not obtain a high improvement in accuracy or precision by setting more centroids but we do obtain a linear increase in computation time, we decide that for a real time application, our lower bound of 600 centroids should suffice.

Finally, we evaluate the incorporation to the \textit{k}-means approach of Principal Component Analysis (PCA) whitening and bootstrapping with a sub-set of our data set composed of similar objects to see clearly to which extend it helps to obtain better predictions. We conclude that given the little computation overhead and the high improvement on prediction results it is a good component to add to the pipeline.

Before evaluating the recognition algorithm we will explain some terms to better understand the results:

\textbf{True positives (TP)}: Equivalent to hits, the algorithm predicted correctly a class.

\textbf{True negatives (TN)}: Equivalent to rejection, the algorithm correctly predicted that a class X was not a class Y.

\textbf{False positives (FP)}: Equivalent to false alarm, the algorithm predicted that a class
was X when it actually was Y.

**False negatives (FN):** Equivalent to miss, the algorithm predicted that a class was not X when it actually was X.

**Accuracy:** How often the algorithm predicts correctly.

\[
Accuracy = \frac{TP + TN}{Total}
\]  

(7.1)

**Recall:** Also known as sensitivity or true positive rate (TPR), it measures the rate of hits.

\[
Recall = \frac{TP}{TP + FN}
\] 

(7.2)

**Precision:** Also known as positive predictive value (PPV), it measures how often the prediction is correct.

\[
Precision = \frac{TP}{TP + FP}
\] 

(7.3)

**F<sub>1</sub>-measure:** A measure that combines precision and recall is the harmonic mean of precision and recall:

\[
F_1\text{-measure} = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR}
\] 

(7.4)

**Confusion matrix:** Table that allows visualization of the recognition algorithm. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class. It helps in seeing if the system is mislabeling two classes.

### 7.1 Experiments set-up

The *RGB-D Object Dataset* [22] is a large scale dataset containing image sequences of 300 objects grouped in 51 categories. Images are captured with a Kinect style 3D camera that records synchronized and aligned 640 × 480 px resolution RGB and depth images. Each object is recorded from three viewing heights (pitch angles: 30°, 45° and 60°) while it rotates on a turntable resulting in approximately 150 views per object. The images are already cropped and segmented so there is no need for object detection or segmentation. Since such a huge dataset is too big given our resources (an average laptop), we make use
of a second dataset also available in the *RGB-D Object dataset* that includes the same objects but in a lower resolution (100 × 100 px approx, it varies depending on the object), we also reduce the dataset to 30 categories and 190 objects corresponding only to objects we can typically find in a kitchen. Reducing the dataset size from 80GB of data to 3.3GB. We will further sub-sample the data by taking every fifth video frame resulting in 26458 RGB-D images.

To generate train and test sets we will randomly select 6/7 of the dataset for training and 1/7 to test. It should be noted that it will also be a balanced selecting, thus ensuring that each category or instance has approximately the same number of samples.

Because of computational restrictions, we will extract 3 random patches from every Region of Interest (ROI) to train the dictionary, with side lengths $w = 6$ and $M = 16$ respectively, as indicated by [12] in every experiment.

### 7.2 Choosing the number of $k$ centroids to learn

For the following experiments we will see the influence of the $k$ centroids to learn in the recognition pipeline, the pre-processing steps will be just standardization of the data, without performing PCA whitening.

We have performed the experiments at category and instance level.

The slight overhead in computation time for an instance prediction in figure 7.1 may be explained by the increase in classes to classify by the Support Vector Machines (SVM), going from 30 categories to 190 instances. The processing of an image for classification is the same whether it is going to be used for category or instance classification, only the labeling changes.

#### 7.2.1 Category classification

At first glance, from the confusion matrices of 600 (fig. 7.2a) and 1200 (fig. 7.2b) centroids we could determine that there is not a significant improvement by increasing the number of centroids. But when plotting the precision vs recall for all the categories, we can see
that using 1200 centroids brings a significant improvement for the most difficult classes to classify (fig. 7.3).

We also obtain a 2% increase in accuracy, precision, recall and the F-measure. We can see the improvement of these measures in figure 7.4.

From figure 7.1 we can conclude that although increasing the number of centroids does improve the results for our particular dataset, the computational cost associated to this increase is too high (linear) compared to the gain. On a real time application might be better to keep the number of centroids to $k = 600$.

### 7.2.2 Instance classification

In this section we will evaluate the instance classification.

Once again from the confusion matrices of 600 and 1200 centroids (fig. 7.5) it is difficult to see the improvement. Precision-recall curves in figure 7.6 show an overall improvement, but for the critical classes.

We only obtain a 1% increase in accuracy, precision, recall and the F-measure (fig. 7.7), although compared to the category classification they are higher.

From figure 7.1 we can see the linear increase of computational time when increasing
Figure 7.2: Depicts the confusion matrices for category classification using the convolutional k-means descriptor with 600 centroids for figure (a) and 1200 centroids for figure (b).

Figure 7.3: Each color depicts the class precision-recall curve for one of the 30 categories using the convolutional k-means descriptor with 600 centroids for figure (a) and 1200 centroids for figure (b). We can observe how the precision improves for difficult to classify classes when the number of centroids increases.

the number of centroids. For the instance classification it is even clearer that the time overhead is not appropriate for a real time application. Again, \( k = 600 \) centroids seems a good choice.
Figure 7.4: Depicts the evolution of the metric results versus the increase of centroids for category classification. Accuracy, recall and the F-measure take the same values.

Figure 7.5: Depicts the confusion matrices for instance classification using the convolutional k-means descriptor with 600 centroids for figure (a) and 1200 centroids for figure (b).

### 7.3 Influence of PCA whitening and bootstrapping

In this section we analyze the influence of learning a dictionary with the PCA whitening and bootstrapping pipeline in the prediction results setting.

We set \( k = 600 \) centroids and the PCA whitening regularization parameter \( \epsilon = 0.1 \). Given our particular setup, where images and patches take values between \([-1, 1]\), this is
Figure 7.6: Each color depicts the class precision-recall curve for one of the 190 instances using the convolutional k-means descriptor with 600 centroids for figure (a) and 1200 centroids for figure (b). Again, the precision improves for difficult to classify classes when the number of centroids increases.

Figure 7.7: Depicts the evolution of the metric results versus the increase of centroids for instance classification. Accuracy and recall take the same values.

**recommended initialization for ϵ according to [16].**

To perform the bootstrapping, we keep the p components which retain 95% of the variance.

We choose a set of categories which are similar in terms of shape and color: apple, bell pepper, lemon, lime, pear, potato and tomato.

Table 7.1 shows the increase in prediction performance for PCA whitening alone (5%) and PCA whitening with bootstrapping (7%). There is also a slight increase in computation
time (around 60 ms) which might be negligible given the improvement of the results.

Figure 7.8: Depicts the confusion matrices for different pipelines to obtain the dictionary. As can be seen there is an improvement when using PCA whitening (b) or bootstrapping (c) in comparison to not using them at all (a).
Figure 7.9: Depicts the precision-recall curves for different pipelines to obtain the dictionary. As it can be seen the highest improvement occurs with bootstrapping (c).

<table>
<thead>
<tr>
<th>Dictionary obtention</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$-measure</th>
<th>Computation time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-means</td>
<td>0.79</td>
<td>0.80</td>
<td>0.79</td>
<td>0.78</td>
<td>329.23</td>
</tr>
<tr>
<td>PCA Whitening</td>
<td>0.84</td>
<td>0.84</td>
<td>0.84</td>
<td>0.83</td>
<td>358.89</td>
</tr>
<tr>
<td>Bootstrapping</td>
<td>0.86</td>
<td>0.86</td>
<td>0.86</td>
<td>0.85</td>
<td>360.22</td>
</tr>
</tbody>
</table>

Table 7.1: Pipelines results
Chapter 8

Budget and sustainability

This section is about budget and sustainability of the project. Depicts a detailed description of the project costs, describing both material and human costs, an analysis of how the different obstacles could affect our budget and an evaluation of the sustainability of the project.

8.1 Budget estimation

In this section we are going to do an estimation of the human resources and material costs.

8.1.1 Human resources costs

Table 8.2 shows the salary of each role involved in the project development, the initially estimated and recalculated costs with the modifications in 5.6 Action plan.

We re-calculate the estimation of the human resources costs from the the temporal planning (tables 5.1, 5.2, 5.3, 5.4 and figure 5.1).

There is an increment in the budget of 2110€ which correspond to an 11% of our first estimated budget. The increments are focused in the analyst and programmer roles since these are the areas where we have found more challenges.
Table 8.1: Total human resources costs

8.1.2 Material resources costs

Direct costs

To calculate the amortization we are going to take into account two factors: useful life and the project duration which is four months and a half.

Hardware resources costs Table 8.2 shows the costs of the hardware that we are going to use in the development of the project which amounts to 153€.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price (€)</th>
<th>Units</th>
<th>Useful life</th>
<th>Amortisation (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laptop</td>
<td>1000</td>
<td>1</td>
<td>4</td>
<td>127</td>
</tr>
<tr>
<td>Kinect</td>
<td>100</td>
<td>2</td>
<td>4</td>
<td>26</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>153</td>
</tr>
</tbody>
</table>

Table 8.2: Hardware resources costs

Software resources costs All the software used in the project is open source, therefore no costs will be associated to software resources.

Indirect resources cost

To calculate the electricity consumption we have taken into account the following:

- The Kinect units take up to 3W and we will use them around 20 hours.
• The laptop takes up to 150W and will be used for the overall duration of the project.

• Two light bulbs take 15W and will be used for the overall duration of the project.

The total amount for the indirect resources is 43€, the details can be seen in table 8.3. After schedule recalculation there has not been any noticeable increment in the indirect costs since it only has been extended 5 days.

<table>
<thead>
<tr>
<th>Product</th>
<th>Price (€) / Unit type</th>
<th>Units</th>
<th>Unit type</th>
<th>Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electricity</td>
<td>0.15</td>
<td>90.84</td>
<td>kWh</td>
<td>14</td>
</tr>
<tr>
<td>Paper</td>
<td>29.03</td>
<td>1</td>
<td>pack</td>
<td>29</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>43</td>
</tr>
</tbody>
</table>

Table 8.3: Indirect resources costs

The estimated cost is the same as the recalculated cost for the indirect resources costs.

8.1.3 Total budget

The total cost re-calculated from the data in tables 8.1, 8.2 and 8.3 sums up to 22046€ with an increment of an 11% from the initial estimated cost.

The increment, as foreseen in the project planning, is only in the human resources side.

Details can be seen in table 8.4.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Estimated cost (€)</th>
<th>Recalculated cost (€)</th>
<th>Cost deviation (€)</th>
<th>Cost deviation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>19740</td>
<td>21850</td>
<td>2110</td>
<td>0.11</td>
</tr>
<tr>
<td>Hardware</td>
<td>153</td>
<td>153</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Software</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Indirect</td>
<td>43</td>
<td>43</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>19936</td>
<td>22046</td>
<td>2110</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Table 8.4: Total cost

59
8.2 Budget control

The budget has suffered a modification since we have needed to extend the development hours. We have used the Gantt diagram to reorganize the time schedule for each task in order to finish the project on time but the budget has seen an 11% increase.

We have not needed any hardware resource apart from the ones listed in the budget estimations. The main deviation could be the laptop breaking down and therefore having to repair it, in which case we would have to rent another one while the reparations are being done to avoid time loss in our project. As a result repairing and renting a laptop might be added to the final budget.

As for software resources, whenever we have needed a new tool or component we have turned to open source applications.

The budget section that we have had more problems with is the one related to the human resources. As stated in section 5 some tasks have taken longer than expected (Descriptor extraction, Principal Component Analysis (PCA) whitening and bootstrapping).

8.3 Sustainability

In this section we are going to evaluate the sustainability of our project in three different areas: economic, social and environmental.

8.3.1 Economic sustainability

In this document we can find an assessment of the costs of our project, taking into account both human and material resources.

The cost stated in the section 8.1 Budget estimation of this document could be the only one. We aim to create a working solution, so it should not need maintenance if no updates are created. However, it is possible that further development is needed. Creating an improved version of the solution with more accuracy, higher speed or adapting it to work in any platform would add more cost to its development.
It would be difficult to do a similar project with lower cost. Since all the software is open source, the hardware is cut to the minimum and the human resources have been set on a tight schedule. Because of that we believe the project is viable from a competitive point of view.

The kitchen object recognition is going to be used as part of a bigger project from the ESAII department in the UPC to make a kitchen that provides assistance to people.

8.3.2 Social sustainability

We foresee the application that we have developed in this project being used in the countries that have the financial capability to provide social aid to disabled people in the form of robotic technology.

Physically challenged people will find their everyday life easier and feel self-sufficient by not needing another person to help them with their kitchen chores.

Although the project aims to improve their lives it might negatively affect the home care sector. The implementation of automation projects like our robotic kitchen will replace low qualified jobs with high qualification work to design, manufacture and set-up the kitchen.

8.3.3 Environmental sustainability

During all the development of the project we are going to have a computer running and during some phases we are going to use two Kinect units. The devices used, the energy spent and the paper used to print the documentation are going to be the only resources used.

As stated in the Indirect resources costs section the energy we need to develop the project is around 91 kWh, which is equivalent to 59 kg of CO2. It is in fact a high amount of energy, but since we need to make extensive use of a computer there is no way to reduce it.
8.3.4 Sustainability score

Given the economic, social and environmental sustainability from the sections above, and the fact that this work has no law issues to take care of, we proceed to score its sustainability.

<table>
<thead>
<tr>
<th>Sustainable?</th>
<th>Economic</th>
<th>Social</th>
<th>Environmental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Results</td>
<td>6</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>Risks</td>
<td>-5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>41</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.5: Sustainability score
Chapter 9

Conclusions

Object recognition is crucial for autonomous systems that have to make decisions about the world that surrounds them. Without it, those systems would have to rely on a myriad of sensors that would make their implantation on an environment shared with humans difficult to say the least.

In this project we set out to implement an algorithm that performs object recognition using volumetric information for a real time application. After careful study of the state of the art we decided to use a popular technique based in a Bag of Words (BoW) approach which incorporates both supervised and unsupervised learning.

Our algorithm was applied over a publicly available dataset of images taken with cost reduced Kinect RGB-D sensors, which depict different objects at different viewpoints.

After performing a number of tests to tune the algorithm parameters, we randomly selected 6/7 of the dataset for training and 1/7 of the dataset for tests and conducted a number of experiments from which we can draw the following conclusions:

- **Centroid selection:** increasing the number of centroids from 600 to 1200 does improve our results but at a higher computational cost. The results obtained with 600 centroids show that the system is able to predict with precision well within the margins while responding in a timely manner, thus making it appropriate for a real-time environment such as our kitchen.

- **PCA whitening and bootstrapping:** they are both widely used techniques and
in our case do improve results with a slight computational overhead, thus making it appropriate for our real-time scenario.

All in all, we have obtained a solution that, although is not on the industrial level, meets our objectives. It can make predictions in less than a second on an average laptop, and the number of false positives and negatives is within the required margins.

9.1 Future work

In the future, given better resources to perform the learning stage of the algorithm, we could try to lower the computation time by performing component reduction in the Principal Component Analysis (PCA) whitening process. Lowering the number of dimensions of the learned centroids would surely yield faster results and maybe we could increase the number of centroids, acquiring more features to discriminate the objects.

Once we are satisfied with the results we could integrate our contribution into Robot Operating System (ROS) and liberate it to the open source community.
Bibliography


