Object recognition applied to mobile robotics

Director:
Dr. Arnau RAMISA

Instituició del director:
Institut de Robòtica i Informàtica Industrial.

Autor:
Ferran RIGUAL

Ponent:
Dr. René ALQUÉZAR

Departament del ponent:
Llenguatges i Sistemes Informàtics

Facultat d’Informàtica de Barcelona (FIB)
Setembre 2012
Agraïments

Aquest projecte de final de carrera representa només una petita part del gran projecte de col·laboració que han portat a terme l’Institut de Robòtica i Informàtica Industrial (IRI) i l’empresa de Barcelona PAL Robotics. Aquesta associació m’ha donat la oportunitat de conèixer membres d’ambdues organitzacions, i gràcies a l’amabilitat, paciència i dedicació tant dels uns com dels altres he pogut aprendre més intensament del que mai ho havia fet fins ara. Per això i pels bons moments que hem passat treballant els ofereixo la meva profunda i sincera gratitud.

Agraeixo especialment al Dr. Arnau Ramisa, director d’aquest projecte, la seva guia i constant supervisió, així com el seu suport en totes les etapes del treball.

Dedicat als meus amics i família.
Contents

1 Introduction 7
  1.1 Motivation: Robocup  . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  1.2 The team: REEM@IRI  . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  1.3 ROS  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
      1.3.1 ROS Packages  . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
      1.3.2 ROS Communication  . . . . . . . . . . . . . . . . . . . . . . . 10
  1.4 Object detection for manipulation  . . . . . . . . . . . . . . . . . . . 11
  1.5 Objectives  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2 Related work 13
  2.1 Viola–Jones object detection framework (2001)  . . . . . . . . . . . 13
  2.2 SIFT (2004)  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
      2.2.1 A brief example  . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
      2.2.2 SIFT implementation: Constructing a scale space  . . . . . . . 15
      2.2.3 SIFT implementation: Laplacian of Gaussian  . . . . . . . . . 15
      2.2.4 SIFT implementation: Keypoints detection  . . . . . . . . . . 19
      2.2.5 SIFT implementation: Constructing the descriptor  . . . . . . 19
  2.3 HOG (2005)  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  2.4 Biologically-Inspired System (2005)  . . . . . . . . . . . . . . . . . . 22
  2.5 Vocabulary Tree (2006)  . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  2.6 SURF (2008)  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  2.7 Far object detection (2009)  . . . . . . . . . . . . . . . . . . . . . . . 24

3 Selection of the final method 27
  3.1 ORTK  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
      3.1.1 Introduction  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
      3.1.2 OpenCV to OpenCV 2.0  . . . . . . . . . . . . . . . . . . . . . . 29
      3.1.3 ROS Communication  . . . . . . . . . . . . . . . . . . . . . . . . 29
      3.1.4 SURF implementation  . . . . . . . . . . . . . . . . . . . . . . . 29
  3.2 RoboEarth  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
      3.2.1 Introduction  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
      3.2.2 RoboEarth structure  . . . . . . . . . . . . . . . . . . . . . . . . 31
      3.2.3 Replacing SURF by SIFT  . . . . . . . . . . . . . . . . . . . . . 32
      3.2.4 Grasping experiments  . . . . . . . . . . . . . . . . . . . . . . . 32
  3.3 ODU-Finder  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
      3.3.1 Introduction  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
  3.4 GIST  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
      3.4.1 Introduction  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
3.4.2 Sliding windows approach .................................. 35
3.4.3 Preparation of the images .................................. 35
3.4.4 Regular results .............................................. 36
3.4.5 Increasing the speed: squared images ....................... 37
3.4.6 Results using squared images .............................. 37
3.5 Color .............................................................. 38
3.5.1 Introduction ............................................... 38
3.5.2 RGB ......................................................... 39
3.5.3 HSV ......................................................... 40
3.6 Moped .............................................................. 40
3.6.1 Introduction ............................................... 41
3.6.2 Bundler .................................................... 41
3.6.3 First results ............................................... 41
3.7 Performance test ............................................... 43

4 Adapting Moped ................................................. 45
4.1 Deeper study of the method ..................................... 45
4.1.1 Training of the method .................................... 46
4.2 Automation of the training process ............................. 47
4.3 Selection of the grasping point ................................. 49

5 Conclusions ......................................................... 51
Chapter 1

Introduction

“I can see the cup on the table,” interrupted Diogenes, “but I can’t see the ‘cupness’.”
“That’s because you have the eyes to see the cup,” said Plato, “but”, tapping his head with his forefinger, “you don’t have the intellect with which to comprehend ‘cupness’.”

Teachings of Diogenes

1.1 Motivation: Robocup

The original motivation of this work was to participate in the Robocup @Home competition.

Robocup is, as stated in its website, “an international scientific initiative with the goal to advance the state of the art of intelligent robots”. It was established in 1997 and originally centred in a soccer competition for robots, as a way to motivate the researchers in this field. Since then, it has been expanding to other themes such as rescue robots (where robots face emergency situations) or assistant robots (where robots help people in their daily lives).

Robocup @Home is the league of Robocup focusing on assistant robots. It aims to develop personal domestic applications. According to their website, “it is the largest international annual competition for autonomous service robots”. The robots participating in this competition are evaluated in a set of tests where they have to autonomously perform some quotidian tasks related to housework.

An example of the tests found in this competition is “Go get it!”. In this test there are four previously known objects randomly spread around a room. The robots go to the room and have to retrieve one of the objects. They have to identify the object before grasping it. Points can be scored for finding the object, for carrying it back to the start location and handing it to the operator of the robot. This is just one of the tests where using object recognition and manipulation can be necessary to solve the problem.
CHAPTER 1. INTRODUCTION

1.2 The team: REEM@IRI

IRI (Institut de Robòtica i Informàtica Industrial) is a joint research center of the Technical University of Catalonia (UPC) and the Spanish Council for Scientific Research (CSIC). It is located in the Parc Tecnològic de Barcelona and organized in four research lines: Automatic Control, Kinematics and Robot Design, Mobile Robotics and Intelligent Systems, and Perception and Manipulation. This work has been done in collaboration with the people of the Perception and Manipulation line.

PAL Robotics is a company dedicated to the research and development of humanoid robots and robotic components located in Barcelona. Their mission is “to provide robotic products and services which can become an integral part of our daily life”. PAL has developed different robots but, currently, their most advanced one is a humanoid assistant called REEM.

REEM (Figure 1.3) is human-sized, it slides through the floor with the wheels of its platform, using several lasers and cameras to avoid the obstacles. It has different ways to communicate: through the touch screen in its chest or using the internal microphones and speakers to maintain a verbal conversation. Its arms can also be helpful when communicating, either to make the dialogue more natural or to indicate the path to take. Also, its hands let it interact with the objects in its environment.

PAL Robotics and IRI have created a collaboration group to prepare the Robocup @Home competition, called REEM@IRI. The team is composed of workers from both organizations, plus some students of the Technical University of Catalonia (UPC). At the beginning, the labour has been split between the members of the group, and the author of this project was assigned “object recognition”.

The team has worked in an organized manner. To save the work and maintain a control of the progress, repositories from both IRI and PAL have been used. The project has been split in different packages, and communication and compatibility between them has been achieved using ROS operating system.
1.2. THE TEAM: REEM@IRI

(a) Full body  (b) Preparing the grasp

Figure 1.3: Pictures from REEM.

Figure 1.4: Part of the team testing the robot.
1.3 ROS

ROS is an open-source, meta-operating system for robots. It provides hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS forms a network with all the processes on the system, which are coupled using the communication infrastructure. It implements several different styles of communication, including synchronous RPC-style communication, asynchronous streaming of data and persistent storage on disk.

1.3.1 ROS Packages

Software in ROS is organized in packages. The goal of these packages is to provide useful functionality in an easy-to-consume manner so that software can be reused. In general, ROS packages follow a principle: they should be coded in a way that can be shared and reused, they also have to be light, but having enough functionality to be useful.

ROS packages tend to follow a common structure:

- bin/: compiled binaries.
- include/: header files for C++ code.
- msg/: Message definitions. Communication will be seen more in detail in Section 1.3.2.
- src/: Source files, both python and C++ code.
- srv/: Service definitions. Communication will be seen more in detail in Section 1.3.2.
- manifest.xml: Package manifest, contains useful information for users and developers, such as the dependencies, the Ubuntu supported versions, etc.

1.3.2 ROS Communication

A node is a process that performs computation, its behaviour is defined by the code of its package. Nodes are combined together into a graph and communicate with one another. A robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one node controls the robot’s wheel motors, one node performs localization, one node performs path planning, one node provide a graphical view of the system, and so on.

A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive
1.4 OBJECT DETECTION FOR MANIPULATION

Types. Messages can include arbitrarily nested structures and arrays (much like C structures).

**Topics** are named buses over which nodes exchange messages. Every node can publish messages to one topic, or subscribe to it to get its published messages. Each topic is strongly typed by the ROS message type used to publish to it and nodes can only receive messages with a matching type. Topics are intended for unidirectional, streaming communication. Nodes that need to perform remote procedure calls, i.e. receive a response to a request, should use services instead.

Request/reply is done via a **Service**, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Like topics, services have an associated service type.

In Figure 1.6, an example of the communication system is shown. There are three nodes communicating with each other. The one on the left, named `teleop_turtle`, is publishing messages through two different topics, named `/turtle1/command_velocity` and `/rosout`. The second node, named `/turtlesim`, is subscribed to `/turtle1/command_velocity` and therefore receiving the messages published by the `teleop_turtle` node. Finally, the node on the right, called `/rosout`, is subscribed to just one topic, named `/rosout`. It is receiving messages from the rest of the nodes, since all of them are publishing to this topic.

1.4 **Object detection for manipulation**

Computer vision seeks to develop algorithms that replicate one capability of the human brain: to infer properties of the external world purely by its vision. Object detection is a field inside computer vision, where we can find objects in an image. It is also possible to segment out regions of space corresponding to particular objects and track them over time, such as a basketball player weaving through the court.

When treating with object detection for manipulation, the objective is to grasp objects once they have been detected. To do so, it is necessary to estimate their position in the real world: determine how far away these objects are, how they are oriented with respect to us, and in relationship to various other objects. Once this knowledge has been acquired, it is necessary to specify the exact point where the robot is going to perform the grasp, called grasping point. In general, this decision depends on the shape of the object and the kind of manipulator. In this work two different manipulators have been used: the REEM robot from PAL robotics and the WAM robot from Barrett technologies, and they compute the grasping point in different ways.
1.5 Objectives

In this work we investigate the possibilities of current object perception methods for mobile manipulator robots from a hands-on perspective. In such scenario, the mobile robot has to identify objects that can be far away, approach them, and perform the grasp. We identify three major problems for practical use of such implementations, namely long-range object detection, automatic object learning and grasping point selection. We provide practical solutions to the two last problems, and evaluate them in a real grasping experiment.

The objectives could be organized in the following bullet list:

- To select a list of candidate methods and prepare an initial evaluation of their performance.
- To select the best method regarding our particular problem.
- Identify those characteristics where the selected method could be improved in order to better respond to our needs.
- Improve, from the previously detected characteristics, those which belong to our field of study.
- Evaluate the resulting method in real experiments.
Chapter 2

Related work

“The hardest problems of pure and applied science can only be solved by the open collaboration of the world-wide scientific community.”

Kenneth G. Wilson, theoretical physicist and Nobel Prize winner.

This chapter contains a review of the methods evaluated during this work that were not elected for the initial test.

In this work, the state of the art methods are surveyed in order to determine which of them are worth taking into account for an initial evaluation of performance. This step is necessary because there are deadlines to accomplish and the evaluation of each method is a hard task.

2.1 Viola–Jones object detection framework (2001)

In [1], an object detection method is presented by P. Viola and M. Jones. The project has focused on creating a rapid system capable of achieving high detection rates (was the first object detection framework to provide competitive object detection rates in real-time).

The article remarks three key contributions: the use of integral images (which will be seen in more detail below), a simple and efficient classifier\(^1\) and a technique to combine classifiers in order to quickly discard background regions of the image while spending more computation on promising face-like regions.

Integral images, or \textit{summed area tables}, create a table where every cell \((x, y)\) has the sum of the intensity values of all the pixels from the origin to that position.

The summed area table can be rapidly generated using the next formula

\[
s(x, y) = i(x, y) + s(x-1, y) + s(x, y-1) - s(x-1, y-1),
\]

where \(i(x, y)\) is the intensity of the pixel in the \((x, y)\) position. The process can be seen in Figure 2.1, where a very simple example is given.

To use the information of this table to know the sum of intensities in a region of an image only three operations and four memory accesses are necessary. Figure 2.2.a

\(^1\)In computer vision, classifiers must associate input images to object classes. Examples of object classes could be \textit{car}, \textit{human}, \textit{eye}, etc.
CHAPTER 2. RELATED WORK

Figure 2.1: Simple image and the resulting summed area table.

shows an example of an image, where each cell of the table represents the intensity of one pixel. Figure 2.2.b shows the resulting summed area table for this image. The objective is to compute the sum of the intensities in the region specified by Figure 2.2.c (the bottom-right area, highlighted in green). For that purpose, we add the value in the top-left corner (A) and the value of the bottom-right corner (D), and subtract the values in the top-right and bottom-left corners (B and C). The result is $5 + 2 + 3 + 6 = 16$. This can seem of little help when working with small areas, but simplifies a lot the work when regions are bigger.

2.2 SIFT (2004)

In [2], the author presents a method for extracting features from images that can be used to perform matching between different views of an object or scene. ‘SIFT’ stands for Scale Invariant Feature Transform, and reveals one of the useful properties of the
2.2. SIFT (2004)

method: features are invariant to scale changes and rotation, and provide robust matching across a substantial range of affine distortion, change in 3D viewpoint, addition of noise and change in illumination. The SIFT descriptor still seems the most appealing descriptor for practical uses, and hence also the most widely used nowadays.

It is important to distinguish between features, keypoints and descriptors:

- **Features** are a prominent or conspicuous part or characteristic of an image.
- **Keypoints** hold the information to locate the features. They have X and Y coordinates (the position where the feature is located within the image), scale and orientation.
- **A descriptor** is a vector for each keypoint, is highly distinctive and partially invariant to illumination, 3D viewpoint, etc. It has 128 elements containing a 0-255 value.

At first, this method finds the features of the images to compare. Once the features are determined, it computes one descriptor for each feature, called SIFT descriptor. Two descriptors should be similar if and only if their corresponding features are similar as well. Secondly, every feature of both images is matched against the features of the other image. Matching is done by comparing the descriptors. Finally, to know if an object is found in both images (objects should have a big number of features on them), positive matches are tested using many well-known robust fitting methods, such as RANSAC or Least Median of Squares.

2.2.1 A brief example

In [3], SIFT is used to find the pictures on Figure 2.3 in the picture on Figure 2.4. As can be seen in Figure 2.5, the matches rightly relate the pictures, even when some scale and rotation transformations had to be applied.

2.2.2 SIFT implementation: Constructing a scale space

The first step of SIFT is to construct a scale space of the input image. A scale space consists in various representations of the image, but at a different level of detail. Getting less detailed images allow the method to look for more general objects while disregarding the little features. Gaussian blur achieves this effect while not adding new false details. An example can be seen on Figure 2.6.

In SIFT, scale space is taken to another level by adding scale transformations. The objective is the same, but the original image is resized to half size each time before applying the Gaussian blur. Images of the same size form an octave. David Lowe, the author of SIFT, recommends working with four octaves and five blur levels, as in the example of Figure 2.7.

2.2.3 SIFT implementation: Laplacian of Gaussian

Next step in SIFT is Laplacian of Gaussian (LoG), which is good for detecting keypoints. Two consecutive images in an octave are picked and one is subtracted from the other. Then the next consecutive pair is taken, and the process repeats. This is done for all octaves. An example can be seen in Figure 2.8.
CHAPTER 2. RELATED WORK

Figure 2.3: Pictures to be found. Correspond to some regions of the test image, but some transformations have been applied on them.

Figure 2.4: Input image.

Figure 2.5: Result. The big rectangles mark matched images. The smaller squares are for individual features in those regions.
Figure 2.6: Progressively blurred out images, result of applying the Gaussian blur multiple times on the same image.

Figure 2.7: Complete SIFT scale space (first octave is trimmed for space reasons).
Figure 2.8: Laplacian of Gaussian step. In RGB representation, images with values close to zero are displayed in black, this is the reason that the difference of Gaussian images are so dark.
2.3. HOG (2005)

In [4], a new method for detecting humans in images is presented. It uses Histograms of Oriented Gradient (HOG) descriptors. When the article was published (2005), the method outperformed the edge and gradient based descriptors of that time, and approached a near perfect score in the dataset that was being used back then (MIT pedestrian database), so it presented another dataset, more challenging, containing 1800 annotated images.

The method is based on evaluating well-normalized local histograms of image gradient orientations in a dense grid, as seen in Figure 2.11. The basic idea is that local

2.2.4 SIFT implementation: Keypoints detection

At this point all the information necessary to get keypoints has been generated. The keypoints are the points where the Laplacian of Gaussians get the local maxima and minima values. The comparison is not only checked against the eight neighbour pixels from the same picture, but against the 26 neighbour pixels from the images of the previous and next blur level, as seen in Figure 2.9.

2.2.5 SIFT implementation: Constructing the descriptor

Once the interesting keypoints of the image have been found, the final step is to create a descriptor for each one. Descriptors should be easy to calculate and a good representation of the keypoint: two descriptors should be similar if and only if the keypoints are similar too.

To do this, a $16 \times 16$ window around the keypoint has to be taken. This $16 \times 16$ window is broken into sixteen $4 \times 4$ windows, as seen in Figure 2.10.a.

Within each $4 \times 4$ window, gradient magnitudes and orientations are calculated. These orientations are put into an 8 bin histogram. Any gradient orientation in the range 0-44 degrees add to the first bin, 45-89 add to the next bin, etc. (Figure 2.10.b).

The amount added to the bin depends on the magnitude of the gradient and on the distance from the keypoint. So gradients that are far away from the keypoint will add smaller values to the histogram, as represented in Figure 2.10.c.

The process is done for all the 16 regions, and every region will have 8 different bins of orientation, each of them with a magnitude. The result is a vector of 128 positions, normalized, that forms the descriptor.

Figure 2.9: The current pixel (marked with an ‘X’) and the 26 neighbour pixels against the maxima/minima comparison is made.
(a) We create sixteen squares of 4x4 pixels around the keypoint. Within each 4×4 window, gradient magnitudes and orientations are calculated.

(b) Orientation of the gradients is added to the corresponding bin.

(c) Magnitude added to every bin depends on the distance from the keypoint to each gradient.

Figure 2.10: SIFT implementation steps.
2.3. HOG (2005)

Object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions.

This method was originally focused on the problem of pedestrian detection in static images, although since then it has been expanded to other challenges, such as human detection on video and detection of a variety of common animals and vehicles. In Figure 2.12 a representation of the method when used on pedestrians is shown.

In [5], the authors introduced an improvement over this technique to be able to recognize deformable part objects. These are classes of objects formed by smaller parts where the instances can vary their form depending on the position of these parts. An example of this kind of objects can be a human face: the eyes, nose and mouth are organized in a common way, but all the faces are different and present little variations in the position of these elements. In a similar way, a human body normally consists in two arms, two legs, one head and the trunk, organized in a common way (see Figure 2.13.a), but depending on the position of the body, these elements can change their position: the
arms can be raised or pointing in different directions, while the legs can be close to each other or separated, if the human is walking (see Figure 2.13.b). With this approach, the method is more robust to variations in the point of view, and can be used to detect non-rigid objects which can vary their form.

However, this is not the problem we are facing. In this work the objects to be detected are well known and are not compound of smaller parts. The method described above is prepared to detect classes of objects (i.e. persons, cats, bikes, etc.) but is not specially intended to detect concrete objects that, furthermore, are rigid. As a result, it uses a lot of information that would not be necessary in our problem, and is hard to train. This also implicates that estimation of object position is not as good as it could be using specialized methods. These are the reasons why this method was discarded in the initial selection of this work.

2.4 Biologically-Inspired System (2005)

In [6], a solution is proposed to one common problem in computer vision: a lot of features have to be compared when trying to detect a long list of objects. This is usually solved using KNN (K-Nearest Neighbors), but this article proposes an alternative over this technique, a novel framework for sharing multiple feature types, such as texture and color features, within and between different object representations. This technique can not be used with distinctive features like SIFT, but with features types that can be repeated a lot of times over an image.

During training, it creates a set of weighted associations between a learned set of vocabulary features and the set of objects to be recognized. During recognition, vocabulary features that are detected at interest points in the image cast weighted votes for the presence of all associated objects at corresponding locations, and the system detects objects whenever this consensus exceeds a learned threshold.

The authors have reviewed how frequently a feature is shared between representations of different objects. In their database, there is a sizeable fraction of features that are shared by many objects, and only few features are not shared at all. This is a favourable condition when using this technique, and necessary to improve the performance of the KNN method.
2.5 Vocabulary Tree (2006)

In [7], a recognition scheme that scales efficiently to a large number of objects is presented. It uses a vocabulary tree\(^2\) to efficiently quantize sift descriptors into visual words, and an inverted file structure to make fast accesses to the database of images (as in a text retrieval approach), even if it is very large (in the article a database of 50,000 images is used).

The algorithm looks for the features in the test image and takes all the descriptors in no particular order. Then quantizes the descriptors into visual words using the vocabulary tree and uses the whole set of visual words to create a histogram where the information about how many times each visual word has been found is saved.

This technique allows to organize the information in an efficient way at query time, as can be seen in Figure 2.14. At the first step, the seed node (represented as a black square) is queried. It has to choose between its three children (represented as green dots) to direct the query to the correct node. In the example, the child on the bottom of the image is chosen, and the rest of the nodes are ignored. The procedure continues until the final node has been reached.

The inverted file structure is used to, for each visual word in the histogram, access to and vote for the set of images that have that visual word in it (an example of the whole method can be seen in Figure 2.15).

2.6 SURF (2008)

In [11] another method for extracting features from images is presented: SURF. Stands for Speeded-Up Robust Features and its name introduces one of its most distinctive characteristics: high velocity. It uses the same strategy as SIFT (extract features from images encoded in descriptors, to compare them), but simplifies some steps and uses techniques to increase the speed of the whole method.

SURF tries to reduce the size of the descriptors while keeping them sufficiently distinctive. It also uses integral images (also known as “summed area table”) when computing the intensity in regions of the image. Summed area table is a technique first introduced in computer vision in [12] by P. Viola and M. Jones (see section 2.1).

When SURF was compared to SIFT in our work, it resulted to be much faster, but it did not achieve the results of SIFT. The number of matches was always smaller and sometimes this made the difference between finding an object or not. In section 3.2.3 the comparison of both methods is available in the implementation of RoboEarth.

\(^2\) A tree is a graph where all the nodes are connected and there are no cycles.
CHAPTER 2. RELATED WORK

Figure 2.15: Vocabulary tree example. At the top the descriptors of the input image are taken. At the middle the vocabulary tree is used to access to the classes of the descriptors (i.e. what is the visual word corresponding to each descriptor). At the bottom the index inverted file is accessed and the images on the right positions are voted.

2.7 Far object detection (2009)

In [17], a method that uses a monocular camera with zoom capabilities is presented. The main contribution of the method is a system focused on far distance detection of objects. It uses a combination of an attention mechanism and zooming as the first steps in the recognition process. The attention mechanism is based on Receptive Field Cooccurrence Histograms and the object recognition on SIFT feature matching.

Receptive Field Cooccurrence Histograms (RFCH) is a technique presented in 2005 by S. Ekvall and D. Kragic[18]. Is a statistical representation of the occurrence of several descriptor responses within an image. Examples of such image descriptors are color intensity and gradient magnitude. If only color descriptors are taken into account, we have a regular color histogram. Figure 2.16 shows an example of a quantized search image, when searching for a red, green and white Santa-cup. The pixels that lie too far away from their nearest cluster are ignored (set to black in this example). The red striped table cloth still remains, as the Santa cup contains red-white edges.

This method seems a good solution for far-distance object detection. However, it requires a camera with zoom capabilities to work. In this project, the cameras used on both robots are not able to get zoomed images.
2.7. FAR OBJECT DETECTION (2009)

Figure 2.16: RFCH Example

(a) Original image  
(b) Pixels that survive the cluster assignment
Chapter 3

Selection of the final method

"I have called this principle, by which each slight variation, if useful, is preserved, by the term of Natural Selection."

Charles Darwin

This chapter presents the methods that have been tested in this project. For each method, its use is justified, its performance is explained and its necessary preparation (where necessary) is detailed. At the end, all the results are compared in order to choose the one that best adapts to our problem.

3.1 ORTK

Object Recognition Tool Kit was the first method used in this project. One of the authors is Arnau Ramisa, the director of this project. Working in collaboration with one of the authors of a project highly simplifies the process of understanding the purpose of every step and the reasons behind each decision.

There is at least another good reason for start working with this method. This method follows the recommendations of David G. Lowe (author of SIFT, explained in Section 2.2), therefore is a good opportunity to learn in depth his work. Most of the methods that have been surveyed in this study use SIFT, and having a good knowledge of it has boosted my comprehension of them.

3.1.1 Introduction

Object Recognition Tool Kit (ORTK) [8] uses visual information (grayscale pictures) for both training and testing. After applying SIFT to the input image, the resulting descriptors are compared to the ones on the database using k-NN\(^1\). There is a first filter in this step, since matches will only be selected if are much better than their second best coincidence. The next steps are three more filters in order to determine which of the current matches are correct.

\(^1\)In pattern recognition, the k-nearest neighbour algorithm (k-NN) is a method for classifying objects based on closest training examples in the feature space.
Now the matches between the features in the input image and the ones in the database are known, but having one simple match does not imply the presence of an object. Ideally, an object is represented by a lot of features, and having a minimum number of matches from one particular object to the input image is necessary to determine its presence. Therefore, a test is performed in this step to generate hypotheses of which matches are correct (correspond to a real relation between the object and the input image) and which are wrong: Generalized Hough Transform.

**Generalized Hough Transform (GHT)**[9] is a method presented in 1981 by D.H. Ballard. It generalizes a previous method, called Hough Transform, used to detect straight lines and curves on images. GHT uses a voting procedure to find imperfect instances of objects. Each match votes for a concrete position, orientation and scale of the object to find. The most popular options are selected as candidates. Candidates (or hypotheses) are groups of matches establishing the presence of an object in the input image. Each candidate determines a position, orientation and scale of the object in the picture. The following filters will be applied to candidates (not to single matches).

**Random Sample Consensus (RANSAC)**[10], is the next filter to be applied, it is an iterative method to estimate parameters of a mathematical model from a set of observed data which contains outliers. One of its applications can be seen in Figure 3.1, where a line is adjusted from a set of points, where some of them are correct and the rest do not describe the line. RANSAC finds the best adjusted line to the set of correct points and discards the outliers (marked in red).

**Iterative Reweighted Least Squares (IRLS)** is the last filter of the process. It is used as a way of mitigating the influence of outliers in a data set. Once all the filters have been passed, the remaining candidates are considered as correct. Figure 3.2 summarizes the steps of this method.
3.1. ORTK

3.1.2 OpenCV to OpenCV 2.0

ORTK was coded in C++ in 2006, six years before the development of this project. The techniques behind the method are the same nowadays, but the libraries used to implement it have changed. The second major release of OpenCV\(^2\) was on October 2009, and OpenCV 2.0 includes major changes to the C++ interface. The first thing done after studying the code was to adapt it to the new interface.

This new interface seeks to reduce the number of lines of code necessary to code up vision functionality as well as reduce common programming errors such as memory leaks (through automatic data allocation and deallocation). This last characteristic implies changing the idea behind memory management: in the previous version of OpenCV the allocation and liberation of memory had to be specifically ordered by the programmer, while in OpenCV all the resources are handled through objects, and objects free their resources when they are out of scope. This is just an example of the type of changes necessary to adapt the project to the new version of OpenCV.

3.1.3 ROS Communication

ROS was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory, one year after the ORTK project was developed. In order to communicate ORTK with the rest of the functionalities of the robots it was necessary to insert the code in a ROS package. Fortunately, this is a common process that had to be previously done with all the software at IRI, so the system was well prepared to facilitate this change.

All the software in IRI is prepared to separate the communication with ROS from the real algorithm of the package. All the communication had to be implemented from scratch, in order to “translate” the output and input of the original project to ROS messages. However, the part of code responsible for the main methods could be reused. This was possible thanks to the work done by the support engineers at IRI.

3.1.4 SURF implementation

When ORTK was initially tested, the results with textured objects were good enough to detect book covers from more than a meter of distance (as can be seen in Figure 3.3). Even if the results were good, the speed of the method was not very satisfactory: every image needed about 3 seconds to be processed. We thought about replacing SIFT descriptors with SURF descriptors, known by their improvement of speed.

OpenCV 2.0 allows the programmer to choose between SIFT and SURF descriptors without a lot of work. The change has to be made everywhere in the code, because the output of both methods (the descriptors) are not compatible, however, the interface of the functions follow the same idea and it is really easy to understand how to use one of them when the other is known. With this characteristic, the change could be made just by replacing all the functions in the code working with SIFT descriptors with the corresponding functions working with SURF.

After applying the replacement, the velocity slightly improved but some previous results were not detected any more, so we kept the original implementation.

\(^2\)OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. The referred implementation can be found at its website: http://opencv.willowgarage.com/
Figure 3.3: First results using ORTK. The green rectangles show the results of the detection process. The searched objects were the two books. Other tests, looking for the mugs on the table, were performed, but ORTK could not find them.

3.2 RoboEarth

After working with ORTK, its results were good but it was necessary to compare it to other projects. Our team project was highly linked with ROS, so we looked for a new project in its software repositories. RoboEarth is being currently developed (when writing these lines) by a big community of researchers, and it was one of the first projects that we got interested in.

RoboEarth provided an improvement in terms of object manipulation, since it offers the position of the detected objects in world coordinates by default.

3.2.1 Introduction

At its core, RoboEarth is a network and database repository where robots can share information and learn from each other about their behaviour and their environment. It includes an object detector for mobile robotics. In this work both the project and the object detector will be referred as “RoboEarth”. On the ROS stack page of the project, there is information about the training and detection process. A couple of tutorials are available, covering the training of the objects and the application to detect them using the camera.

To train the objects, a pattern has to be printed. This pattern contains easily recognizable marks that allow the software to compute the position of the camera with reference to the pattern. The object to train has to be set in the center of the pattern, as seen in Figure 3.4.a. Then the pattern can be rotated so the camera records the object from all the possible points of view. After this process, the software selects the most valuable views and constructs the object model joining them, as can be seen in Figure 3.4.b.

RoboEarth uses SURF descriptors in order to recognize objects in the image. It compares the visual features in the input images with the features recorded during the training stage. Although it uses depth information to construct the object model, it is not used during the recognition stage. As explained in Section 2.6, SURF descriptors are faster than SIFT, but having to compare the input descriptors with the descriptors from all the views of each recorded model makes RoboEarth a shortly scalable method (the speed of the recognition is linearly related to the number of trained models).
3.2. **ROBOEARTH**

3.2.1 RoboEarth structure

As explained in Section 3.2.1, RoboEarth was designed as a network and database repository where robots can share information and learn from each other about their behaviour and their environment. As a result of this approach, its structure has a strong focus on communication between different robots. As can be seen in Figure 3.5, the structure is divided into three different parts: communication with the database, object scanning and object detection.

**Object scanning** is composed by two different modules: *ar_bounding_box* and *re_object_recorder*. The first one segments the object from its environment, by creating a virtual box around the middle center of the marker template and throwing away anything around it. The latter manages the graphical user interface for controlling the recording process.

The application processes every segmented pointcloud of the recorded object. Each pointcloud is treated as a single view (Figure 3.4.a). As different parts of the object are being seen (by moving the camera around it or by rotating the object in front of the camera), the application processes the localization of the camera (using the information of the marker) and joins the views, forming the final object representation. The views can be erased (if incorrect or do not have enough information) and manipulated through...
CHAPTER 3. SELECTION OF THE FINAL METHOD

(a) Single view  (b) Joint and edition of views

Figure 3.6: RoboEarth Training: Recording the object

Object detection is composed by three modules, but only two of them are used, depending on which type of information is being recorded. re_object_detector_gui handles the graphical interface and it is always used. The second module is chosen as follows:

- re_kinect_object_detector is used if depth information (pointclouds) of the scene is available.
- re_vision replaces the previous module when single images are used to detect the object.

Even when there is the possibility of choosing between one module or the other, at this moment only the detection with single images is implemented, so both modules use exactly the same algorithm. The method compares the descriptors found in the input image with all the views of the recorded model.

All the objects used to test RoboEarth in this project were trained by ourselves, so communication node (re_comm in Figure 3.5), has not been used.

3.2.3 Replacing SURF by SIFT

In the ORTK project SIFT was proved to find much better results than SURF. After noticing that RoboEarth uses SURF by default, our reaction was to implement a SIFT version of the object detector. The difference of the results between both implementations was not as clear as in ORTK, but the velocity of the detection process decreased a lot when using SIFT, so we decided to maintain the original implementation.

3.2.4 Grasping experiments

During the experimentation of RoboEarth we performed several detection tests. Each test included the training of the object and the detection using the Kinect camera (even when only single images were being used). At the end, when good results were being achieved, we tried the detection process with the WAM manipulating robot.

In the test, the box of the ASUS Xtion camera was recorded and situated in the middle of the table. The table is recorded from a tilted view by one kinect attached to
3.3. ODU-FINDER

Figure 3.7: Stills of the videos of the grasping experiment using RoboEarth and the WAM robot

the ceiling. The WAM robot is situated near the table, where its position enables the arm to reach most of the table surface.

The images from the camera were used to detect the object and estimate its position using RoboEarth. Then, a software previously implemented by us was used to compute the best grasping point (the one in the middle of the top of the box). The coordinates of the grasping point were sent to the WAM robot, which was configured to use a descending approximation to the object. The result, even at the first try, was a perfect approach of the robot to the grasping position of the object, and the whole process was recorded using a video camera (some stills can be seen in Figure 3.7).

3.3 ODU-Finder

Even when RoboEarth results were good, we considered necessary to test other methods from the ROS repository before making the final choice. ODU-Finder was one of our first options since their authors are well known in the scope of computer vision.

3.3.1 Introduction

In [19], a novel perception system for autonomous service robots is presented. ODU-Finder stands for "Objects of Daily Use Finder". ODU-Finder uses the previously seen SIFT (Section 2.2) and vocabulary tree (Section 2.5) techniques.

The algorithm uses both image and depth information. In our tests, we used a Kinect camera to retrieve the data. In the detection phase, the depth information is used to segment the image in different parts. The intention is to detect the presence of objects lying on a bigger plane, such a table or the marble of the kitchen. In the segmentation, each portion contains a potential object, as can be seen in Figure 3.8.

Once the interesting parts of the images are obtained, ODU-Finder performs object recognition of textured objects by computing the set of SIFT descriptors and then determining the object model in the library which best explains these descriptors in the region of interest. In order to do it, a comparison with all the features in every object view has to be done. The authors consider object recognition as a document retrieval problem, which enables them to use fast data structures (vocabulary trees, as explained in Section 2.5) and retrieval algorithms and apply them to object recognition problems for large libraries of object models.
CHAPTER 3. SELECTION OF THE FINAL METHOD

Figure 3.8: Segmentation used by ODU-Finder.

The version of ODU-Finder that we tested could not be evaluated since it seems to be an unfinished version. There are some parts that need to be configured in the code before obtaining useful results and the object position is not specified (only returns the chances of the object being in the input image, not information about its location).

3.4 GIST

After testing the previous methods, RoboEarth seemed to be adequate when detecting objects from low distance (up to 2 meters). Still, human environments tend to be larger. Distance increases the difficulty of object detection since the resolution of the input images is not good enough (due to the limitations of the processing capacity of mobile robots). For this reason, we started looking for object detection algorithms that work well with objects contained in a small area of the image.

GIST is a well-known algorithm that works only with a region of the input image (sliding windows approach, explained in Section 3.4.2). For this work, the implementation from LEAR\(^3\) was tested [20].

When working with far distance, estimation of object position is not needed, since the object will not be grasped from the current position of the robot. This type of methods are used to detect the presence of the objects in one specific area of the room, in order to determine where to move next.

\(^3\)LEAR is a French joint team of INRIA Grenoble - RhôneAlpes and the LJK laboratory, a joint research unit of the Centre National de Recherche Scientifique (CNRS), the Institut National Polytechnique de Grenoble (INPG), the Université Joseph Fourier (UJF) and Université Pierre-Mendès-France (UPMF).
3.4. **GIST**

Figure 3.9: The first three steps of the sliding windows algorithm. In this example, the window sizes 3x3 pixels. For each step, the image on the top represents the input image and the position of the window (in red), while the image on the bottom represents the saved result.

### 3.4.1 Introduction

In [13], an overwhelming amount of data (80 million images) is used to solve object and scene recognition. The idea is to use nearest-neighbour methods to find similarities between the queried image and a huge database, where all the images are labelled semantically.

Images are represented using a single descriptor called GIST, which uses the whole image to compose its representation. When first presented in [14], GIST was defined as an “abstract representation of the scene that spontaneously activates memory representations of scene categories (a city, a mountain, etc.).”

GIST is inefficient for finding objects in a scene, since minor changes in the image, like the presence of a little object, do not affect the descriptor in a significant way. Furthermore, it would be hard to implement this method in a way it estimates the position of the object in the real world, since it does not have information about the object characteristics (like size or shape).

### 3.4.2 Sliding windows approach

Sliding windows is a technique that focuses in only one region of the picture at once. It is used when trying to find objects that occupy a small portion of the input image. It is an iterative algorithm that select one region of the image (a little window) and performs the desired algorithms using only this region. Then, the region slides and the algorithm is repeated. The process finishes when the entire image has been treated. An explanatory graphic can be seen in Figure 3.9.

### 3.4.3 Preparation of the images

In the training process, the GIST descriptors of the images of the objects have to be computed. The image where the GIST is applied needs to have a determined format (PPM: Portable PixMap), and the size needs to be squared. To train the objects, we took some images of the objects (the objects can be seen in Figure 3.10) and segmented them in a rectangular shape. Then, we modified them to have a squared appearance and 32x32 pixels of size, as can be seen in Figure 3.11.

Once we have the images, it is necessary to adapt them to the requisites of the implementation. Using OpenCV, a little program in C++ was used to prepare the images.
to the different tests. Different sizes were used in order to try the best configuration for the method, sizes where 32x32 pixels, 64x64 pixels and 128x128 pixels.

It is important to notice that we have two different sizes: the size of the original image of the object (used to detect the objects, called “window size” or “patch size”, as seen in Figure 3.12.a) and the squared size used to get the descriptor (not used in the detection process, as seen in Figure 3.12.b).

Then, for each descriptor and for each window size (since each image has a different size, e.g. the milk does not have the same ratio height-width than the book), we have to apply the sliding windows approach (as explained in Section 3.4.2) to find the region of the image containing the object. This is repeated for each window size because we want the studied region of the input image to have the same size as the size defined for the object (its window size).

### 3.4.4 Regular results

After testing GIST with all the possible sizes, we got the results shown in Table 3.1, Table 3.2 and Table 3.3. Rows determine the distance from where the input image was

![Figure 3.10: Objects used to test the methods.](image)

![Figure 3.11: Images used to train GIST.](image)

![Figure 3.12: Different sizes used in GIST.](image)
3.4. **GIST**

<table>
<thead>
<tr>
<th>32x32</th>
<th>Book</th>
<th>Coke</th>
<th>Milk</th>
<th>Mug</th>
<th>Honey</th>
<th>Water</th>
<th>XtionPRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1m</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2m</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3m</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: GIST results when using the 32x32 pixels images.

<table>
<thead>
<tr>
<th>64x64</th>
<th>Book</th>
<th>Coke</th>
<th>Milk</th>
<th>Mug</th>
<th>Honey</th>
<th>Water</th>
<th>XtionPRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1m</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2m</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3m</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: GIST results when using the 64x64 pixels images.

taken. Objects are organized in columns. An X implies a correct detection. As can be seen in the results, objects are best detected when using small images in the training process (32x32 pixels).

### 3.4.5 Increasing the speed: squared images

The sliding windows technique requires a lot of computation. A way of minimizing the effort is to use the same window size for all the images. This allows the method to compute just one GIST descriptor for each region of the input image, instead of as many as objects in the dataset. However, this can lead to a decrease of efficiency because the images used to train the objects do not fit to the objects, as can be seen in Figure 3.13.

### 3.4.6 Results using squared images

In our tests, when using squared images to train the objects the performance of the method highly decreased. Detections could only be performed when training with small size images (32x32), and only two objects could be detected: coke can (from close distance) and Xtion Box (up to 2 meters). Results can be seen in Table 3.4.

<table>
<thead>
<tr>
<th>128x128</th>
<th>Book</th>
<th>Coke</th>
<th>Milk</th>
<th>Mug</th>
<th>Honey</th>
<th>Water</th>
<th>XtionPRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1m</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2m</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3m</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: GIST results when using the 128x128 pixels images.
3.5 Color

A well known technique to detect objects, specially in low-quality images, is color histograms. First the color model\(^4\) is divided into several histogram bins, and each pixel votes for the appropriate one. In order to find object candidate location, a grid of sub-windows is defined over the input image. For every sub-window, the color histogram is computed and compared to the histogram corresponding to the training image. We have evaluated this technique using the implementation available in the OpenCV library.

In computer vision there are some widely used models, choosing between them is essential when working with this algorithm. We experimented with some of them in an attempt to achieve the best performance of this method.

3.5.1 Introduction

In [15], color histograms (Figure 3.14) are used to recognize objects rapid and precisely. The method is divided in a rough detection stage (where the regions of interest, where the object is more likely to be, are found) and a precise detection stage (where the size and orientation of the object are deduced). With this method they increase the speed of the detection, but the method still presents the drawbacks related to color based object recognition:

- Use only a small part of the information, since it disregards gradient-based information.
- Not able to distinguish between objects of the same color.
- Deficient when looking for objects of multiple colors, specially under occlusion.

Furthermore, the implementation of this method is not ready to estimate the position of the object in the real world, therefore, the grasping can not be performed.

\(^4\) A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers.
3.5. COLOR

3.5.2 RGB

The RGB color model (see Figure 3.15) is the most popular one. RGB stands for Red-Green-Blue, and media that transmits light (such as television) uses additive color mixing with these three primary colors, each of which stimulates one of the three types of the eye’s color receptors with as little stimulation as possible of the other two.

In OpenCV, this model is represented using one byte for each color, resulting in three bytes of memory for each pixel. Each byte represents a number between 0 and 255, where 0 implies that the color is not represented in the pixel, and 255 means the maximum possible representation. As a curiosity (since it does not change the performance of the method), OpenCV uses the BGR model, which behaves exactly as the RGB, but swaps the position of the red and the blue bytes.

This was the first model that we tested. We established 512 (8^3) histogram bins
CHAPTER 3. SELECTION OF THE FINAL METHOD

Figure 3.16: Cylindrical representation of the HSV model

<table>
<thead>
<tr>
<th>32x32</th>
<th>Book</th>
<th>Coke</th>
<th>Milk</th>
<th>Mug</th>
<th>Honey</th>
<th>Water</th>
<th>XtionPRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1m</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2m</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3m</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Detection results when testing the color algorithm.

(eight categories for each of the three colors: red, green and blue). This is not the best model to use, since it is fragile to light changes: an increase or decrease of light will be represented in all the fields, leading to an important change in the resulting bin. As a result, after seeing the bad results of this color model, we tried HSV.

3.5.3 HSV

The HSV color model is more robust to changes in illumination. HSV stands for Hue-Saturation-Value. In the cylindrical representation (see Figure 3.16), the angle around the central vertical axis corresponds to “hue”, the distance from the axis corresponds to “saturation”, and the distance along the axis corresponds to “value”. The aim is that representation of color and lighting are independent. Thus, changes in illumination do not affect the color.

In our implementation, the model was divided into 960 bins (30 categories for the hue channel and 32 for the saturation channel), as recommended by openCV. Hue varies from 0 to 179 (180°, to be able to represent the angle value with a byte) and saturation varies from 0 (black-gray-white) to 255 (pure spectrum color).

The method was tested with the initial dataset of objects (see Figure 3.10). As can be seen in Table 3.5, most of the detections were achieved from close distance, when this method was supposed to be used for far distance detections. Because of this unexpected behaviour, the method was discarded from our selection.

3.6 Moped

ROS allows external developers to upload information about their algorithms in the ROS wiki site. IRIS uploads and maintains information about their algorithms there and also references to the repositories from where the software can be downloaded.

Institut de Robòtica Industrial, see Section 1.2.
3.6. MOPED

Carnegie Mellon University (CMU) is a global research university recognized for world-class technology programs, and some of the projects from its personal robotics department can also be found on ROS. This is how we knew about Moped (see Section 3.6.1).

3.6.1 Introduction

In [16], a method for detecting objects and estimate their full pose\(^6\) using only a single image is presented. It was focused in mobile robotics from the beginning, and therefore provides all the information needed for robotic manipulation of the objects.

During the training stage, Moped uses Structure from Motion (SfM) to create 3D object models (an example of a 3D reconstruction obtained using SfM can be seen in Figure 3.17). SfM is the process of finding the three-dimensional structure of an object by analysing local motion signals over time. The problem is similar to the stereo depth estimation, where the correspondences between both images are found and used to determine the distance of each feature to the camera. In SfM, instead of having two images, we have a set of images that have been taken from different positions, but all of them have been taken within a short period of time, as happens in a video recording. Therefore, the difference of each image and the next one is little enough to recognize some of the features on them, which can be easily related. This way the distance of each point to the camera is determined and multiple points of view of the object can be studied.

To detect the objects in a scene, Moped just needs a single image. The object detection stage is based on SIFT feature matching, which has been proven to present good results (although this can be configured to use SURF descriptors instead). One of the advantages of being focused in mobile robots is its speediness: is able to quickly detect the objects on a scene even when using limited hardware. Therefore, Moped is one of selected candidates for this work.

3.6.2 Bundler

Moped is a real-time object recognition and pose estimation system that uses point-based features (e.g. SIFT, SURF) extracted from rigid 3D models of objects. Bundler is a SfM application\(^7\) used in Moped to get the rigid models of the objects. It takes a set of images, image features, and image matches as input, and produces a 3D reconstruction of camera and scene geometry as output. The system reconstructs the scene incrementally, a few images at a time. It can be freely downloaded, but the implementation that we used in this work was included into the package of Moped. Some parts of the code are not completely finished and have to be configured before running the software for the first time. However, after being able to use Bundler to create the rigid 3D models of the objects, the training of Moped could be easily done.

3.6.3 First results

The first results that we got with Moped were similar to the ones of RoboEarth. Objects with enough texture could be easily recognized, even when the pose estimation was not very accurate (later a solution for this problem was found). However, when we tested

---

\(^6\)Pose contains both position and orientation of the object.

\(^7\)Structure from Motion (SfM), is a technique used to get a 3D model from some pictures of an object (see Section 3.6.1).
Figure 3.17: Aerial view of a countryside environment constructed using SfM. The images used were taken from an aeroplane.
the algorithm putting the camera further away from the objects the results were much better than the ones got with any of the previously tested methods.

All the experiments and analysis done with Moped will be seen in more detail in Chapter 4.

3.7 Performance test

In this section the candidate methods are evaluated. To compare their results, we have used the same dataset on all of them, with the same conditions on the test images. We trained every method individually, and tried to achieve the best possible performance.

The dataset contains seven objects, with different characteristics regarding texture, size and color, as can be seen in Figure 3.18. All of them correspond to a household environment and could be used by service robots in their everyday work.

- There are some textureless objects which are more likely to be found by a recognizer not based on feature descriptors.
- Size goes up to the size of a regular box of breakfast cereals.
- Some colors can be found on multiple objects, and objects are of different colors as a rule.

Objects were situated on the floor, separated by a few centimetres between them. We took four pictures from different distances (close distance, one meter, two meters and three meters), at a 1024x600 pixels resolution. The test pictures can be seen in Figure 3.18.

The results could be separated into two sets according to the distance: close detections, when objects are less than two meters away, and far detections when objects are
CHAPTER 3. SELECTION OF THE FINAL METHOD

<table>
<thead>
<tr>
<th>Close distance</th>
<th>Moped</th>
<th>RoboEarth</th>
<th>Color</th>
<th>Gist</th>
<th>Ortk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2m</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3m</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>13</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3.6: Initial evaluation results: Number of correct detections for each distance.

at two or three meters. This way different methods could be used depending on the situation.

Since there were no controversial detections, evaluation was done at hindsight.

MOPED is the method obtaining best results at both distances, as can be seen in Table 3.6. Regarding close distance, there are several methods achieving good results: Color histograms are just one detection below MOPED (out of eight). However, MOPED gets more advantage as distance increases, achieving five far detections while GIST and color histograms get only two (the rest of methods can not detect anything). Given these results, we selected MOPED for the following tests.
Chapter 4

Adapting Moped

“It is not the strongest of the species that survives, nor the most intelligent, but rather the one most adaptable to change.”

Leon C. Megginson, a management sociologist at Louisiana State University, paraphrasing Charles Darwin

As can be seen in Chapter 3, Moped[16, 21, 22, 23](logo on Figure 4.1) is a good object recognition for manipulation method. Its results surpass the most well-known publicly available implementations designed to solve the same problem. However, it is not a commercial product, the work of its developers has been focused on the results more than on the ease of use. Furthermore, the conditions under which the method has been tested may slightly vary to our needs. For these reasons, a process of adaptation and extension of the method is required to obtain optimal results in our tests.

We detected three main possible areas for improvement during the realization of the tests: automatic training, selection of the grasping point and far distance detection. A solution to the two first problems will be proposed in Section 4.2 and Section 4.3, while the latter has been proposed for future work.

4.1 Deeper study of the method

Moped is an object recognition and full 6DoF pose estimation method from a single image that uses SIFT features [2] to find correspondences between the objects in the scene.

Figure 4.1: Moped Logo.
and the learned models. It is based on the original method of Gordon and Lowe [24] and incorporates a model alignment step for accurate localization, automatic initialization and the combination of RANSAC with Mean Shift clustering to improve the performance of the method in the case of multiple instance recognition.

The method to detect known objects in a new image consists of the following steps:

1. SIFT features of the new image are matched with those of the model.

2. Clusters of keypoints with similar rotation and scaling are found. In contrast with the method of Gordon and Lowe, here the Orthogonal Procrustes Decomposition [25] is used instead of the Generalized Hough Transform.

3. RANSAC, modified with a Mean Shift clustering step to consider groups of neighbouring points, and potentially reduce the search time, is used to find all instances of the objects in the image.

4. All instances of an object with similar transformation are fused together.

4.1.1 Training of the method

In order to perform the detections, the test information has to be compared to the model of each object, a representation of its most relevant characteristics. Since the test information consists on an image, these characteristics are a set of visual features, such as the color of the object or SIFT descriptors [2].

According to the recommendations of the author, to train an object model for MOPED the user has to provide between 40 and 60 pictures of the object of interest and, for each picture, an object/background segmentation mask. Then, SIFT descriptors are extracted from the images and filtered with the masks, and matches are established between every pair of images.

As explained in Section 3.6.2, Moped takes advantage of an external software: Bundler [26, 27] to train the objects. By default, Bundler does not assume anything about the organization of the input pictures. In its initial purpose, it was designed to take unordered pictures, so this is the default scenario. However, our set of images are extracted from a video recording, and it’s informative to keep the order to better understand the sequence. Bundler has an option to specify this ordering: it can take a list of input files, where the order of the pictures is determined. This way matching will be more effective and a better representation of the scene can be achieved.

There are some objects that pose additional difficulties to Bundler. This is the case of objects that have the same logo multiple times, or repetitive patterns all along their texture. Visual features tend to be very similar in these cases, and can be easily mismatched. A possible solution for this problem consists in training a model for each side separately. This way the logo will be only once in each set of pictures and no mismatches will occur.

After Bundler is run, MOPED uses its output to generate a model composed of a set of keypoints describing the object. For each keypoint, the following information is recorded:

- 3D position in the final model
- Images where the keypoint has been seen, and its 2D positions
- Average error when trying to match this point in the different images
4.2 Automation of the training process

Robots should be able to work with a large number of objects. Most current object detection methods require an extensive offline training step for every object in order to recognize it in new images. The training process depends on every method, but getting as much information as possible is a common requisite to create a robust object model. Collecting and manually annotating this information can be a tedious and time-consuming task, specially if a lot of objects have to be trained.

Automation of the training process for MOPED intends to speed up, or completely automate, the work of getting the training data for every object. As explained in Section 4.1.1, MOPED needs, for every image of the object, a segmentation mask and the SIFT descriptors. Utilities to take all the SIFT descriptors from the images are available, images can also be obtained in a fast way recording a video while showing the different parts of the object. This leaves the segmentation mask as the single most difficult step to automate.

The segmentation mask consists in a black and white image, where the pixels in white correspond to the object and the pixels in black to the background. The mask is used to discriminate which descriptors have to be included in the model (the ones from the object) and discarded (those from the background). The default method to obtain the masks is through a utility that allows clicking with the cursor on the contours of the object until a closed area is defined. For a cuboid-shaped object, this means that a human operator has to click six times for every training picture. Considering that every object needs, according to the author, between 40 and 60 photos, this can be an infeasible task in practice.

In order to automate this tedious work, we used a rotatory circular wooden platform. The platform is connected to a servo motor, and can be controlled by software to rotate uniformly at different speeds. With this tool, we can record a video of the object, and use plane segmentation or background subtraction in order to get the segmentation masks automatically. In this work we have implemented both methods.

**Plane segmentation** consists in using the depth information provided by the ASUS Xtion camera to separate the object from the rotating platform. The surface of the rotatory disk is found by adjusting a plane to it, and points on top of the plane are considered to belong to the object. The drawback of this method is that, because of the camera specifications, the minimum distance between the camera and the object is of 80 centimetres, which resulted in the objects occupying a small portion of the images and containing a low number of keypoints.

In contrast to plane segmentation, **background subtraction** only uses image information. The requirements of these methods are a camera in a fixed position and orientation during the whole training acquisition process. The method consists in comparing images with and without the objects in the rotating platform. If the background is static (does not contain moving elements), the only differences in the images will be in pixels belonging to the object. With this technique the objects can be as close as necessary to the camera.
We have performed a detection test to compare these two new methods with the original one. The test scenarios show the object on two different conditions: close and far distance. A comparison of the characteristics of the obtained models has been performed in order to better understand the advantages and disadvantages of each training method. Results can be seen on Table 4.1 and on Figure 4.2.

According to the results, plane segmentation seems the worst method and default is slightly improved by background subtraction. The characteristics analysed are:

1. **Points**, how many points does the model have. In general, having a large number of points translates in higher probabilities of detecting the object.

2. **Average error**, the average distance between the descriptors used to match the points.

3. **Number of views**, how many times each point has been seen, on average. The more times a point has been seen, the easier it will be to identify it from various viewpoints.

Detection results show some differences in the models. The default method obtains good results when detecting from a close distance, but it gets worse as the camera is located further away (at two meters it barely works). Plane segmentation does not always recognize the object when this is close to the camera, but works really well at distances around 1.5 meters (for farther distances, again, it barely works). Background subtraction results are similar to the default method, but achieves better results, especially as the distance is increased; this can be attributed to being able to place the object closer to the camera during the training stage.
When the object was trained using plane segmentation, images were acquired one meter away from the camera; this explains well the better results in far detections. On the contrary, when detecting the object from close distances, default and background subtraction methods get the best results and is explained by the fact that the images used to train these methods were taken at close range.

4.3 Selection of the grasping point

Our final objective is to grasp the detected object with a robotic hand, therefore it is necessary to determine a suitable grasping point. This is usually done computing grasp affordances on the object model and then translating the computed grasp to the detected object pose. Since MOPED already estimates the 6 DoF relating the detected object and its model, we could have a precomputed set of good grasping points and select one of them depending on the current scenario. However, we have found that the object pose determined by MOPED is not entirely reliable, and an alternative solution had to be investigated. This is a difficult problem where a lot of factors intervene. In our work, we will assume that our objects are rigid and resistant, and we will consider scenarios where the object to grasp is surrounded by obstacles, but the upper part of the object will be flat and always reachable by our robot.

Given the previous conditions, the strategy selected to grasp the object consists in taking advantage of the 3D information provided by the ASUS Xtion and performing the grasp action from the upper part of the object. Considering that most objects in a household environment have a cuboid shape, the point in the middle of the upper surface is a reasonable grasping point candidate. Furthermore, it can be easily estimated as the mean of the points in the upper surface. To determine the set of points we are interested in, a plane is fit to the upper surface of the object. This way the points belonging to it can be determined, and the grasping point computed as the mean position of all of them.

Using this technique, we have performed a real grasping experiment with our manipulator robot (see Figure 4.3). Over the manipulator there is a fixed Kinect camera. Using the object detector, the technique described above and the depth information of the camera, the grasping point of the object has been computed and the grasp has been performed. Results of the experiments with seven different objects can be seen in Table 4.2. To test the stability of the grasping point detection, we have computed the standard deviation between the points found for 50 consecutive frames.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Xtion box</td>
<td>5/5</td>
<td>0.0125</td>
<td>0.0237</td>
<td>0.0115</td>
</tr>
<tr>
<td>Coke can</td>
<td>2/5</td>
<td>0.0549</td>
<td>0.0599</td>
<td>0.0336</td>
</tr>
<tr>
<td>Milk</td>
<td>3/5</td>
<td>0.0134</td>
<td>0.0123</td>
<td>0.0119</td>
</tr>
<tr>
<td>Cereal box 1</td>
<td>5/5</td>
<td>0.0242</td>
<td>0.0231</td>
<td>0.0274</td>
</tr>
<tr>
<td>Cereal box 2</td>
<td>5/5</td>
<td>0.0197</td>
<td>0.0213</td>
<td>0.0232</td>
</tr>
<tr>
<td>Cellphone box</td>
<td>5/5</td>
<td>0.0237</td>
<td>0.0314</td>
<td>0.0321</td>
</tr>
<tr>
<td>Pringles can</td>
<td>4/5</td>
<td>0.0211</td>
<td>0.0119</td>
<td>0.0107</td>
</tr>
</tbody>
</table>

Table 4.2: Successful grasping results and standard deviation for the detected grasping point positions of the objects in 50 consecutive frames (cm).
Chapter 5

Conclusions

With recent advances in navigation and mobile manipulation, one of the most pressing bottlenecks for service robotics is object perception methods able to cope with the difficulties of unprepared house or office environments, where the robot will have to carry on tasks that involve autonomously learning novel objects, and detecting and manipulating them.

In this work we have addressed the task of setting up a practical perception system for rigid object manipulation, able to compete in current mobile manipulation challenges. For this, first, a state-of-the-art object detection method has been selected among the available ones, and then it has been adapted to the requirements of our scenario by exploring methods for automatic object model acquisition and grasping point selection.

The components of the proposed method have been quantitatively evaluated in a standard robotics oriented object recognition dataset, the Solutions in Perception Challenge\(^1\), as can be seen in Appendix A, and in various in-house datasets. The practical solutions proposed in this work identify and address some of the main, usually forgotten, limitations of current object detection methods when they are put to work.

Many future work lines follow from the contents of the project. For example, improving the automatic object model creation to work in a fully autonomous fashion (e.g. embed the robot with end-to-end object learning capabilities, and curiosity for unknown graspable artefacts). Another option would be to work on far object detection, barely touched in this work, but a real problem in practical situations where the perceptual workspace of the robot is limited to a few meters. Current (very expensive) workarounds to this problem involve randomly navigating the environment hoping to, at some point, find the desired objects close enough to be recognized. An alternative to investigate could be using visual attention methods to find weak object presence cues to guide this exploration.

---

\(^1\)Solutions in perception challenge is a competition to identify the current state of maturity of robotic perception. More information can be found at the official website: [http://solutionsinperception.org/index.html](http://solutionsinperception.org/index.html)
Appendix A: Solutions in perception challenge

Introduction

Solutions in Perception is a challenge sponsored by Willow Garage (the company behind ROS). The goal of this series of competitions is to establish which perception problems have effective solutions and expand the list of solved problems. The current challenge is to recognize the ID and pose of rigid non-shiny, non-transparent textured and non-textured objects. Which seems to correspond to the problem of this project.

Dataset

The dataset contains 15 objects of different shapes and colors, but similar size. It also includes 395 test scenes, from which we randomly selected 251, with a total of 356 object occurrences. The training images show the objects alone and centred in the image while rotating on a turning plate (as can be seen in Figure 5.1), and no background segmentation mask is provided.

Figure 5.1: Set of training images from the Solutions in Perception challenge. The images correspond to the same object (located in the middle, close to the chess board).
Training process

Although any of the techniques discussed in this project for background/foreground segmentation would be suitable for this scenario (objects fixed in the middle of the frame) we used an even simpler filter that discards those keypoints too far from the center of the image. After using Bundler to train the object models, twelve of the fifteen objects were correctly trained.

Results

Results of this evaluation can be seen in Table 5.1.

<table>
<thead>
<tr>
<th>Model</th>
<th>Detected objects</th>
<th>Total objects</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic training</td>
<td>293</td>
<td>356</td>
<td>82.30%</td>
</tr>
<tr>
<td>Manual training</td>
<td>306</td>
<td>356</td>
<td>85.96%</td>
</tr>
</tbody>
</table>

Table 5.1: Results in the Solutions in Perception dataset. No false positives were found.

Figure 5.2: Example results from the Solutions in Perception Challenge using the MOPED object detector. As can be seen, objects are detected, but the pose estimation is not very accurate.
Bibliography


