Combining Depth, Color and Position Information for Object Instance Recognition on an Indoor Mobile Robot

Louis-Charles Caron

Thèse

Combining Depth, Color and Position Information for Object Instance Recognition on an Indoor Mobile Robot

Présentée à l'Unité d'Informatique et d'Ingénierie des Systèmes

pour l'obtention du grade de Docteur de

l'**ENSTA ParisTech**

École Nationale Supérieure des Techniques Avancées

Spécialité : Informatique

par

Louis-Charles Caron

Thèse soutenue le 4 novembre 2015

devant le jury composé de

Président:	Youcef Mezouar	Professeur — Institut Pascal Clermont-Ferrand
Rapporteur:	Fabien Moutarde	Maître-assistant HdR — Mines ParisTech
Rapporteur:	Michel Devy	Directeur de Recherche — LAAS-CNRS
Examinateur:	Guy Le Besnerais	Maître de recherches HdR — ONERA
Superviseur:	David Filliat	Professeur — ENSTA ParisTech
Co-superviseur:	Alexander Gepperth	Maître de conférence — ENSTA ParisTech

Résumé

Des robots mobiles ont déjà fait leur apparition dans nos domiciles pour accomplir des tâches simples. Pour qu'un robot devienne un véritable assistant, il doit reconnaître les objects qui appartiennent à son propriétaire. Dans cette thèse, des algorithmes de reconnaissance d'instances d'objets sont développés pour faire face aux variations (angles de vue, illumination, etc.) qui surviennent dans un contexte de robotique mobile d'intérieur. Différentes manières de tirer profit de ce contexte sont étudiées. Premièrement, un algorithme de segmentation géometrique est proposé pour profiter de la structure inhérente aux scènes d'intérieur et y détecter des objects isolés. Un réseau de neurones pour la reconnaissance d'objets qui fusionne des informations de forme, couleur et texture provenant d'images de couleur et de profondeur est ensuite présenté. Cet algorithme met en évidence l'importance de combiner plusieurs sources d'information et la difficulté à utiliser la couleur en robotique. Enfin, une solution alternative à base de recherche de plus proche voisin, plus facile à entraîner que le réseau de neurones, est détaillée. Cette approche utilise des mesures comme la distance à l'objet pour éliminer certains paramètres dans la recherche de plus proche voisin et le partitionnement de données. L'information provenant de multiples vues d'un même objet sont aussi fusionnés.

L'algorithme résultant fonctionne sur un robot mobile et peut reconnaître 52 objects en 500 ms en moyenne (sur un Intel Core i5 avec 3 GB de RAM) avec un taux de succès de 80%. Les expériences menées dans cette thèse montrent qu'il est essentiel d'identifier et d'utiliser la structure et l'information présentes dans le contexte de la robotique d'intérieur pour faire face aux grandes variations qu'il peut y survenir.

Abstract

Mobile robots have already entered people's homes to perform simple tasks for them. For robots at home to become real assistants, they have to be able to recognize the objects in their owner's home. In this thesis, object instance recognition algorithms are designed to cope with the variations (viewing angle, lighting conditions, etc.) that occur in the context of mobile robotics experiments. Several ways to take advantage of this context are studied. First, a geometric segmentation algorithm that benefits from the structure of indoor scenes to find isolated objects is designed. Then, a neural network based object recognition process that fuses shape, color and texture information provided by color and depth cameras is presented. This step highlights the importance of carefully combining multiple features and the difficulty to use color information in robotics. Finally, an alternative approach using a nearest neighbor classifier, which is easier to train than the neural network, is detailed. It relies on physical measures available to the robot to eliminate some parameters in clustering and nearest neighbor search procedures. Also, it uses information gathered through multiple sightings of the objects to reduce the negative impact of occlusions.

The algorithm can recognize 52 objects with a success rate of 80% and runs in 500 ms on average (on an Intel Core i5 CPU with 3 GB of RAM) on a mobile robot. This work shows that identifying and taking advantage of the structure and information available is essential to handle the variations that happen in indoor mobile robot experiments.

Remerciments

Merci à tous ceux qui m'ont aidé, accompagné, entretenu, supporté, enduré, encouragé, hébergé et inspiré tout au long de ces 3 ans (et demie (et un peu plus)).

Cette thèse a été financée par une bourse d'études supérieures du Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG).

This thesis was funded by a Postgraduate Scholarship from the Natural Sciences and Engineering Research Council of Canada (NSERC).

Contents

1 Introduction **15**
- 1.1 Autonomous mobile robots . . . 15
- 1.2 Software aspects of a robot at home . . . 16
- 1.3 Computer vision on a mobile robot . . . 17
- 1.4 A scenario for a mobile robot at home . . . 21
- 1.5 Contributions . . . 23
- 1.6 Structure of the thesis . . . 26

2 State of the art **27**
- 2.1 Image segmentation . . . 27
  - 2.1.1 Over-segmentation . . . 28
  - 2.1.2 Foreground-background separation . . . 30
- 2.2 Feature-based image comparison . . . 32
  - 2.2.1 From pixels to features . . . 32
  - 2.2.2 Detector: from dense to sparse . . . 34
  - 2.2.3 From an array to a set . . . 35
  - 2.2.4 Vocabularies: from values to representatives . . . 37
- 2.3 Color in computer vision . . . 39
  - 2.3.1 Models of photometric changes . . . 39
  - 2.3.2 Color representations . . . 40
  - 2.3.3 Transformed color representation . . . 42
  - 2.3.4 Color moment invariants . . . 43
- 2.4 Using histograms for density estimation . . . 44
  - 2.4.1 Choosing parameters for histograms . . . 44
  - 2.4.2 Aliasing in histograms . . . 45
  - 2.4.3 Comparing histograms . . . 46
- 2.5 Trends in object instance recognition algorithms for mobile robots . . . 49

3 Methods **53**
- 3.1 Hardware . . . 53
- 3.2 Software . . . 54
- 3.3 Algorithms . . . 54
  - 3.3.1 Parametric model fitting using RANSAC . . . 54
  - 3.3.2 Nearest neighbor search with kd-trees . . . 55
  - 3.3.3 Adaptive occupancy representation using octrees . . . 56

3.3.4 Computing point normals in a point cloud . . . 57
3.3.5 Simultaneous localization and mapping . . . 58
3.3.6 Scale invariant feature transform for texture description . . . 58
3.3.7 Surflet-pair relation for shape description . . . 59
3.3.8 Function approximation with feed-forward neural network . . . 61
3.4 Data . . . 63
3.4.1 Segmentation dataset . . . 63
3.4.2 ENSTA offline object dataset . . . 63
3.4.3 ENSTA online object dataset . . . 66
3.5 Multi-class classification performance measures . . . 66

4 Geometrical indoor scene segmentation on a mobile robot 71
4.1 Introduction . . . 71
4.2 Design choices . . . 71
4.3 Implementation details . . . 73
4.3.1 Offline calibration . . . 73
4.3.2 Pre-processing . . . 73
4.3.3 Floor removal . . . 73
4.3.4 Walls removal . . . 75
4.3.5 Object candidates over-segmentation . . . 77
4.3.6 Object candidates rejection . . . 78
4.3.7 Parameters . . . 78
4.3.8 Process overview . . . 78
4.4 Experiments . . . 78
4.4.1 Segmentation performance . . . 78
4.4.2 Run time . . . 79
4.4.3 Use cases . . . 80
4.5 Discussion . . . 85
4.6 Conclusion . . . 85

5 Global object recognition by fusion of shape and texture information with a feed-forward neural network 87
5.1 Introduction . . . 87
5.2 Design choices . . . 87
5.3 Description of the features . . . 89
5.3.1 Histogram of occurrence of SIFT words . . . 89
5.3.2 Saturation weighted hue histogram . . . 91
5.3.3 Transformed RGB histogram . . . 91
5.3.4 Surflet-pair-relation histogram . . . 91
5.4 Data fusion with a feed-forward neural network . . . 92
5.4.1 Training . . . 92
5.5 Multi-view fusion by position . . . 93
5.6 Object recognition experiments . . . 93
5.6.1 Recognition rates . . . 94
5.6.2 Precision-recall curves . . . 96

5.6.3 Semantic map . . . 96
5.6.4 Run time . . . 100
5.7 Discussion . . . 100
5.8 Conclusion . . . 102

6 Data fusion by shared nearest neighbors for object instance recognition 105
6.1 Introduction . . . 105
6.2 Design choices . . . 106
6.2.1 Color feature . . . 106
6.2.2 Classifier . . . 106
6.2.3 Distance metric . . . 107
6.2.4 Model selection . . . 107
6.2.5 Nearest neighbor search . . . 108
6.2.6 Multi-view fusion . . . 109
6.3 Implementation Details . . . 109
6.3.1 Opponent color descriptor . . . 109
6.3.2 Model selection with physical distance guidance . . . 111
6.3.3 Bounded multi-modal nearest neighbor search . . . 115
6.3.4 Two-step multi-view fusion by position . . . 115
6.4 Experiments . . . 118
6.4.1 Model selection . . . 118
6.4.2 Uni-modal offline experiment . . . 119
6.4.3 Multi-modal offline experiment . . . 120
6.4.4 Online experiment . . . 121
6.4.5 Run time and resources . . . 121
6.5 Discussion . . . 123
6.6 Conclusion . . . 125

7 Discussion and perspectives 127
7.1 Database . . . 127
7.2 Segmentation . . . 131
7.3 Features . . . 133
7.4 Classifiers . . . 134
7.4.1 Offline experiments . . . 134
7.4.2 Online experiments . . . 135
7.5 Multi-modal fusion . . . 136
7.6 Perspectives . . . 137
7.6.1 Common representation of knowledge . . . 137
7.6.2 Autonomous incremental learning . . . 139
7.7 Conclusion . . . 140

A Nearest neighbor experiment results **143**
A.1 Multi-modal offline experiment results 143
A.2 Single-view online experiment results 146
A.3 Multi-view online experiment results 149

List of Figures

1-1 An arena of the Défi Carotte competition. The robot designed for the competition can be seen in the lower left-hand side of the picture. . . 17
1-2 A semantic map. 23
1-3 Block diagram of the robotic system proposed to solve the robot at home scenario. 24

2-1 SLIC superpixels with three different segment sizes. 29
2-2 Tabletop segmentation of a desk. 31
2-3 HOG feature for pedestrian detection. 34
2-4 SIFT detector applied to two similar images 36
2-5 Illustration of the RGB, HSV and opponent color representations. . . 41
2-6 Linear interpolation of samples to a two-dimensional histogram. . . . 46

3-1 Image of the robot 54
3-2 Kd-tree representation of a mug. 56
3-3 Octree representation of the Stanford bunny. 57
3-4 Surface normals computations on a point cloud. 58
3-5 A robot mapping an unknown environment using a SLAM algorithm. 59
3-6 Spatial subsampling and compilation of gradient orientations in the SIFT feature 60
3-7 Angular values involved in the surflet-pair relation. 61
3-8 Original and labeled image example from the segmentation dataset. . 63
3-9 The setup for the offline dataset. 64
3-10 One example for each viewing angle of the "red office chair" object from the offline database. 64
3-11 All 52 objects from the offline object dataset. 65
3-12 The room and objects in which the robot was moving around to collect the images of the online dataset. 67
3-13 Examples of erroneous data removed from the online dataset. 67
3-14 Examples of good views and more challenging ones from the online dataset. 68

4-1 An appropriate scene to perform offline calibration. 74
4-2 Functional overview of the geometric scene segmentation algorithm. . 80
4-3 An indoor scene processed by the geometric scene segmentation algorithm. 81

4-4 Scenes from the segmentation database. 81
4-5 Examples of successful segmentations. 82
4-6 Examples of failed segmentations. 83
4-7 Example of incomplete black object segmentation. 84

5-1 Block diagram of the object recognition process based on the neural network classifier. 90
5-2 Confusion matrix for the multi-view online experiment using the neural network with the surflet-pair-relation histogram, transformed RGB and the occurrence of SIFT word features 95
5-3 Micro-average precision-recall curves for object recognition in the offline experiment. 96
5-4 Micro-average precision-recall curves from the single-view online experiment. 97
5-5 Micro-average precision-recall curves from the multi-view online experiment. 98
5-6 The semantic map resulting from the online experiment. 99
5-7 Zoom on the semantic map. 99

6-1 Block diagram of the common nearest neighbor based recognition. . . 110
6-2 Process of computing feature space similarity thresholds and neighborhoods from physical redundancy. 112
6-3 Multi-modal nearest neighbor search on a simple problem. 116
6-4 Conversion process from search results to fused multi-modal score vector. Each individual feature search generates a score vector that indicates which object is likely to be the one to identify. Even if several models of a given object are returned by a search, the score vector only indicates a 1 for this object. The individual feature score vectors are then combined by a majority operation. The green and red circles indicate which of the objects found in the individual searches are respectively accepted and rejected by the majority operator. 117
6-5 Division of space around an object for the two-step multi-view fusion process. 118
6-6 Selected shape models of the object gray air conditioner. 119
6-7 Selected color models of the object gray air conditioner. 119
6-8 Confusion matrix for the multi-view online experiment. 122

7-1 Setup to capture the point clouds for the RGB-D Object Dataset. . . 128
7-2 Examples of objects from the RGB-D Object Dataset. 128
7-3 Three reconstructed point clouds from the RGB-D Scene Dataset. . . 129
7-4 The setup to capture the point clouds for the Berkeley Instance Recognition Dataset. 129
7-5 Extraction of walls from an occupancy map. 132
7-6 An outdoor scene reconstructed by the OctoMap algorithm. 139

List of Tables

2.1 Changes in lighting conditions 40

3.1 List of freely available software used in this work 55

4.1 Parameters of the geometric scene segmentation algorithm 79
4.2 Performance of segmentation expressed in terms of micro-average precision and recall values for the labels "floor", "wall" and "object". . . 79

5.1 Recognition rates for the offline, single-view online and multi-view online experiments using different combinations of features. 94

6.1 Recognition rate for offline experiments in the form "recall"/"average number of labels" ("number of ties" out of 21 411 examples). 120
6.2 Notable results of the multi-modal offline experiment (21 411 examples) 121
6.3 Notable results of the online experiment (141 examples, 20 objects) . 123

7.1 Comparison of the characteristics of the ENSTA, RGB-D and Berkeley Instance Recognition Datasets 130

A.1 Recognition rates for multi-modal offline experiments using L_2 distance metric (21 411 examples). 143
A.2 Recognition rates for multi-modal offline experiments using χ^2 distance metric (21 411 examples). 144
A.3 Recognition rates for multi-modal offline experiments using Jeffreys distance metric (21 411 examples). 144
A.4 Recognition rates for multi-modal offline experiments using squared-chord distance metric (21 411 examples). 145
A.5 Recognition rates for online experiments using L_2 distance metric without multi-view fusion (141 examples, 20 objects). 146
A.6 Recognition rates for online experiments using χ^2 distance metric without multi-view fusion (141 examples, 20 objects). 147
A.7 Recognition rates for online experiments using Jeffreys distance metric without multi-view fusion (141 examples, 20 objects). 147
A.8 Recognition rates for online experiments using squared-chord distance metric without multi-view fusion (141 examples, 20 objects). 148
A.9 Recognition rates for online experiments using L_2 distance metric with multi-view fusion (141 examples, 20 objects). 149

A.10 Recognition rates for online experiments using χ^2 distance metric with multi-view fusion (141 examples, 20 objects). 150
A.11 Recognition rates for online experiments using Jeffreys distance metric with multi-view fusion (141 examples, 20 objects). 150
A.12 Recognition rates for online experiments using squared-chord distance metric with multi-view fusion (141 examples, 20 objects). 151

Chapter 1

Introduction

1.1 Autonomous mobile robots

Achieving autonomy is a long sought goal in robotics. Many of the most impressive achievements of recent research in robotics involve a robot which can autonomously perform a task. One recent example of a task performed autonomously by a robot is the towel folding done by a PR2[1] robot [47]. In this work, a PR2 robot picks up a towel from a pile of randomly dropped towels on a table and folds it. The process requires that the robot perceives the important features of the towels like corners and manipulates them properly. Merely saying that a robot is autonomous is not very precise statement. The autonomy of a robot has to be studied in parallel with the environment in which it performs its task. In the towel folding example, the robot is not capable of placing the towels on a shelf or to open a dryer to fetch a towel. The environment in which the robot operates is controlled. Certain parameters are chosen so that it is possible for the robot to operate without having to solve every single problem that could come with this task. The towels are placed on a flat surface in front of the robot, for instance. This is one example of a controlled parameter. Consequently, the towel folding robot is not yet a completely autonomous laundry-service robot. This first step, though, can serve as the basis for an enhanced robot which deals with a higher number of aspects of a real laundry-service robot. It is not rare to see impressive robots that are the result a series of designs aimed at solving increasingly difficult tasks. An inspiring example is the development of autonomous cars, dating back to the DARPA Grand Challenge[2] held in 2004. The challenge was a robotics competition where a robot had to autonomously travel 142 miles in the desert of Primm in the United States in complete autonomy in less than 10 hours [85]. The challenge focused on autonomous navigation in an unknown environment, involving no interaction between the car and external factors except the dirt road. In 2007, the DARPA Urban Challenge[3] added new difficulties to the problem by having the robots operate in a city environment involving traffic and obstacles. A few years

[1] https://www.willowgarage.com/pages/pr2/overview
[2] http://www.darpa.mil/newsevents/releases/2014/03/13.aspx
[3] http://archive.darpa.mil/grandchallenge/

later, Google unveiled the now famous Google car [52]. This succession of events was a great success for fields robotics. It struck the imagination of people by showing that the day a car will drive itself autonomously is not that far.

The strategy of designing robots in a simplified but real environment is a great way to tackle realistic challenges without being overwhelmed by complexity. This thesis applies such a strategy to the design of robots at home. A robot at home is a robot that shares the home of its owner to assist him in its daily chores. A robot that accomplishes useful tasks in a home is a very complex system. It involves hardware (wheels or legs, hands for manipulation, a structure, several sensors, etc) and software (perception of the environment, mapping and localization, manipulation, path planning, mobility, etc.) parts that work hand in hand to produce a well behaving robot. This thesis will concentrate on the software aspects of a robot at home. The hardware will be composed of off-the-shelf solutions chosen to fit the task to solve.

1.2 Software aspects of a robot at home

For a robot at home to be of any use, it needs a minimum set of abilities. Imagine the robot's owner asks it to get a bottle of milk in the refrigerator. This requires, as a bare minimum, that the robot can: communicate, move and manipulate. First, the owner must be able to communicate its demand to the robot. Next, the robot has to move towards the refrigerator. Then, it must open the refrigerator and pick the bottle up. Finally it must move again to hand the bottle to its owner. Each of these tasks can be split into even smaller parts. Communication implies that the robot first perceives the owner's voice (or any other media used to communicate the owner's will) and then analyzes its meaning. Also, before the robot can even start to move around, it has to figure out where it is. This requires perceiving the environment around it. Moving to the right place in front of the refrigerator implies detecting the appliance first. Obviously, the software required to control such a robot also is very complex. To reduce the complexity involved in working on a robot at home, some software aspects like manipulation and control will be overlooked in this thesis. Other aspects such as localization and mapping will be solved by freely available state-of-the-art solutions. The thesis will concentrate on perception and more precisely scene understanding. Scene understanding is the problem of identifying the different elements that compose a scene. It is an important ability for a robot at home to understand its environment. Detecting the floor, for example, can help the robot plan trajectories. A robot at home must also be able to recognize the objects and pieces of furniture around it. Scene understanding and more specifically object recognition are domains of computer vision. In this thesis, theories and algorithms from computer vision will be applied in a robot at home scenario.

Part of the software developed in this thesis was designed during the PACOM project[4] for the Défi Carotte[5]. Défi Carotte is a robotics challenge organized by the

[4] http://cogrob.ensta-paristech.fr/pacom/

[5] http://www.defense.gouv.fr/dga/actualite-dga/2010/defi-carotte-la-robotique-en-competition?nav=web

Figure 1-1: An arena of the Défi Carotte competition. The robot designed for the competition can be seen in the lower left-hand side of the picture.

french Armament Procurement Agency and Research Funding Agency. The challenge was to design a robot (or several robots) to autonomously explore and map an unknown arena of about 100 m^2. Additionally, the map had to be annotated with the type of floor (concrete, wood, etc.), type of walls and objects that the robot encountered. The construction of the arena was a single floor with flat walls, but the presence of ramps on the floor, of windows and mirrors made the task more difficult. Figure 1-1 shows a picture of an arena that had to be explored during the challenge.

1.3 Computer vision on a mobile robot

Object category and object instance recognition are two important research domains of computer vision. In object category recognition, the goal is to determine the category of an unknown object from a list of possible categories. Examples of categories are "chair", "table", "bag" or "dog". The goal of an object instance recognition algorithm is to determine the identity of an unknown object from a list of possible objects. Object category recognition focuses on the ability to capture the features that characterize a given type of objects. Tables, for instance, tend to be composed of a flat surface mounted on four legs. The goal of object category recognition is to learn these characteristics on a large number of examples in order to be able to automatically determine the category of an object that was never seen before. Object

category recognition focuses on the elements that are common to all instances in a category. The difficulty in object category recognition lies in the fact that the categories like "table" do not translate into an fixed set of characteristics. Some tables, for instance, have a single leg. In object instance recognition, the goal is to find the features that are specific to each instance of an object. For example, the shape of a four-legged table would be a good feature to distinguish it from a single-legged one. Object instance recognition focuses on the differences between several objects in order to be able to tell them apart. For a robot at home, it is essential to be able to specifically recognize the pieces of furniture that are in its owner's home. A robot at home must be able to tell whether it is facing one of the dining room chairs or the one from the office. Furthermore, a robot at home will not often be facing objects it has never seen before. The only objects it is concerned with are the ones in its owner's home. Object instance recognition is therefore a more useful ability than object category recognition for a robot at home.

Computer vision algorithms typically rely on the information provided by a video camera. Recently though, the availability of inexpensive depth cameras changed this trend. The Kinect, for instance, is an RGB-D camera that is nowadays widely used in robotics. An RGB-D camera combines a color camera and a depth camera, providing more information than a regular video camera. The added information is very useful for a variety of tasks from mapping [39] to scene understanding [31] and object recognition [75, 16]. In this thesis, the depth information provided by an RGB-D camera will be combined with color to perform a more reliable recognition. One pitfall of RGB-D camera like the Kinect is that the depth sensor cannot be used outdoors. Direct sunlight saturates the sensor and provokes false measurements. A robot at home operates indoors and failures from the depth camera should not happen often.

Object instance recognition algorithms, whether they use depth information or not, were not designed with robotics in mind. Rather, robotics is one possible application of these algorithms. Applying computer vision algorithms to robotics means that the specificities of the application must be identified and considered. Mobile robotics in particular is considered to be a difficult application of computer vision. In mobile robotics, the operating conditions are much less controlled than in usual computer vision tasks.

The main problems that come with the lack of control in robotics are:

Sensor noise, motion blur
: Measures on a robot come from sensors, and all sensors are subject to noise. In particular motion blur a is type of noise specific to moving cameras (or images of dynamic scenes) that causes edges to be dull.

Partial views, occlusions
: Partial views happen when an object does not appear entirely in a single image. They happen when an object is partly outside the field of view of the camera and when an object occludes another one. Self occlusions, where a part of an object is hidden behind another part of the same object, can also happen.

Illumination changes, shades
The illumination of an object in a given indoor environment can vary due to weather, the presence of windows and blinds, room lights, the position of the object and the position of the camera. These factors can also produce shades on the object which change its visual appearance in an unpredictable way.

Viewpoint variations, scale factor
As a mobile robot moves around, it will see objects from any possible angle of view. Also, the same object can be seen from afar or be right in front of the camera and occupy the whole field of view.

Clutter
Indoor scenes tend to contain many objects of interest that might be clumped together and difficult to tell apart.

Number of classes
A robot at home should be able to recognize each piece of furniture and casual objects present in its owner's home. This leads to a high number of object instances to recognize.

Limited physical resources
Mobile robots have energy and weight constraints that tend to limit the power of their embedded processor and sensors. Sensors on a mobile robot typically have a lower precision than usual and processing power lower than what is available in computer vision.

Limited time
Robots behave in the environment and must react to it. Accordingly, processes on a mobile robot are required to run as fast as possible. High-level processes like object recognition do not have strict real-time constraints, but their run time still should be in the order of seconds.

These difficulties are usually tackled in isolation in computer vision, but must be dealt with simultaneously in robotics. On the other hand, some specificities of robotics can be taken advantage of and benefit to the computer vision task. One of the main advantages of robotics is that its many facets can benefit to one another. Here are a few examples of how some aspects of a robot can benefit to vision:

Manipulation
If this is possible for the robot, manipulating an object can enhance perception. Having the robot pick up the object and move it in front of its camera is a way to generate a set of images that will make recognition easier [45]. The fact that the robot itself moves the object gives information about the exact pose of the object through proprioception. The additional information provided by manipulation, compared to a single static image, is enormous. One difficulty of this approach is that the robot's end effector has to be in direct contact with the object in order to manipulate it. It must thus be eliminated from the

perceptions, otherwise it will pollute them. The robot basically has to be able to recognize its own end effector before it can recognize the objects it manipulates.

Movement

The mobility of the robot is another source of information. One way to use the movement of the robot is by performing environment reconstruction [35, 83, 96]. By moving around, the robot can get a more complete view of the environment and use this to create a virtual version of it. Reconstructed environments do not suffer from occlusions too much. As the robot moves around, it is likely that it gets to a place where a part of the environment that previously was occluded now becomes visible. Observing the environment for a long time also helps filter out sensor noise. Another way to link perception and movement is through active sampling, or active selection of a good view for the object recognition. A robot can first coarsely analyze a scene and then decide that it is beneficial to zoom in on a particular region of the image or to get closer to a certain object in order to confirm a first hypothesis [18, 97].

Multi-modality

On a robot, more information can be collected simply by using more sensors. This is usually referred to as multi-modal processing. The multiple modalities might not necessarily come from different sensors, as some sensors do provide information that is multi-modal in nature. This is the case of the brightness and color information provided by a color camera and the color and depth information provided by an RGB-D camera. Many feature combinations can be advantageously fused. This is the case for gradient and shape [30], texture and color [45], color and gradient [41]. Most uses of color involve the HSV representation and texture is often described using local features. These features are described respectively in chapter 2, sections 2.3.2 and chapter 3 3.3.6.

An aspect of robots at home which can be highly beneficial to a vision task is the fact that such robots are confined to indoor environments. Indoor scenes are highly structured and their analysis tends to be much simpler than that of outdoor scenes. This is exploited in many robotic systems, but the work in [56] is directly related to this thesis. They use a tiltable laser range finder to produce two-dimensional depth maps on a mobile robot. The depth map is used to produce a three-dimensional reconstruction of the indoor environment. The resulting map is analyzed and the main structuring elements like walls, floor and ceiling are marked. A certain number of objects are also recognized. This thesis follows a similar approach but focuses much more on the object recognition part and explores the use of a multi-modal perception to improve on this aspect. A special focus is given to the use of color for object recognition in difficult conditions.

One of the domains that spawned from object instance recognition is image retrieval. Image retrieval designates a system which accepts a query image and returns as many related images from its database as possible. This process sometimes is referred to as "google of images". If queried with an image of the Eiffel Tower, it

should return all images from its database on which the Eiffel Tower appears. Image retrieval is closely related to object instance recognition as both work with an instance of specific objects (the Eiffel Tower, not any tower). Images used for image retrieval tend to show the objects under various points of view, scales and illumination conditions. Image retrieval system must also deal with cluttered images and a large number of classes. Image retrieval systems, though, are typically used in situations where background is very informative. Images of the Eiffel Tower, even if you remove the Eiffel Tower, will be very similar to each other because they will show the other buildings around the Eiffel Tower. On the other hand, object instance recognition system absolutely cannot rely on background information. Most object instances, be they chairs or coffee mugs, move around and are not bound to a specific background. The most accomplished image retrieval systems use feature-based image comparison algorithms. Feature-based processing of images is reviewed in chapter 2, section 2.2. The solutions used to solve the image retrieval problem will be studied in detail in this thesis and will be adapted to take advantage of the robotics context.

Recent work in robotics demonstrated the usefulness of segmentation as a pre-processing to object recognition. The pioneering work in [56] is a good example. The table-top recognition tasks also make heavy use of geometric segmentation processes [31]. Indoor scenes are highly structured by the presence of a floor, ceiling and walls. They lend themselves very well to geometric segmentation. The usage of a segmentation pre-processing step tends to reduce the overall processing time of object recognition algorithms. Segmentation strategies are discussed in chapter 2, section 2.1.

1.4 A scenario for a mobile robot at home

This thesis proposes a procedure and the algorithms needed for a robot at home to be operational in a new home. For the robot to be operational, it must be able to localize itself in the home and recognize the pieces of furniture and objects it encounters as it moves around. The procedure starts by having the owner show the robot the objects he wants it to recognize. Each object must be placed in front of the robot in a clear area. The owner turns the object so that the robot gets a view of the object under many viewing angles. The owner can also provide a name for the objects in order to have a common language with the robot. A database of 52 commonplace objects and pieces of furniture was compiled to simulate this demonstration step by the owner. The database contains snapshots of the objects seen under different viewing angles and at different distances. The manipulations implied to capture these images are simple and can be performed by the non-expert owner of a robot at home. These images form the offline database of objects. The offline database contains images taken in somewhat controlled conditions. The objects, in the offline database, are clearly visible, for instance. They are not occluded by other objects. As the robot moves around in the home, it will see these same objects, but in an online context. Online images can suffer from all the problems listed in section 1.3. The algorithms presented in this thesis use the information from an offline database in order to

recognize objects seen in an online context. Validation of the algorithms is done by moving the robot around in a room where a certain number of objects are scattered and measuring the recognition rate. A robot at home should be able to autonomously move around, but remote controlling the robot is one way to reduce the complexity of the overall task. It should be possible to couple the vision algorithms of this thesis to an autonomous exploration module in a future work.

An important aspect of the scenario is the user interface. The user interface is the means of communication between the owner and its robot. The interface has an input and an output mechanism. The input interface is the way the owner can communicate semantic information to the robot. Semantics is the study of meaning and is the basis for communication. Two persons can communicate only if they agree on the semantics first. More concretely, if they give the same meaning to the words they use. For the owner to communicate with its robot, both must have a common semantics. The approach taken here will be to have the owner input semantic information to the robot about the furniture and objects in his home. This will be done in a modest way using a simple interface: a keyboard. Using a keyboard, the owner will provide its robot with the name of the pieces of furniture in its home. For instance, the owner will enter "kitchen trashcan" while having the robot face the actual trashcan so it can associate the appearance of the object to its name.

Another aspect of communication concerns the way the robot can give feedback to the user. This will be a little bit more elaborate and will take the form of a semantic map. A semantic map is a metric map in which some sort of high level information is included. In this scenario, the additional information will be the name of the pieces of furniture the robot encountered while performing its tasks, their location and a snapshot of them. An example of a semantic map is shown in figure 1-2. Semantic map does not mean a map with more information, it means a map in which the information has a structure. The map shown is not a regular map on which object snapshots and labels were overlaid. Each object pictured in the map is a separate entity with its own label, snapshot and position information. The main use of the semantic map here it to reflect the robot's understanding of the environment. It is a way for the user to glimpse into the robot's "mind" and validate that the robot correctly learned what he has taught it. A second reason to use a semantic map is that it is a good way for the robot to store and organize its knowledge for its own use. The semantic maps built in this work are a good starting point to add even more knowledge that will serve the robot for other purposes. An example of this is to store properties of the recognized objects. For instance, if the robot recognizes a light object in front of it, it could store the fact that such an object can be picked up or pushed. This can be useful knowledge for navigation as moving the object out of the way might be the best solution to get to a desired location.

A block diagram overview of the solution to the robot at home scenario proposed in this thesis is shown in figure 1-3.

Depth images provided by an RGB-D camera can be represented as a point cloud for convenience. Point clouds convert the depth value and coordinate of the pixel in the depth image into a 3-dimensional position in space. The point cloud representation is more convenient for operations that work on the point's positions. This is

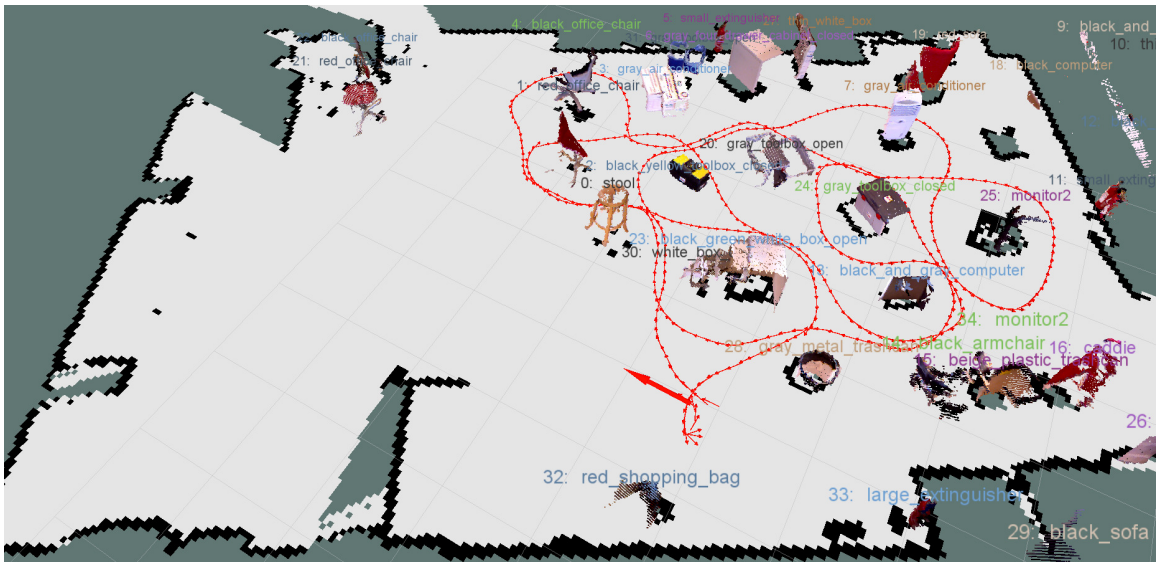

Figure 1-2: A semantic map. The image shows a metric map with semantic annotations. The light region is the mapped regions. It is delimited by a black boundary. The semantic annotations are a snapshot and the name of the objects present in the scene The object images hollow because they are the actual point clouds captured by the Kinect. The red trail is the path followed by the robot to produce this map.

the case for plane detection methods, for instance. Both representations contain the same information. The information from the color can also be added in the point cloud. The terms image and point cloud are used interchangeably in this work.

1.5 Contributions

The goal of this thesis is to develop and integrate the software required for a mobile robot to perform in an object instance recognition scenario. The focus will be on the adaptation of successful computer vision algorithms to the context of indoor mobile robotics. Specifically, the contributions are:

- Compile the ENSTA dataset for object instance recognition in mobile robotics in both controlled and realistic situations
  - This database contains the data to emulate the scenario of section 1.4. It is composed of an "offline" and an "online" part. The offline database contains point clouds of 52 objects and pieces of furniture taken in a controlled environment. These point clouds correspond to what the owner would have to provide to the robot during the learning phase. The online database contains 141 point clouds of a subset of 20 objects from the offline database. The online point clouds were taken as the robot moved around in a room where the objects were scattered and no control was exercised on the object views and lighting of the scene. The online part serves as a validation dataset for algorithms trained on the offline dataset.

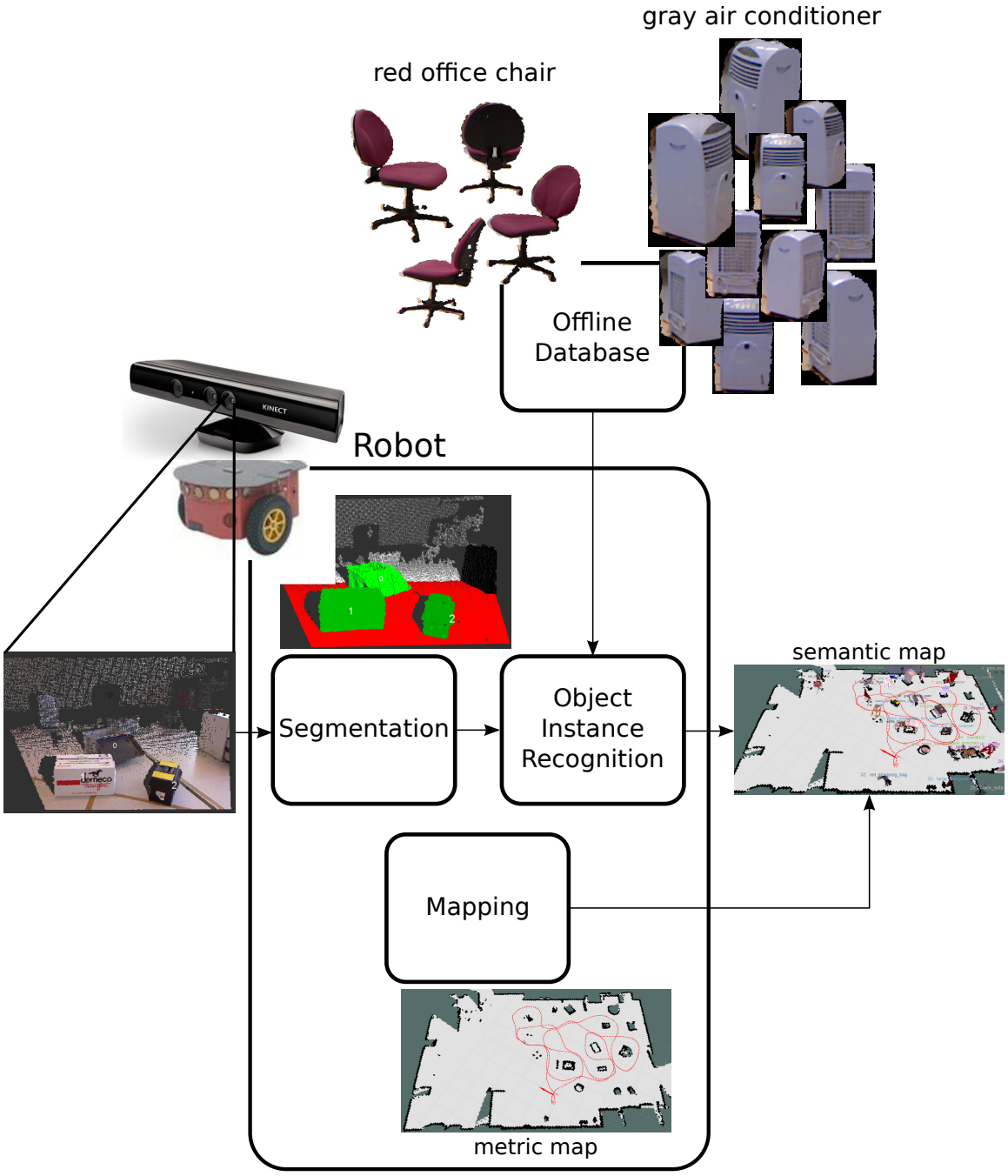

Figure 1-3: Block diagram of the robotic system proposed to solve the robot at home scenario.

- Demonstrate that geometric segmentation for indoor mobile robots works as well as sophisticated probabilistic segmentation

  - Taking advantage of the high structure of indoor scenes, a geometric segmentation algorithm is developed for scene analysis. The algorithm finds, in a point cloud picturing an indoor scene, the structural parts that are the floor and the walls. It also identifies what remains in the scene as separate objects. All the processing is based on geometric considerations. It produces a segmentation which quality is comparable to what a Markov random field approach yields, but for a fraction of the computational cost.

- Show evidence that offline databases give misleading performance measures in realistic scenarios and that the drop in performance can be compensated by combining multiple features and multiple detections

  - Traditionally, computer vision algorithms are trained on a subset of a database and performance results come from testing the algorithm on the remaining examples. The offline part of the ENSTA database is comparable to such a standard computer vision database. The results of experiments conducted on the offline database using the standard scheme are compared to experiments where an algorithm is trained on the offline database and tested on the online database. The comparison shows that an algorithm that performs well on offline data does not necessarily do so on online data.

  - The robotic system developed in this thesis is used as the running demonstration for the Robotics and Computer Vision team at ENSTA ParisTech for more than two years. It is trained on offline data and successfully performs object instance recognition in an online context on a mobile robot.

- Propose a physically grounded metric for clustering in high dimensional space and exploit it in a bounded nearest neighbor search to reduce the number of parameters in the algorithms

  - A computer vision algorithm running on a mobile robot has access to many physical quantities such as the position of the robot in the room. In this thesis, the distance to the object is used to guide an unsupervised clustering algorithm and find appropriate threshold values in a high dimensional space.

  - Clustering and nearest neighbor search algorithms involve a number of parameters such as the number of clusters to create and the number of neighbors to return. The physically grounded metric used in this work eliminates the need to manually set such parameters.

1.6 Structure of the thesis

The rest of the thesis is structured is follows. The state of the art is reviewed in chapter 2. First, different segmentation algorithms are presented. Several segmentation methods from this state of the art will be combined in this thesis. Then, the main concepts behind feature-based image comparison are detailed. Feature-based image comparison is a popular solution for object instance recognition problems. The representation of color in computer vision is reviewed next. The reason why color is a difficult piece of information to use are explained and different representations that help alleviate the problem are presented. The use of histograms as a density estimation method is also detailed. Several distance metrics used to compare histograms are described, as the experiments in this thesis will test these different metrics. A review of the main trends for object recognition on mobile robots is also conducted.

In chapter 3, the hardware and software components of the robots are detailed. The database that was compiled for the robot at home scenario is also described. Also, the performance measures used to evaluate the object recognition algorithms are introduced.

A geometric indoor scene segmentation algorithm for mobile robots is proposed and evaluated in chapter 4. The implementation details and list of parameters for the algorithm are given. The algorithm is tested on an actual robot and also on a segmentation database. The performance of the geometric segmentation algorithm is compared to that of a Markov random field approach.

Chapter 5 describes an object instance recognition algorithm based on a neural network. The chapter details the features used for recognition and the training procedure for the neural network. A method to fuse the recognitions done on different views of the same object is described. The recognition algorithm is evaluated in three series of experiments. A first series uses offline data both to train and test the algorithm. The second experiment simulates a robotics task by training on offline data and testing on online data. The last experiment adds multi-view fusion to the second one.

A nearest neighbor classifier developed as a replacement for the neural network is detailed in chapter 6. The classifier involves a model selection method that uses physical measures to guide the unsupervised clustering of high dimensional data. A multi-modal fusion mechanism for nearest neighbor search is presented. Also, an improvement on the multi-view fusion is proposed. The results of the same three experiments that were conducted in chapter 5 are given.

Finally, chapter 7 reviews the main contributions of this thesis and suggest improvements. The database is compared to other object instance recognition databases. The geometric segmentation algorithm is reviewed in the light of its advantages and drawbacks. The different features used in this thesis are also evaluated and replacements are suggested. The properties of the neural network and the nearest neighbor classifiers are then compared. The chapter ends on suggestions of future work for the object instance recognition algorithm.

Chapter 2

State of the art

This chapter will review the research done in the main areas covered by the thesis. Image processing in general, and object instance recognition in particular, is a lengthy operation. One way to reduce to amount of computation required for object recognition is to conduct a scene analysis first to eliminate uninformative elements from the input. This process is called image segmentation and is reviewed in the first section. Then, the ideas behind feature based object recognition used in image retrieval systems are explained in section 2.2. Feature based recognition involves several key points that inspired most of the object recognition aspect of this work. In particular, the problem of vocabulary creation, which involves unsupervised clustering in a high-dimensional space, is introduced. The multi-modal approach taken in the work involves using color information. Color is obviously a source of useful information for object instance recognition, but is known to be difficult to use in realistic conditions. A review of the most popular way to process color is presented in section 2.3. Finally, most of the features used in this work take the form of a histogram. The main steps involved in the use of histograms are detailed in section 2.4. The review focuses on histogram comparison as it is a tricky step with many possible solutions.

2.1 Image segmentation

Segmentation algorithms use simple rules to split an input into coherent subgroups. The rules are simple and can be quickly applied to yield a first analysis of the input. The aim is that this crude analysis can guide the subsequent processing steps and save time. As processing time seems to be an issue in mobile robotics, this pre-processing step is essential. In this review, two different processes will be distinguished: over-segmentation and foreground-background separation. In the literature, image segmentation can designate any or both of these processes, but the more specific terms are used in this text for clarity. Foreground-background separation finds part of the input that are uninformative so that they are ignored by the following processing step. Over-segmentation is the general process of splitting the input into a reasonable number of subgroups using simple similarity criteria. The review focuses on foreground-background separation because it is a process which greatly benefits

from the strong context priors that exist for indoor mobile robotics. Both types of segmentation will by combined in the indoor scene segmentation algorithm described in chapter 2.

2.1.1 Over-segmentation

The process of over-segmentation takes an input and divides it into distinct parts, or segments. The role of this kind of segmentation is to group the very small pieces of information that compose the input (pixels of an image or points of a point cloud) into fewer, larger groups. The groups formed by over-segmentation might not yet form meaningful entities. Rather, they are composed of points that share a common property. The usefulness of such algorithms comes from the fact that it is likely that points with similar properties should undergo the same processing in the following stages of a pipeline. If this is the case, then the grouping might allow these other stages to process the segments in batch (e.g., in parallel) rather than on a point-per-point basis.

Seeded region growing [2] is an early over-segmentation algorithm. The seeds are the pixels which form the initial regions to grow. They are selected by the user or automatically chosen to lie on homogeneous regions with regard to the segmentation criterion [20]. The algorithm lists the neighboring pixels of all the seeds and processes them sequentially. A pixel is automatically added to a region if it is a neighbor of this region and no other one. If a pixel touches two regions, it is added to the region it is most similar to. Similarity can be based on intensity, color or normal orientation, for instance. An iteration of the algorithm ends when all the neighboring pixels are processed. The neighbors of the newly obtained region are then found and the algorithm is repeated.

Recent over-segmentation algorithm apply similar principles than the seeded regions growing. Two recent methods that are considered to be fast are reviewed here. The popular SLIC superpixels [1] produce segments that are regularly spread in the image. The procedure involves a parameter to control the size of the resulting segments. SLIC superpixels use a variation of the K-means algorithm to group the pixels based on a user defined similarity criteria. The number of computations is reduced by limiting the search radius to the specified size parameter. The result of applying SLIC superpixels with three different segment sizes is shown in figure 2-1. The second over-segmentation algorithm is a graph-based method [22]. Each pixel of the input image is a vertex and neighboring pixels are linked by a weighted edge which reflects the similarity criteria. All the edges in the graph are processed in order of decreasing weight value. Segments are formed by merging vertices if the ratio of inter-segment similarity to intra-segment similarity is small enough.

Over-segmentation methods typically vary by the procedure used to create the segments and the order in which the pixels are processed. Both reviewed methods use a user defined similarity criteria which can depend on any information such as color, position, alignment, etc. This makes over-segmentation algorithms versatile as they adapt to the available information.

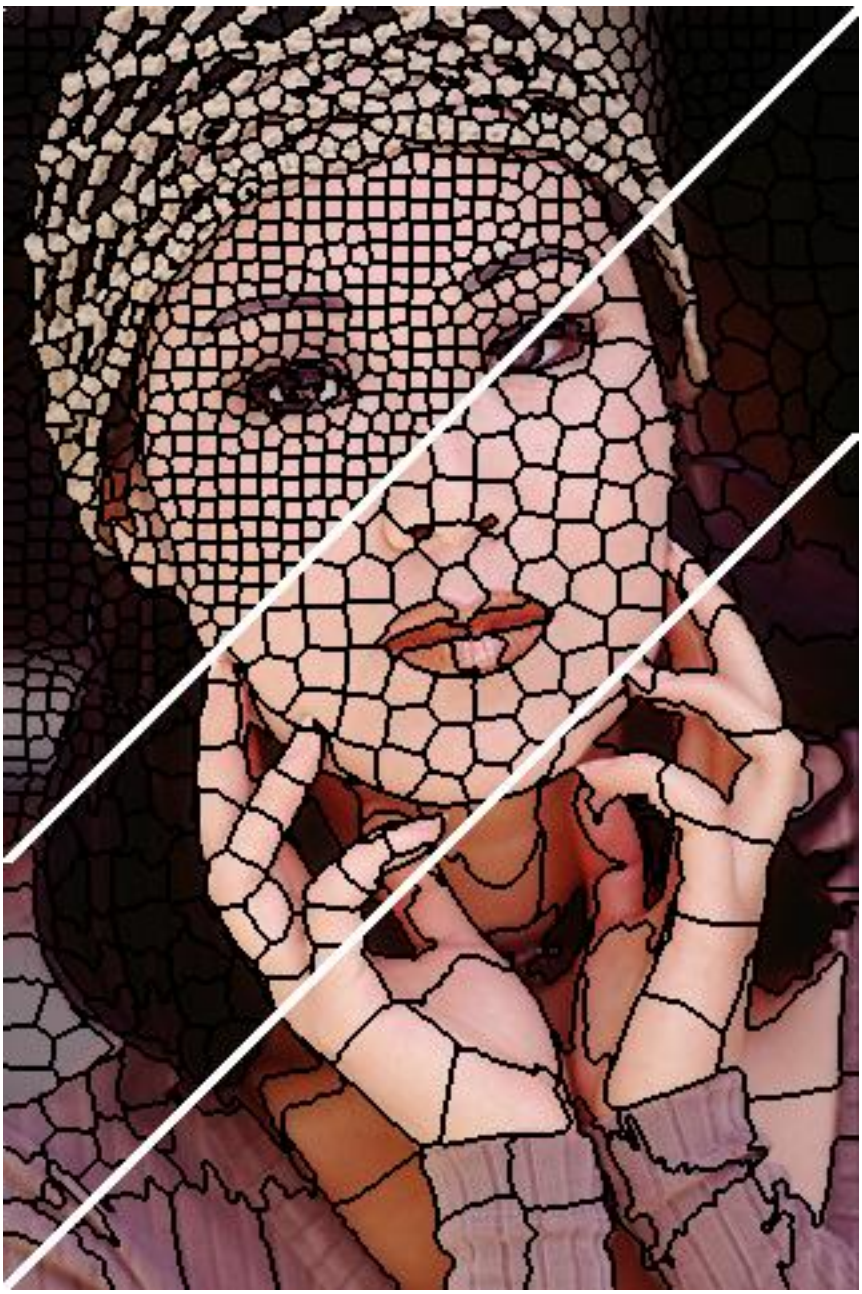

Figure 2-1: SLIC superpixels with three different segment sizes. Viewed 17 November 2015 http://ivrl.epfl.ch/files/content/sites/ivrg/files/research/images/RK_SLICSuperpixels/intro_pics/302003_combo.jpg.

2.1.2 Foreground-background separation

In foreground-background separation, the goal is to determine for each pixel of an image whether it is part of the foreground or of the background. Pixels are labeled as part of the background if they are considered uninformative. They are usually discarded. The foreground pixels are the ones that carry the useful information and are passed to the subsequent processing stages.

In some contexts, it is easier to define properties that characterize the background. In [14], a camera mounted in front of an autonomous car, facing the road, is used for navigation. Before more intricate computations are done, a foreground-background separation step first removes pixels representing the sky. The separation is done by removing all pixels above the horizon line. This is possible because the camera will always be in a position where the sky is in the upper part of the image.

A lot of work focuses on identifying the foreground. This is the focus of object discovery. Object discovery algorithms vary in the knowledge they use to formulate a separation criterion. It is common to use an over-segmentation step beforehand and form object candidates by assembling the obtained segments. This is the case in [28, 53] where convex regions are built based on the assumption that objects in certain context tend to be compact and have a rather simple, convex shape. In [36], objects are formed by combining segments based on six measures: compactness, symmetry, smoothness, local convexity, global convexity and recurrence. It is also common to see objects of interest be surrounded by background. This intuition is used in [46], where regions enclosed into larger parts of the image are considered as foreground. In [89], the input is first split into smoothly curved surfaces. The resulting segments are simple geometric surfaces like planes and cylinders. Then, these segments are grouped into objects by analyzing which surfaces seem to cut through other ones. If one surfaces cuts another one, they are assumed not to be part of the same objects.

Alternately, frameworks like Markov random fields can be used for foreground-background separation [6, 17]. In the previous examples, both a measure and a specific threshold had to be specified for grouping or splitting segments. Markov random fields, once adapted rules are formulated, can be trained to learn the right parameter values from a set of representative images. The solution described in [17] is reviewed here because it will be compared to the segmentation method developed in this thesis (see chapter 4, section 4.4). A Markov random field is a classifier which can assign to every input pixel one of several labels. In [17], indoor scenes are analyzed and the labels are floor, wall and object. Defining a Markov random field at the pixel level would lead to a precise segmentation, but involves too many computations. Rather, SLIC superpixels [1] are first used to produce segments that are similar in appearance. The Markov random field is defined over the obtained superpixels. The Markov random field seeks to find the right label for each superpixel based on their properties, the label of neighboring superpixels and a prior probability for each label. The superpixels properties are: their position in the image relative to the image's center, their shape, color and texture. The influence of neighboring superpixels in the field is weighted by the similarity of their depth measure and of the orientation of their surface normals. The parameters of these diffusion factors are set manually.

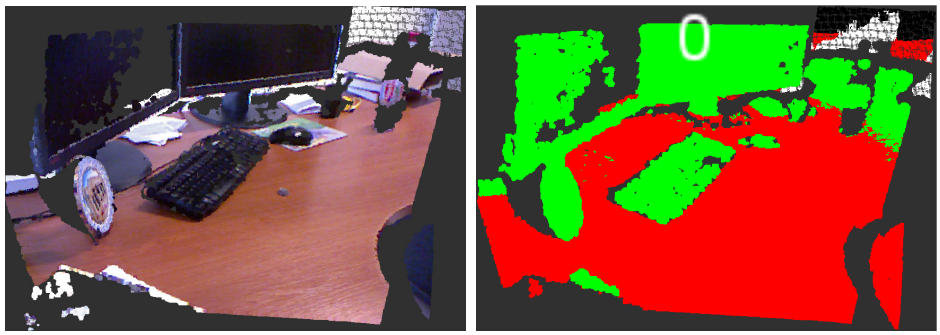

Figure 2-2: Tabletop segmentation of a desk. **Left** The color input image. **Right** The segmented image with support plane in red and objects in green.

Different values are tested leading to weaker or stronger regularization. The rest of the parameters is learned on a dataset. Inference is conducted on a relaxed version of the problem according to the approach in [93]. Using Markov random fields is one way to take advantage of available data to tune a foreground-background separation formulation.

In robotics, a popular foreground-background separation algorithm came about with the increasing popularity of object recognition in table-top settings [31, 88]. In a table-top scenario, objects are laid on a desk, table or any horizontal support plane and must be identified. From such a definition, it is clear that the horizontal support plane, no matter its nature, does not help in the identification of the objects. It can help locate the objects, because they should be in contact with the plane, but pixels belonging to the plane itself can safely be discarded. The support plane in an unknown scene can for instance be detected using a RANSAC algorithm (see chapter 3, section 3.3.1). The point normals can be provided to the detection algorithm to refine the detection to points lying on a plane and having a normal perpendicular to it. It is possible to reduce the size of the foreground even further by removing everything that falls outside the horizontal boundaries of the plane [3]. The boundaries can be obtained by finding a tight enclosing convex hull of the detected plane. Table-top segmentation works very well in most cases, as long as a large enough portion of the support plane is visible. If it is not the case, the detection will most likely fail. An example of the tabletop segmentation of an office scene, centered on a desk, is shown in figure 2-2. The table-top approach can be extended to indoor scenes in general. For an indoor segmentation, the floor acts as the support plane. Then, other important structures such as walls and ceiling can be identified and removed [56].

Another interesting heuristic for foreground-background separation is the detection of change. In the context of foreground-background separation, change can be defined as the difference between a baseline point cloud and the one to segment. Practically, the baseline information can be obtained by an environment reconstruction algorithm. Such algorithms build a stable three-dimensional reconstruction of an environment by averaging several perceptions. This has the effect of removing noise and also to eliminate from the reconstruction moving or transient elements. If such a baseline is available, then the corresponding scene can be subtracted from

an unknown point cloud to remove any stable information. The procedure effectively yields a change detector [23] which can be used for foreground-background separation. Such a segmentation is highly interesting for object recognition because object tend to change or move consistently. Rigid objects, for instance, move and change as a whole and distinct objects tend to move separately. Change detection can thus easily determine the exact segmentation of whole rigid objects. Change detection should have a good estimate of the noise level so that noise does not trigger a detection. This conversely means that a change detector cannot detect elements whose magnitude is smaller than noise level. Finally, the change detector described here needs a baseline reconstructed scene, which might be hard to acquire (due to partial visibility or to computational limitations).

Foreground-background separation can save huge amounts of processing by discarding large regions of the image, but its action is irremediable. Discarding a region containing essential information can lead to a global failure of the system. The utility and safety of a foreground-background separation algorithm depends on how reliable the separation criteria are. When the available information is scarce, the criteria must be formulated in terms of generic considerations. In indoor mobile robotics, strong assumptions can be made about the structure of the input and specific criteria about planes can be used to get a dependable and fast foreground-background separation.

2.2 Feature-based image comparison

An RGB image is a very rich source of information. Take the example of a color image with 640×480 pixels, each of which having three color channels which can take any value in the interval $[0, 255[$. The number of possible configurations is enormous. But every image from this set does not represent something meaningful, and many images represent the same scene or convey the same high level information. Changing the value of a single pixel from an image would most likely not change its meaning, for instance. Yet, changing the value of every single pixel, by rotating or flipping the image, can also keep the image's meaning intact. At the pixel level, it is not clear exactly what changes an image can suffer before its meaning gets altered. Consequently, it is difficult to extract the meaning of an image by analyzing it at the pixel level. Feature-based image comparison introduces four concepts that help alleviate this difficulty. These concepts will be put to use throughout this thesis.

2.2.1 From pixels to features

The most important aspect of feature-based image comparison is the extraction of features. Feature extraction designates the sequence of operations used to transform pixel values into a feature. This section explains why using features is beneficial and briefly describes a few of the most popular features used in computer vision.

The goal of feature extraction is to replace the hard to interpret pixel values into more useful feature values. Interpreting the pixels, or image, means finding the causes that explain why the camera perceived this specific image. The causes include the

position of the camera in the world and the presence in the scene of objects or other elements like walls. A single pixel value has a very high number of possible causes. One of the goal of extracting features is to build from pixel values new quantities that have fewer causes. This is typically done by combining the information from several close-by pixels. One simple way to transform a group of pixels into a feature is to compute a histogram. A histogram approximates the probability distribution function of an image by quantizing the pixel values. A patch of neighboring pixels can be turned into a color histograms [84], for instance. A second result of using several pixels is that geometrical information such as lines and corners can also be encoded. These structures can carry a lot of information about the scene and form what is now called interest points. The most popular interest point descriptor is the scale invariant feature transform (SIFT) [44]. In the SIFT feature, the patch is split into smaller regions and the relative orientation of the gradients in these regions are computed. Lines and corners can also form contours or silhouettes. The histogram of oriented gradients (HOG) [15] feature is one way to encode the silhouette depicted in a patch. The HOG feature also splits the region into cells and computes a histogram of local gradient. The HOG feature however encodes absolute gradient orientation whereas in the SIFT, the local gradients are computed relative to the patch's overall gradient orientation. For this reason, the SIFT is better at encoding interest points, which are small distinctive points of an image or object to recognize. When several SIFT features known to appear in an object of interest are found in an image, the confidence in the presence of the object increases. The HOG feature, on the other hand, describes a precise silhouette at a given scale and orientation which on its own is a strong indication of the presence of the object of interest. Examples of a HOG feature for pedestrian detection is shown in figure 2-3.

These features are successful not only because the set of causes that explain their presence in an image is restricted, but also because they present some form of invariance. Each of these features are designed to be invariant to certain changes. Invariance means that if the input pixel values suffer this change, the feature's output values will not change, or at least not too much. As an example, color histograms are invariant to symmetries, rotations and scaling. Rotating the input patch in pixels will absolutely not change the resulting color histogram. Features based on gradient are invariant to color changes and most illumination changes. The SIFT feature specifically is designed to be invariant to rotations and includes mechanisms to enforce invariance to scale changes. It is the fact that the patch is divided according to a grid aligned with its main gradient that give SIFT most of its rotation invariance properties. The exact orientation of the grid is not chosen ahead of time, but is decided, for each patch, according to its actual contents. Invariance properties can be beneficial, but can also be hurtful depending on the application. Both SIFT and HOG features encode gradients. A gradient is a measure of change and does not retain any information about the magnitude of the input values. This gives gradients in images a certain level of invariance to illumination changes. For object instance recognition, using features which are invariant to lighting conditions is beneficial. Features should be specific to the objects shown in the scene, not to the nature of the light source. However, gradients are also invariant to color changes. This is problematic in object

Figure 2-3: HOG feature for pedestrian detection. **Left** Example of an image of a pedestrian. **Right** A HOG feature computed over many pedestrian images. The silhouette of a typical pedestrian appears in the feature. The silhouette has two heads due to variations in the pose of the pedestrians in the images used to compute the feature.

instance recognition where the object of same shape but different color must be told apart.

In addition to invariance to certain changes, feature extraction can reduce, or rather fix, the dimensionality of the data to work with. The SIFT feature is encoded in 128 values, for instance. This can be useful as many decision making algorithm need that all data samples, or examples, have the same size. When an interest point is encoded by a SIFT feature, the output always has size 128 no matter the number of pixels in the original patch. If a single patch gets encoded by a fixed number of values, a whole image still is represented by an unknown number of values which depends on the number of features present in this image. Vocabularies (see section 2.2.4) are one way to encode whole images using a fixed number of values.

2.2.2 Detector: from dense to sparse

A second concept in feature-based image comparison is that all patches from an image do not carry information of the same quality. This means that it might actually not be worth performing the feature extraction and further processing steps on certain patches. Useless computations can be avoided by running a feature detector prior to the feature extraction step. In this sense, feature detectors act like foreground-background separation algorithms. The difference is that foreground-background sep-

aration uses knowledge about the application whereas a detector is tailored to the specific features used. If a texture detector is used, for instance, patches of an image that do not have any gradient will be discarded as if they were background. It does not mean that uniform regions of an image carry no information. Uniform patches do carry information, but texture descriptors are more sensitive to noise when the gradient is low. It might therefore be better not to extract texture features for these patches. The main purpose of the detector is to select regions containing information that can effectively be encoded by the feature.

Many popular feature detectors trigger on change. The SIFT detector is one of them. It analyzes a scale-space representation of the image's contents obtained by convolving the image with a Gaussian kernel of varying width. The positions and scales where the difference between the output at one scale and the next scale is a local extremum are considered as points of interest. This kind of detectors are best coupled with texture descriptors. An example of SIFT features detected on two very similar but not identical images is given in figure 2-4. The invariances of the SIFT detector allows for interest points to be detected at corresponding places on both images. The maximally stable extremal region detector [49] finds locations where the contents of the image is stable. It analyzes the sequence obtained by thresholding the input image using different threshold values (from the minimum value in the image to the maximum). The detector analyzes the regions formed by the thresholding operation. It finds the threshold values for which the area of the regions is the most stable. These are the maximally stable extremal regions. The maximally stable extremal region detector triggers on patches with stable characteristics. It can effectively be coupled with a color [49] descriptor to encode the contents of the patch and a shape descriptor [25] to encode its contour. Many other interest point detectors exist, like SURF [8] and FAST [65]. See for example [50, 74] for in-depth reviews.

The main advantage of using a feature detection is the reduction of the amount of information to process in further stages. The detector acts as a filter telling whether or not a given patch should be considered or ignored. This leads to the drawback of feature detectors: they can actually reduce the amount of useful information. If the risk of discarding valuable information is too high, or the benefit or feature detection too low, the use of a detector can be omitted. In this case, features can be calculated according to a predefined pattern. Features can be extracted, for example, for every pixel of the image, on a regular grid, or according to a specific pattern that replicates the sampling done by the human eye. Alternately, a single feature can be computed on the whole input image.

2.2.3 From an array to a set

An image is a 2-dimensional array of pixels. The 2-dimensional nature of images reflects the spatial arrangement of the captured scene. It is likely that neighboring pixels in an image come from neighboring elements in the scene. This fact can be exploited when analyzing the content of a single image. When comparing two images, however, the spatial arrangement can hardly be used. A change of viewpoint from one image to another can invalidate any neighbor correspondence. Additionally, scale

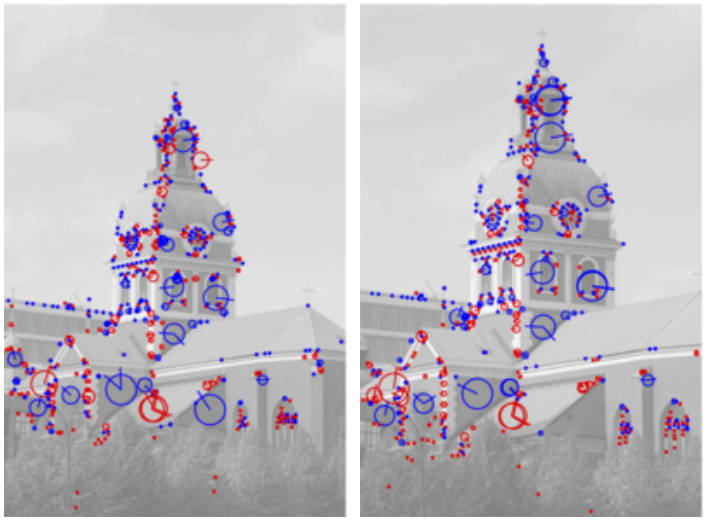

Figure 2-4: SIFT detector applied to two similar images. Many of the interest points detected are common on the two images. Viewed 17 November 2015 http://www.scholarpedia.org/w/images/thumb/5/53/Jakob2-Laplace-500pts.png/200px-Jakob2-Laplace-500pts.png and http://www.scholarpedia.org/w/images/thumb/a/a0/Jakob1-Laplace-500pts.png/200px-Jakob1-Laplace-500pts.png.

changes affect the spatial arrangement between two images. Any of these changes can cause two neighboring pixels in one image to be far from each other in a second image. For comparison purposes, it is common to treat images as sets rather than arrays [13]. The spatial information is discarded and the images are compared solely based of the values present in the sets. This is usually referred to as the representation of images as bags. This technique stemmed from information retrieval used in text analysis [78].

Using a bag representation, two images can be compared using a measure of similarity over sets. As an example, the intersection measure counts the number of common elements between two sets. The intersection of sets $0, 2, 2, 3$ and $1, 0, 1, 0$ is 1 because each set has one element of value 0. It does not matter where this element is in the sets. It is however not very informative to apply this procedure to pixels because two completely different images can have a very high number of common pixel values.

The bag representation becomes powerful when applied to feature values. Feature values are designed to be much more informative than individual pixels. Features tend to be specific to the presence of certain shapes or textures in the images. Furthermore, most features internally contain some spatial information from the image, since they are computed on patches of neighboring pixels. This last fact partially counterbalances the loss of spatial information that happens from using the bag representation. It is not reasonable however to measure the similarity of two sets of features by computing their intersection as done in the previous example. Indeed, the intersection finds in the sets elements that are strictly equal. The equality oper-

ation is not suitable for feature comparison. No matter how well-designed a feature is and the level of invariance it shows, noise and other parasitic effects are very likely to affect its calculation. Two features encoding the same interest point can thus be slightly different and fail to be considered as equal in the intersection comparison. Features should be compared using a distance metric which measures how different they are with real-valued number. The choice of a distance measure is not straightforward and section 2.4.3 is dedicated to this subject. The L_2 distance is often used [44, 13, 58], but the squared chord distance recently was suggested as a replacement [5].

2.2.4 Vocabularies: from values to representatives

Considering every possible value of every single pixel leads to an extremely large number of images. Vocabularies exploit the idea that it is not necessary to be able to distinguish every single of the possible images from each other. Many images from the set of all possible images are very similar to each other. If a certain level of imprecision is acceptable, a single image could be used to represent each of these subsets of almost similar images. When an image from one of the represented subsets is encountered, it is replaced by the corresponding representative. These representative images are called words and they form a vocabulary. In practice, vocabularies are not built on images, but on features. This strategy is called vector quantization of features because any real-valued feature gets transformed into one element from the finite, discrete set that is the vocabulary [79, 55]. Also, the practical way to attribute an input feature to the corresponding word is not to check if it is part of the set of real-valued features represented by this word. It is much more convenient to define a similarity measure and find the one word from the vocabulary which is most similar to the input feature.

Using vocabularies can make many operations much more simple than if they were applied directly on real-valued vectors. This is because the size and contents of the vocabulary are fixed once it is created. This gives two possible ways to represent an input feature. First, it can be substituted with its corresponding word. Second, a binary indicator vector can be used. The indicator vector is a one-shot binary vector of the same size as the vocabulary which tells which element from the vocabulary represents the input feature. The elements of the indicator vector have value 1 if the corresponding word from the vocabulary represents the input feature and value 0 otherwise. Using indicator vectors, two features can be compared by performing a vector multiplication. The result is 1 if the features are equal, that is, if they are represented by the same word, and 0 otherwise. Also, a set of features can be represented by the vector sum of the indicator vectors of the features. A whole image can be encoded in this way by summing the indicator vector of all the features extracted from this image. Summing several indicator vectors yields a histogram of word occurrence. The histogram of occurrence of an image reports how many times each of the words from the vocabulary was matched to one of the features extracted from this image. The fact that the contents of the vocabulary is fixed has other benefits. The space covered by the words in the vocabulary can be analyzed to accelerate further processing. In this work, the words can be stored in a data

structure such as a kd-tree and an octree so that searches for nearest neighbors can be performed more quickly [55]. See chapter 3, sections 3.3.2 and 3.3.3 for more information on kdtrees and octrees. Alternately, the vocabulary can be stored as an inverted index, a common procedure in bag of words representation. The inverted index stores, for each word from the vocabulary, the list of images from the training set in which the word is found [98].

Creating a vocabulary from a list of features, extracted from a given set of images, is usually done using an unsupervised clustering algorithm. The process of summarizing a set of features using a smaller number of representatives is a clustering operation. A desired property of the representatives is for them to have a statistical distribution in space that fits that of the initial set. This is what unsupervised clustering algorithms aim to produce. Some clustering algorithms do so by analysing the full $F \times F$ similarity matrix obtained by computing the distance between each of the F input features. This is the case of spectral clustering and hierarchical agglomerative clustering, for instance. While the use of the similarity matrix ensures that the statistics of the input set of features can be preserved, it comes at a high computational price and is intractable when the F is large. The most effective method for vocabulary generation is k-means [13, 79]. In the k-means algorithm, the number of desired representatives, or cluster center, k is set by the designer. There are different initialization schemes for k-means, such as randomly selecting k of the input features. A more elaborate initialization, called k-means++ [7], chooses the first cluster center randomly. Then, the other $k-1$ centers are iteratively chosen also at random, but by increasing the probability to pick a feature which is far from any already selected center. Once it is initialized, the k-means algorithm assigns all the features from the input list to the closest center. The centers are then updated as the center of mass of all the features assigned to them. This two-step process repeats until the assignment stabilizes. This simple algorithm works well in most cases, but as the number of input features F becomes very large, even it becomes intractable. One alternate solution is hierarchical k-means. Hierarchical k-means [55] clusters the input features using $n << k$ centers using the standard k-means algorithm. Then a separate k-means is run on the features assigned to each cluster, producing a total of n^2 clusters. This process is repeated until the desired number k of clusters is reached. This happens after i steps if $k = n^i$. There always is a possibility that k-means algorithms produce bad clusterings due to the random nature of the initialization. A clustering is bad when the statistical properties of the cluster centers do not approximate the initial distribution very well. Hierarchical k-means is even more critical in this regard because any sub-optimal clustering happening in the first levels will propagate to the final clustering. Another solution is approximate k-means [58]. The approximate k-means algorithm uses an approximate nearest neighbor search algorithm to find the correct assignment of features to cluster centers. Approximate k-means tends to produce results very similar to exact k-means, but for a fraction of the computational cost. Still, bad initialization can lead to sub-optimal clustering. There is no practical solution to guide the initialization in order to ensure the resulting clustering will optimally approximate the initial set of features. A second drawback of many popular clustering algorithms is that they implicitly build 'spherically-shaped' clusters, which

is oftentimes badly adapted to the actual geometry of features' distribution in space.

2.3 Color in computer vision

Color is a very informative piece of information, but it is not a dependable one. Using color in an object recognition system requires caution and specific processing. The considerations described in the following sections have guided the design of the color based features of this thesis.

2.3.1 Models of photometric changes

Color is an intuitive piece of information to use for object recognition. It seems obvious that it is useful to use color when comparing two objects. Color however is difficult to deal with, because it is not an intrinsic property of an object. The perceived color of an object depends on the color, intensity and position of light sources, the reflectance of the object and the position and sensitivity of the sensor. Simply put, the perceived color $\left(R^p, G^p, B^p\right)^T$ of an object is a possibly complicated function of its intrinsic color:

$$\begin{pmatrix} R^p \\ G^p \\ B^p \end{pmatrix} = f \begin{pmatrix} R^i \\ G^i \\ B^i \end{pmatrix} \tag{2.1}$$

The influence of the numerous factors can be described by physics, but the laws involved are too complex to use for a general model in computer vision [51]. Rather, simplified models exist that give a form to the function $f()$ depending on which factors are considered. These models are best understood when put in relation to the type of changes that can happen between two perceptions of the same object. Changes related to light intensity are characterized by a single factor applied to all color channels. When the factor is different for the channels, then it is a change in the light's color. The following list is based on the analysis in [90]:

If color information is to be used for object recognition, one has to find ways to be invariant to some of the possible photometric changes the object's perceived color can suffer. However, more invariance does not mean better recognition potential. More invariance properties only means it is more likely that an algorithm's output remains the same for a given object even when the light's properties change. But it can also mean that the exact same output will be obtained even if the object changes. For instance, being invariant to light color change also means being invariant to object color change. A red office chair and a blue version of the same office chair will look exactly the same for a light color change invariant descriptor. The following sections introduce color representations that provide invariance to some of the changes stated in table 2.1.

Light intensity scaling	$\begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix} \begin{pmatrix} R^i \\ G^i \\ B^i \end{pmatrix}$
Light intensity offset	$\begin{pmatrix} R^i \\ G^i \\ B^i \end{pmatrix} + \begin{pmatrix} o \\ o \\ o \end{pmatrix}$
Light intensity scaling and offset	$\begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix} \begin{pmatrix} R^i \\ G^i \\ B^i \end{pmatrix} + \begin{pmatrix} o \\ o \\ o \end{pmatrix}$
Light color scaling	$\begin{pmatrix} s_R & 0 & 0 \\ 0 & s_G & 0 \\ 0 & 0 & s_B \end{pmatrix} \begin{pmatrix} R^i \\ G^i \\ B^i \end{pmatrix}$
Light color scaling and offset	$\begin{pmatrix} s_R & 0 & 0 \\ 0 & s_G & 0 \\ 0 & 0 & s_B \end{pmatrix} \begin{pmatrix} R^i \\ G^i \\ B^i \end{pmatrix} + \begin{pmatrix} o_R \\ o_G \\ o_B \end{pmatrix}$

Table 2.1: Changes in lighting conditions

2.3.2 Color representations

This section will introduce the most commonly used color representations, or color spaces, as well as some useful color transformations. These different representations serve to expose different properties of colors or to provide invariance to certain photometric changes. An illustration of the first three color representations described in this section, RGB, HSV and opponent colors, is shown in figure 2-5.

RGB

The RGB color representation is the most widely used to encode images. RGB stands for red, green and blue. In RGB, each pixel of an image is assigned a triplet of values between 0 and 256 (excluding 256) which indicates the level of red, green and blue in the perceived light. A value of 0 means there is no energy in the wavelengths corresponding to that channel. A pixel with value $\{0, 0, 0\}$ is black whereas a pixel of value $\{255, 255, 255\}$ is a bright white. Pure red, for example, is encoded as $\{255, 0, 0\}$. These examples point the fact that the value in the red channel is not a direct indicator of how red the pixel is, as a white pixel also has a value of 255 in the red channel. The RGB representation does not provide any invariance. This makes color analysis in RGB representation complicated, and motivates the use of different color representations. The RGB representation is shown in figure 2-5.

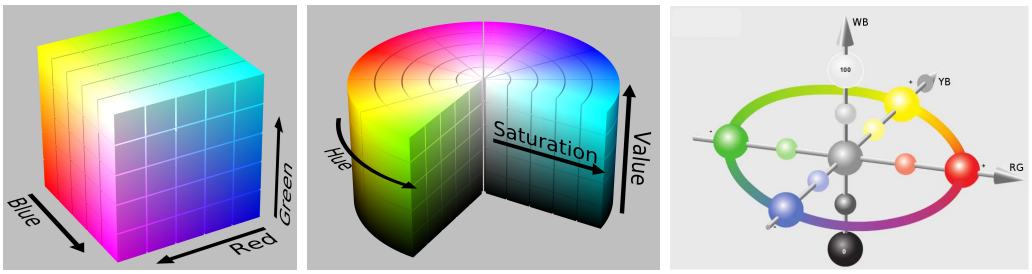

Figure 2-5: Illustration of the RGB, HSV and opponent color representations. **Left** The RGB representation. **Middle** The HSV representation. **Right** The opponent color representation with RG (red-green), YB (yellow-blue) and WB (white-black) axes.

HSV

The HSV representation is a step towards separately encoding color properties such as lightness and colorfulness. HSV stands for hue, saturation and value. The hue scale indicates the color (red, purple, reddish-orange, etc.) without any indication whether it is light or dark, sharp or dull. The value scale indicates the brightness (from black to white) and the saturation the colorfulness (from pure color to gray). The separation of the hue means that the same color, in a darker or a brighter version, will always have the same H value. This gives the H channel of HSV invariance to light intensity scaling and offset. The HSV color space can be seen in figure 2-5. The H, S and V values are calculated as follows:

$$H = \begin{cases} undefined & \text{if } max - min = 0 \\ \frac{G-B}{max-min} \mod 6 & \text{if } max \text{ is } R \\ \frac{B-R}{max-min} + 2 & \text{if } max \text{ is } G \\ \frac{R-G}{max-min} + 4 & \text{if } max \text{ is } B \end{cases} \tag{2.2}$$

$$S = \begin{cases} 0 & \text{if } max = 0 \\ \frac{max-min}{max} & \text{otherwise} \end{cases} \tag{2.3}$$

$$V = max \tag{2.4}$$

where max and min are the maximum and minimum values between R, G and B. The modulus operation "$y \mod x$" successively adds/subtracts x to/from y until the result lies in the interval $[0, x[$. Note that using these formulae, the range is $[0, 6[$ for the H channel, $[0, 1]$ for the S channel and $[0..256[$ for the V channel.

Equation 2.2 shows that H is undefined when S is 0, which means that the color of the pixel is undefined in this case. Even if the hue is properly defined as soon as $S > 0$, its value is unstable with low values of S. This is because low values of S indicate that all three RGB values are close-by, meaning that the channel with maximum value can easily change. The value of S actually indicates the confidence in the value of H [90]. The higher the S, the more colorful a pixel is and the more stable

the value of H. Another particularity of the H channel is that it is cyclical, meaning that close-by values indicate close-by colors, but the lowest and highest values also encode the same color (red, according to equation 2.2). The main advantage of the H channel in HSV is that it is invariant to light intensity scaling and offsets. However, the H channel alone does not give any indication of whether an object's color is white or black or gray, as the H is undefined in these three cases.

Opponent colors

The HSV representation introduced the concept of separately encoding different properties of a color. Opponent colors encode mostly the same properties, but using some specificities of the human visual system. Color representations based on the opponent colors theory split the hue into two channels, and encode luminosity in a third one. The two color channels are inspired by the human eye. In the human visual system, some cells are sensitive to a color scale from red to green while other cells are sensitive to the scale from yellow to blue. Each of these two pairs, red-green and yellow-blue, form color opponents. A third type of cells is sensitive to the brightness, which can be regarded as a scale from white to black. A visual representation of the opponent color space is shown in figure 2-5. A simple way to compute opponent colors from RGB values is:

$$RG = \frac{R - G}{\sqrt{2}} \tag{2.5}$$

$$YB = \frac{R + G - 2B}{\sqrt{6}} \tag{2.6}$$

$$WB = \frac{R + G + B}{\sqrt{3}} \tag{2.7}$$

The RG (red-green) and YB (yellow-blue) channels are invariant to light intensity offset, but not to scaling. The WB (white-black) channel has no invariance. The scaling coefficients $\sqrt{2}$, $\sqrt{6}$ and $\sqrt{3}$ were obtained by performing principal component analysis on natural images [67].

More intricate calculations exist to get opponent color values for which the scaling corresponds with human perception. This is done in Lab color representations [86] and the likes (Hunter Lab, CIELab, CIEL*a*b*, etc.). This however comes at the price of increased computational cost and is aimed at color description in human-centric applications.

2.3.3 Transformed color representation

Transformed color indicates in this thesis the process of normalizing color information to zero mean and unit variance [51]. The transformed color representation, unlike the other representations described in this section, is not a property of a single pixel. Rather, it describes the pixels' color with regard to the statistics of a set of pixels S

by applying the following transformation

$$R_T = \frac{R - \hat{m}_R}{std_R} G_T = \frac{G - \hat{m}_G}{std_G} B_T = \frac{B - \hat{m}_B}{std_B} \tag{2.8}$$

where $\hat{m}_x$ and std_x respectively are the mean and standard deviation for color channel x of pixels in the set S. Because it describes a pixel in relation to other pixels, the transformed color representation really is more of a feature than a color representation. The transformed color representation can be applied to using any color space as input. As the transformation already yields invariance to color scaling and offsets no matter what input space the input is, the transformed color representation is usually used on the most easily available color values. Converting the input from an RGB camera to a different color representation before transforming it according to equation 2.8 would involve more computations without providing more invariance properties.

2.3.4 Color moment invariants

Color moment invariants are transformations of RGB information that can lead to geometric and photometric invariances. Color moments that have been demonstrated to work for object recognition are combinations of generalized color moments of degree lower than 2 and order lower than 1 [51]. The generalized color moment of degree $a + b + c$ and order $p + q$ is

$$M_{pq}^{abc} = \int \int x^p y^q R(x,y)^a G(x,y)^b B(x,y)^c dxdy \tag{2.9}$$

where $R/G/B(x,y)$ is the value for the R, G or B channel of pixel with coordinates (x,y).

Moments of degree 0 only contain spatial information and are invariant to any photometric change. Conversely, moments of order 0 do not contain any spatial information and are invariant to geometric changes. By judiciously combining moments of different degrees and orders, it is possible to obtain measures which presents a desired invariance, they are called moment invariants. These invariants are classified based on whether or not they are tolerant to geometric changes and the type of photometric change they are invariant to. The moment invariants only apply for planar surfaces and far away light sources. The geometric changes considered in [51] are affine transformations. The photometric changes include all the ones stated earlier plus the more general light affine change defined as:

$$\begin{pmatrix} s_{RG} & s_{RG} & s_{RB} \\ s_{GR} & s_{GG} & s_{GB} \\ s_{BR} & s_{BG} & s_{BB} \end{pmatrix} \begin{pmatrix} R^i \\ G^i \\ B^i \end{pmatrix} + \begin{pmatrix} o_R \\ o_G \\ o_B \end{pmatrix} \tag{2.10}$$

From all the moment invariants of a given category, it is possible to extract a list of independent moments invariants. All other moment invariants of this category are function of the independent ones. For the category of invariance to geometric

changes and light color scaling and offset, for instance, there are 18 independent moment invariants.

2.4 Using histograms for density estimation

Density estimation is the process of estimating a probability density function based on data samples. A histogram is built by first defining the domain to which it will be applied. That is, defining a minimum and maximum value for each dimension of the input data. Then, this interval is divided into a certain number of sub-intervals, called bins. The width of the bin allow a compromise between precision of the estimate and memory usage. Once the limits and bin width for each dimension is chosen, each bin is assigned a value equal to the number of samples from the input data which lies inside the interval covered by that bin. If all samples were generated by the same stochastic process, the resulting histogram effectively represents an estimate if this process's probability density function.

Histograms also allow some flexibility when representing multi-variate data. It is straightforward to chose whether the different variables should be represented as a multi-dimensional histogram or as the concatenation of several one-dimensional ones. The multi-dimensional representation is the right choice when the variables are dependent, but the memory to store the resulting histogram usage increases exponentially with the number of dimension. If the variables are independent or if the exact estimation of the joint distribution of the variables is not required, then the concatenated one-dimensional histograms is preferred.

2.4.1 Choosing parameters for histograms

There are two parameters for histograms, the limits and the bin width. The limits tend to be the most simple parameter to determine. Sometimes, the data is naturally bounded to lie in a given interval. In this case, it is usually a good idea to choose this interval as the limit of the histograms. If data is not naturally bounded, then the histogram's limits can be chosen by inspecting data. Data inspection can reveal that sample values never or rarely overpasses a certain range. The choice of the exact limits values is then arbitrary. If the limits are chosen such that some samples fall outside the histogram, these samples can either be ignored or their value clipped to the histogram's interval.

The bin width is more difficult to choose. The first consequence of choosing a bin width is to fix the memory footprint of the histogram, no matter what function it should approximate. Reducing the bin width will increase the memory requirement. Thus, the bin width should be as large as possible. Increasing the bin width, though, has a filtering effect. A given function approximated with a histogram with smaller bin width will show much more details. It is possible to determine an optimal bin width if the statistical distribution of the function to approximate is known. For example, a procedure to determine the optimal bin width to approximate Poisson distributed data is detailed in [76]. If the distribution is unknown, choosing the

appropriate bin width then is a chicken-and-egg type of problem as the shape of the function to approximate should be known for the right bin width to be chosen but the function cannot be estimated without choosing a bin width. There are rules of thumb to chose a bin width based on a sub-sample of data points [26, 73]. These methods respectively use the inter-quartile range and standard deviation of the data sample as an estimate of the dispersion of the data. Both rules tend to perform poorly if the underlying distribution is multi-modal, however. Another strategy to determine the appropriate bin width is to have it match the noise level of the data. This method does not ensure an optimal memory usage, as the bin width might end up being much smaller than necessary to capture the details of the function. But at the least, it will ensure that the noise naturally gets filtered out by the quantization due to the process of compiling the histogram. Finally, one practical way to choose the bin width is to take the maximum possible number of bins determined by the computational resources and the available memory. This method gives the most precise histogram given the physical limits of the system.

2.4.2 Aliasing in histograms

When compiling a histogram, it is possible that samples with very similar values fall into two adjacent bins. This is called aliasing. Aliasing happens naturally in histograms because of their discrete nature. It comes from the truncation, or discretization, that happens when a real-valued number is associated with a discrete bin. A simple way to rid a histogram of aliasing effects is to use spreading, or interpolation. Interpolation is used during the compilation of the histogram. Instead of counting the number of samples falling into a given bin, each sample contributes to the bins that cover intervals close to its value. One simple and popular interpolation scheme is linear interpolation [15]. To illustrate linear interpolation, take the example of a one-dimensional histogram. In this case, each sample contributes to the bin it falls into and also to the two bins adjacent to this one. The value which is contributed to the adjacent bins is proportional to the distance from the sample's value to the opposite bin boundary. As a concrete example, a sample of value 1.25 is compiled into a one-dimensional histogram with limits 0 and 3 and bin width 1. The sample will contribute 1 to bin 1, $|1.25 - 2| = 0.75$ to bin 0 and $|1.25 - 1| = 0.25$ to bin 0. Note that the contribution to the center bin is equal to the sum of the contributions to the two adjacent bins and the total contribution to the histogram is 2^{dim}, where dim is the number of dimensions in the histogram. This scheme applies to histograms of any dimension. In a two dimensional histogram, the contribution to adjacent bins thus is proportional to the area opposite to the bin. This is shown in figure 2-6.

Note that this spreading strategy has no parameter. The width of the spreading always is equal to three bins, in every dimension. This makes implementing this method very simple. Considering that the interpolation has a low-pass filtering effect, it also means that the amount of detail lost in the process is not parametrizable.

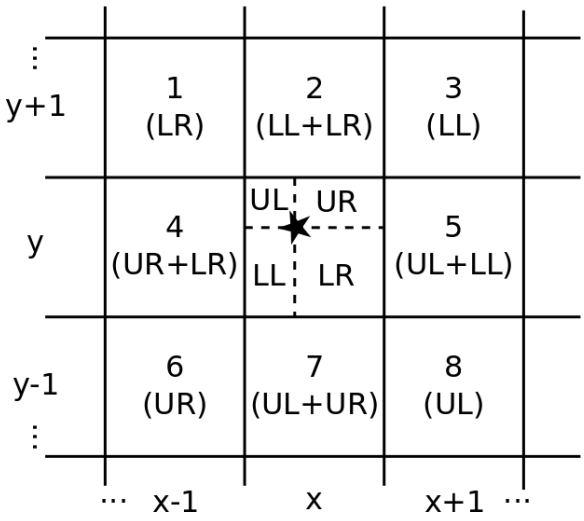

Figure 2-6: The linear interpolation of samples to a two-dimensional histogram. The sample falls into bin (x,y) and contributes to this bin's eight neighbors. The star shows the sample's exact value. UL, UR, LL and LR represent the areas of the quadrants formed by the element in the bin. Bin (x,y) is increased by 1, bin (x-1,y-1) is increased by LR, bin (x,y-1) by LL+LR, etc.

2.4.3 Comparing histograms

Histograms in this work are used to encode images in order to compare them and perform object recognition. The actual comparison thus happens between histograms and a comparison scheme must be defined. There are many different ways to compare histograms. Histograms have a structure comparable to vectors and matrices. Any vector or matrix distance can be used to compute the distance between two histograms of compatible dimensions. Furthermore, histograms of any size can be seen as a point in space and point distances such as the Euclidean distance can also be used. Histograms also are widely used in probability distribution estimation and probabilistic measures such as the logarithm of likelihood are often used. All the previous measures are bin-to-bin measures: they compare the content of corresponding pairs of bins in two histograms. Some similarity measures specific to histograms also exist. They usually are cross-bin measures. A cross-bin metric compares corresponding bins and also close-by ones to measure the shift that might exist between two histograms. Cross-bin measures make sense in histogram comparison because noise in the input data translates into shifts in the histogram. This section will review some bin-to-bin measures first and then introduce two cross-bin measures. Although not all of the measures reviewed are proper distance measure, the word distance will be used equivalently to similarity (or dissimilarity) measure.

Bin-to-bin distances

Four bin-to-bin distance measures will be reviewed: the L_2, χ^2, Jeffreys and squared-chord. All of them are isotropic, meaning they treat all dimensions equivalently. The

equations will be given to compute the distance between histograms P and Q of equal size N, with P_i denoting the i^{th} element of P. The distances measures are named according to [11]. The four distances reviewed here were chosen because they present different properties that might lend themselves well to histogram comparison. Many more distance measures present the properties that will be discussed here, but not all of them can be reviewed here. The four chosen measures are among the most simple representatives for each property. The properties will be introduced when describing the distance measure that represents it.

The L_2 distances is computed according to equation 2.11.

$$d_{L_2}(P,Q) = \sqrt{\sum_{i=0}^{N} (P_i - Q_i)^2} \tag{2.11}$$

The L_2 distance measure is widespread, easy to understand and fast to compute. It is a symmetric distance, meaning that P and Q can be interchanged in the equation. It gives more importance to larger elements in the difference $P - Q$. This means that the L_2 distance is larger when some elements in the difference $P - Q$ are larger than others compared to the case where all the elements have similar values. Also, the L_2 distance only considers absolute differences. That is, it does not consider the magnitude of the values of P and Q, only the difference between them. This means that the L_2 distance between very large values will be the same as the distance between very small values if the difference $P - Q$ is equal.

The χ^2 distance adds a weighting to the L_2 measure which makes it dependent on the magnitude of P and Q. Each square difference from the sum in the L_2 distance is weighted by the inverse of the sum of the two values, yielding equation 2.12. This has the effect of giving a larger weight to small differences when they result from comparing small values. This property is the reason why the χ^2 distance is included in this list. It is symmetric and only works with positive values. The χ^2 distance between two elements P_i and Q_i cannot be larger than the largest element. It is exactly equal to the largest element when the other one is equal to 0. If both elements are 0, the χ^2 distance is 0.

$$d_{\chi^2}(P,Q) = \sqrt{\sum_{i=0}^{N} \frac{(P_i - Q_i)^2}{P_i + Q_i}}, \text{ where} \tag{2.12}$$

$$\frac{(P_i - Q_i)^2}{P_i + Q_i} = 0 \text{ if } P_i = 0 \text{ and } Q_i = 0 \tag{2.13}$$

Many distances such as the Kullback-leibler, Jeffreys and log-likelihood involve a logarithm function that measures the order of magnitude of the ratio between the compared values. This review uses the Jeffreys distance, computed according to

equation 2.14 as the representative of this class.

$$d_J\left(P,Q\right) = \sum_{i=0}^{N}\left(P_i - Q_i\right)\ln\frac{P_i}{Q_i}, \text{ where} \tag{2.14}$$

$$\left(P_i - Q_i\right)\ln\frac{P_i}{Q_i} = \begin{cases} 0 & \text{if } P_i = 0 \text{ and } Q_i = 0 \\ \inf & \text{if } P_i \neq 0 \text{ and } Q_i = 0 \text{ or } P_i = 0 \text{ and } Q_i \neq 0 \end{cases} \tag{2.15}$$

The presence of the logarithm gives even more importance to the relative difference between P and Q than the χ^2 distance already does. To the point that when any element of P is 0 while the corresponding one in Q is not (and conversely), the resulting distance is infinite. This makes sense in a probabilistic setting, where the occurrence of a 0 means that an event is absolutely impossible. But in histogram comparison, this is not acceptable and is avoided by adding a very small offset to zero-valued bins before computing the distance. Jeffreys distance is not symmetric.

The squared-chord distance does not present any additional property when compared to the other distances cited here. It is of interest because it can be seen as a modification of the L_2 which confers it the relative difference weighting property. The squared-chord distance is computed as follows

$$d_{SC}\left(P,Q\right) = \sum_{i=0}^{N}\left(\sqrt{P_i} - \sqrt{Q_i}\right)^2 \tag{2.16}$$

The square-chord distance combines the speed of the L_2 distance while presenting the interesting properties of the other distances. The absence of the external square root is inconsequential, it is the square root applied to the elements of P and Q that changes the properties of the measure. It is the suggested distance to use when comparing SIFT features for image retrieval [5]. It is a symmetric measure, that is only valid if $P_i \geq 0\ \forall i \in [1..N[$ and $\sum_{i=0}^{N} P_i \leq 1$.

Cross-bin distances

All the bin-to-bin distances have a major flaw: they do not take into account the fact that close-by values in a histogram might have a similar meaning. When using a histogram to approximate a function, consecutive bins encode a very similar information, namely the value of this function at different by close-by positions. Cross-bin distances take this information into account.

The earth mover's distance [66] is a cross-bin distance which takes it root in the transportation problem. It computes the amount of effort required to transform one histogram into the other by moving the values inside them. The effort is computed as the total of the product of units of histogram values (the earth) and units of bin distance (ground distance) that result when transforming one histogram into the

other. Equation 2.17 shows how to compute the earth mover's distance.

$$d_{EMD}\left(P,Q\right) = \frac{\sum_{i=0}^{N}\sum_{j=0}^{N} f_{ij}d_{ij}}{\sum_{i=0}^{N}\sum_{j=0}^{N} f_{ij}}, \tag{2.17}$$

where f_{ij} is the amount of histogram values to move from bin i to bin j and d_{ij} is a measure of the distance from bin i to bin j. The Earth mover's distance requires the definition of a distance measure, such as any bin-to-bin distances discussed in the previous section. The distance can also be an arbitrary measure of similarity tailored for the objects being compared.

The diffusion distance [43] is a cross-bin distance which makes an analogy to heat diffusion. It simulates the diffusion of the values in the histogram by successively smoothing the histogram with a Gaussian kernel and downsampling it by a factor 2. It is computed using equation 2.18.

$$d_{diff}\left(P,Q\right) = \sum_{l=0}^{L} |d_l|_1, \tag{2.18}$$

where $d_0 = P - Q$, d_l is the smoothed and downsampled version of d_{l-1} and $|d|_1$ is the L_1 norm of d. The Gaussian kernel has a variance σ which must be adapted to the application.

2.5 Trends in object instance recognition algorithms for mobile robots

A robot is an assembly of a large number of elements ranging from electronics to motors, sensors and algorithms. It is very difficult to evaluate whether or not a certain algorithm that works well on a robot would be as effective for a different one. Furthermore, the efficacy of a method depends on the context in which it was used. In robotics, the context is the world in which the robot behaves. However, some methods are much more popular than others and can be trusted to perform well in a variety of contexts. For a robot to perform object recognition, it has to implement most of the aspects that were reviewed in this chapter. This section highlights a few examples of object recognition robots and gives an overview of how each element is integrated to yield a functional system. This review will follow the same order as the first sections of this chapter (segmentation/detection, features, comparison/classification). A final section summarizes other technical aspects of the algorithms like the number of objects to recognize and the training procedure.

It is generally known that object recognition is a computationally expensive process. The first step of most object recognition algorithms on mobile robots aims at reducing the size of the input image. Both segmentation and detection algorithms achieve this goal. When a segmentation procedure is applied, it is generally simple.

Some examples are the color thresholding algorithm in [71] and the geometric segmentation in [56]. Most of the time, when a detector is used, the SIFT detector is preferred [82, 71, 80, 25, 21]. The SIFT detector has the benefit of choosing points from an image is a way that is invariant to rotations and scaling. The Harris and SURF detectors have such a property and are sometimes used on mobile robots [71]. Another way to reduce the overall computations is to choose random points from the image as is done in [9]. Saliency algorithms can also help rating different regions of an image as informative or uninformative. In [4, 25], a saliency algorithm based on color is described. The red-green, yellow-blue and intensity channels they use recall the opponent color representation of colors described in section 2.3.2.

Features are used to encode an image or a patch prior to classification. Features must be computed rapidly and be stable under the numerous variations that can occur during the operation of a mobile robot. Here again, the SIFT feature is very popular [82, 80, 25, 21]. SIFT features only encode intensity information and they are sometimes coupled with color features. Different color features are seen used on mobile robots but most of the time they rely on the HSV representation [94, 57, 9, 4]. As stated earlier, a representation similar to the opponent color is used in [25], but within a saliency detection step. Simple features such as lines [57] are also sometimes used as they are very quick to compute and provide a good amount of information in some settings. In several cases, color is simply not used [92, 62, 80, 25, 21]. These algorithms cannot distinguish objects that differ only by their color.

The classification or comparison step determines whether or not a given object is present in a test image. When features are used, the features from the test image are somehow compared to the feature from a set on training images. In many cases, a vocabulary is used to speed-up the comparison process [25, 9]. Examples also exist where the features are directly compared to each other [82, 42, 80, 94, 21]. Neural networks are also usable on mobile robots [4] because they are very fast to use once the training is done. Probabilistic classification can also be performed. The use of a particle filter results in an algorithm that can adapt to the resources available on the mobile robot [57]. Feature-less comparisons like image cross-correlation [71] are also seen, but much less frequently.

The algorithms reviewed in this section all make some assumptions about the objects or the configuration of the scenes to analyze. The most common assumption is that the objects present in a scene do not touch each other [4]. In addition to this first constraint, the use of a uniform or regular background is often seen [94, 57, 25]. An even more restrictive hypothesis is that images show a single object [42, 62, 80, 9]. Only a few algorithms can detect and recognize objects in cluttered scenes [92]. Almost every reviewed work uses an offline training database where the objects are seen in controlled conditions and tests on data collected on the robot in a more or less controlled environment. The level of control in test data is a second distinctive point between the algorithms. Most algorithms are invariant to a change in the objects position in the image, viewing angle and distance. Only a few examples are robust to occlusions [62, 4] and changes in the illumination [42, 92] of the objects. To improve the robustness to changes between train and test data, the algorithm in [9] is retrained as data is collected by robot. This allows the algorithm to adapt to the new

environments as they are encountered. Finally, the number of objects to recognize is typically low in mobile robot experiments, ranging from 3 to 10.

Chapter 3

Methods

This chapter contains technical information about the hardware and software used in this thesis. The datasets used for training and validation of the developed algorithms are also presented along with the measures of performance used.

3.1 Hardware

The robot used is a Pioneer 3-DX robot[1]. The pioneer 3-DX is a sturdy differential-drive wheeled robot. It comes with frontal and rear sonars and an onboard computer, but they are not used in this work. The pioneer can move at a speed of 1.6 meters per second.

The robot is equipped with a Kinect RGB-D camera[2]. The Kinect combines a color and a depth camera in a single sensor. The color camera provides an RGB image with resolution up to 1280×960 pixels. The depth camera provides an image of resolution up to 1280×960 pixels where each pixels contains the distance, in millimeters, to the nearest obstacle. The depth camera does not provide reliable measurements if an obstacle is closer than 0.8 meters or farther than 4 meters. Both sensors provide images at a rate up to 30 Hz. They have a field of view of 43° vertically and 57° horizontally. In this work, both cameras are set to a resolution of 640×480 and frame rate of 3 Hz. The Kinect is mounted on the robot about 1 meter from the ground. It faces the front of the robot and is slightly tilted downward.

The robot is also equipped with an Hokuyo UTM-30LX laser range finder[3]. The Hokuyo has a detection range of 0.1 meters to 30 meters. Its accuracy is of ± 30 millimeters for distances up to 10 meters and of ± 50 millimeters beyond 10 meters. It has a field of view of 270°, an angular resolution of 0.25° and provides a full scan every 25 milliseconds. In this work, the angular resolution of the range finder is limited to 250°, cutting 10° on each side, to clear the robot structure. The laser range finder is attached to the robot structure, centered and at the front of the robot.

[1] http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx
[2] http://msdn.microsoft.com/en-us/library/jj131033.aspx
[3] https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html

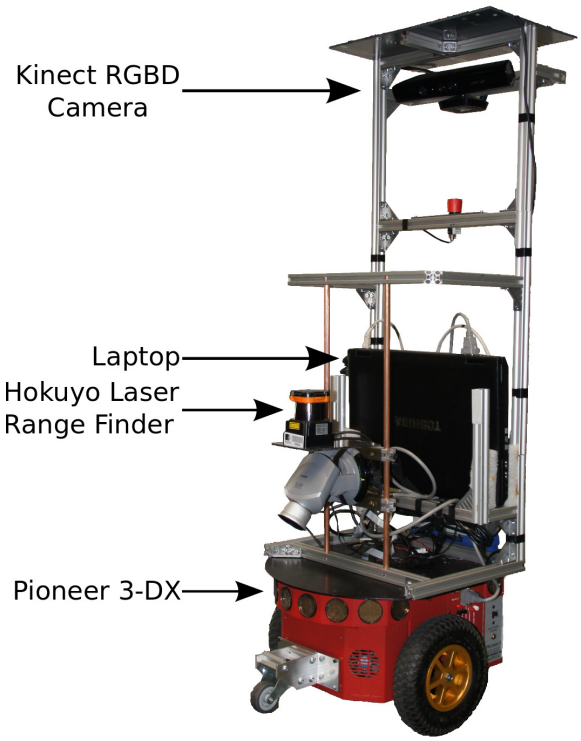

Figure 3-1: Image of the robot

The robot also carries a Toshiba Tecra laptop computer (Intel Core i5, 3 GB RAM). Figure 3-1 shows the robot with sensors and computer.

3.2 Software

Table 3.1 lists the software used in this work.

3.3 Algorithms

3.3.1 Parametric model fitting using RANSAC

RANSAC (RANdom SAmple Consensus) is an iterative method to find, given a dataset and a parametric model, the values of the model's coefficients that fit the dataset best and the set of samples from the dataset that support this model (the inliers) [24]. The dataset may be noisy, and even contain outliers. Outliers are samples that should not be taken into account when fitting the model. This could be the case of samples that belong to a different model, suffer unusual large noise or are the result of a measurement fault, for instance. The outliers in RANSAC are identified as samples having a model error larger than a user defined threshold. The model error is the distance between the sample and a given model (distance between a point and a plane, in a plane fitting application). The RANSAC algorithm proceeds as follows.

Software	Version	Use
Ubuntu[a]	12.04	Operating system
ROS[61]	Hydro Medusa	Hector_slam[37], visualization, integration
PCL[69]	1.7	Surflet-pair relation features, normals calculation, region-growing clustering, visualization, point cloud manipulation
OpenCV[10]	2.4	SIFT, clustering, histograms, image manipulation
PyBrain[72]	0.3	Feed-forward neural networks
Eigen[b]	3.2.2	Matrix operations
libnabo[19]	1.0.5	Nearest neighbor search

Table 3.1: List of freely available software used in this work

[a] http://www.ubuntu.com/
[b] http://eigen.tuxfamily.org

The minimal number of points needed to define the parametric model are randomly picked from the dataset (three points for a plane). The model's parameter are computed from these points. Then, the model error for each point from the dataset is computed. The point which model error is smaller than the error threshold are added to the inlier list. The sum of the model error of all inlier points is the overall error for this iteration. The next iteration repeats these steps using different randomly select initial points. After a number of iterations specified by the user is reached, the model with smallest overall error is returned. The main advantage of RANSAC is the explicit distinction between inliers and outliers. This makes RANSAC less sensitive to noise than typical minimum squared error fitting. The disadvantages of RANSAC are its random nature, its application specific parameters and its iterative nature. In this thesis, RANSAC is mainly used for plane detection in a point cloud.

3.3.2 Nearest neighbor search with kd-trees

The search for nearest neighbors is useful in many processing steps. In this work, it is used during normal computations, point cloud spatial sub-sampling, and feature matching. Given a set of points in a search space and a query point, a k-nearest neighbor search algorithm returns the k points from the set that are closest to the query point. The search can end when k neighbors are found or when all the neighbors within a predefined neighborhood radius are found. A naive nearest neighbors algorithm will compute the distance to all points in the set to the query point. It will then sort the points in ascending order of distance and return the k first points, or all the ones whose distance is lower than the defined radius. Since the query point is not known beforehand, the search cannot be simplified by pre-ordering the points in the set. However, the space in which those points lie can be analyzed so that useless computations are avoided. A kd-tree is a data structure which recursively splits a space into cells [19]. The splits are aligned with the space's axis and are made such that after a cell is split, an equal number of points lie in the two created sub-cells.

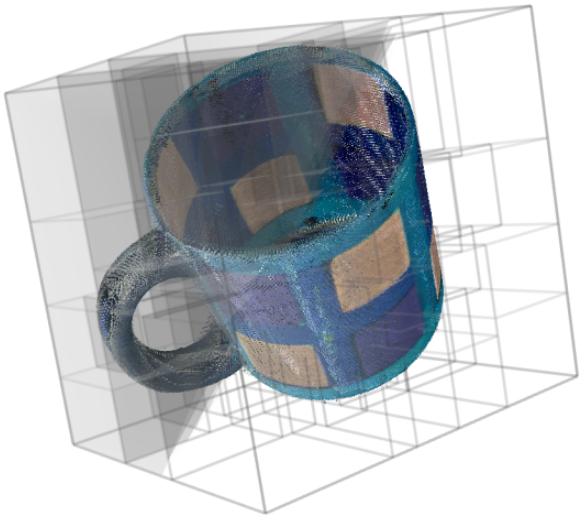

Figure 3-2: Kd-tree representation of a mug. Viewed 24 April 2015 http://www.pointclouds.org/assets/images/contents/documentation/kdtree_mug.png.

Once the kd-tree is created, it allows to rapidly determine which point is the closest to a query point along each dimension. Once the k number of nearest neighbors are found, or once the distance to the next closest point exceeds the maximum search radius, the search is stopped. The points lying further away will be completely ignored by the algorithm, saving computations. The software used in this thesis to implement kd-trees is given in table 3.1. Figure 3-2 shows the space occupied by a mug divided using a kd-tree.

3.3.3 Adaptive occupancy representation using octrees

A point cloud captured by a Kinect does not have a fixed spatial density. This is the case for all projective measurement devices, because their field of view forms a cone (or a pyramid) in space. The number of sampled points is constant, but the size of the cone increases as it extends farther away from the camera. The point density in a Kinect point cloud thus depends of the distance to the obstacles in front of the camera. A higher density of points means that smaller details can be represented, but it also implies a higher memory footprint. If a certain spatial resolution is known to be sufficient for the application, it might be beneficial to turn a point cloud into an octree. An octree is a data structure that recursively splits a 3-dimensional space into cubes of adaptive size [33]. The size of the cubes is adaptive in the sense that it locally depends on the presence of points. If a cube contains no data point, it will not be further split into smaller cubes. Cubes containing data points are split until a given minimum size is reached or until they contain no more points than a pre-defined capacity. Compared to a kd-tree, the places where splits can occur are predefined in an octree, the data only determines whether or not the split will happen. By sacrificing some spatial resolution, octree can help limiting the memory footprint of a point cloud. Octrees simplify some operations on point clouds also.

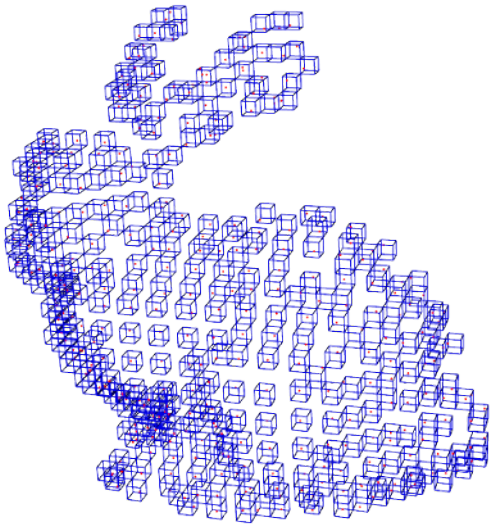

Figure 3-3: Octree representation of the Stanford bunny using a fixed resolution. Viewed 24 April 2015 http://www.pointclouds.org/assets/images/contents/documentation/octree_bunny.png.

Adding two point clouds means to concatenate the list of points they contain. This can lead to duplicates (or near-duplicates) and a waste of memory resources. In an octree, the addition of new points leads to an increase of memory only if the location of the points is not already represented. As stated in table 3.1, the PCL library is used to implement the octrees in this thesis. Figure 3-3 shows the use of an octree to represent the Stanford bunny.

3.3.4 Computing point normals in a point cloud

In a point cloud, a point normal designates the coefficients of the vector perpendicular to the surface at a given point. Surfaces in a point cloud are not explicitly represented. Before a normal can be computed, the surface must be estimated. The surface at a point of interest is typically taken to be the plane which best fits the points in a neighborhood around this point. A larger neighborhood filters out the noise in the points position but also removes small surface details that can be important. The size of the neighborhood can thus be adapted to the estimated level of noise to preserve useful details. In a point cloud captured with a Kinect, points farther away from the camera tend to be noisier that close-by points. The noise level is estimated to grow with the squared distance [32]. The size of the neighborhood used to compute a surface normal can thus be chosen to be proportional to its distance to the camera. Once the neighborhood is determined, a plane is fitted with a least squared error estimation. The coefficients of the fitted plane explicitly are the surface normal at this point. The normal computations in this thesis are done using the "average 3d change" of the PCL library. Figure 3-4 illustrates the process of computing surface normals in a point cloud.

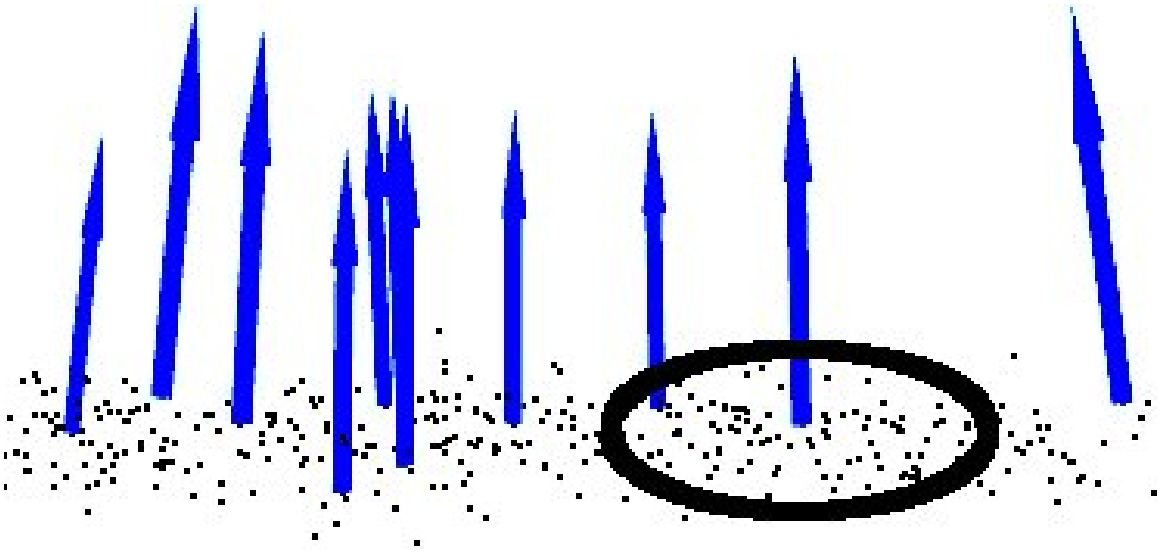

Figure 3-4: Surface normals computation on a point cloud. Viewed 24 April 2015 http://www.pointclouds.org/assets/images/contents/documentation/features_normal.png.

3.3.5 Simultaneous localization and mapping

Simultaneous localization and mapping (SLAM) in static environment was a major research topic of the past decade. As a result, off-the-shelf solutions now exist that tackle the problem of SLAM for wheeled robots equipped with a laser sensor. These SLAM algorithms use the information from the laser, and optionally odometry measures, to simultaneously build a map of an unknown static environment and estimate the position of the robot within this map. Hector SLAM [37] is one of these publicly available libraries. Hector map uses a scan matching approach. With each new scan, an estimate of the transformation between the map and the received scan is computed. The new scan is then integrated into the map. This method is fast and can benefit from the high sampling rate of modern laser range finders. The algorithm outputs a metric map and the current estimate of the robot's position in this map. Figure 3-5 illustrates the process of building a map using a SLAM algorithm.

3.3.6 Scale invariant feature transform for texture description

The Scale Invariant Feature Transform (SIFT) is an image descriptor based on gradients. It was introduced in [44] and is nowadays widely used. The term SIFT usually designates two things, the descriptor itself, and a detector. The SIFT detector is based on the detection of scale-space extrema in a multi-scale difference of Gaussians pyramid. A pyramid of Gaussians is obtained by repeatedly convolving with a Gaussian kernel and subsampling an image. The difference of Gaussians pyramid is the result of computing the difference between consecutive levels in this pyramid. The detector returns the points where the difference of Gaussians is a local extremum within a $3 \times 3 \times 3$ region of the scale-space. The returned points typically correspond

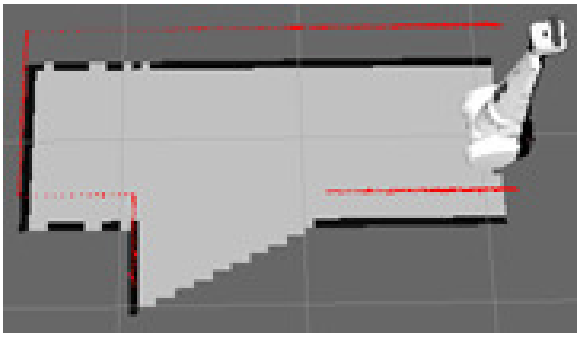

Figure 3-5: A robot mapping an unknown environment using a SLAM algorithm. The red dots show the impacts of a laser scan. A map is built incrementally based on these impacts. As the robot moves, it can localize itself within the map and update it based on new laser scans. Viewed 24 April 2015 http://cstwiki.wtb.tue.nl/images/Gmap2.gif.

to specific image features such as edges, corners and blobs. In SIFT, the detection on edges are removed because they present too many symmetries to be reliably matched. Points lying in a region of low contrast are also rejected because they tend to be produce noisy SIFT features. The SIFT detector is invariant to rotations and scans different scales of the image to provide scale invariance. The detector produces a set of locations and scales where a SIFT feature should be computed. The SIFT feature encodes the spatial arrangement of gray scale gradients around the feature location. The size of the neighborhood used to compute the SIFT feature depends on the scale value given by the detector. The feature itself is rotation invariant because all computations are referenced to the dominant orientation of the gradients in the neighborhood. If several dominant orientations are detected, several SIFT features will be computed, one for each orientation. A grid is aligned with the dominant orientation and laid out around the feature location to define an array of smaller neighborhoods where local gradient orientations will be computed and compiled into histograms. The concatenation of these small histograms form a SIFT feature. Figure 3-6 shows an example of a grid used to compute local gradient orientations on an image.

3.3.7 Surflet-pair relation for shape description

Surflet-pair relations are a way to encode the relative position of a pair of oriented points [91]. An oriented point can be obtained by pairing a point with its surface normal. The surflet-pair relation computes 4 values that encode the surflets' configuration without reference to an external point of view. A surflet-pair relation is composed of three angular values and one linear value. For a given pair of surflets $\{p_1, n_1\}$ and $\{p_2, n_2\}$, where p_i is a point's position as a coordinate in three-dimensional space and n_i is a unit-norm vector giving the orientation of the point's normal. The first step is to choose one of them as the point of reference. To make this operation invariant to point of view, the reference point is the one which normal forms the smallest

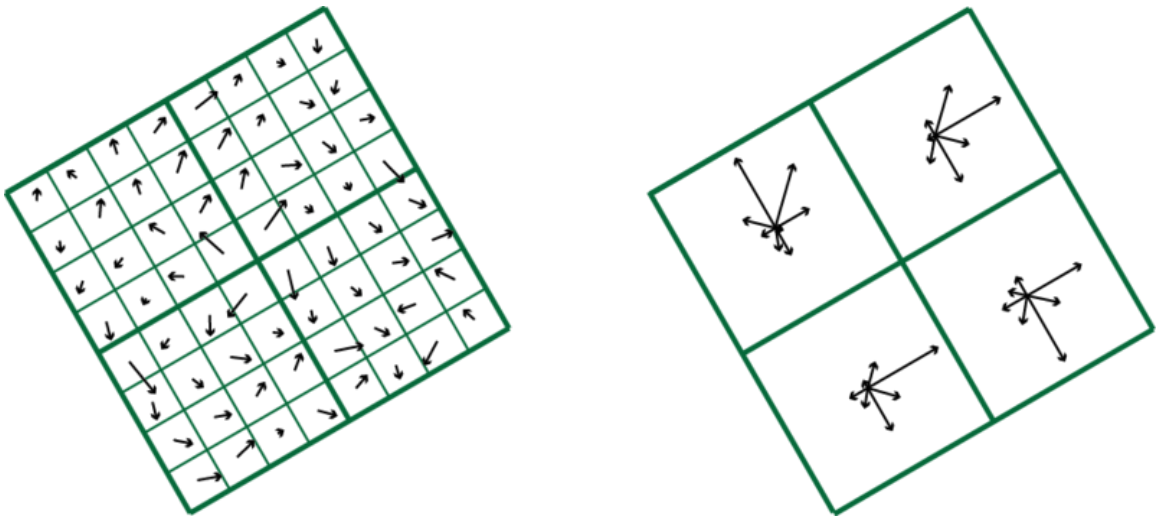

Figure 3-6: Spatial subsampling and compilation of gradient orientations in the SIFT feature (using a 2×2 grid). Viewed 24 April 2015 http://www.scholarpedia.org/w/images/thumb/0/04/Sift-descr-ill.png/500px-Sift-descr-ill.png.

angle with the difference vector $p_1 - p_2$. That is,

$$\{p_s, n_s\} \leftarrow \begin{cases} \{p_1, n_1\} & \text{if } |n_1 \cdot (p_1 - p_2)| \leq |n_2 \cdot (p_1 - p_2)| \\ \{p_2, n_2\} & \text{otherwise} \end{cases} \tag{3.1}$$

The other surflet is renamed $\{p_t, n_t\}$. Then, a frame of reference can be fixed to this reference point with axes defined as:

$$u = n_s \tag{3.2}$$
$$v = u \times \frac{p_t - p_s}{\|p_t - p_s\|_2} \tag{3.3}$$
$$w = u \times v \tag{3.4}$$

With this frame of reference, the surflet-pair relation can be computed as:

$$\alpha = v \cdot n_t \tag{3.5}$$
$$\phi = u \cdot \frac{p_t - p_s}{\|p_t - p_s\|_2} \tag{3.6}$$
$$\theta = \arctan(w \cdot n_t, u \cdot n_t) \tag{3.7}$$
$$d = \|p_t - p_s\|_2 \tag{3.8}$$

Figure 3-7 shows the angular measures α, ϕ and θ for two surflets. The mapping is one to one: one particular instance of a surflet-pair relation can only be generated by one particular surflet pair, and the opposite is also true.

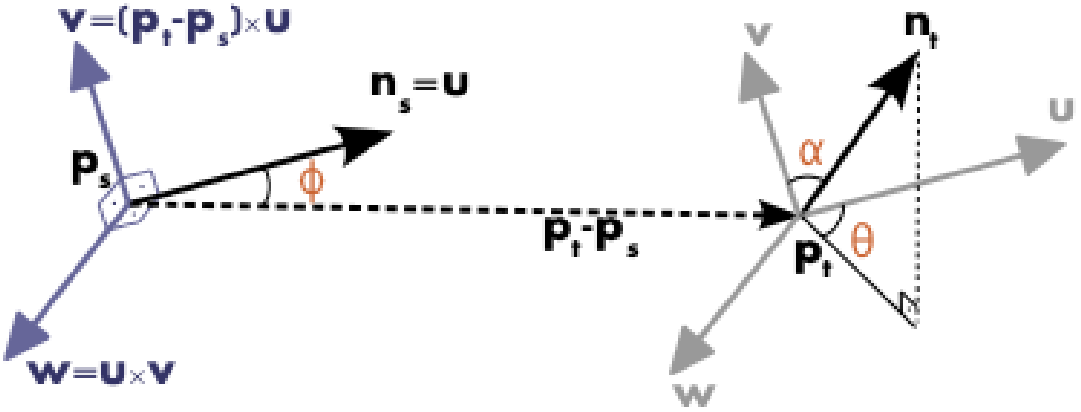

Figure 3-7: Angular values involved in the surflet-pair relation. Viewed 24 April 2015 http://pointclouds.org/documentation/tutorials/_images/pfh_frame.png.

3.3.8 Function approximation with feed-forward neural network

A feed-forward neural network is a function approximation machine. It learns the parameters of a parametric model that best transforms the input sample from a training database into the right output values. A feed-forward neural network has a layered structure which contains a certain number of neurons. Neurons are simple units with an arbitrary number of inputs and a single output. The output value results from performing a weighted sum of the inputs and passing the result through a non-linear function [29]. It is the value of these weights that are learned through a gradient descent process from a training database. The non-linear function typically has a squashing effect to restrict inputs of any value to a fixed range of output values. An example of such a function is the sigmoid function. The operation performed by a sigmoid unit is as follows:

$$o^{sigmoid}(i) = \frac{1}{1 + \sum_{n=0}^{N} w_n i_n}, \tag{3.9}$$

where $o_{sigmoid}$ is the output value, N is the number of inputs, w_n is the n^{th} weight and i_n is the n^{th} value of the input vector i. Typically, three layers are used. They are called the input, hidden and output layers. The input layer contains as many units as there are input variables, they have a single input and no non-linear transformation. In fact, the input layer does not modify the input values. The size of the hidden layer is a meta-parameter to be chosen by the designer. The inputs to the units in the hidden layer are the output values of all units from the input layer. The form of the non-linear transformation is a second meta-parameter of a feed-forward neural network. There are as many units in the output layer as there are output variables in the problem to learn and they take input from all hidden units. The output function

is also chosen by the designer, and sometimes is linear. A bias unit must be added which is connected to all units from the hidden and output layers through another set of weights.

Training a feed-forward neural network is an iterative process which starts by assigning each weight a random initial value. The back-propagation learning algorithm for neural networks [68, 63] works as follows. Each sample from the training database is presented to the neural network in turn. For each input sample, the neural network's output is calculated and the difference between the network's answer and the target value specified in the database is computed. This is the neural network's error. This error can be used to change the weights of the neural network such that it becomes better at computing the good answer for this specific input sample. The weights are modified by back-propagating the error through the network, from the output layer to the input layer, by multiplying it by the current weights. This back-propagation effectively is a way to assign credit to the weights for the provided answer. It produces, for each neuron, a quantity which indicates how the weight should be changed in order to improve the final answer of the neural network for this input sample. Once the weights are modified, the process is repeated for a different sample from the database until none is left. An epoch is the fact of processing the entire database. Several epochs are required for a feed-forward neural network to converge. Changing the weights is done only by considering the current sample from the database. The next sample might cancel any change done by previous ones. To increase the chances that the neural network converges to a solution that gives a good output value for all input samples, the order in which the samples are presented to the network is randomly changed between each epoch. The resilient back-propagation algorithm is a variant of the standard learning rule in which the weight change is increased if the sign of the error gradient stays the same for several iterations [34]. Training can be performed until the neural network gives the exact right output value for each input sample, but this is likely to be too long (as well as generate overfitting problems).

The standard way to stop the optimization is by using early stopping [60]. The early stopping algorithm monitors the mean-squared error obtained on a validation set. The validation set is a subset of the training set that is not used in the training process, but only serves as a measure of the quality of the current state of the network. The optimization process should, at each epoch, decrease the mean square error on the training set. Monitoring the mean square error on a validation set, containing samples similar but different to that of the training set, is one way to measure how well the network generalizes to unseen data. The early stopping method provides a rule to stop the training of the network when the generalization quality start to decrease. For instance, if the mean square error on the validation set increases for 5 epochs in a row, the training process is stopped. It is normal that this error increases from time to time during the process, but if it steadily increases for a few iterations, it is a sign that the network starts overfitting.

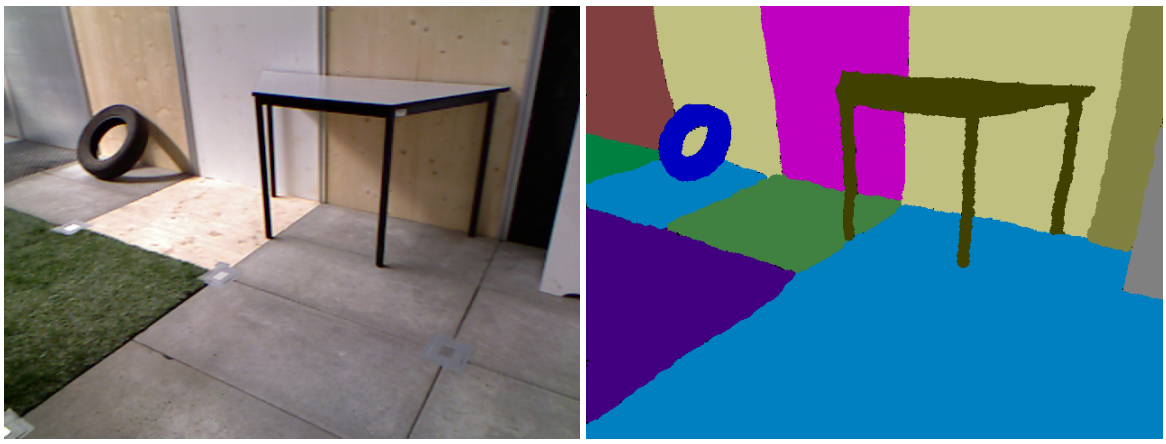

Figure 3-8: Original and labeled image example from the segmentation dataset.

3.4 Data

In object recognition, datasets are used to train, validate and measure the performances of algorithms. In this work, three different datasets are used.

3.4.1 Segmentation dataset

The segmentation database was collected during the Carotte competition (see section 1.2). It it used to evaluate the performance of the segmentation algorithm. It consists in 70 point clouds captured as the robot was exploring the arena. The point clouds are hand-labeled into categories floor, wall and object. The labeling for the walls is based on appearance, meaning that if the color or texture of a wall changes, the assigned label will change accordingly. There are only 70 images in this dataset, but the annotations are of high quality. An example of a scene from the segmentation dataset is shown in figure 3-8.

3.4.2 ENSTA offline object dataset

The ENSTA offline object dataset is used for training and for basic validation of object instance recognition algorithms. It consists of a total of 21 411 point clouds of 52 objects and pieces of furniture like chairs, boxes, small cabinets, etc. The data was collected using the robot described in section 3.1 and consists in point clouds taken with the Kinect camera. Objects were shot under 6 different viewing angles while the robot was moving back and forth towards them. The setup can be seen in figure 3-9. For each angle, 100 point clouds were saved, from a distance varying from 1 to 4 meters. Figure 3-10 shows one of the objects under the 6 angles of view and varying distance. One snapshot from each of the objects is shown figure 3-11. The database is a collection of examples of point clouds grouped by object and viewing angles. The object showed in each point cloud is extracted by using the segmentation algorithm detailed in chapter 4.

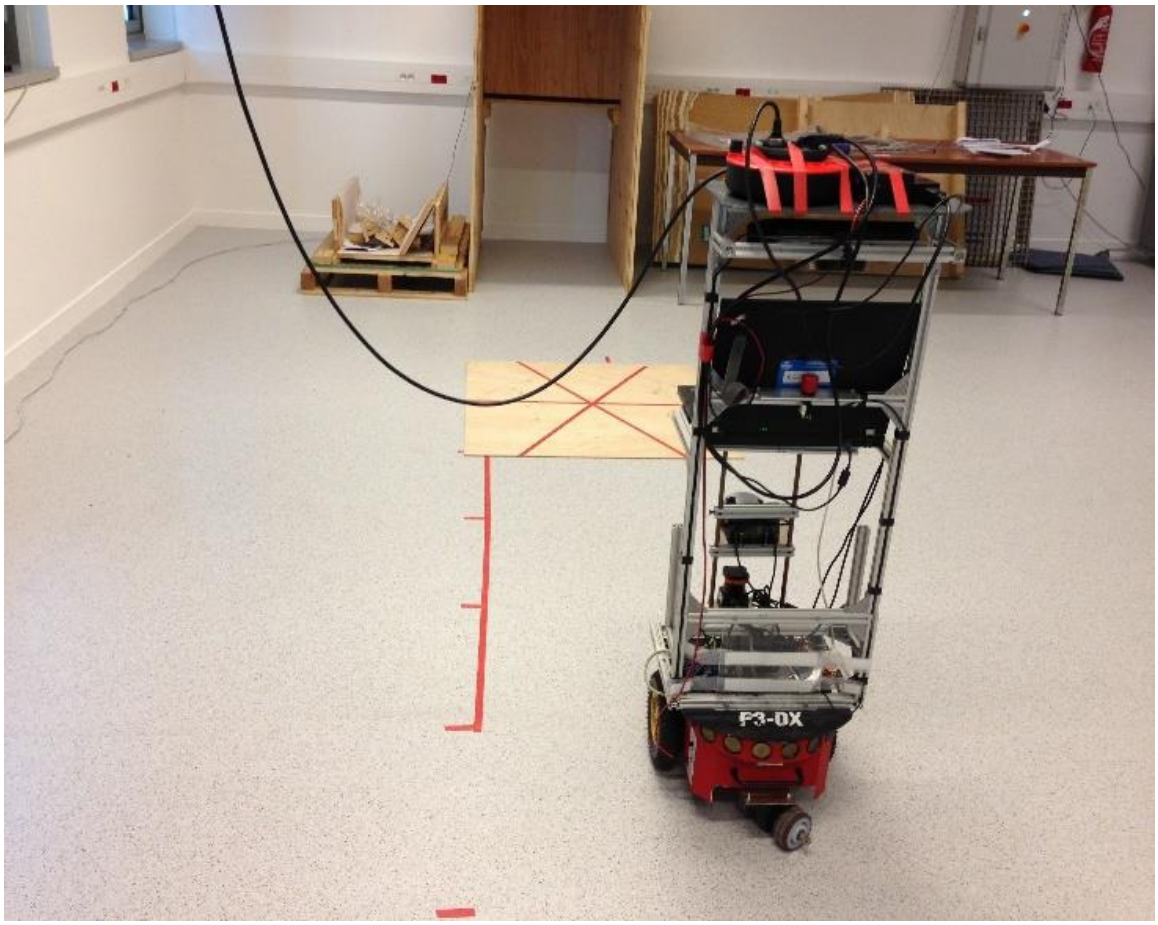

Figure 3-9: The setup for the offline dataset.

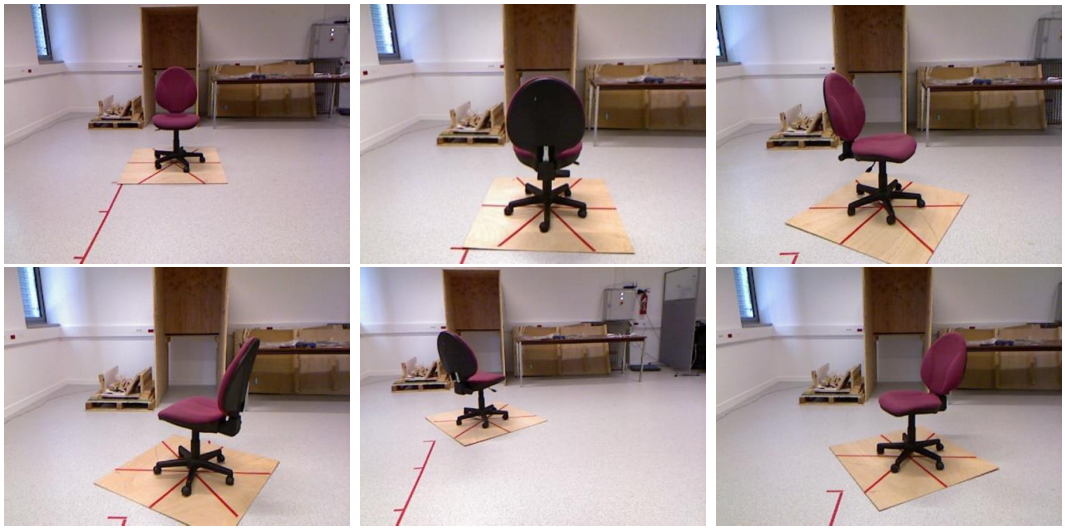

Figure 3-10: One example for each viewing angle of the "red office chair" object from the offline database.

Figure 3-11: All 52 objects from the offline object dataset.

An interesting aspect of the ENSTA offline object dataset is the mix of similarities between the objects it comprises. For example, there are examples of objects that have exactly the same shape, but not the same color (the sofas, bottom-left part of figure 3-11). Many objects have a very similar shape and color, like the cabinets and some of the boxes. Finally, there are several objects that share an almost identical color while having different shapes. The presence of such combinations of objects in the dataset makes it the perfect ground to explore the process of combining features for object recognition.

3.4.3 ENSTA online object dataset

The ENSTA online object dataset is a small collection of point clouds taken in less stable conditions than the offline dataset. It is used as challenging validation data for object instance recognition algorithms, after training on the offline dataset. The point clouds were taken as the robot was moving around a room where 23 objects from the offline dataset were scattered. The room is shown in figure 3-12. Each point cloud is segmented using the method detailed in chapter 4. The resulting object views are checked for errors and erroneous images are removed. Out of 157 views, 16 were removed. Examples of removed views are shown in figure 3-13. Some object were completely removed because all their views were erroneous. For instance, a box and a trashcan that were too close together were removed because the segmentation was unable to separate them. There is a total of 20 objects and 141 views left in the online database.

The remaining views are challenging for several reasons. There are important changes in the lighting conditions. The objects are seen from unknown angles of view, which might not exist in the offline dataset. Also, some amount of motion blur happens when the point cloud was taken as the robot turned rapidly. Finally, there are partial views of some objects and heavy occlusions. Figure 3-14 shows examples of images that were kept in the database. Some of them show very well segmented objects, but some images are heavily occluded and badly segmented.

3.5 Multi-class classification performance measures

The experiments conducted in this work fall in the category of multi-class classification. In multi-class classification, each sample from a dataset must be assigned a single label from several distinct ones. The performance measures used in this work are variations of standard binary classification measures, as suggested in [81]. Two types of statistics can be compiled: per-label ones and global ones. To obtain global measures, the classifier's answers are first compiled on a per-label basis and then summed up. Per-label answers fall in three categories:

- True positive (tp_i), the correct labeling of a sample from class i
- False positive (fp_i), the labeling as i of a sample from class $j \neq i$

Figure 3-12: The room and objects in which the robot was moving around to collect the images of the online dataset.

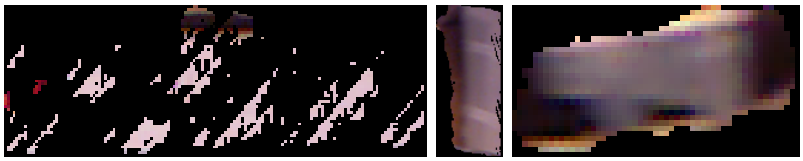

Figure 3-13: Examples of the 16 erroneous object segments removed from the online dataset. These image segments are impossible to recognize even by a human and were considered as failures from the segmentation step that should not affect the object recognition's performance measures. **Left** A part of a wall that was only partly detected. **Center** and **right** Parts of unknown objects.

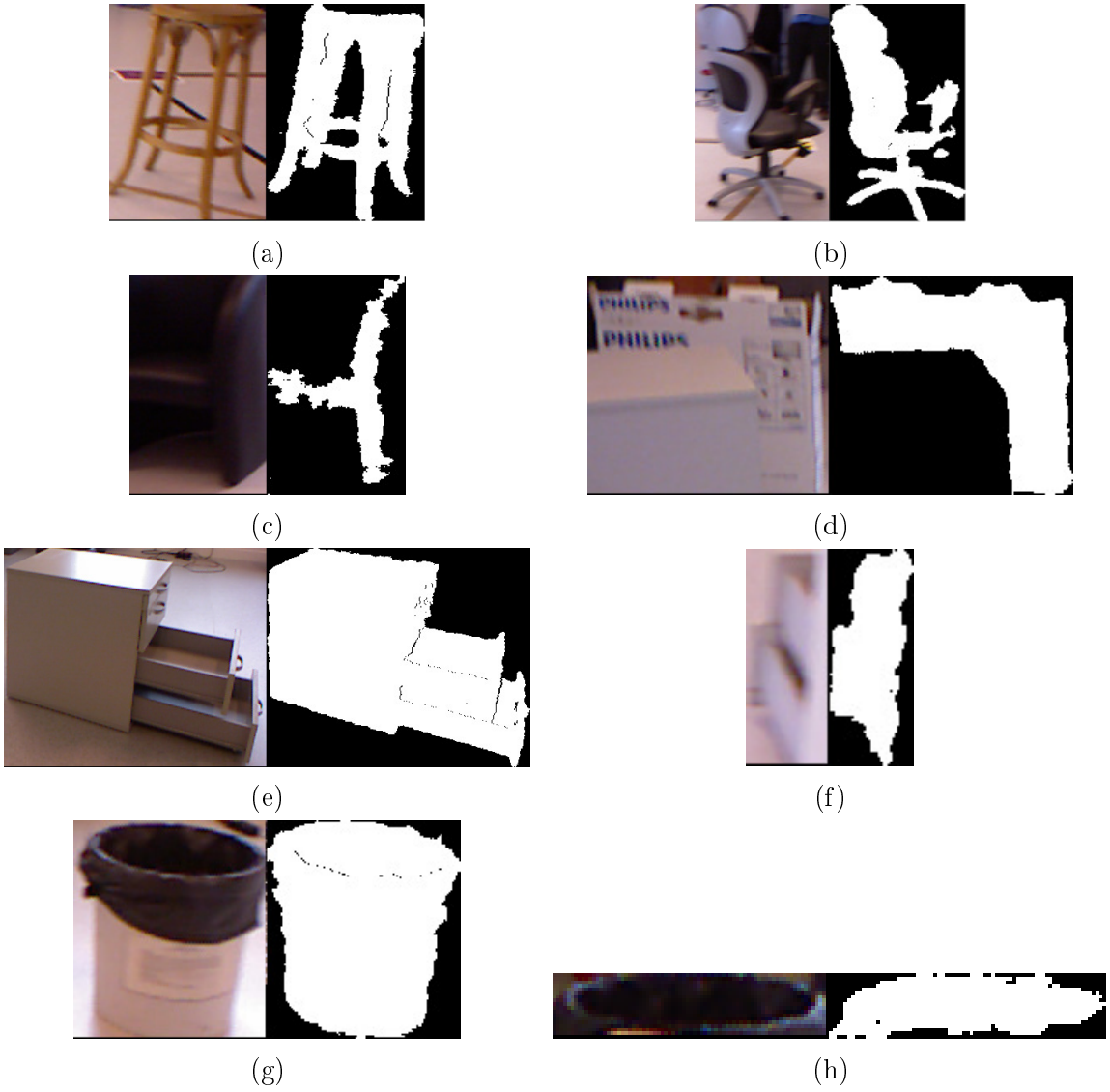

Figure 3-14: Examples of good views and more challenging ones from the online dataset. All examples show a color image on the left and the binary segmentation mask on the right. The algorithms only have access to the pixels from the color image which are white in the segmentation mask. **a** A correctly segmented stool. **b** A correctly segmented black office chair. **c** An incomplete black sofa due to the bad perception of black objects by the depth camera. **d** An occluded thin white box. **e** A correctly segmented cabinet. The cabinet is seen against the light, making it appear much more contrasted than in the offline database. **f** A partial view of the cabinet due to occlusion. **g** A correctly segmented trashcan. **h** A partial view of the trashcan due to occlusion.

- False negative (fn_i), the labeling as $j \neq i$ of a sample from class i or the failure to label a sample from class i

True positives represent right answers, and false negative and false positive are two different types of wrong answers. True negatives are nonexistent here as they would refer to correctly determining that an object is not part of the database. This does not happen here because all examples do come from the database. True negatives anyhow are not involved in the performance measures described below. A useful note for the computations of the performance measures is that the sum of true positives and false negatives for a class is equal to the total number of samples of that class.

Precision measures the ability of a classifier to not incorrectly label as i samples from classes $j \neq i$. When a classifier shows a high precision to class i, it can be trusted that samples the classifier labeled as i really are of class i. Precision is undefined if a classifier does not label any sample as i. Precision is the ratio of correctly labeled samples of class i to the total number of samples labeled i:

$$precision_\mu = \frac{tp_i}{tp_i + fp_i} \tag{3.10}$$

Recall measures the ability to correctly label samples of class i. When a classifier shows a high recall to class i, it can be trusted that samples the classifier labeled as $j \neq i$ really are not of class i. Recall is the ratio of correctly labeled samples of class i to the total number of samples of class i:

$$recall_\mu = \frac{tp_i}{tp_i + fn_i} \tag{3.11}$$

Macro-averaging is one way to combine per-label measures into an overall performance. A macro-average is obtained by computing a per-label measure for each label and then summing the result. Macro-averaging gives each label an equal importance in the overall measure. Macro-average precision and recall measures are obtained s follows:

$$precision_M = \sum_i \frac{tp_i}{tp_i + fp_i} \tag{3.12}$$

$$recall_M = \sum_i \frac{tp_i}{tp_i + fn_i} \tag{3.13}$$

Micro-averaging is another way to combine per-label measures. Micro-averaging is obtained by summing the contributions from all labels first and then computing the appropriate ratio. Micro-averaging gives each sample equal importance in the overall

measure. Micro-average precision and recall measures are obtained with

$$precision_m = \frac{\sum_i tp_i}{\sum_i tp_i + fp_i} \tag{3.14}$$

$$recall_m = \frac{\sum_i tp_i}{\sum_i tp_i + fn_i} \tag{3.15}$$

Precision-recall curves are a tool to visualize the effect of accepting or rejecting the answer of a classifier according to a confidence measure which varies between 0 and 1. Most classifiers implicitly provide such a measure. An example of a confidence measure is the actual activation value of the highest activation neuron of a neural network. If this value actually is a good confidence measure, it should be low when the classifier is wrong and high when it is right. This way, the measure can be used to filter the answer of the classifier by applying a threshold. If the confidence is higher than the threshold, the answer is accepted, otherwise, it is rejected. A precision-recall curve is drawn by plotting the precision and recall values obtained while the threshold is varied from 0 to 1. With a threshold of 1, all answers are rejected. This means the recall is 0 and precision is undefined. For visualization purposes, the precision will be fixed to 1 in this case. With a threshold of 0, all answers are accepted. This yields the highest possible recall and lowest possible precision for the classifier under study. The curve allows to visualize the compromise the classifier offers between precision and recall so that a good threshold can be chosen for a given application. Examples of precision-recall curves can be seen in figures 5-3, 5-4 and 5-5, from chapter 5, section 5.6.

A precision-recall curve shows the evolution of the precision and recall measures as the value of the threshold changes from 0 (all answers accepted) to 1 (no answer accepted). If the classifier behaves correctly, increasing the threshold value should decrease the recall and increase the precision. The rate at which these two values change is informative about whether or not bad answers tend to be filtered before good ones. The area under a precision-recall curves measures this behavior. A larger area under the curve indicates the thresholding method works well in the sense that bad answers are avoided while good ones are kept.

Chapter 4

Geometrical indoor scene segmentation on a mobile robot

4.1 Introduction

This chapter details the implementation and testing of a geometrical scene segmentation algorithm. The segmentation algorithm's goal is to pre-format and reduce the amount of data to be processed by the subsequent stages of the pipeline. Reducing the amount of data this early in the processing should reduce the overall computational complexity of the pipeline and reduce the run time. For this stage to be beneficial, it must be fast and eliminate as much of the uninformative parts of the point cloud as possible while preserving the useful ones. These points are evaluated qualitatively in a first time by inspecting some outputs produced by the algorithm. A quantitative evaluation is also conducted by computing the precision and recall of the algorithm on the segmentation dataset (see chapter 3, section 3.4.1). Using these scores, the geometric segmentation algorithm is compared with Markov random field based segmentation algorithm (see chapter 2, section 2.1.2). Section 4.2 presents the justification for the main choices involved in the design of the algorithm. Then the implementation details, values for all parameters and a functional view of the algorithms is given in section 4.3. Finally the experiments are described in section 4.4.

4.2 Design choices

The first fact that influences the design of the segmentation algorithm is that it will serve in an indoor environment. Indoor environments tend to be highly structured. Indoor scenes typically contain a floor, some walls and a ceiling. In this work, the Kinect is tilted downwards and the ceiling is never visible. The floor and walls are structures that can be recognized by their shape: they are planar. Furthermore, they do not convey any information about the objects to recognize and can be safely removed from the point cloud to process. This fact motivates the design of a geometric segmentation algorithm to detect the floor and walls. To make the algorithm as simple

as possible, only the geometric information is used. Information from the color image is completely ignored at this stage. This decision is supported by the success of geometric table-top segmentation algorithms (see 2, section 2.1.2).

The second important information is that the algorithm will run on a wheeled robot. This means the position of the Kinect and other sensors should be somewhat stable: there should be little variation in the distance from the Kinect to the ground, for instance. The same holds for the roll and pitch of the Kinect with regard to the floor plane. This stability allows a first foreground-background separation step to be done very easily. The floor plane can be completely removed from an indoor scene, if an estimate of the Kinect's position is known. This estimate is provided by an offline calibration procedure. Note that the Kinect will indeed move slightly during operation. This will happen, for instance, if the robot accelerates or bumps into something. For this reason, an online estimation of the coefficients of the floor plane will still be done for each input point cloud. As noted, the walls should also be recognizable due to their shape. However, there is not as much a priori information about a wall's position as there as for the floor. Hence, a more elaborate search strategy, based on RANSAC, will be used.

Thirdly, the algorithms in this thesis work with the assumption that objects to recognize do not touch each other. Consequently, the segmentation algorithm includes a step to segment the objects based on a distance criterion. Each cluster obtained this way is called an object candidate. This distance clustering step will be accomplished by a region growing algorithm which splits objects if they are farther apart than a given distance. This distance criterion will not be applied directly, though. Oftentimes, objects seen by the Kinect are split into several distinct blocks because the parts linking these blocks are invisible to the camera (they can be occluded by other parts of the object). To avoid having the region growing algorithm split these objects, the point cloud is first projected to the floor plane. This simple trick successfully recombines candidate objects into a single cluster in most situations. This has the side effect of merging objects that are on top of each other, but this happens less often than the split object problem stated earlier. Another issue with the region growing algorithm is that it involves computing the distance between a large number of pairs of points. Because the computational complexity of this step is high, the point cloud is first down-sampled before any other processing is done. The only fact to keep in mind is that the down-sampling resolution (the average distance between two points in the down-sampled point cloud) should not be larger than the distance threshold used to form object candidates. If this resolution is a few times smaller than the over-segmentation distance threshold, the down-sampling has no negative effect on the quality of the over-segmentation.

Finally, a few more criteria can be used to further eliminate uninformative or useless parts of the point cloud. Object candidates that are too small, for instance, do not carry enough information to be recognized and can be removed. Candidates that are larger than the largest object of interest can also be removed. Furthermore, object candidates that touch a border of the point cloud are very likely to be partly outside the field of view of the Kinect and thus be hard to recognize. It is realistic to say that during a robotics experiment, each object in the scene will be seen at

least once while it is clearly visible in the field of view of the Kinect. Candidates touching a border can thus be safely discarded from the point cloud. Furthermore, the Kinect tends to be noisy, especially as the distance increases. Thus, parts of the point cloud that are too far away from the Kinect are removed so that the focus is on close-by objects, which are more likely to be successfully recognized. Again, there should be a moment during the experiment where objects that have previously been discarded from a point cloud will be clearly visible and will get through the segmentation process.

The following section explain the implementation details of the algorithm and the experiments.

4.3 Implementation details

This section gives all implementation details of the geometrical scene segmentation algorithm.

4.3.1 Offline calibration

The floor plane removal requires an estimate of the floor plane parameters. This information is provided by an offline calibration phase. The calibration is done by having the robot face a large area of visible floor and running a RANSAC plane detection algorithm with parameters shown in table 4.1. Figure 4-1 shows an acceptable configuration for calibration as well as the detected floor plane. The calibration phase has to be done only once for a given placement of the Kinect on the robot. It provides the floor plane's normal coefficients (a, b and c) and distance (d) in the form $ax + by + cz + d = 0$.

4.3.2 Pre-processing

The removal of far-away points is done first as it is the most simple step to implement. To avoid unnecessary computations of exact Euclidean distances, a threshold is applied on the points' z-coordinate. The z-axis is not exactly parallel to the floor plane, as the Kinect is tilted downwards, which means the distance is likely to be underestimated. The removal of far-away points in no way requires high precision and this simple thresholding provides a usable result.

A second pre-processing step is the computation of point normals. The normals are used during the floor removal step. They are calculated according to the procedure detailed in chapter 3, section 3.3.4.

4.3.3 Floor removal

The floor removal algorithm proceeds in three steps: selecting the points that are close to the calibration floor plane, using these points to compute the refined floor plane coefficients and removing the points lying close to this estimated floor plane. Points

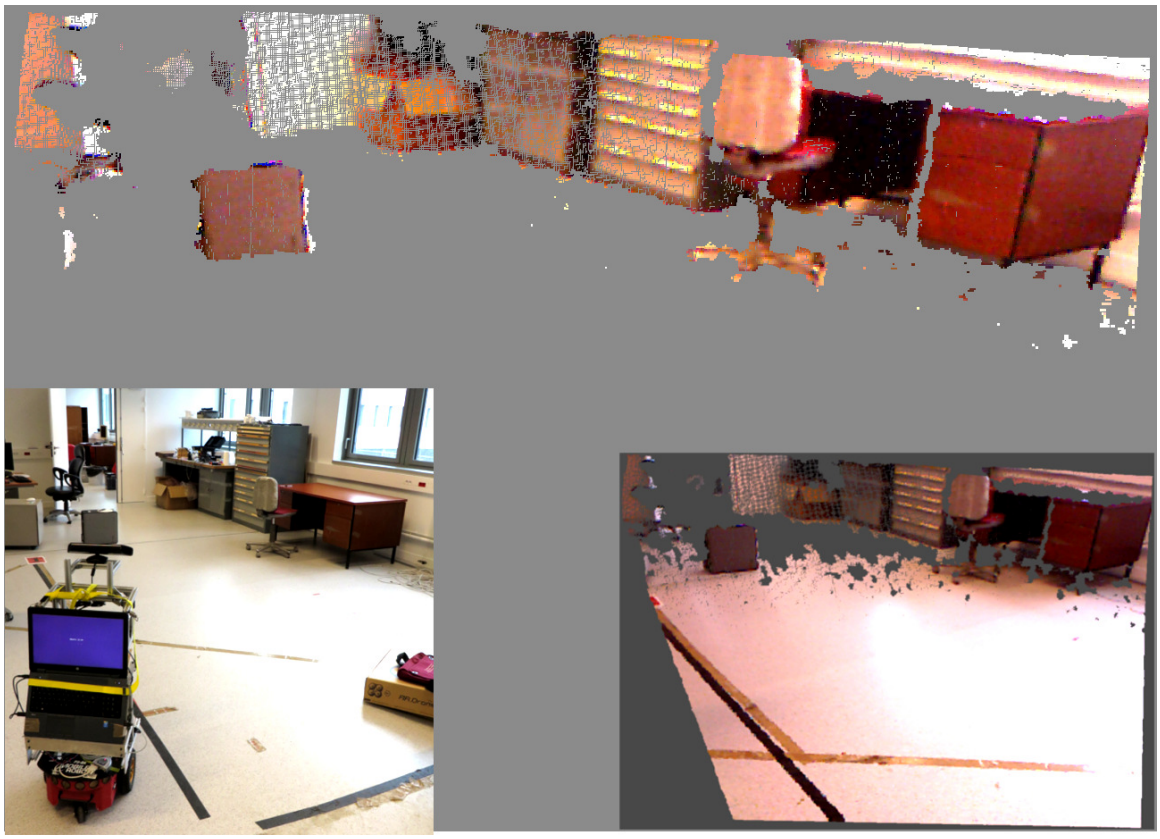

Figure 4-1: An appropriate scene to perform offline calibration. The main image shows the scene with the floor plane removed after calibration. The bottom-left inset shows the robot's surroundings and the bottom-right inset shows the input point cloud.

from the point cloud are selected for floor plane estimation if they are close enough to the calibration floor plane and if their normal is aligned with the floor plane normal. If the number of points in the list is large enough, the refined floor plane is estimated from the list. If the point count is low, it means that the floor is not clearly visible because of objects occluding it. In this case, the calibration floor plane is used for this point cloud. The refined floor plane estimation is done by performing a principal component analysis and selecting the last eigenvector. This eigenvector corresponds to the direction of lowest variance and effectively is a least-squares estimation of the plane normal. Finally, all points from the point cloud that lie below or just over the floor plane are removed. This process is detailed in algorithm 1. The values of the parameters used for the algorithm are given in table 4.1.

4.3.4 Walls removal

There is not as much a priori information about position and orientation of walls as there is for the floor plane. The walls in a scene cannot be found using a technique as simple as algorithm 1. Walls have to be found using other heuristics: walls are planar surfaces, they are perpendicular to the floor plane and they are large structures. With these heuristics, and the knowledge of the coefficients of the floor plane, walls can be searched in the point cloud. The search is done using RANSAC for plane detection. The RANSAC algorithm is stochastic: it has a high chance to find a plane if this plane occupies a large portion of the point cloud. If it is the case, finding and removing the walls will be very beneficial. If RANSAC exits without finding anything, then it most likely means that the walls (if there are any of them) are small and that failing to remove them should not be catastrophic. The removal process keeps two copies of the input point cloud. The first is a temporary point cloud used during the process from which all the planes found by RANSAC are removed. The second is the result point cloud from which only points belonging to walls are removed. The process of finding walls is iterative, as there might be more than one in the scene. First, RANSAC is run to detect a plane. If RANSAC does not return a plane, the process ends. Otherwise, the plane is removed from the temporary point cloud. If the plane's size is large enough and if it is perpendicular to the floor plane, then it is considered as a wall and removed from the result point cloud. In any case, the process is repeated using the temporary point cloud as the input to the RANSAC of the next iteration. The process ends when RANSAC returns no plane or the plane it returns is too small or when there are too few remaining points in the point cloud. When the iterative process is done, the result point cloud is returned. This procedure is presented in algorithm 4. To reduce the amount of computation, the size check is performed on the number of points forming the plane, not on the physical size of the plane. Precisely estimating the physical dimensions of a point cloud is a complex process which typically involves computing a convex hull that fits the point cloud as tightly as possible. The dimension of the object could also be approximated by a 3-dimensional bounding box, but this would still require visiting each point from the object point cloud. The values of the parameters used for the algorithm are given in table 4.1.

Algorithm 1 Floor removal

Input: *point_cloud*, input point cloud with point normals
calib_floor_normal, floor plane normal from offline calibration
calib_floor_distance, distance to floor plane from offline calibration
estimate_dot_prod_threshold, minimum dot product value between points' normal and floor plane normal
estimate_distance_threshold, maximum distance between points and floor plane for plane estimation
estimate_size_threshold, minimum number of points in the list of floor points
remove_distance_threshold, maximum distance between points and floor plane for point removal

Output: *current_floor_normal*, the estimated floor normal
current_floor_distance, the estimated distance to the floor plane
point_cloud, output point cloud with points from the floor plane removed

floor_points_list ← an empty list of points
for all *point* in *point_cloud* **do**
  point_normal ← point normal of *point*
  normal_dot_prod ← *point_normal* · *calib_floor_normal*
  if *normal_dot_prod* > *estimate_dot_prod_threshold* **then**
    distance_along_normal ← *point* · *calib_floor_normal*
    distance_from_floor ← | *distance_along_normal* − *calib_floor_distance* |
    if *distance_from_floor* < *estimate_distance_threshold* **then**
      add *point* to *floor_points_list*
    end if
  end if
end for
if size of *floor_points_list* > *estimate_size_threshold* **then**
  pos_mean ← the mean of points' position in *floor_points_list*
  pos_cov ← the covariance matrix of points' position in *floor_points_list*
  compute eigenvalues and eigenvectors of *pos_cov*
  current_floor_normal = eigenvector with smallest eigenvalue
  current_floor_distance = *pos_mean* · *current_floor_normal*
else
  current_floor_normal = *calib_floor_normal*
  current_floor_distance = *calib_floor_distance*
end if
for all *point* in *point_cloud* **do**
  distance_along_normal ← *point* · *current_floor_normal*
  distance_from_floor ← *current_floor_distance* − *distance_along_normal*
  if *distance_from_floor* < *remove_distance_threshold* **then**
    remove *point* from *point_cloud*
  end if
end for

Algorithm 2 Walls removal

Input: *point_ cloud*, input point cloud with point normals
floor_ normal, floor plane normal
distance_ threshold, maximum distance for the RANSAC
size_ threshold, minimum number of points in a wall
dot_ product_ threshold, maximum dot product value between wall normal and floor plane normal
Output: *point_ cloud*, output point cloud, with points from the walls removed

tmp_ point_ cloud ← *point_ cloud*
result_ point_ cloud ← *point_ cloud*
repeat
  Find *plane* and *plane_ normal* in *tmp_ point_ cloud* using RANSAC with *distance_ threshold*
  if a *plane* is found **then**
    if size of *plane* > *size_ threshold* **then**
      remove all points of *plane* from *tmp_ point_ cloud*
      angle_ dot_ prod ← | *plane_ normal* · *floor_ normal* |
      if *angle_ dot_ prod* < *dot_ product_ threshold* **then**
        remove all points of *plane* from *result_ point_ cloud*
      end if
    end if
  end if
until size of *tmp_ point_ cloud* > *size_ threshold* **and** a *plane* is found **and** size of *plane* > *size_ threshold*

4.3.5 Object candidates over-segmentation

This stage starts with the down-sampling of the point cloud using a voxel grid filter implemented using the PCL library. The voxel grid filter uses an octree structure to reduce the spatial density of points in a point cloud to a desired value. This results in a point cloud in which two points are no closer than a certain distance. The reduced number of point accelerates the remaining steps for the segmentation, but for the object recognition, the original resolution should be restored. For this purpose, an index is built during the voxel-grid filtering that lists, for each down-sampled point, the points from the original point cloud which it represents. This is the only information required to allow the restoration of the original point density. The next step after down-sampling is to project the points to the floor plane. The projection uses the down-sampled point cloud as input and consists in applying the following transformation to each of the points' coordinates p:

$$proj_{m,d}\left(p\right) = p - d\left(m \cdot p\right), \tag{4.1}$$

where m and d respectively are the coefficients and the distance of the floor plane. Finally, a greedy over-segmentation algorithm, as described in chapter 2 section 2.1.1 is used to segment the point cloud into object candidates using a maximum distance criterion. The distances are computed using the projected point cloud, but the groups are formed in the non-projected, but down-sampled, one. The original resolution will be restored, but this can be delayed so that the last step of the segmentation algorithm benefits from the lower point count. A functional view of the object candidate over-segmentation step can be seen in figure 4-2 and the parameter values used are shown in table 4.1. Take note that in this particular case, the over-segmentation algorithm is expected to produce segments that do correspond to the objects. The method used effectively is an over-segmentation algorithm, but the distance criterion used leads to a perfect segmentation of the input.

4.3.6 Object candidates rejection

This foreground-background separation step removes object candidates whose size is too small or too large, as well as candidates that are too close to a border of the point cloud. The size of the candidates is measured by counting the number of points. As the point cloud is still in its down-sampled state, the threshold on size must be adapted to the voxel grid's resolution. After the removal of these undesirable object candidates, the remaining points are replaced by the corresponding list of points from the down-sampling index to restore the original point cloud density. The parameter values for this processing stage are shown in table 4.1.

4.3.7 Parameters

Table 4.1 lists all the parameters of the geometric scene segmentation algorithm and the values used throughout the experiments.

4.3.8 Process overview

Figure 4-2 shows a block diagram of the geometric segmentation algorithm and figure 4-3 shows the segmentation of an indoor scene by the algorithm.

4.4 Experiments

The geometric scene segmentation algorithm was tested on the segmentation database described in section 3.4.1. A segmentation algorithm based on Markov random fields [17] is also tested on the database for comparison purposes. The segmentation performance results are given in the next section and interpreted in section 4.5.

4.4.1 Segmentation performance

The algorithm assigns one of three different labels to each pixel: floor, wall or object. Some examples of segmented images are shown in figure 4-4. For the sake of the

Table 4.1: Parameters of the geometric scene segmentation algorithm

Step	Name	Value
Calibration	RANSAC distance threshold	5 cm
Distance filtering	Maximum z-coordinate	3 meters
Point normals calculation	Depth change factor	0.02 (default in PCL)
	Smoothing factor	10 (default in PCL)
Floor removal	Estimation dot product threshold	0.98
	Estimation maximum distance	5 cm
	Estimation minimum size	25 000 points
	Removal maximum distance	3.5 cm
	Removal maximum height	2 m
Walls removal	RANSAC distance threshold	5 cm
	Size threshold	50 000 points
	Dot product threshold	0.05
Object candidates over-segmentation	Voxel grid resolution	2 cm
	Maximum distance	5 cm
Object candidates rejection	Minimum size	100 points
	Maximum size	10 000 points
	Minimum distance from border	10 cm

Method	Precision			Recall			Run time
	Wall	Floor	Object	Wall	Floor	Object	(per image)
Geometric	93.3	97.8	65.0	87.7	91.2	98.1	300 ms
[17] strong regul.	96.4	89.5	46.8	77.6	79.5	41.6	2 s
[17] weak regul.	94.7	89.4	88.1	82.8	80.4	23.5	2 s
[17] no regul.	94.7	88.8	64.8	81.1	78.5	32.1	2 s

Table 4.2: Performance of segmentation expressed in terms of micro-average precision and recall values for the labels "floor", "wall" and "object".

comparison with the annotated images, all floor types, wall types and objects are grouped into these three labels. The precision and recall for this experiment are reported in table 4.2. The measures are computed on a pixel-wise basis, for each label, and over the whole dataset.

4.4.2 Run time

The run time for the whole segmentation algorithm varies between 200 ms and 600 ms with an average around 300 ms. The most lengthy steps to run are the wall removal, as it is an iterative processing, and the object candidates over-segmentation. Consequently, the number of valid object candidates and the failure to remove a wall are the factors that affect run time the most. Walls that do pass the wall removal step untouched usually get removed by the object candidate rejection, but the increase computation required to downsample and cluster them can really hurt performance.

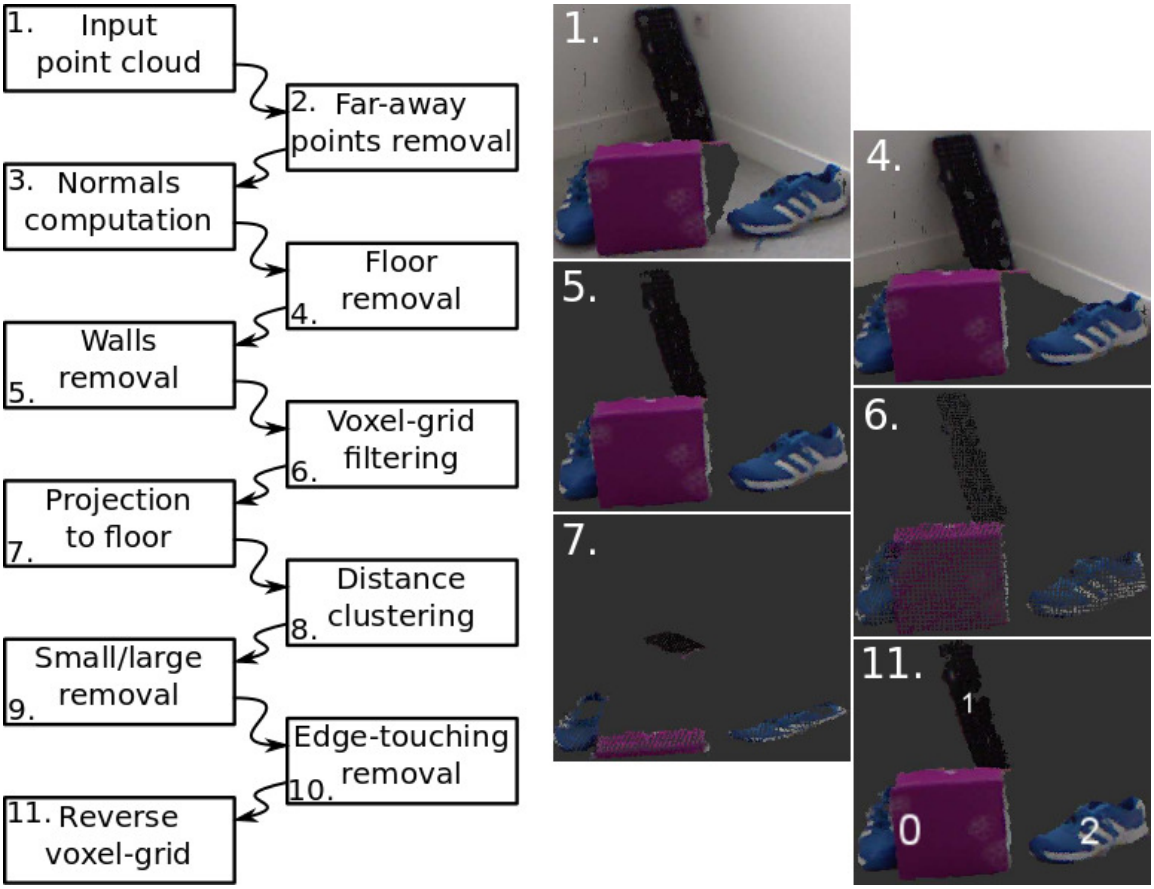

Figure 4-2: **Left:** A functional overview of the geometric scene segmentation algorithm. **Right:** The resulting point cloud after keys steps of the segmentation.

The presence of large valid object candidates increase run time for the same reason. It is possible to imagine a scene where the algorithm takes much longer to run, but this case was not encountered during the online experiments.

4.4.3 Use cases

Figures 4-5 and 4-6 show specific examples of successes and failures of the segmentation algorithm. See the figure captions for detailed interpretation. Note that the top part of 4-5 shows a part of a large cupboard labeled as a wall. This is considered as a success because the cupboard is too large a piece of furniture to be considered an object of interest here. It is desirable that the segmentation algorithm removes it from the point cloud. The fact that a part of it is removed during the wall removal step is good, since it involves fewer computations to remove it at this point compared to removal during the object candidates rejection step.

Black objects also are problematic for the geometric segmentation algorithm. This is due to the fact that black objects are generally badly perceived by the Kinect's depth camera. This causes black objects to be incomplete as shown figure 4-7.

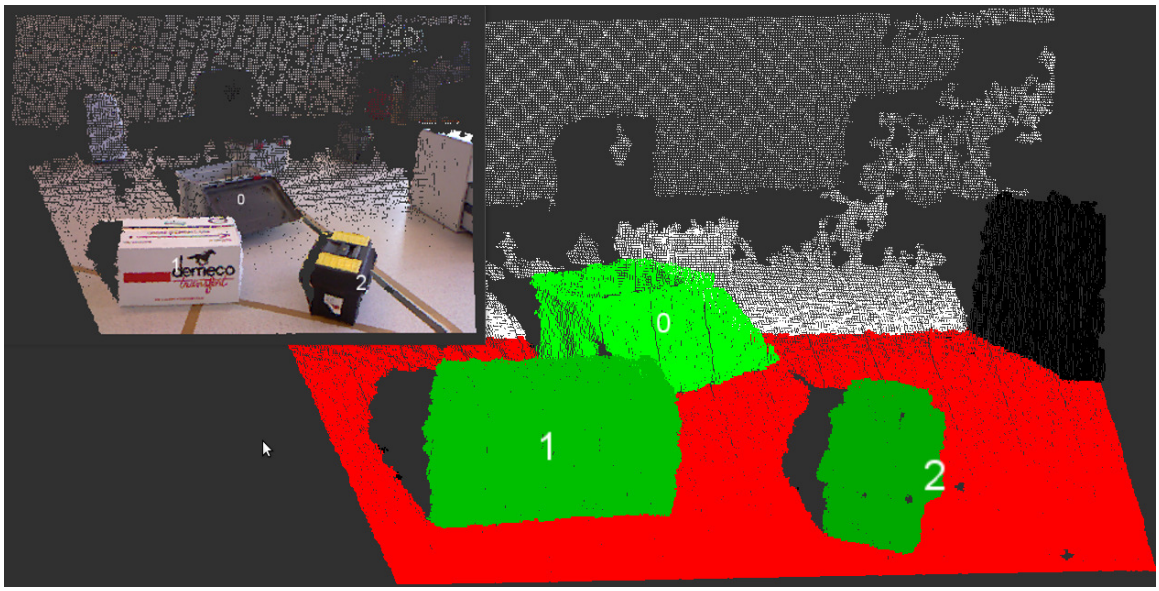

Figure 4-3: An indoor scene processed by the geometric scene segmentation algorithm. The top-left inset shows the original point cloud. In the interpreted scene, the floor is red, the objects are in green (and are numbered). Points colored in white are filtered out because they are too far from the robot and the object in black is discarded because it touches a border of the point cloud.

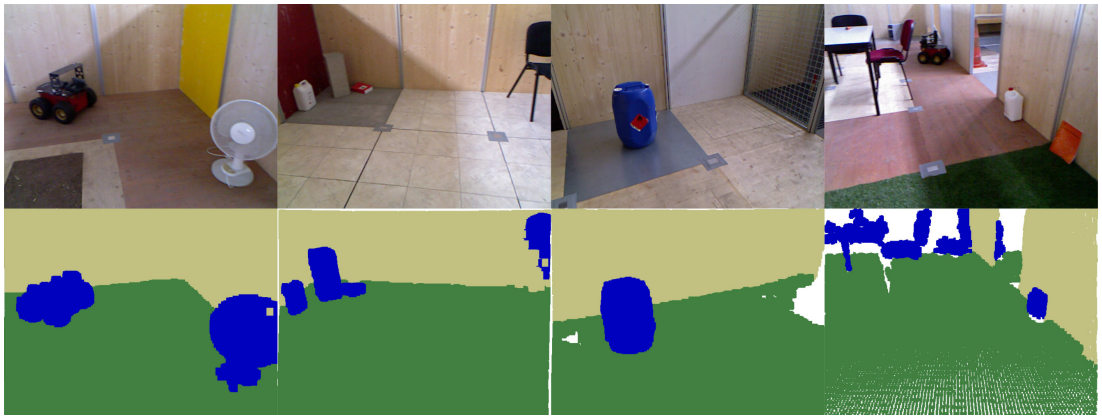

Figure 4-4: **Top.** Scenes from the segmentation database. **Bottom.** Segmentations with floor in green, walls in beige and objects in blue.

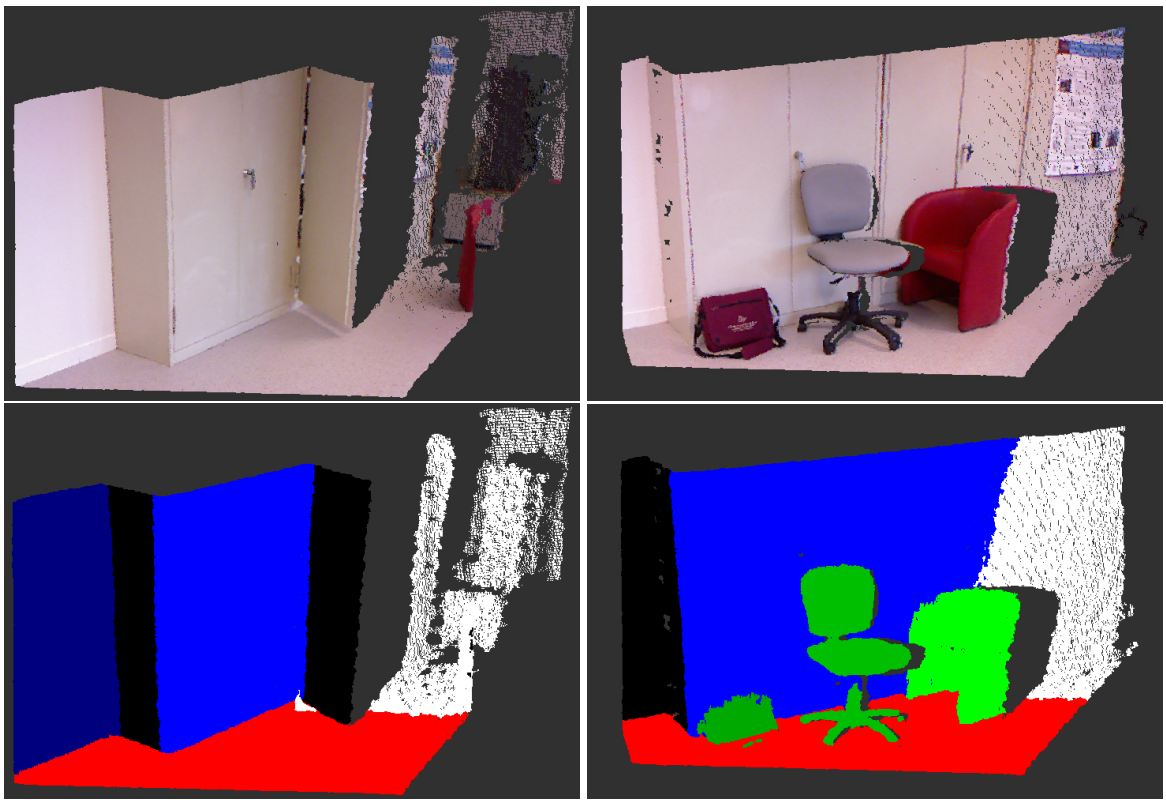

Figure 4-5: Examples of successful segmentations. The left column shows the original point cloud and the right column shows the scene interpretation where red is "floor", blue is "wall", green is "object", black is "rejected" and white is "too far". Different shades of blue and green indicates different walls and objects. **Left.** The scene shows a piece of wall and a large cupboard. The segmentation algorithm successfully identifies the floor and the piece of wall. The entire cupboard also is labeled as non-object (partly because it qualifies as a wall and because it touches the point cloud's border). **Right.** The scene shows three objects touching the cupboard. As the cupboard is identified as a wall, the objects are successfully segmented even though they are in direct contact with it (but apart from each other). The wall removal step cuts a part of the red bag, but a significant part of it still is identified as an object. Note that all three objects are successfully identified as individual objects (the shade of green for the bag is slightly darker than for the chair).

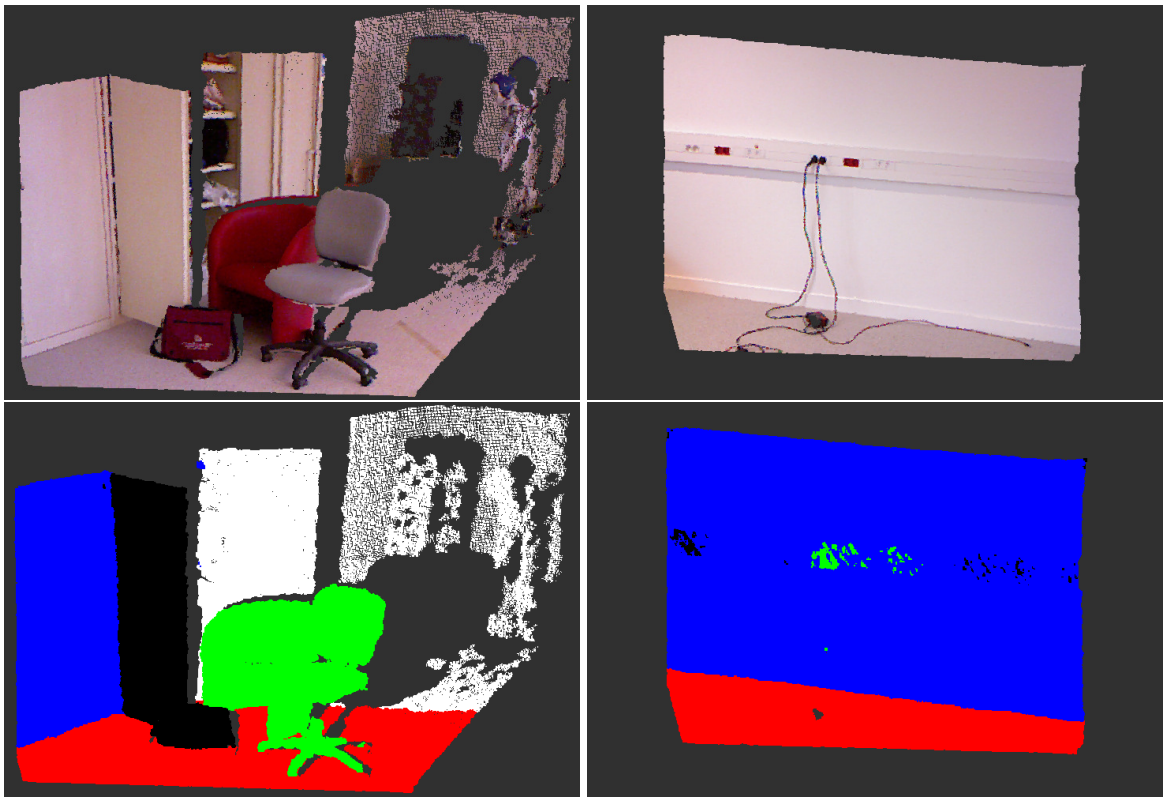

Figure 4-6: Examples of failed segmentations. The left column shows the original point cloud and the right column shows the scene interpretation where red is "floor", blue is "wall", green is "object", black is "rejected" and white is "too far". Different shades of blue and green indicates different walls and objects. **Left.** The scene shows the sofa and chair touching each other and the bag and cupboard door touching each other. The sofa and chair are merged in a single object candidate. The bag and door also are merged into a single object candidate and are rejected as a whole because the door touches the top border of the point cloud. **Right.** The scene shows an uneven wall and some power cords. Most of the wall is labeled correctly, but the uneven part is wrongly labeled as an object. The power cords are too small to be perceived by the depth camera.

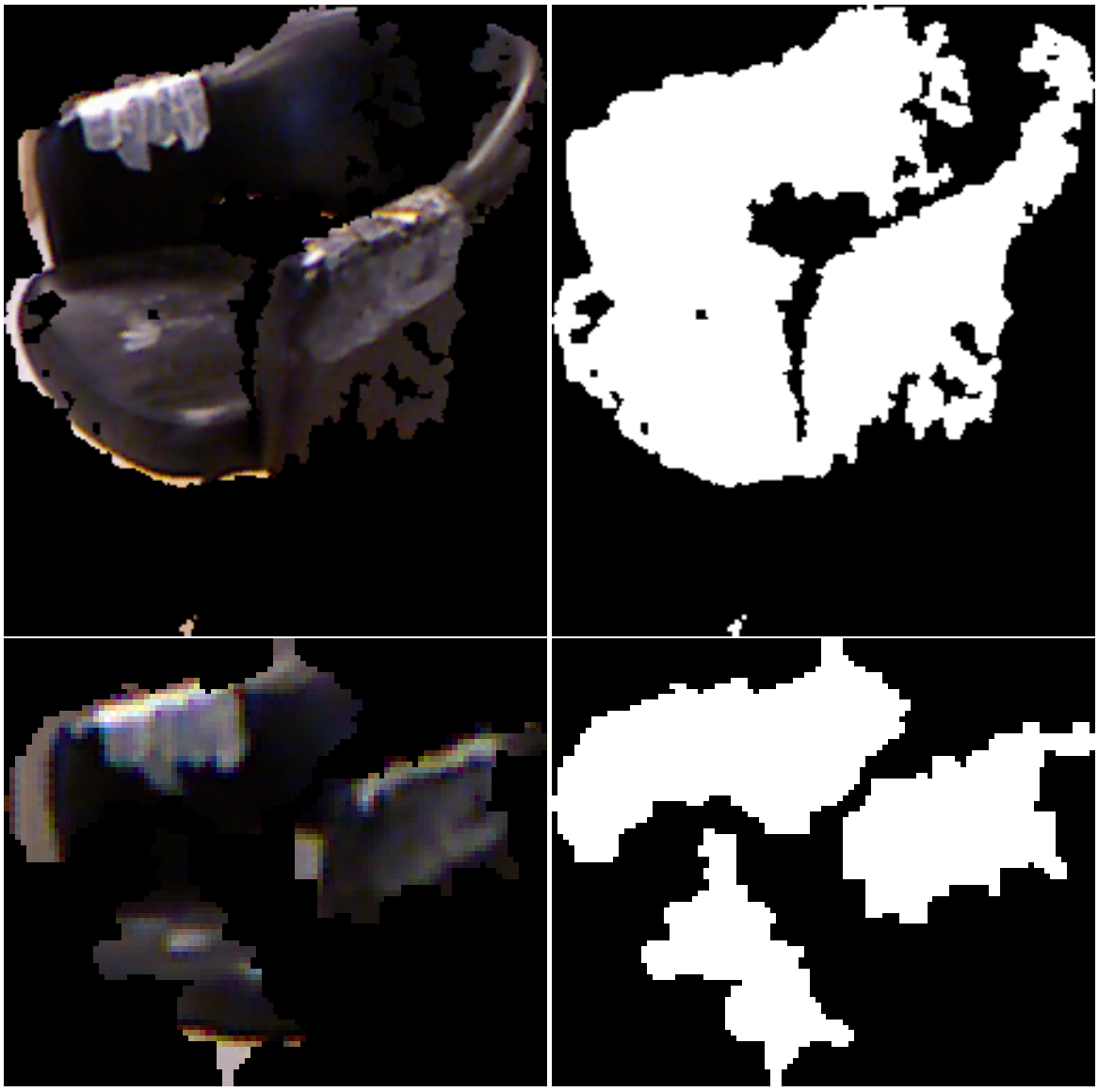

Figure 4-7: Example of incomplete black object segmentation. The left images show the color image of the segmented black chair and the right image the segmentation mask. **Top** A rather good segmentation of a black armchair. **Bottom** A bad segmentation of the same black armchair due to bad perception from the depth camera.

4.5 Discussion

The two most important qualities for the segmentation algorithm for the application is a low run time and a high recall for objects. First, the processing speed of the segmentation algorithm must be as low as possible because part of its utility is to reduce the processing time of the overall pipeline. Second, the recall for object is more important than the other measures because of the way the segmentation is used in the processing pipeline. The segmentation step simply discards pixels that are not labelled as an object. Consequently, any actual object that the segmentation algorithm fails to label as such will not be processed by the subsequent object recognition step. This situation is irreversible and undesirable. A high recall for object ensures it happens as rarely as possible. Other mislabellings have smaller consequences. For instance, pixels that are mistakenly labelled as objects will be treated by the object recognition step and hopefully will be filtered out by it.

The geometric segmentation algorithm scores higher than the MRF algorithm both for object recall and processing speed: 98.1% object recall versus 41.6%, and 300 milliseconds versus 2 seconds. The precision for the object class can also be regarded, as a high precision means that fewer useless pixels (pixels that are not objects) will have to be treated by the object recognition. In this regard, both the geometric and the MRF algorithms achieve a comparable performance of about 65%.

It is not surprising that the geometric algorithm performs better than the MRF. The geometric segmentation is specifically designed to operate on indoor scenes, and the test dataset is entirely composed of such images. The MRF algorithm, on the other hand, could be used with some success on any type of images, given a training set is provided. Secondly, the MRF algorithm aims at identifying objects in a scene based on their appearance and shape. This is a difficult task, because objects of almost any form and color exist. It is very unlikely that an algorithm can effectively capture the statistics of such a diverse class. On the other hand, the geometric algorithm aims at finding in an indoor scene everything that is not an object. Floor planes and walls have much simpler characteristics that can be identified. In indoor scenes, this is a much more reasonable objective which leads, as demonstrated here, to better performance.

4.6 Conclusion

A geometric indoor scene segmentation algorithm was presented and evaluated on a database. The database is composed of images taken by a robot during an exploration task in the Carotte competition. The algorithm is similar to some table-top segmentation algorithms used for robot manipulation and object recognition. It consists in thee main steps: the removal of floor and walls, the clustering of points into object candidates and the rejection of unwanted object candidates.

For the purpose of object recognition, the most important quality is its capability to quickly and correctly segment objects. The distinction between wall and floor is not useful here because both are part of the background. The geometric scene

segmentation algorithm described here, with a recall score of 98.1% for the object category and a run time of 300 milliseconds, clearly is suitable as the first stage of an indoor object instance recognition algorithm. The comparison with the MRF algorithm also shows that it was appropriate to develop a simple algorithm based on geometry. As the application context is defined, the robot should operate indoors, such a simple algorithm can be perfectly adapted. It would most likely fail in a different environment where an MRF based algorithm would adapt appropriately.

The output of the algorithm is a list of point clouds. Each of these point clouds depicts a single object to recognize. A first object recognition algorithm based on a neural network classifier is presented in chapter 5. An alternative approach using a nearest neighbor classifier will also be described in chapter 6.

Chapter 5

Global object recognition by fusion of shape and texture information with a feed-forward neural network

5.1 Introduction

This chapter introduces and tests the neural network based classifier for object recognition in a robotics experiment. This classifier is the second part of the object recognition pipeline. The first part, the geometric scene segmentation, is described in chapter 4. The output of the segmentation stage is a list of point clouds, each one representing an object from the scene in front of the robot. The role of the classifier is to identify the object from each of the point clouds provided by the segmentation. Two constraints for the classifier to work are that: the objects come from a given list of objects of interest, the point clouds do show one and only one of these objects. Then, the classifier associates to each input image a list of values that indicate how likely it is that each object of interest be the one represented in the point cloud. The next section presents and defends the design choices for the classifier. Implementation details are given in sections 5.3, 5.4 and 5.5 which respectively describe the features used, the neural network and the multi-view fusion method. The recognition rate and precision and recall figures for two different types of experiments are given in section 5.6. A first series of experiments are lead to validate the design of this classifier on offline data. The second series of experiments test the whole object recognition prototype in an online robotics experiment. This second experiment specifically aims to test whether is it possible to perform object classification on a robot when the learning is done on data taken in controlled conditions.

5.2 Design choices

The neural network based classifier for object recognition is a two-step process: feature computation and classification. This is a widespread structure in computer vision. There are quite a few choices involved in the design of a feature-based object

recognition algorithm: the features to use, the way these features are combined, and the classification procedure.

First, the choice of the features is discussed. Features are the result of mathematical operations applied to a given input. Once the features are computed, the original input is discarded. That is, any information not conveyed by the features themselves is ignored by the classifier. Features in computer vision are crafted so that they are invariant to some specific changes in the input. Being invariant to a change means that the resulting values will not change if the input suffers this change. That is, the feature is blind to certain aspects of the input. It is rather difficult to design features which are blind solely to superficial changes. Oftentimes, the desired invariance also induces undesired ones. For instance, working with levels of gray provides invariance to many illumination changes, but it also renders blind to the actual color of objects. Rather than hand-crafting a flawless feature, it is chosen here to use several imperfect but complementary ones. In this chapter, the combination of a shape, a texture and two color features will be tested. The texture descriptor is a histogram of occurrence of SIFT words, described in section 5.3.1. SIFT is the feature of choice for object retrieval algorithms and should work as well in the current context. A shape feature also is chosen to benefit from the 3-dimensional data provided by the Kinect. The surflet-pair relation histogram is based on the Surflet-pair relation feature described in chapter 3, section 3.3.7. It is chosen for its invariance to viewpoint changes. A feature based on the hue channel of the HSV color representation is also used and described in section 5.3.2. It is a simple way to add color information while presenting some invariance to light changes. Finally, a feature that approximates the probability density function of each color channel, namely the transformed RGB feature, adds more invariant information from color. It is detailed in section 5.3.3. All used features take the form of histograms. In this chapter, the spreading strategy described in chapter 2, section 2.4.2.is not used. This results in a faster algorithm, but as subject to aliasing. The spreading procedure will be put to use in the features of chapter 6.

If several features are used, a single answer still needs to be provided by the classifier. This raises the question of combination, or fusion of the information. Each feature can lead to different or contradicting results, and a mechanism must exist to funnel them into a single output. For this prototype, the choice of using a feed-forward neural network makes the fusion easy: the neural network can take care of it. The versatility of the neural network comes very handy in this situation. It seamlessly accommodates inputs of different nature and learns, from the database, how to combine the different pieces of information to correctly predict the object's class. The ease of use is a prime argument in using a neural network for this prototype. One of the limitations of a neural network is that input and output sizes are fixed. That is, the number of values used to encode a given input point cloud cannot change from one example to another. This can be problematic when using SIFT features with a detector. The SIFT detector finds regions in an input image where it is judicious to compute SIFT features. The number of regions found depends on the image itself and is not fixed. One way to make the use of a detector compatible with the neural network is to build a histogram of occurrence of SIFT words (see chapter 2, section 2.2.4). It retains the benefits of using a detector while allowing to fix the number of

values presented to the neural network.

When a test example is presented to the neural network, the activation of the output neurons should indicate how likely it is that each object from the database be the one represented in the input point cloud. This list of activations must be transformed into a single prediction of what object is there. It is natural that the predicted object be the one represented by the neuron with the highest activation value. It is also natural to mitigate the answer if two or more neurons have a similarly high activation. The softmax activation function is a simple mechanism to transform the activation of the individual output neurons based on the value of the others output neurons. It increases the output value of the neuron with highest activation and decreases the output of all other neurons. It is a way to turn the output values of the neural network into a confidence value which depends on the activation of all the output neurons. The final classification strategy is to assign to the input point cloud the label corresponding to the highest activation neuron, knowing that this activation is computed using a softmax function. The exact value of this activation should also function as a measure of confidence.

One final aspect of this prototype is the fusion of recognition scores from several detections of the same object. The multi-view fusion of recognition scores is one way to benefit from the use of a robot. As a robot moves around, it is likely to see the objects under different, sometimes more favorable, angles of view. One way to exploit this is to perform an early fusion of the data for example by using a scene reconstruction algorithm. Scene reconstruction is a lengthy process and there is not always enough information to effectively reconstruct all the objects in scene. If the object recognition algorithm is designed to work on reconstructed scenes, it is likely to fail in the cases where reconstruction is not possible. To prevent this, a late fusion is performed here. The object recognition algorithm works on each view of the objects and the results of this operation is fused based on position information. A newly detected object is fused with a previously seen object if their map positions are close to each other.

Figure 5-1 shows a block diagram of the proposed object recognition process.

5.3 Description of the features

5.3.1 Histogram of occurrence of SIFT words

The histogram of occurrence of SIFT words is similar to the first part of the widely used bag of visual words (BOW) for visual recognition [79] technique. The method consists in an offline vocabulary building step and an online feature calculation step. The vocabulary is obtained by computing SIFT features on all images of the offline database and clustering them using K-means and L_2 distance. The resulting clusters are called words and they form the vocabulary. A number of words of 100, 1 000 and 10 000 were tested without notable difference in the behavior of the algorithm, so the final implementation uses a vocabulary of 100 words.

The online recognition of an image proceeds in three phases: feature extraction,

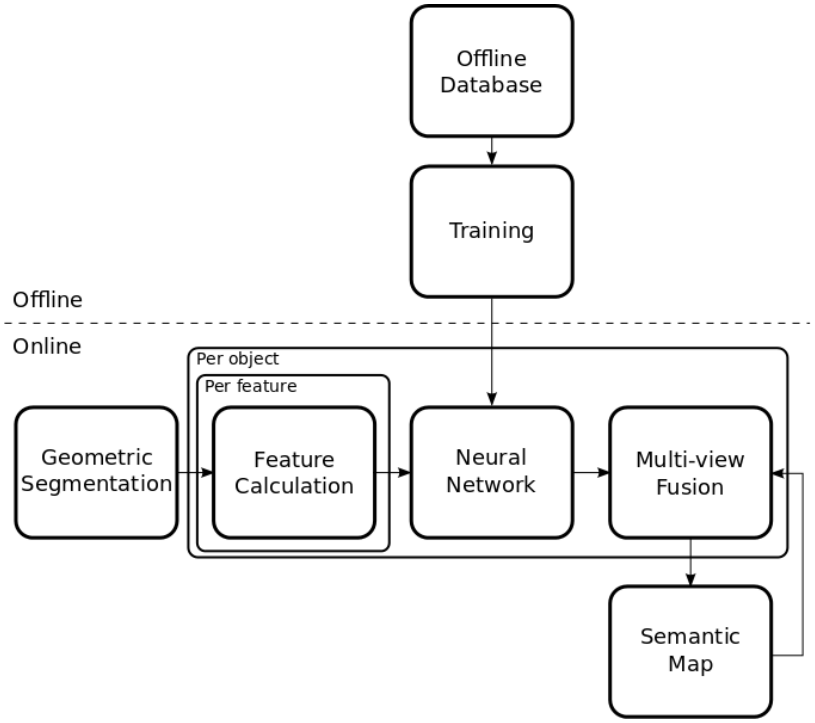

Figure 5-1: Block diagram of the object recognition process based on the neural network classifier. The offline section is executed once for a given database and set of parameters. The online section is executed in loop during operation. The frames show which operations are duplicated for the number of features used and the number of objects returned by the segmentation step.

vocabulary matching and histogram compilation. The extraction is where SIFT features are extracted from the image. During matching, each of the extracted SIFT feature goes through a nearest neighbor matching process with the words from the vocabulary. That is, for each extracted feature, a search is performed to find the most similar word from the vocabulary. The L_2 distance is used to compare SIFT features. The histogram of occurrence of SIFT words is the result of compiling the number of times each word from the vocabulary was matched to a feature extracted from the unknown image. The resulting histogram has as many bins as there are words in the vocabulary (in this case: 100). The histogram is normalized so that the sum of its values is 1 (L_1 normalized).

5.3.2 Saturation weighted hue histogram

The color feature used here is based on the HSV representation, described in chapter 2, section 2.3.2. The hue channel gives direct information about the color of a pixel and is invariant to light intensity changes. The hue value however tends to be unstable as the colorfulness of a pixel decreases. The saturation channel can be used as a measure of confidence in the hue value, since saturation increases with colorfulness. The feature used here thus is a histogram of hue weighted by saturation. The bin to which a pixel contributes to the histogram depends on its hue, and the weight of this contribution equals its saturation. The hue channel is naturally bounded by 0 and 6 and the histogram splits this interval in 16 bins. After the 16-bin histogram is compiled, it is L_1 normalized.

5.3.3 Transformed RGB histogram

The histogram of hue described in the previous section encodes the color of the object, but is not invariant to light color changes and shifts. To test the usefulness of features with invariance to all light source changes mentioned in chapter 2, section 2.3, a feature based on the transformed RGB representation is also designed. This feature is computed by applying the three following steps to each of the R, G and B color channels. First, the mean and standard deviation for the channel is computed for the whole input image. The value of each pixel is then normalized using the computed mean and standard deviation. The resulting values are not naturally bounded and must thus be clipped before being added to the histogram. Clipping values at -3 and 3 gives a good trade-off between distortion due to clipping and precision of the approximation. Values are then cumulated in a separate 16-bin histogram for each of the R, G and B channel. The transformed RGB histogram is the concatenation of these three histograms and is also L_1 normalized. The transformed RGB feature thus is a 48-bin histogram.

5.3.4 Surflet-pair-relation histogram

A histogram of surflet-pair-relation feature is a point cloud descriptor of shape that is invariant to rotations and distance changes. It is obtained in the following way. First,

10 000 pairs of points are randomly drawn from the object's point cloud. The number of point pairs must be large enough to capture the statistics of the shape of the obtain while remaining computationally tractable. For the size of the objects in the ENSTA objects dataset, using 10 000 pairs was deemed reasonable. For each pair, the surflet-pair-relation feature described in chapter 2, section 3.3.7 is computed. The feature is computed using the PCL [69]. The surflet-pair-relation feature is a 4-dimensional descriptor. The 3 first values of the descriptor are naturally bounded, since they are angular measures. The last value is a distance measure which must be bounded. One way to do so is to turn it into a relative distance by dividing it by the size of the object (the size is defined as the maximum distance between any two points of the object's point cloud). All dimensions are split into 5 intervals (as recommended in [91]), giving rise to a 625-bin histogram. The histogram is L_1 normalized.

5.4 Data fusion with a feed-forward neural network

The feature histograms described in the previous section are concatenated and form the input of the feed-forward neural network. A three-layer network is used. The size of the input layer is the sum of the size of the features used. The output layer contains one neuron for each object instance to recognize. The hidden layer is chosen to contain 50 neurons. Hidden layer sizes of 25 and 75 were also tested without any notable difference in the behavior of the algorithm. An elaborate strategy to find the optimal size of the hidden layer was not conducted because the neural network is used only as a proof of concept leading to the developments detailed in chapter 6. The results obtained with a hidden layer of 50 neurons was satisfying and this configuration was kept. All the neurons have a sigmoid activation function except the last layer's where a non-local softmax function is used. The softmax function modifies the whole vector of activation values x according to equation 5.1.

$$o_i^{softmax} = \frac{e^{x_i}}{\sum_{j=0}^{K} e^{x_j}}, \tag{5.1}$$

where $o_i^{softmax}$ is the ith element of the output softmax vector, x is the output neurons' activation vector and K is the number of output neurons. The softmax function modifies each element of the activation vector based on the value of all the other elements. The softmax function produces values in the range $[0, 1]$. It tends to set the value of the highest input element close to 1 and the other elements close to 0..

5.4.1 Training

The network is trained on a subset of data called the training set. This set is split in two parts. The first part consists in 90% of the samples from the training set and is used for learning the neural networks' weights. The training is done using the resilient back-propagation algorithm (RProp) and early stopping (see chapter 3, section 3.3.8). The RProp algorithm is run for one epoch (that is, it is applied to the entire training set), and then the other part of the training set (the remaining 10%)

is used to compute a mean square error (MSE) measure for the current weights of the neural network. The MSE is stored and the process is repeated until the MSE increases for 5 consecutive epochs. This early stopping procedure helps preventing overfitting by the neural network.

5.5 Multi-view fusion by position

During the online experiments, the recognition algorithm is provided with map information thanks to the Hector SLAM module (chapter 3, section 3.3.5). Hector SLAM outputs a metric map of the environment and the robot's position within this map. Knowing the relative positions of detected objects with regard to the robot, it is possible to compute their absolute position in the metric map. The position of an object is computed by calculating the centroid of the points in the object's point cloud. This absolute position will be used to save the objects' recognition scores given by the neural network in the map. Each time an object is seen, its absolute position is compared to all the positions stored in the map. The new object's recognition scores and position are fused with that of the closest stored object if they are not farther than 30 cm. Otherwise, a new object is created in the map at the corresponding position. The fusion of the recognition score is a weighted average of all views of the object. The number of points in an object's view is used to weight the average. A high number of points either means the object is closeby or the point cloud shows a more complete view of it. In both cases, the recognition should have more chances to be successful and it is worth it to advantageously weight it during the averaging operation. The numbers reported for the map-online experiments which use fusion by position are the final ones saved in the map.

5.6 Object recognition experiments

The object recognition framework is benchmarked using the ENSTA object recognition database (offline and online). The first series of experiments is done on the offline database. The goal of these experiments is to find which combination of features gives the best recognition rate. The tests are conducted using all possible combinations of features, including single features. In the figures, the features names are abbreviated: "sift" is the bag of SIFT descriptor (section 5.3.1), "trgb" is the transformed RGB histogram (section 5.3.3) and "hue" is the hue weighted by saturation histogram (section 5.3.2), and "sprh" is the surflet-pair-relation histogram feature (section 5.3.4).

For all experiments, a cross-validation process is conducted by training the neural network two times. Each time, a random subset consisting of 90% of the offline database is selected as the training set and the remaining 10% acts as the validation set. This split is done randomly. Note that according to the procedure in 5.4.1, training sets are further split in two parts to allow for the early stopping procedure. For each fold of the cross-validation, training is performed twice using a different random initialization of the network's weights. This means that a total of four neural net-

works are trained for each combination of feature to evaluate. The results presented here are the average of the performance using these four neural networks.

For the offline experiment, the network is trained on the training set and performance measures are computed on the validation set. All performance measures are micro-averages of all experiments done for a given combination of features. The recognition rates are shown in table 5.1 and the precision-recall curves are shown in fig 5-3. The online experiment uses the networks obtained from the offline experiment and tests them on the online database. In a first experiment, recognition is done on each individual object from the online database. In a second experiment, the multi-view fusion method described in Sec. 5.5 is used.

5.6.1 Recognition rates

For all three experiments and feature combinations, table 5.1 summarizes the recognition rates. Also, figure 5-2 shows the confusion matrix of the online experiment using the combination surflet-pair-relation histogram, transformed RGB and the occurrence of SIFT word features. The results presented here are commented in section 5.7.

Table 5.1: Recognition rates for the offline, single-view online and multi-view online experiments using different combinations of features.

Features				Offline	Single	Multi
sift	trgb	hue	sprh		online	online
✓				76%	29%	51%
	✓			87%	26%	31%
✓	✓			92%	42%	68%
		✓		76%	16%	18%
✓		✓		92%	32%	36%
	✓	✓		94%	24%	26%
✓	✓	✓		95%	34%	42%
			✓	92%	**65%**	70%
✓			✓	96%	62%	72%
	✓		✓	**97%**	63%	**79%**
✓	✓		✓	**97%**	61%	**80%**
		✓	✓	**97%**	51%	53%
✓		✓	✓	**98%**	46%	65%
	✓	✓	✓	**98%**	51%	47%
✓	✓	✓	✓	**98%**	54%	68%

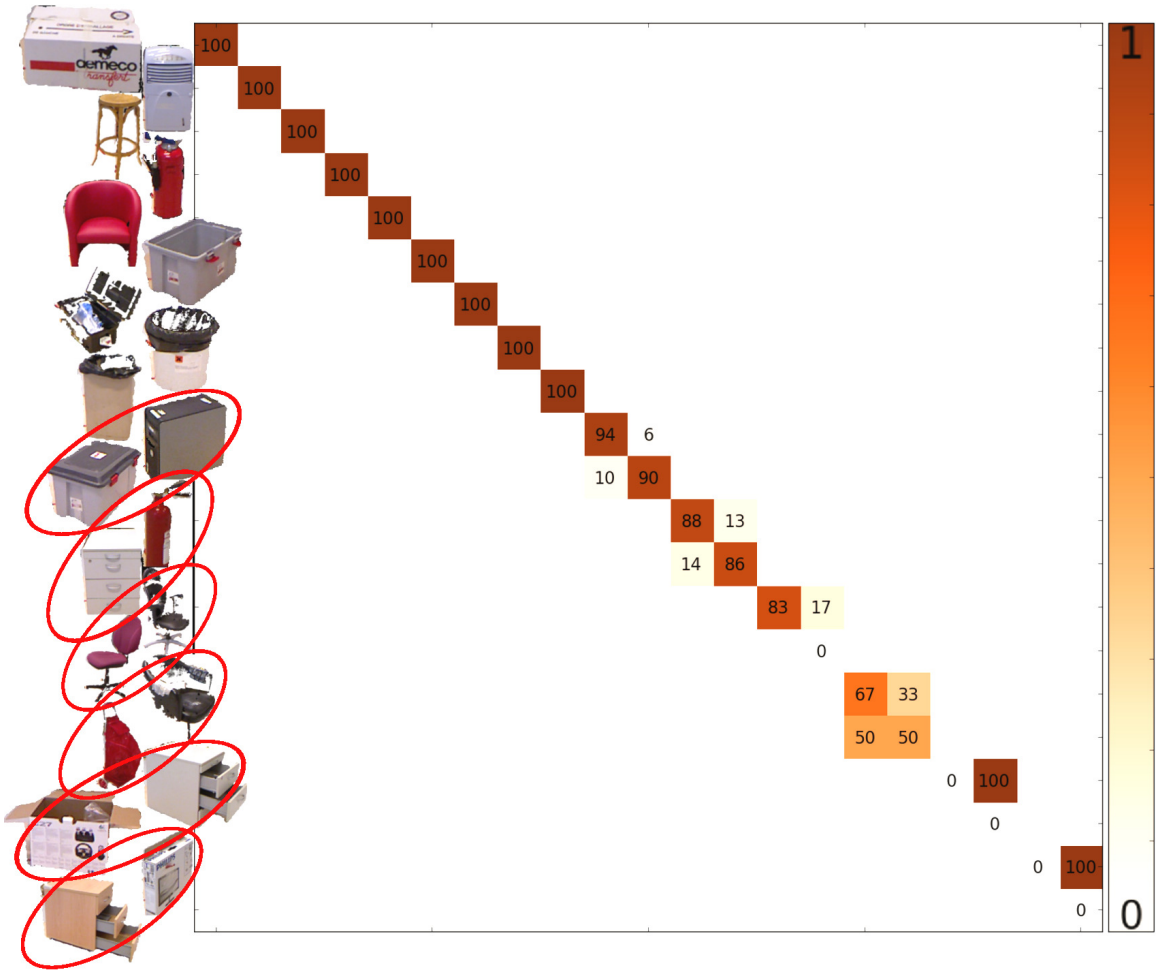

Figure 5-2: Confusion matrix for the multi-view online experiment using the neural network with the surflet-pair-relation histogram, transformed RGB and the occurrence of SIFT word features. The vertical axis shows the objects that were present in the experiment and the horizontal axis reflects the answers given by the classifier. The circles objects indicate the confusions that were made by the classifier.

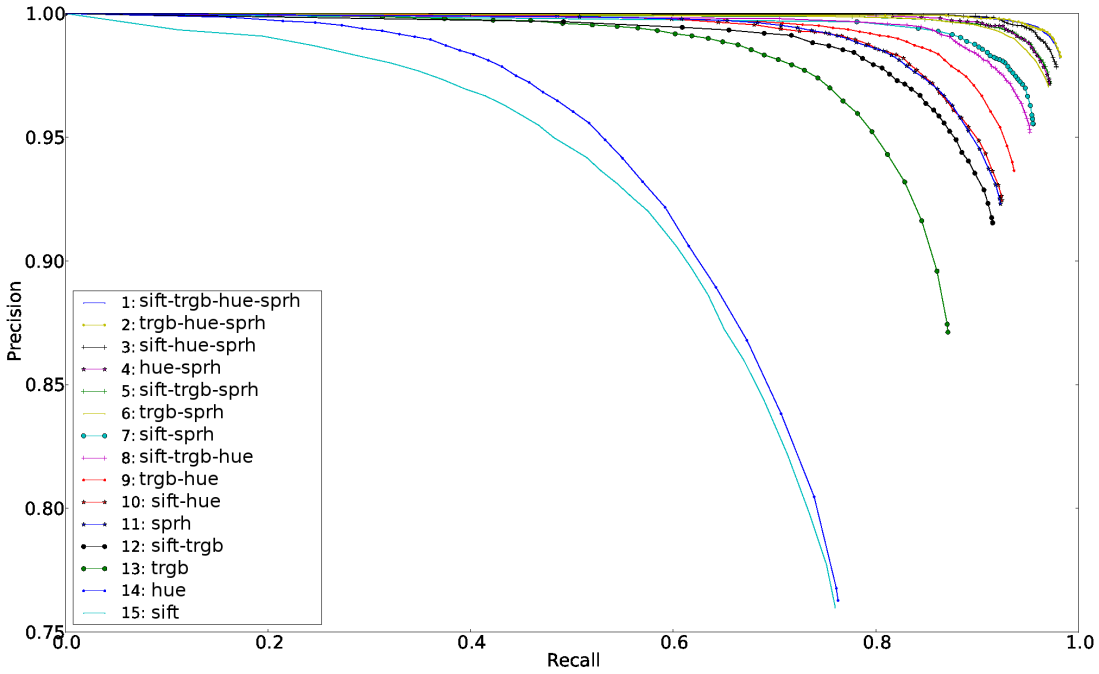

Figure 5-3: Micro-average precision-recall curves for object recognition in the offline experiment. The entries in the legend are ordered with the best performing feature set first (measured by the area under the curve). Features names are abbreviated: "sift" is the texture descriptor described in section 5.3.1, "trgb" is the transformed color histogram (from section 5.3.3) and "hue" is the hue-weighted-by-saturation histogram from section 5.3.2, and "sprh" is the shape feature from section 5.3.4

5.6.2 Precision-recall curves

Precision-recall curves for the three experiments are generated by using the activation of the winning neuron as a threshold for rejecting answers. The curves are shown in figures 5-3, 5-4 and 5-5 and will be commented in section 5.7.

5.6.3 Semantic map

The final result of the online experiment is a semantic map containing the knowledge acquired by the robot. The room in which the experiment was conducted in shown in figure 3-12 in chapter 3, section 3.4.3. The resulting semantic map is depicted in figure 5-6.

Figure 5-7 shows some successes and failure cases in the semantic map. The notable successes are the correct segmentations of objects in contact with walls. The main failure cases are the duplication of certain objects, the failure to remove background clutter and the incomplete segmentation of black objects. The duplication of some objects is due to a bad calibration of the camera which measures the distance to the objects and the laser used to build the map. Imperfect synchronization between localization and the capture of point clouds can also lead to variations in the computation of an object's position.

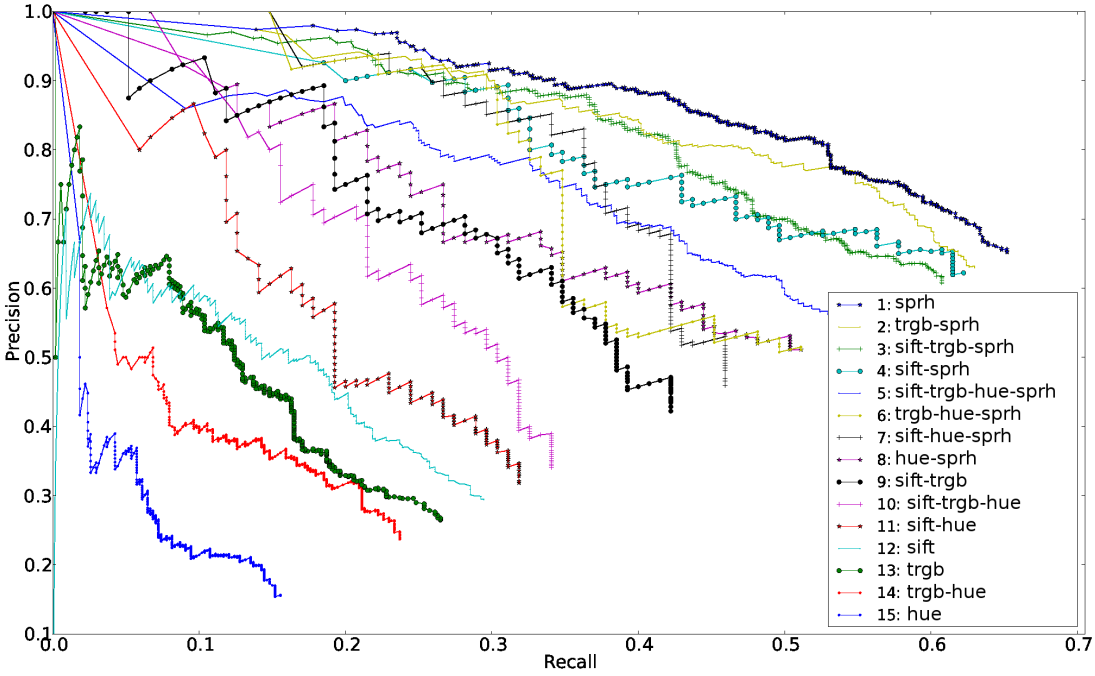

Figure 5-4: Micro-average precision-recall curves from the single-view online experiment. The entries in the legend are ordered with the best performing feature set first (measured by the area under the curve). Features names are abbreviated: "sift" is the texture descriptor described in section 5.3.1, "trgb" is the transformed color histogram (from section 5.3.3) and "hue" is the hue-weighted-by-saturation histogram from section 5.3.2, and "sprh" is the shape feature from section 5.3.4

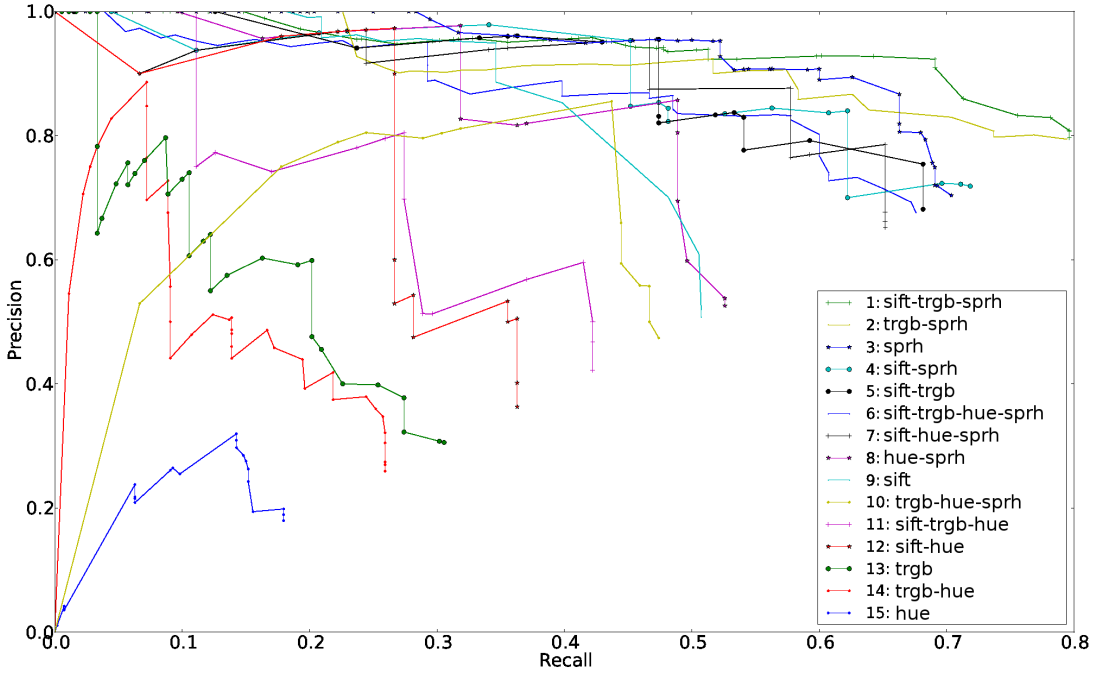

Figure 5-5: Micro-average precision-recall curves from the multi-view online experiment. The entries in the legend are ordered with the best performing feature set first (measured by the area under the curve). Features names are abbreviated: "sift" is the texture descriptor described in section 5.3.1, "trgb" is the transformed color histogram (from section 5.3.3) and "hue" is the hue-weighted-by-saturation histogram from section 5.3.2, and "sprh" is the shape feature from section 5.3.4

Figure 5-6: The semantic map resulting from the online experiment. The map is built using Hector SLAM and the laser range finder. One top of the maps, all the objects returned by the geometric segmentation algorithm are pictured. The object images hollow because they are the actual point clouds captured by the Kinect. The red trail is the path followed by the robot. This image is a different view of the semantic map shown in chapter 1, figure 1-2

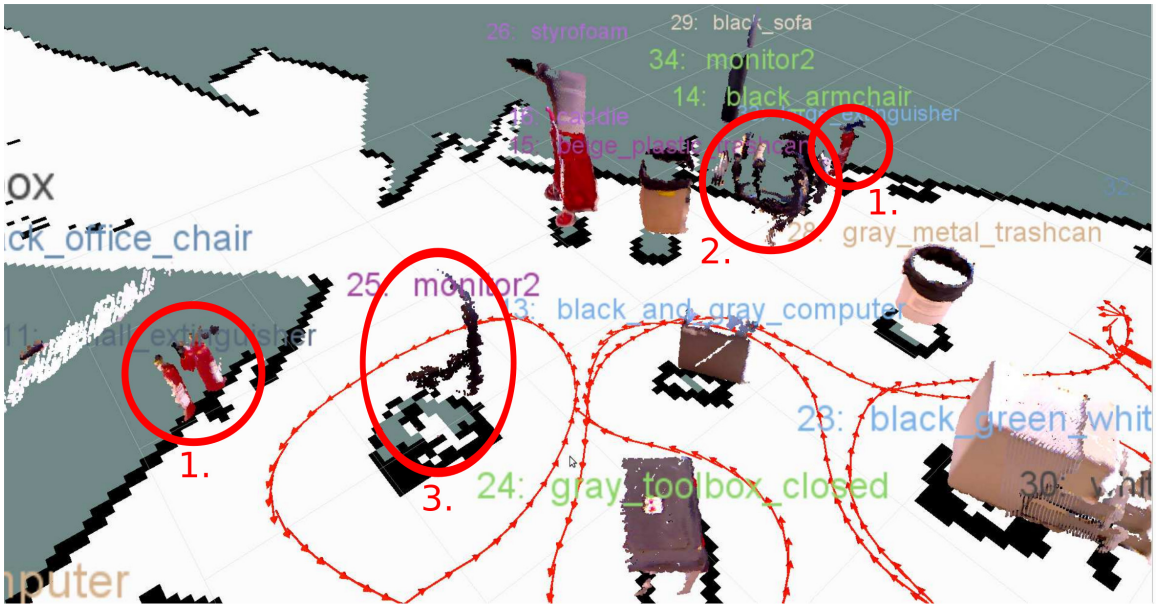

Figure 5-7: Zoom on the semantic map. The numbered red circles interesting points in the map. **1.** Successful segmentation of objects in contact with a wall. Also duplication of the extinguisher on the left. **2.** Failure to remove background clutter. **3.** Incomplete segmentation of a black object.

5.6.4 Run time

The training of a single network takes about one day using one core of a Core 2 Duo CPU @ 3 GHz, so the cross-validation process for a given set of parameters takes around four days. The feature computation takes about 100 milliseconds. The exact duration of the feature computation depends on the feature used and the size of the objects in the scene. The surflet-pair relation histogram, for instance, used a fixed number of randomly drawn points from an object's point clouds. The time required to compute this feature thus is adjustable and does not depend on the size on the input point cloud. The time to compute the SIFT feature increases with the size of the input point cloud and the number of interest points found by the detector. The two other features are computed on all points from the object, their run time increases linearly with the size and number of objects in the scene. The neural network has a fixed computational cost. This cost is proportional to the number of weights in the network. In the robotics experiment, the classification of the features from a single point cloud took less than 50 milliseconds.

5.7 Discussion

The main goal of this chapter was to develop a classifier to recognize objects segmented out of indoor scene. The classifier is composed of feature calculators and a feed-forward neural network. For an online scenario, an additional fusion by position step can integrate the recognition scores of several views of the same object. A complementary set of four features is studied: occurrence of SIFT words for texture, saturation weighted hue and transformed RGB for color and surflet-pair-relation features for 3-dimensional shape.

The offline experiments show that the neural network can seamlessly fuse the information provided by these diverse sources. The surflet-pair-relation histogram feature performs best individually (92%), but the best recognition rates are obtained when several complementary features are used. All combinations of features that include the surflet-pair-relation histogram give between 96% and 98% of good recognitions on offline data. This emphasizes the suitability of the shape feature and its compatibility with other visual features. The results of the online experiments indicate which feature is more suitable for recognition on online data even if training is done on offline data. The surflet-pair-relation histogram is again the best performing feature in the simple online experiment. The saturation weighted hue histogram is by far the worse feature in this experiment. All combinations that include hue perform worse or equal than the ones excluding it. In this first online experiment, the recognition scores for each individual view of the objects are considered. Some of the inputs to the classifier are very bad view of the object including occluded and blurry views. The good recognition rate of the surflet-pair-relation histogram (65%) suggest this feature is less sensitive to this kind of perturbation than the other features. Since all other features perform much worse than the shape feature, combining them does not work very well either. The figures change for the multi-view online experiment. In

this experiment, the best performing feature combination is the surflet-pair-relation histogram and transformed RGB with (80%) and without (79%) the occurrence of SIFT word feature. This leads to the conclusion that SIFT does not give much more information on the object than the transformed RGB does, when they are used in combination with the shape feature. Recognition rates for the combinations that include hue still are lower than when it is not used. From the two online experiments, it can be concluded that the hue is too specific to the precise conditions where an object is and cannot be used as the input to a neural network for robotics experiments. A different feature should be used to encode color information is such a scenario.

The large gap between offline and online performance (from 98% to 65%) is easily explained. As can be seen in the examples provided in chapter 3, section 3.4, the nature of the offline views differs greatly from that of the online ones. The differences include lighting conditions, shapes, occlusions and viewing angles, among others. The training procedure is designed to tune the neural network so that it outputs right answers when provided with views that resemble that from its training dataset. As the training dataset does not include any view from the online database, the neural network cannot learn about their statistics. The only way the training on offline data can generalize to online data is by using features that are invariant to the differences between the two databases. If this is the case, the actual numbers provided to the neural network will be the same for offline and online views and the trained neural network will perform equally for both. This is not exactly the case here because the variations in the online database and too severe to be entirely filtered by the features. Consequently, the offline performance is not a dependable indication of the algorithm's performance in the online experiment. Indeed, the best performing setup for the offline experiment is the combination of all four features. This is not the case for both online experiments, as the use of the surflet-pair-relation histogram alone yields a recall about 10% higher than the use of all features.

The confusion matrix shows that most objects were correctly identified. However, among the six pairs of confused objects, only three are reasonable. That is, three of the confusions involve objects that resemble each other to some extent. For example, an instance of a gray box was confused with a gray and black computer. Both objects are rather colorless (gray and black) and have a rectangular prismoid shape. On the other hand, three confusions concern objects that seemingly have no common characteristics. For instance, the red caddie and the black office chair have very different shapes and colors. These awkward confusions are the result of the incapacity of the neural network to extrapolate to unknown situations. It is probable that these objects were seen under a unknown viewing angle, or were partly occluded. Such a situation can yield feature values that are sensibly different from data from the offline database. This unnatural data, when input in the neural network, produces unprecedented activations in the hidden layer which can result in an output that is absolutely not in phase with the input. This behavior from the neural network, that the outputs are a function of the whole vector of input values, is the reason why it sometimes produces unreasonable confusions. If one part of the input values does not resemble the training data, for instance if a single feature value is different from what it usually is, it can totally disrupt the output of the neural network.

The precision-recall curves also give information about the method. The precision-recall curves are obtained by using a decision threshold to accept or reject the classifier answers based on a confidence measure. The confidence measure is the activation of the highest activation output neuron (after applying the softmax function). If the confidence measure is higher than the threshold, the answer is accepted. Otherwise, it is rejected. One point on the precision-recall curve corresponds to the precision and recall measure when filtering the answers using a given value of the decision threshold. The whole curve is obtained by varying the threshold from 0 to 1. The smooth, non increasing profiles of the offline curves (figure 5-3) indicate that the confidence measure used behaves well. It is an indication that good answers generally come with a high confidence measure (a high activation value for the highest activation neuron). On offline data, the highest activation neuron's activation value thus is a good indication of the network's confidence. However, the curves for online (figures 5-4 and 5-5) data have a completely different profile. The sawtooth curves indicate that both good and bad answers sometimes come with a high confidence value. They suggests that activation is a rather bad confidence measure for the online experiments. Since the behavior is the same for all feature combinations, it can be assumed that the problem comes from the neural network. The neural network itself is not the cause, as its behavior on the offline data is acceptable. Rather, it is the fact that the neural network was trained on offline data and tested on online data that made this flaw appear.

In the online experiment, the multi-view fusion consistently increases recognition rates. It seems to be a very important aspect to incorporate to any vision system applied to robotics. The choice to perform late fusion seems reasonable in the light of the obtained recognition rates. Maybe an early fusion algorithm like scene reconstruction would increase the performances even more, but it might also provoke failures of the classifier in situations where not enough data is available for proper reconstruction. The late fusion brings benefits for a fraction of the computational cost of a scene reconstruction. However, there is an underlying hypothesis to the proposed multi-view fusion method. The averaging produces sensible results only if all the views from a given object are independent sources of information. If each view shows the object under a different angle, for instance, it is reasonable to average the recognition scores. It is also a valid approach if not all views are independent but there is a similar number of views taken from each of the independent angles of view. In a robotics experiment, it is very likely that the number of view be unbalanced. For instance, it is possible that a large proportion of the total number of views were taken as the robot stood still in front of the object. Averaging the scores from all views in this case would largely favor the views taken from this position. The next chapter introduces a simple modification of the multi-view fusion that reduces the effect.

5.8 Conclusion

Experiments on the ENSTA online database are the proof-of-concept of this object recognition pipeline. The 80% recognition rate on the online database is high enough

to confirm that the approach is valid. This prototype also is used as the main robotics demonstration for the Robotics and Computer Vision team at ENSTA ParisTech for more than two year. This long term use proves the viability of the method. The experiments show that the surflet-pair-relation histogram feature is excellent for the application. Also, the transformed RGB color descriptor works well for the online setting when used in combination with the shape feature. It seems that the histogram of hue descriptor is too dependent on the lighting conditions and does not transfer well from offline training to online testing. Also, the use of the histogram of occurrence of SIFT words does not notably improve the performance of the algorithm. Note that if the histogram of hue is discarded, then the whole pipeline is blind to color information. Indeed, the end result of the transformed RGB feature does not contain any information about the initial color of the pixels. The next chapter will introduce a new color feature to use in replacement of the histogram of hue.

The feed-forward neural network has many benefits. Its online computation time is very low and the fact that it fuses any type of feature without extra effort is appreciable. However, it is very slow to train the network. Producing the recognition rates for a given set of parameters, using the two-fold cross-validation procedure, takes around four days. This makes the neural network an inconvenient tool to use for a designer. Moreover, changing the dataset also requires training the neural network over again. This means that the neural network also is troublesome for the end user who would like to have the algorithm learn new objects. Finally, the purely statistical nature of the neural network causes its behavior on online data to be impossible to predict based on offline experiments. In the end, the black-box nature of the neural network was useful to have an idea of which features work well and which do not, but it is too troublesome to use for the designer and the end user. The next chapter describes the work done to replace the neural network by a nearest neighbor voting mechanism. The nearest neighbor mechanism will integrate physical information about the objects to recognize that will help alleviate the problem of transferring from offline to online experiments.

Chapter 6

Data fusion by shared nearest neighbors for object instance recognition

6.1 Introduction

In this chapter, a nearest neighbor search strategy for object classification on a mobile robot is developed and tested. The nearest neighbor strategy is meant to replace the neural network classifier presented in the last chapter. The nearest neighbor classifier will use the features introduced in the last chapter, at the exception of the hue color feature. A color feature based on opponent color theory is presented in this chapter to substitute the hue based color feature.

These changes are meant to provide two properties to the classifier. First, the nearest neighbor strategy should allow the independent parametrization of the different features. This is not the case for the neural network; if a parameter from any of the features had to be changed, the neural network needed to be trained over again. Also, it should be much easier to change the input database of the classifier with the nearest neighbor strategy than it was with the neural network. Second, the opponent color feature should be able to better transfer from offline training to online testing. The property of an algorithm to transfer from a dataset to another is verified by choosing the best parameters for the first dataset and testing whether or not these parameters also work well for the second dataset. If an algorithm transfers well from offline data to online data, it means that it can be optimized on data available at design time and be expected to work well during standard operation, facing slightly different data. The hue color feature did not transfer very well to online data (see chapter 5, section 5.6). It showed very good performance on offline data but failed to give good recognition rates on online data.

The next section justifies the changes introduced in this chapter. Section 6.3 gives details about the implementation of these changes. Then, a series of offline and online experiments are run to highlight the properties of the method in section 6.5.

6.2 Design choices

6.2.1 Color feature

The histogram of hue values based on the HSV color representation proved not to transfer very well to online recognition. The problem may be caused by the fact that the hue does not encode the absence of color. Black, gray and white pixels have an undefined hue value (see equation 2.2). This is the reason why the hue is weighted by the saturation value when the hue color histogram is compiled. The weighting completely removes the contribution of pixels with undefined hue. The result is a histogram dominated by information from colored pixels even if there are very few compared to the number of colorless pixels. The opponent color representation encodes colors at the same time as absence of color in the same space. In an opponent color histogram, the colorless pixels will contribute just like colored ones. As the database contains many objects that are mainly colorless, the opponent color space seems more appropriate.

6.2.2 Classifier

The motivation for removing the neural network is twofold. First, neural networks do not offer much control. There is no theoretical guarantee that the training of neural network will not overfit to the training data. Early-stopping helps reducing the chances of overfit, but the procedure still involves manually chosen parameters that are hard to back up with theoretical arguments. Even when all parameters are chosen, the result of the training depends on the initial values of the weights, which are randomly chosen. Secondly, the neural network is a bulky tool to use both for the designer and the end user. Training takes more than a day and is not incremental, meaning it a new training step must be undertaken as soon as a change happens in the pipeline. These changes include the features used for recognition and their parameters, but also the content of the training dataset. This means the set of objects the robot can recognize is fixed until a brand new training procedure is started. This is tedious for the designer and limits the number of tests that can be done in a given given period of time. It also does not fit very well in the usage scenario where changes to what the robot should be able to recognize are meant to happen on a regular basis.

The chosen strategy to replace the neural network classification is a nearest neighbor voting mechanism. A simple nearest neighbor classifier computes the distance between a test input and the examples from a training database and returns the most similar ones. Classification is then performed by assigning to the input the label of one of the returned examples from the database. An important aspect of the nearest neighbor strategy in this work is its usage with multiple features. To replace the neural network, the nearest neighbor search must fuse the features used to describe objects. Each of the features computed on a point cloud lie in their own separate space, and fusing the information they provide can be done in several ways. In this work, a separate nearest neighbor search will be performed in each feature

subspace. The result of the searches will then be fused. This allows for each search to be parametrized specifically for each subspace. Each nearest neighbor search returns a list of possible labels. The list of possible labels will be different for each search because the different features do not encode the same information. The method for fusing these answers is based on the following observation. Most objects from the database cannot be distinguished based on a single feature since many objects share a common shape, color, texture, etc. The objects which are similar according to certain features can only be distinguished based on the answer provided by other features. Knowing that an object is red, or that it is square-shaped is not enough to classify it. It is the combination of square and red that can lead to the classification of a red box. Single feature searches will thus most of the time return several answers. If the search strategy is successful, it will return the set of examples that share one feature with the test input. One of the returned answer should be the right one. As several features are used, the right label should be part of each set of answers from the individual searches. One way to combine the information provided by several features is to take the intersection of the searches' answers. Another solution is to only consider answers returned by a majority of the single feature searches. Each example returned by the searches that passes the majority test is a possible label for the test input. According to this strategy, the most important quality of a single feature nearest neighbor search is that the right label be always part of the returned ones. Hence, the single feature searches will be parametrized such that their recall is high. The precision of the overall classifier will be obtained by the multi-modal fusion.

6.2.3 Distance metric

The feature space distance metric still has to be chosen, and several ones will be tested by comparing the recall on single features. Only bin-to-bin distances will be tested because cross-bin distances are much slower to compute than bin-to-bin distances [43]. Another motivation not to use cross-bin distances is the fact that the distance metric in this work are only involved in nearest neighbor searches. Nearest neighbors tend to be very similar to each other. The histograms of two neighbors should differ only by small variations. It is expected that the complicated process of computing cross-bin distances is not required for nearest neighbor searches. Still, to compensate the drawback of bin-to-bin to distances, the spreading of values during histogram compilation will be used (see chapter 2, 2.4.2). The spreading has a diffusion effect which should incorporate a small cross-bin effect to the bin-to-bin distances.

6.2.4 Model selection

In the database, many examples are redundant. It is hurtful to have redundant examples in a nearest neighbor search procedure because these examples do not add extra information but slow down the search process. It is not straightforward to design a method that automatically finds redundant examples. This is because redundant examples are not exactly identical, they only are very similar. Distance metrics, like

the ones described in chapter 2, section 2.4.3, measure how similar two examples are, but the problem remains of defining how similar two examples should be before being considered redundant. The similarity threshold to determine redundancy changes depending on the sensor, the form of the feature space used to describe the objects and the distance metric used to measure similarity. It is also very likely that the similarity threshold depends on the exact point in feature space where the similarity is measured. The number of unknowns is high, and the behavior of distance metrics in high-dimensional space is very counter-intuitive. It is impossible for a human to tell if a similarity threshold of 0.5 is more reasonable than a threshold of 0.05, for instance.

Detection of redundancy is a similar problem to that of creating a vocabulary from an input list of features (see chapter 2). The typical solution to this is to run an unsupervised clustering algorithm [58]. This algorithm however acts blindly and can give different results depending on the initialization. It is not even guaranteed to remove redundant examples from the list.

In this work, a method to detect redundancy is proposed that leverages on the availability of physical measures to guide the selection of a similarity threshold. The method takes advantage of the fact that a subset of the redundant examples exist because some examples show the same object from the same angle of view and distance. These are called physically redundant examples as opposed to feature space redundant examples. All redundant examples are feature space redundant and a subset of them are also physically redundant. It is much easier to design an algorithm to automatically find physically redundant examples because the redundancy has an easy to interpret cause. It is easy to understand that the same object seen at a distance of 1 meter might look different than when it is seen from a distance of 2 meters. Conversely, if the difference in the physical distance to the object between two views is 10 centimetres, it should be reasonable to consider them to be redundant. Physically redundant examples have the same properties than feature space redundant ones. They are affected by the same sensor noise, for instance. The right similarity threshold to automatically detect feature space redundancy can thus be estimated from physically redundant examples.

Once the set of feature space redundant examples are found, a selection procedure must choose which ones to keep and which ones to eliminate from the database. The redundancy detection and model selection procedure is explained in section 6.3.2.

6.2.5 Nearest neighbor search

A simple nearest neighbor search returns a single example from the database. The returned example is the model from the database that is most similar to the input. The amount of information provided by such a search is limited. It does not tell whether or not other models were almost as similar to the input than the returned one. To circumvent this, it is useful to return a small number of neighbors rather than a single one [59]. However, it is not known a priori how many models should be returned by the search to provide optimal results. In fact, it is very likely that the optimal number of neighbors to return depends on the exact value of the test

input. If the input lies in a dense region of the database, it might be better to return a larger number of neighbors. The similarity threshold found during the model selection procedure can be used again here to define an appropriate radius for the nearest neighbor search. The resulting nearest neighbor search strategy is described in section 6.3.3.

6.2.6 Multi-view fusion

The multi-view fusion proposed in the last chapter suffers from a major drawback: it considers each view of a given object as being independent. In a robotics experiment, this hypothesis is very likely to be wrong. This will happen for instance if the robot stops moving and several snapshots of an object are taken from the same location. These snapshot are not independent from each other because they come from the exact same source of information. For two snapshots to be independent, they must show a different part of the object. In this chapter, the multi-view fusion will group the views of a given object based on what face of an object is shown. All snapshots showing the same face of an object will be averaged first, and the averages from each face will be fused to produce the final answer.

6.3 Implementation Details

This section gives all implementation details for the new color feature, the model selection method, the nearest neighbor search and the new multi-view fusion. Figure 6-1 shows a block diagram of the recognition process using the common nearest neighbor classifier.

The surflet-pair relation histogram shape feature and the transformed RGB color feature described in chapter 5, sections 5.3.4 and 5.3.3 are reused in this chapter. The nearest neighbor strategy allows for more parameters to be tested and different sizes of histograms will be used in the experiments of section 6.4. A surflet-pair relation is composed of 3 angular measures (α, ϕ and θ) and one distance measure (see chapter 3, section 3.3.7). The histogram sizes for the surflet-pair relation histogram tested in this chapter take the form A×D, where A designates the number of bins used for the three angular measure and D is the number of bins used for the distance measure. The size for the transformed RGB histogram is specified by a single number which indicates the number of bins used per channel. The actual size of the histogram is three times this number.

6.3.1 Opponent color descriptor

In opponent color theory, colors are encoded by using two scales: one from red to green and the other from yellow to blue. These two scales are invariant to light intensity offsets. The opponent color descriptor is computed by calculating equations 2.5 and 2.6 for all points of the input point cloud. The result is compiled in a 2-dimensional N by N histogram. For simplicity reasons, the number of bins is the same for both

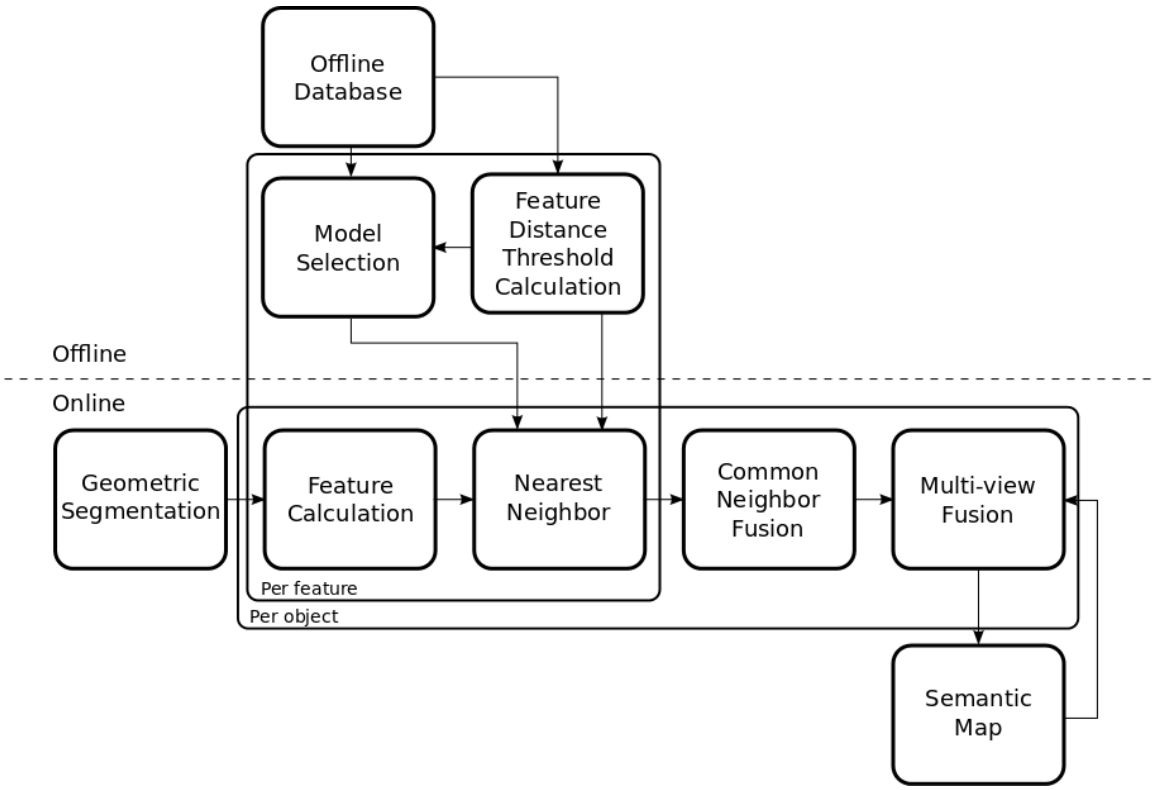

Figure 6-1: Block diagram of the common nearest neighbor based recognition. The offline section is executed once for a given database and set of parameters. The online section is executed in loop during operation. The frames show which operations are duplicated for the number of features used and the number of objects returned by the segmentation step.

dimension of the histogram. Histograms of size of 8×8 and 16×16 were tested. The values are clipped to -256 and 255.

6.3.2 Model selection with physical distance guidance

The model selection is a procedure that eliminates from the database the examples that are redundant. It is performed independently for each feature. First, a physical neighborhood is determined for each example of the database by applying a user defined physical distance threshold. The physical neighborhood for each example is used to compute its feature space similarity threshold. Using the similarity threshold, a feature space neighborhood is then determined for each example. Figure 6-2 illustrates the process of determining feature space neighborhoods from physical space ones. The process is detailed in this section and in algorithm 3. Using the feature space neighborhoods, a greedy model selection method discards useless examples. The model selection method is also described below and a pseudocode is given in algorithm 4.

Estimation of feature space similarity thresholds

Physical neighborhoods are computed by considering all examples from each object and each viewing angle. Two examples are physical neighbors if they show the same object from the same viewing angle and a similar physical distance. The physical distance *dist* is the Euclidean distance between the object's point cloud centroid and the camera. If two examples' physical distance differs by less than a given physical distance threshold t_{phy}, then they are physical neighbors. In this work, the distance threshold to determine physical neighborhood is 10 centimeters. Equation 6.1 shows how to determine the physical neighborhood pn for object o, viewing angle a and example e.

$$pn_{o,a,e} \leftarrow \{n : |dist_e - dist_n| \leq t_{phy}, \forall n \in db_{o,a,\backslash e}\} \tag{6.1}$$

where $db_{o,a,\backslash e}$ is the list of examples for object o and viewing angle a, excluding example e. From these physical neighborhoods, a feature similarity threshold can be found. The feature similarity threshold for redundancy detection is the maximum feature distance measure between an example and any of its physical neighbors.

$$t_{o,a,e} = \max_{n \in pn_{o,a,e}} feature\_distance(\{o,a,e\},\{o,a,n\}) \tag{6.2}$$

where $feature\_distance$ can be any measure of distance between histograms.

Figure 6-2 and algorithm 3 detail the process of estimating feature space similarity thresholds from a physical distance threshold.

Greedy model selection

Once the similarity thresholds are computed, they are used to remove redundant examples from the database. First, feature space neighborhoods are found for each

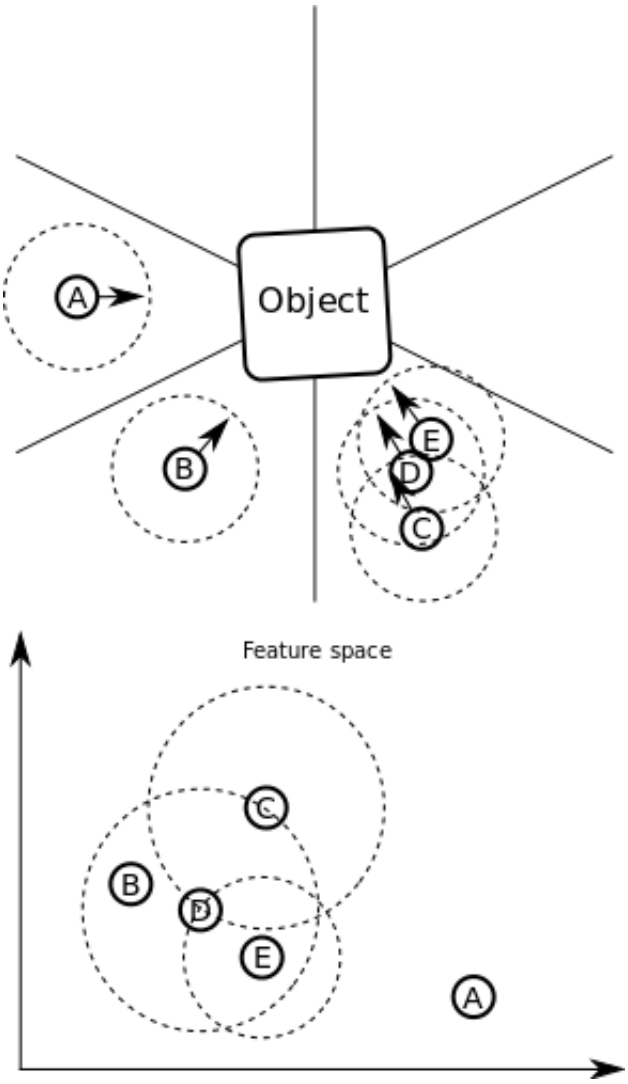

Figure 6-2: The process of computing feature space similarity thresholds and neighborhoods from physical redundancy. The top part of the figure shows the physical layout of 5 views of the object. The circled letters indicate the location where the robot was with regard to the object when a view was captured. The views are taken from 3 different viewing angles. View B is taken from a different angle than views C, D and E but is very similar to them because of symmetries in the object. The dotted circles show the neighborhoods of each view. A and B have no physical neighbors. The other views have the following neighborhoods: C:{D}, D:{C,E}, E:{D}. The bottom part of the figure shows the views in feature space. The similarity thresholds for each view is determined as the largest distance from the view to any of its physical neighbors. The resulting feature space neighborhoods are: C:{D}, D:{B,C,E}, E:{D}.

Algorithm 3 Estimating feature space similarity thresholds

Input: db, offline database of objects
t_{phy}, distance threshold to determine physical neighborhoods
Output: t, feature space similarity threshold for each example from the database

for all object o in db **do**
  for all viewing angle a in db_o **do**
    for all example e in $db_{o,a}$ **do**
      Compute point cloud centroid ctd of example $db_{o,a,e}$
      Compute example physical distance $dist$:
      $\cdot\, dist_e \leftarrow \sqrt{ctd_x^2 + ctd_y^2 + ctd_z^2}$
      Find physical neighborhood pn for example o, a, e:
      $\cdot\, pn_{o,a,e} \leftarrow \{n : |dist_e - dist_n| \leq t_{phy}, \forall n \in db_{o,a,\setminus e}\}$
      Find feature space similarity threshold for example o, a, e:
      $\cdot\, t_{o,a,e} = \max_{n \in pn_{o,a,e}} feature\_distance(\{o, a, e\}, \{o, a, n\})$
    end for
  end for
end for

example. Unlike the search for physical neighbors, the search for feature neighbors is done on all the examples of a given object, no matter the viewing angle. This allows feature neighbors to include examples from a different viewing angle that are similar because of symmetries in the object.

$$fn_{o,a,e} = \{n : feature\_distance(\{o, a, e\}, \{o, n\}) \leq t_{o,a,e}, \forall n \in db_{o,\setminus e}\} \tag{6.3}$$

where $db_{o,\setminus e}$ is the list of examples for all viewing angles of object o, excluding example e. The model selection is a greedy iterative process that generates two lists: the selected list and the redundant list. The procedure iteratively selects examples to add to the selected list. Every time an example is selected, all the examples that are part of its feature space neighborhood are added to the redundant list. Starting with empty lists, the procedure selects the example which has the highest number of examples in its feature space neighborhood. These neighbors are added to the redundant list. For the next iteration, a new example must be selected. The examples considered for selection are all the ones that are not already in either the selected list or the redundant list. From these examples, the one which has the highest number of feature space neighbors that are also not part of the selected or redundant list is selected. In other words, the procedure selects the example which will cause the largest number of examples to be added to the redundant list. If several examples have the same number of feature space neighbors, a score is computed to select one of them. The score is the variance of the feature space neighbors of each example. The example with lowest variance is selected. The model selection procedure stops when selecting a new example would cause a single example to be added to the redundant

list. The procedure is summarized in algorithm 4.

Algorithm 4 Greedy model selection

Input: db, offline database of objects
t, feature space similarity threshold for each example from the database
Output: *selected*, list of selected models

for all object o in db **do**
  for all viewing angle a in db_o **do**
    for all example e in $db_{o,a}$ **do**
      Find feature space neighborhood for example o, e:
      $\cdot\ fn_{o,e} = \{n : feature\_distance(\{o,e\},\{o,n\}) \leq t_{o,a,e}, \forall n \in db_{o,\setminus e}\}$
    end for
  end for
  $selected$: {}
  $redundant$: {}
  $done = false$
  while $done == false$ **do**
    Find example with largest non-redundant, non-selected neighborhood:
    $\cdot\ sel = arg\max_{e \in db_o} \Big(count(fn_{o,e}) - count(fn_{o,e} \cap redundant) - count(fn_{o,e} \cap selected) \Big)$
    if there are several examples in sel **then**
      for all example e in sel **do**
        Compute example's neighborhood feature space variance:
        $\cdot\ \mu_e = \sum_{n \in fn_{o,e}} \frac{feature(n)}{count(fn_{o,e})}$
        $\cdot\ var_e = \sum_{n \in fn_{o,e}} \frac{feature(n) - \mu_e}{count(fn_{o,e}) - 1}$
      end for
      Pick example with smallest variance:
      $\cdot\ sel = arg\min_{e \in sel} var_e$
    end if
    if $count(fn_{o,sel}) \leq 1$ **then**
      $done = true$
    else
      Select sel as a model:
      $\cdot\ selected\ +\!= sel$
      $\cdot\ redundant\ +\!= fn_{o,sel}$
    end if
  end while
end for

6.3.3 Bounded multi-modal nearest neighbor search

Nearest neighbor searches are done in individual feature spaces where only selected models exist. These models are fixed and known. Therefore, the search can be accelerated by analyzing the space spanned by the models. In this work, a kd-tree is built for each feature space (see chapter 3, section 3.3.2). Once the kd-trees are built, a nearest neighbor search is performed to find all models which have the input as a feature space neighbor. This is implemented as a two step procedure. First, a search is issued with a search radius equal to the largest similarity threshold of all model's similarity thresholds. Then, the returned models are filtered by checking whether or not their similarity with the input is lower than their own similarity threshold. All models that pass this test are returned by the search procedure. This search process is illustrated in figure 6-3. The set of models returned by the search is a list of the models from the database that are feature space neighbors of the test input. Each of these models represents one of the 52 objects from the database and there can be several models of the same object in the search space. For recognition purposes, it is only useful to know which objects (or labels) are represented in the set of neighbor models. The set of models is thus transformed into a score vector. The score vector indicates, for each of the object of the database, whether or not one of its models is present in the set returned by the search. At this point, the score vector only contains 0's and 1's. The information about how many models of each object were in the set is discarded because it depends on the number of models selected by the model selection procedure. Some objects have more models than others, which should not influence the recognition scores. If no models were returned by the search procedure, then the score vector is all 0's. It is possible that the score vector contains more than one 1's. This is called a tie in the recognition results of section 6.4. The ties can be resolved by the fusion steps.

The multi-modal fusion step combines the score vectors from all the individual feature searches into a single one. The fusion simply removes the 1's which do not appear in a majority of individual score vectors. When three features are used, which is the case in this work, the fused score vector contains 1's only in places where two individual score vectors showed a 1. If only 0's remain, then the example cannot be classified. It is possible that the score vector contains several 1's. The ties present at this step can still be resolved by the multi-view fusion step. Figure 6-4 illustrates the conversion from the individual search results to multi-modal score vectors.

6.3.4 Two-step multi-view fusion by position

When a new object is stored in the semantic map, the area around it is divided into 12 sectors. Each time this object is seen, the position of the robot is used to compute in which sector the score vector should be stored. All the score vectors from a given sector are averaged to yield this sector's score vector. Just as in chapter 5, the average is weighted by the number of points in the view's point cloud. This first averaging operation acts like a filter to remove noise from the views from each sector. Each sector is then assumed to provide independent information about the object's

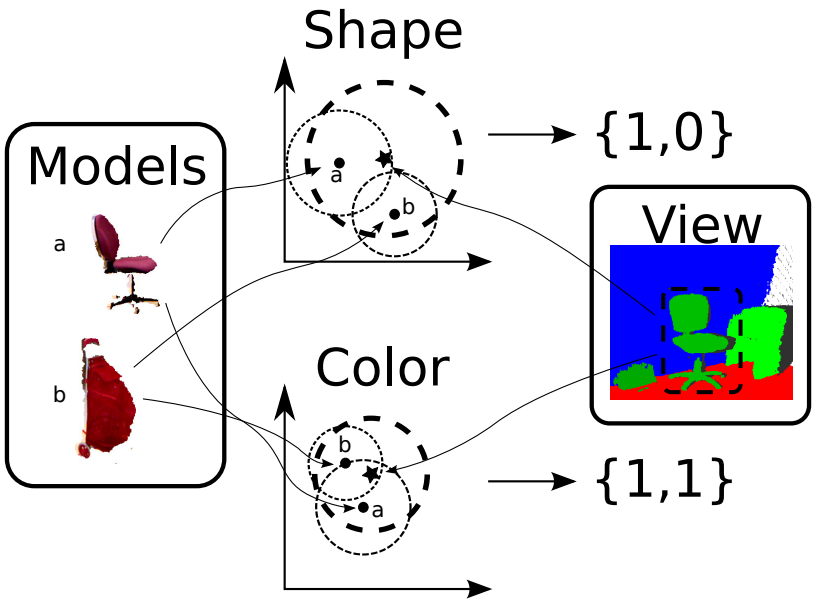

Figure 6-3: Illustration of the multi-modal nearest neighbor search on a simple problem. The box on the left represents the models selected from the database. A shape and a color feature are computed for each model. Each model becomes a point in the feature spaces. The view of the object to recognize also is a point in the feature spaces (represented by the star). The bolder circles are the radii used for the search and the thinner ones are each model's similarity threshold. The search first finds all potential neighbor models. Then, each returned model is kept only if the star is inside their own similarity limit. In color space, the search finds both model and the star in inside each model's similarity threshold radius. The search in color space returns both models a and b as neighbors, as indicated by the score vector: {1,1}. In shape space, the search also finds both models, but the star is outside the threshold for model b. The shape space search only returns model a ({0,1}).

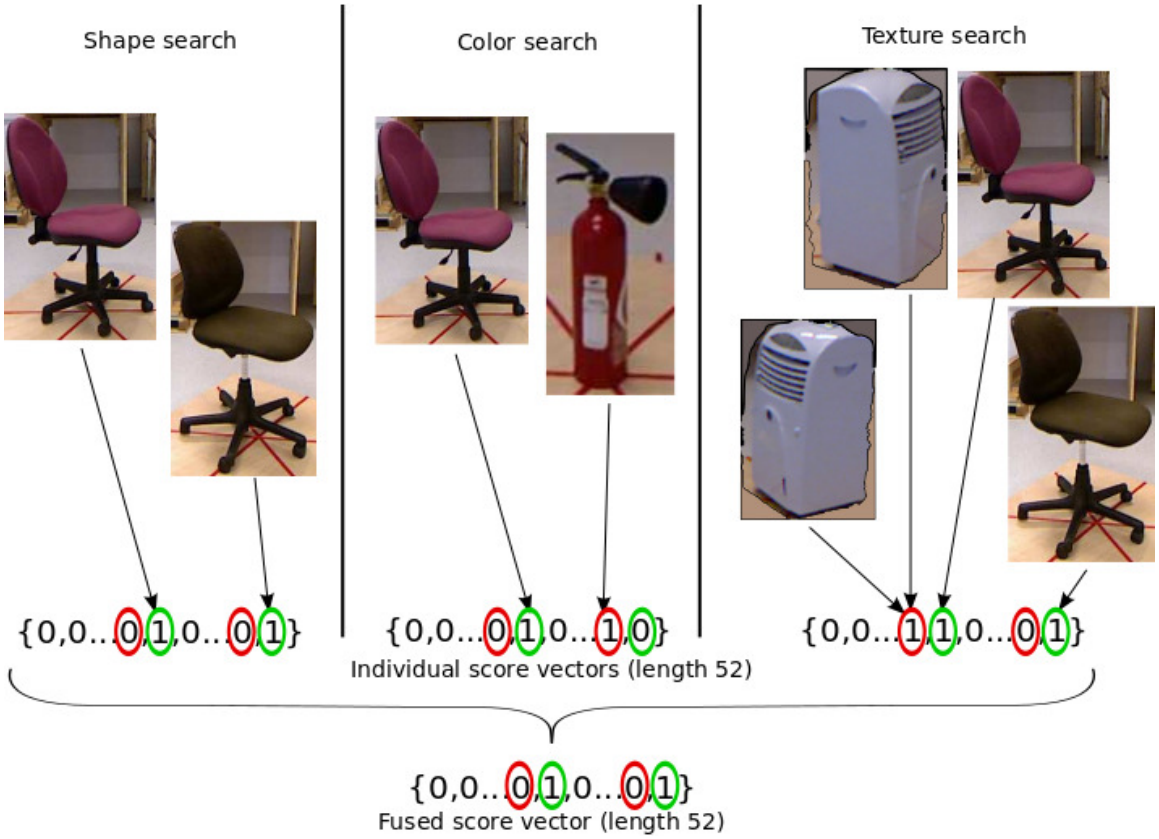

Figure 6-4: Conversion process from search results to fused multi-modal score vector. Each individual feature search generates a score vector that indicates which object is likely to be the one to identify. Even if several models of a given object are returned by a search, the score vector only indicates a 1 for this object. The individual feature score vectors are then combined by a majority operation. The green and red circles indicate which of the objects found in the individual searches are respectively accepted and rejected by the majority operator.

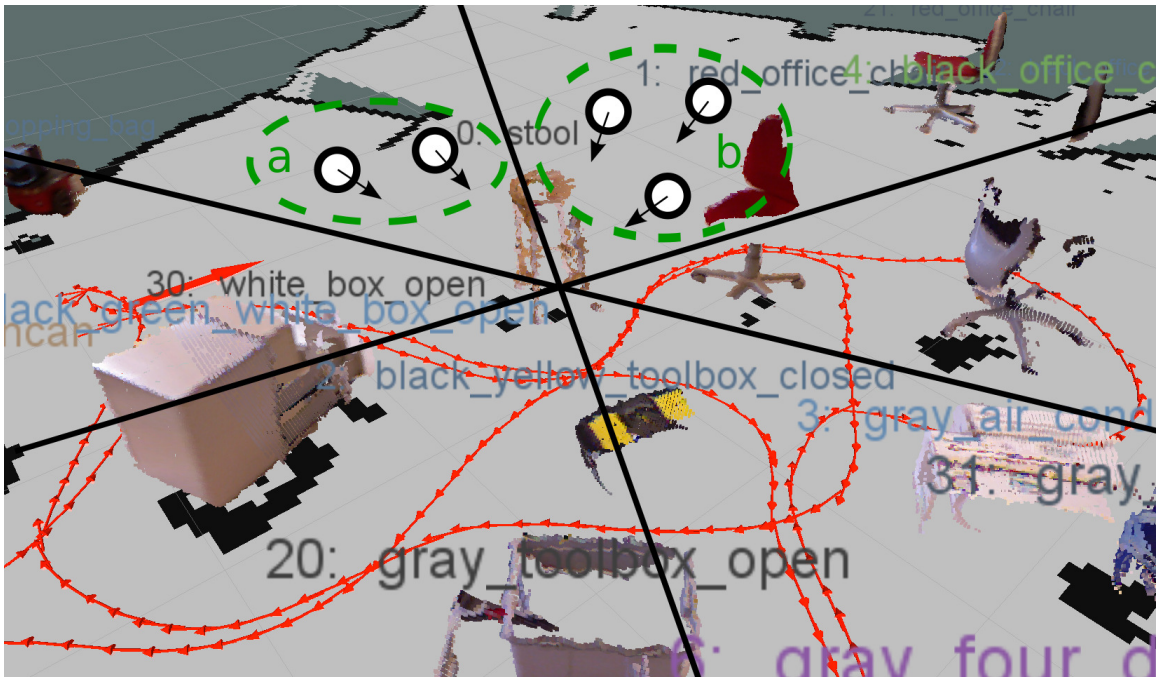

Figure 6-5: The division of space around an object for the two-step multi-view fusion process. In this image, the space is divided into 6 sectors. The white circles indicate the positions where the robot was when each view was taken. The score vector for all views in the same sector are averaged first (dark-green circles). Then, these sector score vectors are averaged to get the final classification score for the object.

identity. The final score vector for an object is the average score vector over all sectors that have at least one view. The division into sectors is illustrated in figure 6-5. The score vectors at this point contain real numbers between 0 and 1 which indicate how confident the classifier is that the test input is each of the objects from the database. The test input is given the label which has highest confidence value. If more than one label have the highest confidence, then there is a tie and it is not possible to determine the input's label until another view of the object is seen.

6.4 Experiments

The first run of experiments is conducted on the offline database. The goal of this first series is to determine the best size of histograms for the features. Each feature is individually studied. Spreading is done according to the linear interpolation detailed in chapter 2, section 2.4.2.

6.4.1 Model selection

The proposed model selection method ensures that at least one model is selected for each object from the database. It can select several models for the same object. The model selection is conducted independently for each used feature. Figure 6-6

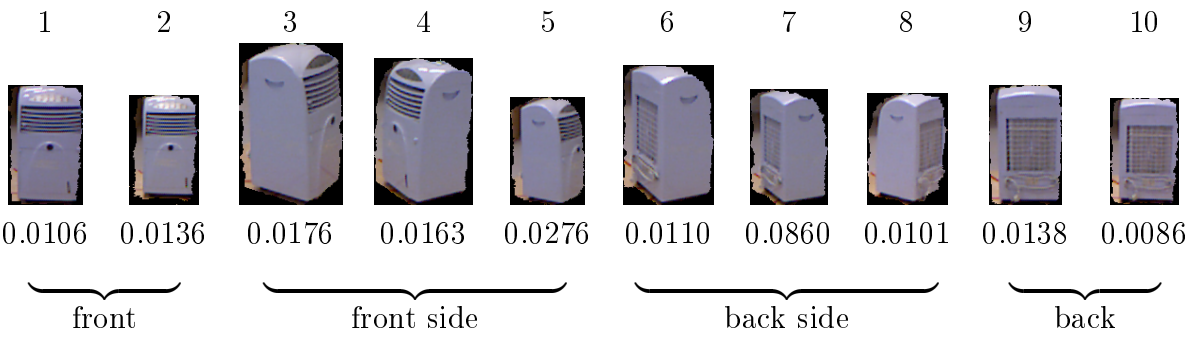

Figure 6-6: Selected models of the object gray air conditioner for 3×3 surflet-pair relation feature with physical neighborhood of 10 centimeters. The feature space similarity threshold for each model is displayed under the image. The method selected 10 examples out of 440.

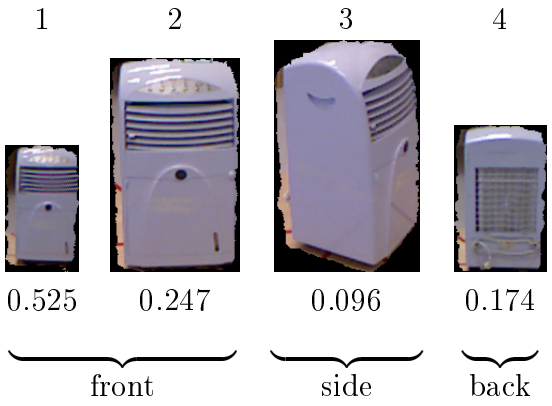

Figure 6-7: Selected models of the object gray air conditioner for 16×16 opponent color feature with physical neighborhood of 10 centimeters. The feature space similarity threshold for each model is displayed under the image. The method selected 4 examples out of 440.

shows the models selected by the 3×3 surflet-pair relation histogram for the gray air conditioner object using the squared-chord distance. Figure 6-7 shows the selected models for the 16×16 opponent color feature for the same object and distance metric. The figure also show the feature space similarity thresholds for the selected models. There is a total of 440 examples of the gray air conditioner in the offline database and the two model selection procedures presented here select only 10 and 4 of them. After the model selection procedure is run on the whole database of 21 411 examples, only 200 to 400 models remain, depending of the feature and distance metric used.

6.4.2 Uni-modal offline experiment

The model selection method is first run on the offline database for each feature and parameter. Using the selected models, the nearest neighbor classifier is tested on the entire offline database. The recognition rate for all parameter configurations are

Table 6.1: Recognition rate for offline experiments in the form "recall"/"average number of labels" ("number of ties" out of 21 411 examples).

Surflet-pair relation

distance	3×3	4×4	5×5
L_2	73.48 /4.68 (10441)	86.10 /8.16 (14777)	86.36 /6.76 (14585)
χ^2	72.31 /4.17 (9230)	72.44 /**3.31** (7178)	70.84 /**3.27** (**5842**)
Jeffreys	73.04 /4.16 (9241)	73.89 /**3.46** (8119)	74.29 /**3.34** (7698)
S-chord	85.85 /8.11 (15535)	**89.97** /8.31 (16142)	**89.87** /8.38 (16306)

Opponent color

distance	8×8	16×16
L_2	80.27 /6.29 (**14759**)	86.43 /6.88 (**14799**)
χ^2	88.39 /6.50 (**14620**)	89.86 /6.54 (15498)
Jeffreys	88.68 /6.45 (**14168**)	**91.65** /6.46 (15942)
S-chord	87.20 /6.69 (15306)	90.16 /6.15 (15719)

Transformed RGB

distance	8	16	32
L_2	73.10 /4.14 (10507)	63.98 /3.33 (5719)	48.83 /**2.56** (**2684**)
χ^2	78.22 /4.47 (11627)	72.56 /3.52 (7245)	64.04 /3.68 (4510)
Jeffreys	85.88 /7.14 (18194)	**90.39** /6.96 (18703)	**90.43** /6.93 (18490)
S-chord	84.75 /6.98 (15683)	**89.56** /7.26 (17133)	88.15 /7.59 (16552)

given in table 6.1. The table shows results for different histogram sizes. The results are given in the form "recall"/"average number of labels" ("number of ties"). The recall is computed according to equation 3.11 as $\frac{tp}{tp+fn}$. The true positives tp here are all score vectors that contain the right answer. The recall given here is the highest possible recall value obtainable after disambiguation by multi-modal fusion. The number of ties is the number of test inputs for which the score vector contains more than one 1 entry. The number of ties indicate the number of classifications that must be disambiguated. The average number of labels is the average number of 1's in the score vectors. A high number of labels means the number of objects the test input is confused with is high. All configurations are tested on a total of 21 411 examples from the offline database. The best performances for each feature are shown in bold.

6.4.3 Multi-modal offline experiment

A second series of offline experiment is run in multi-modal mode. This time, the combination of surflet-pair relation, opponent color and transformed RGB feature are tested. All combinations of the features' parameters are tested. All results are given in appendix A, section A.1. The recall rate, precision rate and number of ties for selected configurations are given in table 6.2. The selection shows the combinations

Table 6.2: Notable results of the multi-modal offline experiment (21 411 examples)

Distance	SPRH	OppCo	TRGB	Recall	Number of labels	Ties
L_2	3×3	8×8	16	78.63%	2.46	5664
L_2	3×3	8×8	32	74.06%	2.35	3972
L_2	4×4	16×16	8	90.98%	3.20	11247
χ^2	3×3	16×16	8	86.88%	2.85	8608
χ^2	3×3	16×16	16	84.59%	2.51	6301
χ^2	5×5	16×16	32	81.20%	2.35	**3471**
Jeffreys	4×4	8×8	8	89.57%	3.15	10185
Jeffreys	4×4	8×8	32	91.12%	3.1	11021
Jeffreys	5×5	16×16	32	**92.30%**	3.13	12497
S-chord	3×3	16×16	8	86.80%	2.87	8623
S-chord	5×5	8×8	32	82.49%	2.38	4318
S-chord	5×5	16×16	32	84.17%	**2.33**	4449

for each distance that give the best recall and ties for the offline experiment as well as the ones that give the best online results (see the next section).

6.4.4 Online experiment

The online experiment is run in multi-modal mode with all combinations of feature parameters. The single-view and multi-view conditions are compared. The results of all online experiments are given in appendix A sections A.2 and A.3. The appendix shows the recall rates and ties for all feature parameter combinations and all 4 distance metrics. A summary of the results for the single- and multi-view experiments is given in figure 6.3. The single-view results show the recall and number of ties. It also shows the per-object recall. The per-object recall indicates for each object if at least one of the views was correctly recognized. It represents the highest possible number of objects that each configuration may recognize after multi-view fusion. The multi-view results show the recall as well as the number of unambiguously recognized objects. The number of recognized objects indicates how many correctly labeled objects would appear on the semantic map. It does not include ambiguous recognitions as these would not have a label specified in the map.

The confusion matrix for the squared-chord distance with the 5×5 surflet-pair relation histogram, 16×16 opponent color and 32 transformed RGB features using multi-view fusion is shown in figure 6-8.

6.4.5 Run time and resources

The nearest neighbor search when 3 features are used and a total of about 1 000 models takes less than 100 milliseconds to execute. Storing the model's feature values uses up about 1 GB of RAM. The fusion operations take negligible time.

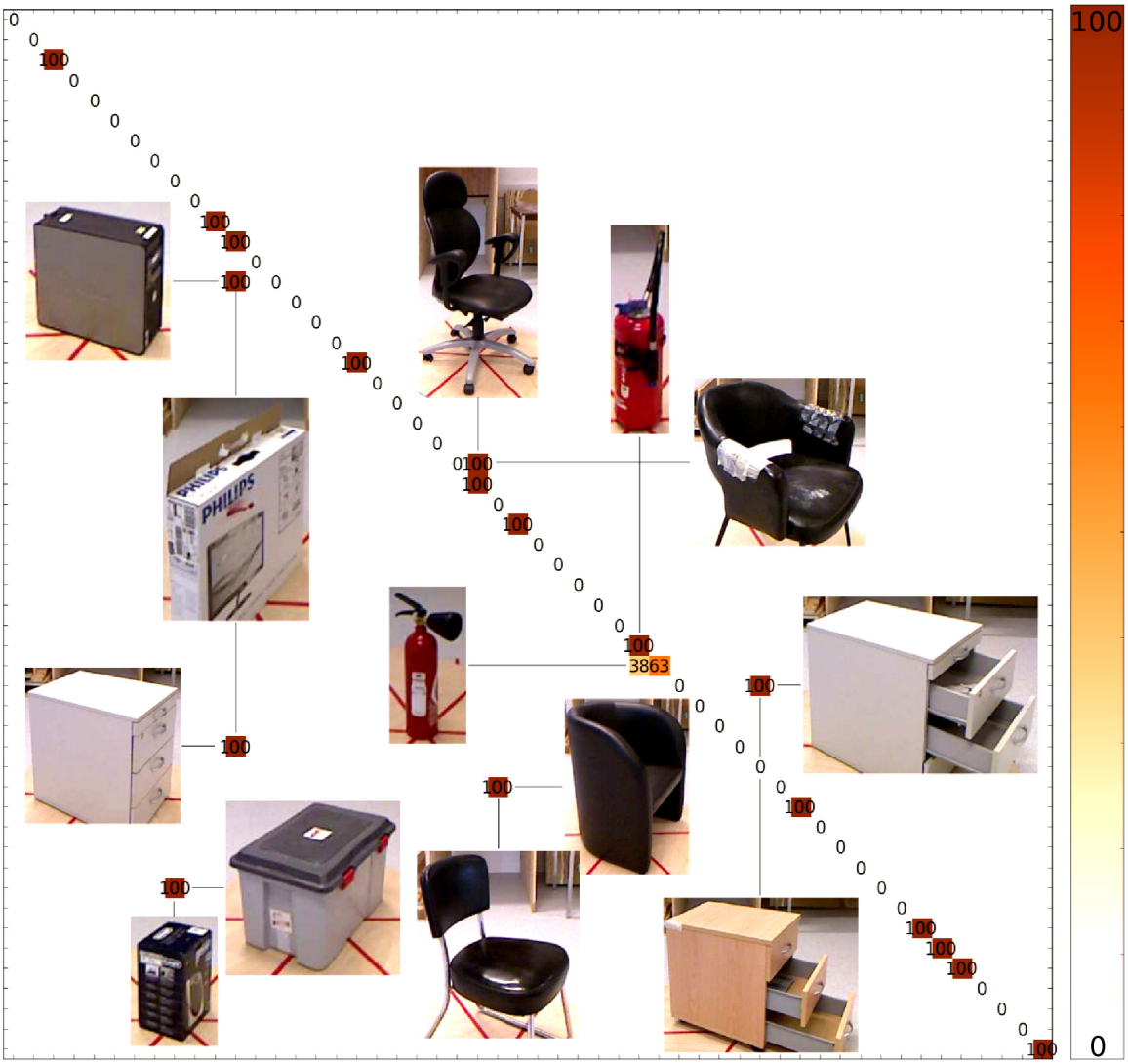

Figure 6-8: Confusion matrix for the multi-view online experiment. An image of all confused objects is overlaid to the confusion matrix. Images associated to a vertical line represent objects that were present in the scene while images linked to an horizontal line show the labels assigned by the classifier. The matrix mostly contains 0 and 100 entries because the multi-view fusion combines all the views from each object into a single score. This is not the case for one of the extinguishers because there were two instances of this object in the scene.

Table 6.3: Notable results of the online experiment (141 examples, 20 objects)

				Single-view			Multi-view	
Distance	SPRH	OppCo	TRGB	Recall	Ties	Object recall	Recall	Correct objects
L_2	3×3	8×8	16	17.02%	4	12	47.52%	9
L_2	3×3	8×8	32	13.48%	**3**	11	43.97%	9
L_2	4×4	16×16	8	29.79%	38	**17**	33.33%	5
χ^2	3×3	16×16	8	32.68%	18	16	51.07%	10
χ^2	3×3	16×16	16	29.79%	5	15	56.74%	12
χ^2	5×5	16×16	32	17.73%	**3**	12	48.23%	11
Jeffreys	4×4	8×8	8	28.37%	20	15	34.75%	7
Jeffreys	4×4	8×8	32	27.66%	21	15	**66.67%**	10
Jeffreys	5×5	16×16	32	**50.35%**	55	**17**	51.77%	10
S-chord	3×3	16×16	8	31.21%	10	**17**	38.30%	6
S-chord	5×5	8×8	32	21.28%	6	12	46.10%	11
S-chord	5×5	16×16	32	20.57%	8	14	58.87%	**13**

6.5 Discussion

This chapter proposed a replacement classifier for the neural network. The main goal was to find a strategy which is more flexible than the neural network and allows the training on offline data to transfer to online data. Replacing the neural network involved designing a model selection method and the multi-modal nearest neighbor search strategy. The analysis of the offline uni-modal performance gives hints about the behavior of the model selection and indicates which distance metric works best with each feature. Comparing the uni-modal performance to the multi-modal performance shows if the multi-modal fusion strategy works properly. Finally the online results indicate if the method transfers well to online data.

The uni-modal performance is the result of the model selection method and the uni-modal nearest neighbor search strategy. The examples of selected models shown in 6-6 and 6-7 indicate that the model selection algorithm seems to work properly. The figure shows the 10 models kept for the 3×3 surflet-pair relation feature and the 4 ones for the 16×16 opponent color feature for the gray air conditioner object using the squared-chord distance metric. There are more models for the shape descriptor because the shape of the object is slightly different when seen from a varying distance, whereas the color remains essentially the same. The shape changes with the distance because a larger portion of the top part of the object becomes visible as it approaches the camera. The difference can also come from the varying point density. As can be seen in figure 6-6, the models that show the object from the same angle of view are taken from different distances (this can be seen by the size of the image changing). This suggests that the method indeed eliminates redundant examples. Notice that most symmetrical views are eliminated, as both sides of the side views do not appear. Only the 6^{th} and 7^t views of the surflet-pair relation feature seem to be replicates.

The figures also show the feature space similarity thresholds computed by the model selection method. It can be seen that the thresholds can be very different for the same object depending on the feature used and the exact view. The strength of the method lies in the fact that these feature space thresholds are computed from a physical distance threshold. The physical distance threshold is unique for all objects, features and distance metric used, and it is easy to define by the designer.

The uni-modal offline experiments results are an indicator of which distance metric is the best. The metric which gives the best average recall is the squared-chord distance (above 89% for at least one parameter set of each feature). It gives the best or second best recall for all features. The Jeffreys distance gives very similar results, except for the surflet-pair relation feature. In this case, the squared-chord (89%) vastly outperforms the Jeffreys distance (74%). The squared-chord distance, along with the Jeffreys distance, however consistently have a very high number of ties. The number of ties for the squared-chord distance is above 15 000 (out of 21 411 examples) for all features and parameters. A high number of ties is expected for uni-modal experiments. Single features cannot single-handedly disambiguate a vast majority of the objects from the database. Being able to disambiguate all objects based only on single features would be a sign of over-fitting rather than a good behavior. For all cases in the uni-modal experiment, a decrease in the number of ties comes with a decrease of the recall rate. This is a natural trade-off between recall and precision.

The comparison of uni-modal and multi-modal offline results gives indications on the efficacy of the fusion method. The recall rates globally are lower in the multi-modal case. This is not surprising since the fusion is an "and"-like operation which can only filter out answers. In general, the decrease is rather low. The best parameter sets for each distance metric give recall rates that are consistently over 85% (it is around 90% for uni-modal). The number of ties, though, is most of the time vastly reduced. The squared-chord distance, which had 15 000 and more ties now sometimes has as few as 4 500. The average number of labels involved in the ties also drastically decreased from more than 6 in the uni-modal experiment to less than 3 for the multi-modal. The best configuration in the multi-modal offline experiment is the squared-chord distance with the 5×5 or 4×4 for surflet-pair relation histogram, 16×16 for the opponent color and 16 for the transformed RGB features.

The overall best performance for the multi-view online experiment is obtained with the squared-chord distance with the 5×5 for surflet-pair relation histogram, 16×16 for the opponent color and 32 for the transformed RGB features. Using these settings, 13 objects out of 21 are unambiguously recognized. This specific configuration also gives good results on the offline database. However, it is not realistic to say that offline results can lead to choosing this combination without hesitation. Many other configurations give similar results on the offline database and do not perform quite as well in the online experiment. For example, the 5×5 for surflet-pair relation histogram, 16×16 for the opponent color and 8 for the transformed RGB features configuration gives one of the best recall rates in the multi-modal offline experiments, but can only disambiguate 6 objects (13 object recall with 7 object ties) in the online experiment.

Results from the single-view and multi-view online show the usefulness of the

multi-view fusion by position. The fusion results in a higher recall rate, but the increase is unpredictable. It is impossible to predict if a situation will significantly benefit from it or not. All cases do improve at least a little bit when using the multi-view fusion. The motivation to use a two-step fusion mechanism comes from a practical problem that was encountered during the demonstration of the neural network classifier from chapter 5. During demonstration experiments, it was frequent that the robot would remain still while an object was in view. Since the multi-view fusion from chapter 5 averages the score vectors from all views of the objects, the scores obtained from this specific viewpoint very quickly came to outweigh any other score vector. The two-step method used here makes sure this situation does not happen. It does not have much influence in the online experiment where the robot was constantly moving but does help greatly during demonstrations.

The confusion matrix of the multi-view online experiment shows that most confusions concern objects that indeed have a similar shape and color. Note that for the classifier black, gray and white are the same color. The most surprising confusion is the closed gray toolbox with the closed black box. The two objects look pretty different to a human, but in fact they are both cubic and mostly colorless, which explains that the nearest neighbor classifier confuses them. This shows the difficulty to transpose from a human perception to the robot's perception of its surroundings.

6.6 Conclusion

The chapter introduced a model selection method and an object instance classifier based on a nearest neighbor strategy. The model selection method uses the physical distance between the camera and the object to automatically compute a similarity threshold in feature space for each example from the database. The similarity threshold is then used to limit the radius of the searches for nearest neighbor both during the model selection and recognition steps. The use of the physical distance removes the need for the designer to manually choose a hard threshold or determine the number of neighbors that should be returned by the searches. Setting a threshold for complex distance measures in high dimensional space is counter intuitive and prone to errors.

The nearest neighbor classifier yields good recall rates on the offline database. The multi-modal fusion decreases this recall a little but also greatly reduces the number of ties. The squared-chord distance produces the best results on the offline database. The combinations of features and parameters that work well offline however do not always do so on the online database. It is not reasonable to choose the parameters based on the offline performance and consistently get good results on the online database. The best configuration for online recognition is able to correctly label 13 objects out of 21.

The next chapter will review the algorithms presented in this thesis and compare them to recent related work.

Chapter 7

Discussion and perspectives

7.1 Database

The database is a fundamental aspect of an object recognition experiment. The database determines the nature and the difficulty of the problem to be solved. The ENSTA offline dataset contains a total of 21 411 point clouds of 52 different objects. The dataset is diverse. It contains some objects that have the same shape, like boxes and cabinets, and some objects that have the same color. These objects can easily cause confusions for object recognition algorithms that rely on a single feature. The dataset contains a very large number of colorless objects. Colorless objects are black, gray or white. Having several objects sharing the same color (or absence of color) is interesting because they are confused by algorithms that are invariant to light intensity changes. This causes a difficult situation where invariance is helpful to counter the changing conditions from the online database but is also hurtful because of the large number of colorless objects. Finally, it is possible that having only six different viewing angles in the offline database makes generalization to unseen angles difficult. How difficult depends on the exact implementation of the object recognition algorithm, but having a more precise coverage of viewing angles would certainly be beneficial.

Research in object instance recognition for indoor applications increased in the recent years and several datasets of home objects now exist. Comparing the ENSTA database with other ones can help asses the difficulty of the recognition task that was tackled in this thesis. Two datasets are highlighted here: RGB-D Object Dataset [40] and the Berkeley Instance Recognition Dataset [77]. The database most similar to the one compiled for this thesis is the RGB-D Object Dataset. It contains point clouds from 300 common household objects like food packages, fruits and vegetables, scissors, pliers, etc. The data was collected using an RGB-D camera by placing the objects on a turntable. A complete revolution of each object was captured with the camera placed at three different heights. Figure 7-1 shows the turntable used to collect the data a several examples of objects from the database.

One of the notable aspects of the RGB-D Object Dataset is that is also contains an online part. The RGB-D Scenes Dataset contains 22 reconstructed indoor envi-

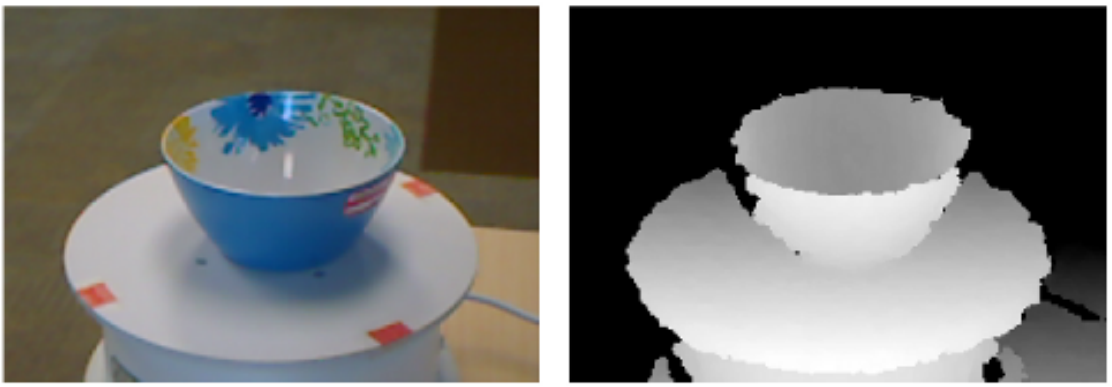

Figure 7-1: The setup to capture the point clouds for the RGB-D Object Dataset. The picture shows a bowl placed on the turntable. Image viewed 9 April 2015 http://rgbd-dataset.cs.washington.edu/imgs/rgbd.png.

Figure 7-2: Examples of objects from the RGB-D Object Dataset. Image viewed 9 April 2015 http://rgbd-dataset.cs.washington.edu/imgs/rgbd_dataset2.png.

Figure 7-3: Three reconstructed point clouds from the RGB-D Scene Dataset. Image viewed 9 April 2015 http://rgbd-dataset.cs.washington.edu/imgs/rgbd_scenes_v2.png.

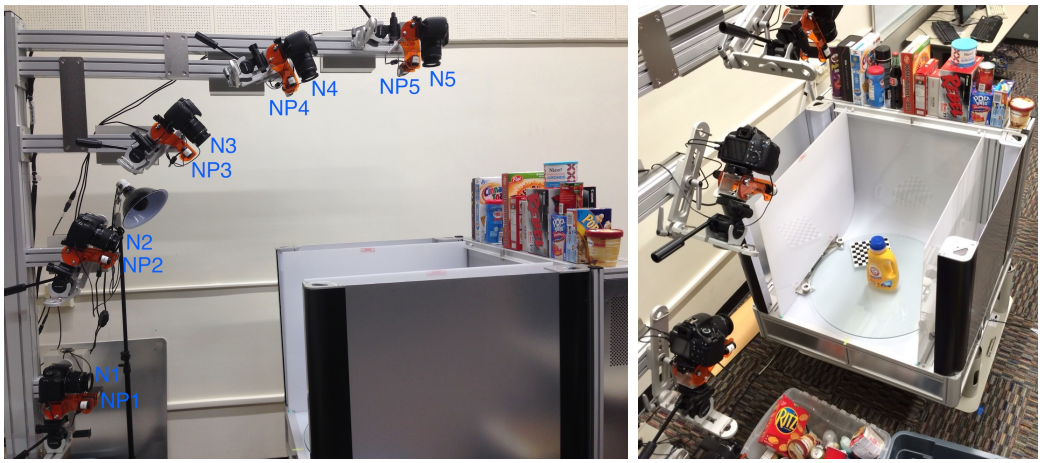

Figure 7-4: The setup to capture the point clouds for the Berkeley Instance Recognition Dataset. **Left** The array of cameras. **Right** One object from the dataset being filmed. Other objects of the dataset can be seen at the bottom and the top of the image. Images viewed 9 April 2015 http://rll.berkeley.edu/amazon_picking_challenge/images/ortery_side_labeled.jpg and http://rll.berkeley.edu/amazon_picking_challenge/images/system_and_objects.jpg.

ronments where some of the objects from the RGB-D Object Dataset are placed. The scenes were taken in various context like in a kitchen, a meeting room or an office. Figure 7-3 shows three scenes from the dataset.

The Berkeley Instance Recognition Dataset contains point clouds from 125 objects like food packages and detergent bottles. The data is collected using an array of RGB and distance cameras as the objects are placed on a turntable. Figure 7-4 shows the setup and some objects from the database.

These two datasets of objects used a turntable setup where the objects move in front of a stable camera. This is beneficial compared to the strategy used in this work where the camera moves in front of the objects. The still camera cannot suffer from motion blur and from a shift between the depth and color camera. The only phenomenon that is not captured by the fixed camera is the different point densities provoked by varying the distance of the object. One point to note is that the ENSTA dataset is not well suited to position estimation of the object. The fact that the

Dataset	Objects	Point clouds	Variations
ENSTA offline	52	21 000	Distance, point density, 1 angle
ENSTA online	20	141	Distance (large), 1 angle, illumination (large), occlusions (large)
RGB-D Object	300	250 000	2 angles
RGB-D Scenes	5	22 sequences	Distance (small), 1 angle, occlusions (small)
Berkeley	100	60 000	2 angles

Table 7.1: Comparison of the characteristics of the ENSTA, RGB-D and Berkeley Instance Recognition Datasets

camera is moving means that its position would have to be estimated for every point cloud for a ground truth to be computed. The RGB-D Object Dataset and Berkeley Instance Recognition Dataset are a much better choice to train algorithms for object pose estimation. Another difference is that the ENSTA dataset contains images of rather large objects compared to the two other datasets. The choice was made for the ENSTA dataset to use objects that would matter in navigation task and make sense to mark in a map. In essence, the RGB-D Object Dataset and the Berkeley Instance Recognition Dataset are comparable to the offline dataset introduced in this work. They contain more objects and the point clouds are of higher quality due to the still cameras, but the general idea of varying the angle of view and height (or distance) is the same.

The RGB-D Scene Dataset can be compared with the ENSTA online dataset. Both show the objects from their offline counterpart in a different environment. The way the objects are laid out is comparable. They are simply placed on a flat surface and do not touch each other. A major difference lies in the way the images are captured. In the RGB-D Scene Dataset, the camera is hovering over the scene at a more or less constant distance from the objects. The objects typically are well visible, although a certain level of occlusion sometimes happens. In the ENSTA online dataset, the camera is literally traversing the scene. This causes much more occlusions and partial views to happen. More importantly, it causes large changes in scale. This kind of variation is absent from the RGB-D Scene Dataset. The lighting conditions are also not controlled at all in the ENSTA online dataset. Some objects are viewed against the light, causing large illumination changes. The objects are exposed to direct sunlight from a close-by window, influencing their perceived color. This makes the online dataset very challenging for object recognition based on color. It is important that the online part of an object instance dataset contains this kind of challenges as this is the sort of situations a robot at home will be confronted to. The main problem of the ENSTA online dataset is the very small number of views it contains.

Table 7.1 compares the main characteristics of the three datasets.

7.2 Segmentation

The geometric segmentation algorithm was designed to segment objects from an indoor scene as fast as possible. The constrained and structured environment of a robot at home lends itself well to simple segmentation rules. The algorithm presented in this work is very similar to the popular desktop segmentation algorithms that appeared in recent years (see chapter 2, section 2.1.2). This thesis applied the algorithm to mobile robotics. The main additions are the wall detection step and the projection to the floor step. The projection to floor plane reduces the likelihood of splitting complex objects during the distance segmentation step. The resulting process is not perfect, but it very rapidly removes large parts of the input point cloud. One failure case is caused by the depth camera which cannot perceive some objects that are black, reflective or transparent. Addressing this issue would require using information from the color camera, and therefore completely changing the nature of the segmentation algorithm. The use of color might help reducing the number of failures, but it would also make the algorithm slower, as more computations would be required to process the color information. Two other problems of the geometric segmentation algorithm are wall detection and the generation of object candidates.

The detection of walls in the algorithm is problematic. During the experiments and demonstrations, it is the step of the segmentation that failed the most often. It is actually very difficult to design good rules to detect walls from single point clouds captured by a Kinect. The most distinctive feature of a wall is its very large size. A wall literally is the size of an entire room. This criteria alone should lead to reliable wall detection. However, walls seldom are completely visible in a point cloud. They are often occluded by large objects and the small field of view of a Kinect limits the size of a wall that can actually appear in a single point cloud. It becomes impossible to distinguish a wall from a large planar object based solely on geometric cues. One solution to this is to use a sensor that has a large field of view, like the laser range finder, to detect walls. A second solution is to run wall detection on a reconstruction of the environment, like a map. The metric map that serves as the basis for the semantic map in this work could be used for wall detection. Walls appear almost completely on maps and can be detected using line detection algorithm just like a plane detection algorithm was used in the point cloud. An example of walls detected from a map is shown in figure 7-5. The walls are extracted by running a line detection algorithm and assembling the parallel ones into larger segments. The bigger picture provided by a reconstructed environment helps reducing the number of failures compared to wall detection in single point clouds. Furthermore, the process of detecting walls on a map does not have to be repeated for every input point cloud like it is in this thesis.

The most important limitation of the object recognition algorithm presented in this work is that it fails when objects touch each other. The geometric segmentation algorithm cannot separate objects that touch each other and the classifier considers every candidate provided by the segmentation algorithm as a single object. If the segmentation algorithm could separate objects even if they are in contact, this limitation of the object recognition would disappear. It is not straightforward to separate two objects in an early stage of processing like the segmentation step. The

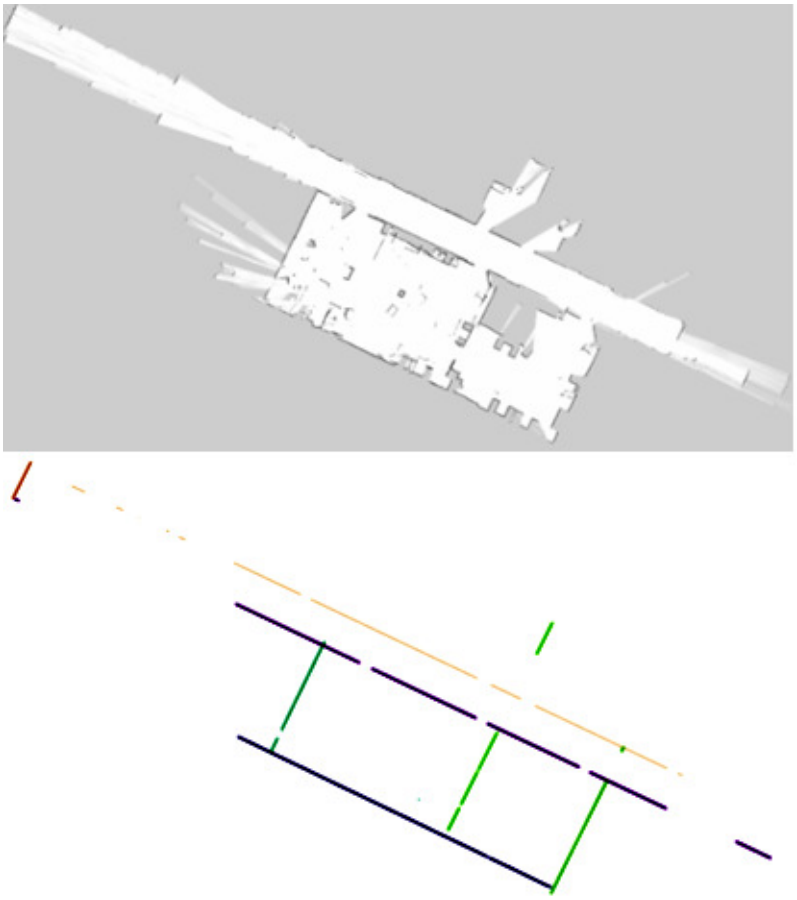

Figure 7-5: Extraction of walls from an occupancy map. **Top** The input map. The map was produced by a SLAM algorithm running on a robot and contains some errors. **Bottom** The walls found using a line detection algorithm (different walls have a different color). The most apparent walls are well segmented by the algorithm

best that can be done is to further split object candidates according to a given rule and run the object recognition on each part. The choice of a splitting however is not straightforward and can induce other problems. Some algorithms discussed in chapter 2, section 2.1.2 use criteria like compactness and convexity to detect objects in a scene. These algorithms lead to segments whose boundaries fit with apparent features of the point cloud like sharp angles and color changes. For a given scene, the segments produced by such a method can differ because they depend on the point of view and external factors like illumination. Another way to split a point cloud into segments is to follow a pre-defined grid. This is the kind of segmentation that an octree produces. Octrees, as will be discussed in section 7.6.1, can be used as the basis for environment reconstruction in robotics. Reconstruction algorithms based on octrees always split the space into cubes according to the same fixed grid. This could be preferred over appearance based splitting strategies to ensure that the produced segments correspond to the same objects or parts of objects no matter the viewpoint. This makes the accumulation of information from different views of the same object much more straightforward than an appearance based segmentation.

7.3 Features

The feature calculation step is where the point clouds are turned into descriptors. The descriptors encode a specific information about the input point cloud: the color, shape, texture, silhouette, etc. In this work, several features were used and the focus was put on choosing features that would provide complementary information. A shape feature, three color features and one texture feature were tested. The shape and texture features used in this work performed reasonably well for the task at hand. It is possible that using different features would lead to better results, but the improvement will probably be very limited. The flexibility of the nearest neighbor classifier offers the possibility to easily test different feature combinations. This section suggests possible replacements for these features. The color feature is discussed in more detail because it is a more difficult feature to exploit. Finally, the combination of these features is also addressed.

The increasing use of RGB-D cameras for object recognition caused several shape features to be presented recently (see [87] for a review). Shape features typically encode information about the relative position of points in the point cloud as well as the orientation of their normal. This is exactly the kind of information that is captured by the surflet-pair-relation histogram feature used in this thesis. It is possible that using a different feature would produce different recognition results, but it is hard to tell what feature would be the best. There also is a large choice of texture features. Among the most popular ones, local binary patterns have proved to be very efficient for texture recognition in difficult situations [54]. Local binary patterns are fast to compute and invariant to large changes in illumination and to rotations. They may be a good replacement for the transformed RGB feature or the histogram of SIFT words. Another very popular descriptor for object recognition are histograms of oriented gradients (HOGs). Histograms of oriented gradients have

been demonstrated to perform very well in tasks like pedestrians detection [27]. They however are not invariant to rotations, and might not be applicable to the current scenario.

The only feature type which did not seem to reach the best possible results are the color features. Color features are seldom used in real-life object recognition tasks. Color is a difficult property to work with because it is affected by many factors. The opponent color feature seemed to work better than the hue feature. But it might be possible to attain even better ones using another feature. A feature based on invariant color moments (see chapter 2, section 2.3.4) could be considered. The theory behind invariant color moments is sound and provides an appropriate set of features for a chosen combination of desired invariance. The drawback of invariant color moments is that they are high dimensional features. When the dimensionality of the feature increases, it becomes impossible to work with histograms because of the required amount of memory. One solution is the use of a vocabulary, as was done for the SIFT features in chapter 5. The vocabulary however adds an extra step in the design. Vocabularies are an approximation step which affect the precision of the feature. They must be carefully built. Still, it may be worth trying the invariant color moment features for object recognition. It may be interesting to use different types of features than the ones used in this work also. An example of an unused type of feature is size. It was purposefully omitted in this work. This is because it is difficult to estimate the size of the whole object to recognize. The only quantity which can be estimated at the step where features are computed is the size of an object candidate. However, the segmentation step is not guaranteed to correctly segment objects. If the view is a partial view of an object, the candidate's size will be smaller than the size of the actual object. If the view shows more than one objects, the size will be larger than the actual object's size. The size of an object candidate thus is not a reasonable piece of information to use in the current context.

7.4 Classifiers

Two classifiers were tested in this thesis: a neural network and a nearest neighbor classifier. Both classifiers had different behaviors on the set of experiments that were performed. The next sections will review and compare each classifier's advantages and drawbacks.

7.4.1 Offline experiments

The neural network yielded higher recognition rates on offline data than the nearest neighbor did. The neural network is a fine-tuned function approximation machine and it naturally performs well on data that is similar to the one it was trained on. Neural networks have a high number of parameters that are adjusted during training to adapt as well as possible to the training data. The parameters can play many different roles. They can select certain inputs over other ones, cancel some noise, take into account the local density of the feature space, and so on. This makes the

neural network very effective for offline experiments.

The case of the nearest neighbor is very different. The performance of the nearest neighbor classifier on offline experiments is dictated by how well the selected models cover the area in feature space spanned by the offline database. The higher the level of coverage, the higher the recall for offline experiments. A high coverage, however, comes at the expense of using a higher number of models, which increases the processing time. This is why the model selection method was designed to enforce a rather low number of models rather than ensuring a high level of coverage. This choice is implemented in the criterion that stops the selection method. In the algorithm described in chapter 6, section 6.3.2, the stopping criterion is based on a measure of the benefit of selecting a new model. If the benefit is too low, the method is stopped, no matter the coverage level of the set of already selected models. This design choice favors speed over recall, which is reflected in the lower offline performance compared to that of the neural network.

7.4.2 Online experiments

Both methods gave decent results on the online experiment, but the neural network had a sensibly better recall than the nearest neighbor method. Performing well on the online dataset means to be able to generalize over unseen viewing angles, occlusions and illuminations. The generalization over illumination probably is tied to the features more than to classifiers. The other generalizations are the result of all processing steps from features to multi-modal fusion, classification and multi-view fusion. It is difficult to conclude about which classifier is better in this situation. One observation is that the performance for both classifiers improves drastically from the single-view to the multi-view experiments. It seems that a large portion of the success of both methods in the online experiments is due to the ability to capitalize on one or a few good views of the objects. The multi-view fusion will be discussed in more detail in section 7.5.

More specifically, the online experiments expose the fact that the neural network does not extrapolate very well to unseen examples. If a test input falls outside the region or area of feature space covered by the training examples, there is no way to predict the behaviour of the neural network. This fact explains the drop in performance from the offline to the online experiments. It is also visible in the confusion matrix shown in figure 5-2, chapter 5. The confusion matrix shows that half of the confusions made by the neural network concern objects that do not look alike. The confusion matrix of the nearest neighbor search does not show this bad behavior. All confusions involve objects that have both a similar color and shape. This shows that, even if the nearest neighbor makes more mistakes than the neural network, the mistakes it does are much more sensible. The wrong classification from the nearest neighbor are still a good indication of the appearance of an object. This is not the case for the neural network.

Another downside of neural networks in online experiments is the necessity, if the neural network is to be confronted with objects that are not part of the offline database, to design a background class. A background class is a collection of images

that do not represent objects that should be recognized by the neural network. It is the only way to allow algorithms like neural networks to classify an input as "none of the known classes". In a robotics experiment, it will be common for the robot to encounter unknown objects. It is thus important for the classifier to have the capacity to detect such cases. Bad segmentations can also cause unknown objects, such as parts of the background, to be processed by the neural network. A background class was not designed in this work because it is an error-prone process that was not essential to reach the goals of the thesis. An alternate method was explored in the thesis though: to use of a threshold on the activation of the highest activated output neuron. However, the precision-recall curves shown in section 5.6.2 suggest that the decision threshold works well for offline data, but behaves very badly on online data. The problem of rejecting answers thus still is open for the neural network approach. The nearest neighbor, on the other hand, does not require such a procedure. The nearest neighbor search naturally rejects, or fails to answer to, examples that fall outside the region in feature space spanned by the selected models.

7.5 Multi-modal fusion

The exact features used have an impact on performance, but the way they are combined is probably much more important. Two fusion methods were used in this work: the neural network and a majority operator on the answers provided by separate nearest neighbor searches. The neural network provides a very good way to fuse different sources of information. The training procedure of a neural network learns the correct parameters to combine the information provided by the different features (if it is possible to do so). The structure of the neural network and high number of weights allows it to adapt to very diverse situations. This high adaptation potential is the reason why the neural network was used in the first version of the object recognition algorithm. With the replacement of the neural network, an equivalent procedure had to be designed. The choice was to accomplish the fusion of multi-modal information with a majority operation. The majority operator values all features equally. More importantly, it values all combinations of features equally. If a majority of features contains a given answer, the operator will return that answer no matter exactly which features are concerned. In a situation where several features have a common source of information, this is not the best solution. For example, the transformed RGB and the opponent color features are computed from the color information in the point cloud. Both features have the same source, but not the same properties: the transformed RGB has more invariances than the opponent color feature. The different combinations of search results can be interpreted with this fact in mind. If the transformed RGB search contains an answer but the opponent color feature does not, it might be a sign the light source in the scene is different than in the offline database. It is possible that the answer is correct. The opposite situation, where the opponent color feature search contains an answer that is not returned by the transformed RGB feature, is different. It is much less likely that an answer returned only by the opponent color feature be right because this feature is supposed to fail if the transformed

RGB fails. The features should therefore be arranged in a sort of hierarchy based on their invariances. A feature whose set of invariances is contained in the set of another feature's invariance should only serve to confirm the results of the more invariant feature. Its vote should not be considered for the answers that are not also returned by the more invariant feature. Features with different sources of information should be in separate branches of the hierarchy. The majority operator could then be applied on the different branches.

7.6 Perspectives

There are many ways to improve on the algorithms proposed in this thesis. Two major perspectives are discussed here. The first improvement should concern the integration of the several sources of knowledge available to the robot. As mentioned in the introduction of this thesis, a robot is an assembly of elements coming from diverse fields of research. Most of the different aspects of a robot come with their own theoretical background and implementation strategy. This patchwork of different algorithms can tackle many of the typical problems encountered in a robot at home scenario, as shown in this thesis. However, it would probably be very beneficial as a first improvement to unify the different pieces of software. More precisely, a unique representation of knowledge should be chosen to be common to all information processing in the algorithm before any other modification should happen. The second major improvement is to take advantage of the possibility offered by the algorithms to learn in an incremental and autonomous way. These two improvements are detailed in the following sections.

7.6.1 Common representation of knowledge

The software aspect of the robotic system presented in this work is a patchwork of algorithms. There is some exchange of information between the different processing cells, but there is no common representation of knowledge. Take the example of the semantic map. It is composed of the occupancy grid provided by the SLAM module and the objects identity from the object recognition pipeline. The occupancy grid is a discretization of space as a dense grid. Each grid cell takes the value 0 if it is empty and 1 if it is occupied. The semantic map is augmented by adding information about the objects found by the object classification algorithm. The knowledge about objects takes the form of a list of coordinates that correspond to the center of mass of each object. Each entry in the list also contains the recognition scores of the object. The unit of knowledge is completely different between the map and the objects. The map has units of information which are indexed by their position. The object recognition's unit of information is indexed by a unique identification number. Each piece of information in the map corresponds to an area in the environment. The list of objects represents objects by point masses, without any reference to the volume or area they occupy. Yet, the two modules describe the environment in which the robot operates. Moreover, the knowledge produced by one module could benefit to the other

module. As stated earlier, the segmentation module should take advantage of the big picture provided by the map to detect walls more efficiently. Additionally, the SLAM module can provide information about moving objects and help the segmentation process [48]. Also, the location and class of the objects recognized by the recognition algorithm could benefit to the SLAM module by providing landmarks [70, 12]. If the modules do not represent the knowledge in a common form, conversion must happen for every exchange of information. Conversions waste computational resources and lead to approximation and possible accumulation of error.

Before more complex tasks can be undertaken, a common unit of knowledge representation should be chosen. There are many possible representations. The representation must encompass map making and semantic information about objects and properties of the environment. In robotics, the scale of the environment and the processing power limits the possibilities for the unit of representation. Two examples of large scale localization and mapping approaches that have proven to work in such a setting are feature-based mapping [39, 38] and octree based methods [64]. The feature-based method stores information much like the object information in this work. The visual appearance of the places visited by the robots is stored as a list of features. The representation is compact, it only contains the minimum amount of information required for processing. On the other hand, octree based methods are similar to the grid occupancy map of the SLAM module of this work. An octree map splits the 3-dimensional environment in cubes that store occupancy information. It intrinsically contains information about the 3-dimensional structure of the environment. Any other information can be stored in the octree. For the purposes of this work, the recognition scores for each object would be stored in the octree cubes that form the object. One point to consider for the choice of a representation is that a robot at home should be able to share information with its owner. A feature-based approach is optimized for software use and does not allow much interaction with a human unless additional information is stored specifically for this purpose. A reconstructed environment produced by an octree, on the other hand, is naturally understandable by a human. This makes them especially well suited for robots at home. An example of an area mapped using OctoMap [33] is shown in figure 7-6. A last advantage of the octree representation is that it produces a geometric segmentation of the scene. As already stated in section 7.2, this segmentation could be useful to split object candidates into small parts in a repeatable way.

A third category of representation for reconstructed environments are meshes. For real-time reconstruction from RGB-D cameras, meshed can be computed from a truncated sign distance function representation [95]. They have been used to map indoor environments but are typically demonstrated on systems with substantial computational resources and memory. Meshes represent the surfaces in a scene and are useful for physical simulations and for visualization purposes. The surfaces information is hardly useful for the creation of a semantic map.

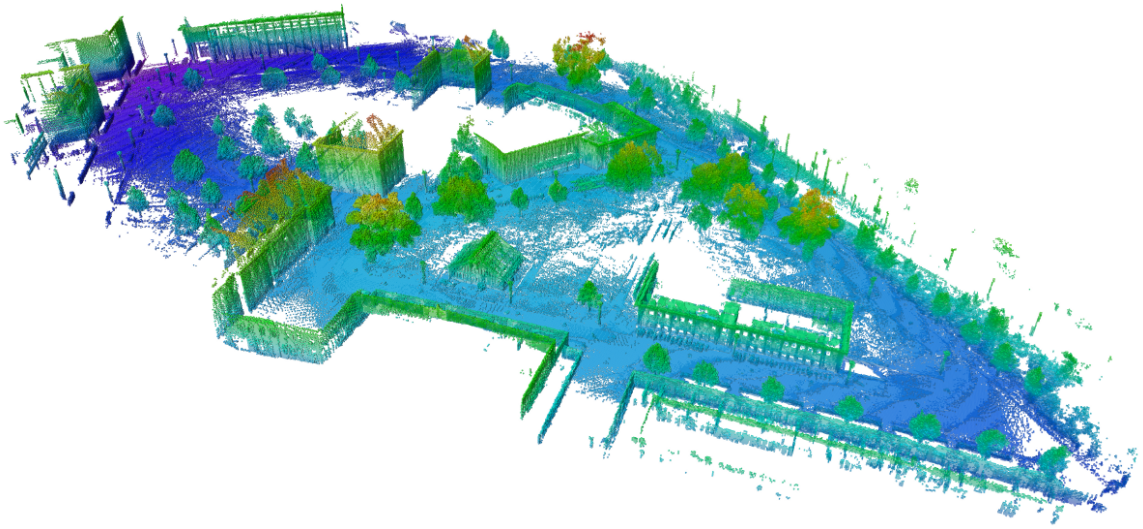

Figure 7-6: An outdoor scene reconstructed by the OctoMap algorithm. The height of the points are color coded (blue is low, red is high). Image viewed 9 April 2015 https://octomap.github.io/freiburg_outdoor_big.png.

7.6.2 Autonomous incremental learning

This thesis worked on the basis that its owner could show the objects to the robot before any recognition can happen. During this operation, the objects should be presented in such a way that the robot can easily segment them from the scene and save the features that characterize them. After this operation, the knowledge of the robot is fixed. Another phase of learning could be done by the owner if he so desires, but the robot will not update its knowledge base by itself. Endowing the robot with the ability to update and improve its own database seems necessary to increase the recognition rates above the scores showed in this thesis. This ability is facilitated if the algorithms lend themselves to incremental learning. Incremental learning means to be able to selectively update a knowledge base. It is very difficult to do incremental learning with the neural network based approach. If objects are added to the database, the neural network can be adapted by running the training procedure with the augmented set of views. But the very small number of added views for a new object compared to more than 30 000 views of already existing objects means that the performance for the new object will be difficult to guarantee. The convergence of the training will most likely be very long. This was an important reason to replace the neural network by a more flexible classifier. The nearest neighbor strategy that was adopted is a very good example of an algorithm that is well adapted to incremental learning. Adding new objects to the nearest neighbor classifier does not require altering what is known about the other objects. The model selection method can be run on the examples of the new object alone and the appropriate views added to the list of models. Of course, as more models get added to the database, the memory usage will increase and the nearest neighbor searches will slow down. That might be a problem in very long term experiments. Whether or not the number of models

needed for a robot at home to perform properly is too large is to be tested.

The fact that the robot incrementally learn does not mean it can learn autonomously. For a robot to learn autonomously, a mechanism must exist to tell it when to learn and when not to. Otherwise, each view of an object would be saved to memory and the robot would rapidly run out of resources. In this thesis, it is the owner that replaces this mechanism. The owner places an object in front of the robot and tells it to save this view in memory. It then shows the object differently before telling the robot to learn again. For the robot to be able to autonomously learn new models of an object, a mechanism must be implemented to determine when a given view is different enough from the set of already known ones and is worth adding to the knowledge base. The model selection method proposed in this thesis implements such a mechanism. In the model selection method, the physical position of the robot with regard to the object is used to determine whether or not two views should be considered as the same. The feature space similarity threshold of the model selection procedure could be used as a learning threshold. If a new view is more similar to a known model than this model's feature space similarity threshold, it is not learned. Otherwise, the view and all the views of this object that are taken from close-by positions go through the model selection process. One of the views is chosen according to the selection procedure and its feature space similarity threshold is computed and saved.

7.7 Conclusion

In this thesis, an object instance recognition algorithm using an RGB-D camera was presented and tested in a robot at home scenario. The robot used is a pioneer 3D-X equipped with a Kinect RGB-D camera, an Hokuyo laser range finder and a Toshiba Tecra laptop computer (Intel Core i5, 3 GB RAM). A database of 52 commonplace objects and pieces of furniture was compiled to simulate a recognition task for a robot a home. The database contains an online part where 20 out of the 52 objects are seen in a different context than in the offline part. The views of the objects in the online part are subject to large occlusions, illumination changes and scale factors. Training a recognition algorithm on the offline dataset and testing it on the online one is a challenging task representative of what a robot at home would have to do. The algorithm developed in this thesis successfully recognizes 13 out of the 20 objects in the online database. The algorithm processes a point cloud from the Kinect in 500 milliseconds in average. The run time depends on the content of the point cloud.

The success of the algorithm demonstrates the efficiency of an indoor scene segmentation algorithm based on geometric cues only. This kind of segmentation algorithm takes advantage of the structure in indoor scenes. The results obtained with a neural network classifier showed that it is very difficult to recognize objects when they are seen in a different context than the one from the training database. The use of complementary features helps achieve good recognition rates in this situation. Also, the fusion of the recognition scores obtained from different views of the same object greatly improves the algorithm's performance. Finally, the physical distance

between the objects and the camera is used to guide the unsupervised clustering of the database during a model selection procedure. This method prevents the use of the K-means algorithm and does not require to set the number of clusters to form.

Future work should concentrate on a better integration of the mapping and the recognition algorithms by using a common representation of the information. Also, adapting the model selection method could allow the robot to autonomously learn models of new objects.

Appendix A

Nearest neighbor experiment results

A.1 Multi-modal offline experiment results

Table A.1: Recognition rates for multi-modal offline experiments using L_2 distance metric (21 411 examples).

SPRH	OppCo	TBGR	Recall	Number of labels	Ties
3×3	8×8	8	82.89%	2.75	7864
3×3	8×8	16	78.63%	**2.46**	5664
3×3	8×8	32	74.06%	**2.35**	**3972**
3×3	16×16	8	85.08%	2.81	8214
3×3	16×16	16	81.34%	2.48	6106
3×3	16×16	32	76.89%	**2.35**	**4260**
4×4	8×8	8	88.11%	3.09	10911
4×4	8×8	16	84.28%	2.82	8870
4×4	8×8	32	81.05%	2.70	7304
4×4	16×16	8	**90.98%**	3.20	11247
4×4	16×16	16	87.71%	2.89	9488
4×4	16×16	32	84.65%	2.76	7766
5×5	8×8	8	87.78%	3.02	10056
5×5	8×8	16	84.35%	2.72	8172
5×5	8×8	32	81.26%	2.57	6431
5×5	16×16	8	**90.59%**	3.12	10691
5×5	16×16	16	87.92%	2.82	8933
5×5	16×16	32	85.03%	2.66	7169

Table A.2: Recognition rates for multi-modal offline experiments using χ^2 distance metric (21 411 examples).

SPRH	OppCo	TBGR	Recall	Number of labels	Ties
3×3	8×8	8	**86.85%**	2.93	8239
3×3	8×8	16	84.69%	2.54	6328
3×3	8×8	32	82.10%	2.48	5327
3×3	16×16	8	**86.88%**	2.85	8608
3×3	16×16	16	84.59%	2.51	6301
3×3	16×16	32	82.12%	2.44	5190
4×4	8×8	8	**86.28%**	2.81	7177
4×4	8×8	16	84.28%	2.46	5163
4×4	8×8	3	81.97%	2.41	4065
4×4	16×16	8	**86.68%**	2.73	7547
4×4	16×16	16	84.36%	2.43	5068
4×4	16×16	32	82.28%	**2.35**	3806
5×5	8×8	8	**85.65%**	2.79	6816
5×5	8×8	1	83.64%	2.45	4795
5×5	8×8	32	81.07%	2.40	3700
5×5	16×16	8	**85.63%**	2.70	7202
5×5	16×16	16	83.34%	2.42	4701
5×5	16×16	32	81.20%	**2.35**	**3471**

Table A.3: Recognition rates for multi-modal offline experiments using Jeffreys distance metric (21 411 examples).

SPRH	OppCo	TBGR	Recall	Number of labels	Ties
3×3	8×8	8	89.16%	3.16	10982
3×3	8×8	16	91.10%	3.12	11785
3×3	8×8	32	91.28%	3.13	11918
3×3	16×16	8	90.00%	3.15	12228
3×3	16×16	16	91.93%	3.14	13012
3×3	16×16	32	**92.15%**	3.16	13180
4×4	8×8	8	89.57%	3.15	**10185**
4×4	8×8	16	91.11%	**3.10**	**10808**
4×4	8×8	32	91.12%	3.11	11021
4×4	16×16	8	90.43%	**3.10**	11505
4×4	16×16	16	**92.05%**	3.11	12232
4×4	16×16	32	**92.04%**	**3.10**	12442
5×5	8×8	8	89.90%	3.18	**10206**
5×5	8×8	16	91.26%	3.15	**10682**
5×5	8×8	32	91.49%	3.15	11015
5×5	16×16	8	90.50%	3.13	11499
5×5	16×16	16	**92.08%**	3.13	12189
5×5	16×16	32	**92.30%**	3.13	12497

Table A.4: Recognition rates for multi-modal offline experiments using squared-chord distance metric (21 411 examples).

SPRH	OppCo	TBGR	Recall	Number of labels	Ties
3×3	8×8	8	85.73%	2.93	8441
3×3	8×8	16	84.94%	2.62	7167
3×3	8×8	32	82.66%	**2.42**	5585
3×3	16×16	8	**86.80%**	2.87	8623
3×3	16×16	16	**86.53%**	2.59	7033
3×3	16×16	32	84.47%	**2.41**	5622
4×4	8×8	8	85.40%	2.84	7340
4×4	8×8	16	84.08%	2.55	6000
4×4	8×8	32	82.32%	**2.36**	**4326**
4×4	16×16	8	**86.45%**	2.78	7490
4×4	16×16	16	85.48%	2.52	5869
4×4	16×16	32	83.84%	**2.33**	4447
5×5	8×8	8	85.48%	2.85	7378
5×5	8×8	16	84.52%	2.56	6096
5×5	8×8	32	82.49%	**2.38**	**4318**
5×5	16×16	8	**86.40%**	2.78	7503
5×5	16×16	16	**85.88%**	2.52	5949
5×5	16×16	32	84.17%	**2.33**	4449

A.2 Single-view online experiment results

Table A.5: Recognition rates for online experiments using L_2 distance metric without multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Ties	Object recall
3×3	8×8	8	22.70%	14	14
3×3	8×8	16	17.02%	**4**	12
3×3	8×8	32	13.48%	**3**	11
3×3	16×16	8	24.11%	18	15
3×3	16×16	16	19.15%	9	11
3×3	16×16	32	15.60%	**6**	10
4×4	8×8	8	27.66%	31	15
4×4	8×8	16	22.70%	25	13
4×4	8×8	32	19.15%	20	13
4×4	16×16	8	**29.79%**	38	**17**
4×4	16×16	16	24.11%	25	13
4×4	16×16	32	20.57%	19	12
5×5	8×8	8	26.95%	24	15
5×5	8×8	16	21.28%	16	12
5×5	8×8	32	17.73%	14	12
5×5	16×16	8	26.95%	29	**17**
5×5	16×16	16	21.99%	28	13
5×5	16×16	32	18.44%	17	12

Table A.6: Recognition rates for online experiments using χ^2 distance metric without multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Ties	Object recall
3×3	8×8	8	24.11%	16	**15**
3×3	8×8	16	21.28%	4	13
3×3	8×8	32	17.73%	4	13
3×3	16×16	8	**32.62%**	18	**16**
3×3	16×16	16	29.79%	5	**15**
3×3	16×16	32	24.82%	5	**15**
4×4	8×8	8	22.70%	15	13
4×4	8×8	16	21.28%	**2**	13
4×4	8×8	3	14.89%	**2**	11
4×4	16×16	8	27.66%	18	14
4×4	16×16	16	26.24%	6	13
4×4	16×16	32	19.86%	5	12
5×5	8×8	8	17.73%	9	12
5×5	8×8	16	17.02%	**1**	11
5×5	8×8	32	12.77%	**1**	11
5×5	16×16	8	26.95%	17	14
5×5	16×16	16	24.82%	4	13
5×5	16×16	32	17.73%	3	12

Table A.7: Recognition rates for online experiments using Jeffreys distance metric without multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Ties	Object recall
3×3	8×8	8	29.08%	28	13
3×3	8×8	16	31.91%	29	13
3×3	8×8	32	27.66%	**19**	14
3×3	16×16	8	51.06%	60	**18**
3×3	16×16	16	**52.48%**	56	**18**
3×3	16×16	32	**51.77%**	59	17
4×4	8×8	8	28.37%	**20**	15
4×4	8×8	16	30.50%	**24**	15
4×4	8×8	32	27.66%	**21**	15
4×4	16×16	8	50.35%	55	**19**
4×4	16×16	16	49.65%	54	**19**
4×4	16×16	32	47.52%	56	18
5×5	8×8	8	29.79%	**21**	14
5×5	8×8	16	31.21%	**24**	14
5×5	8×8	32	27.66%	**17**	14
5×5	16×16	8	**51.77%**	53	**18**
5×5	16×16	16	50.35%	53	**18**
5×5	16×16	32	50.35%	55	17

Table A.8: Recognition rates for online experiments using squared-chord distance metric without multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Ties	Object recall
3×3	8×8	8	**31.91%**	19	15
3×3	8×8	16	31.21%	10	**17**
3×3	8×8	32	25.53%	**8**	**16**
3×3	16×16	8	**32.62%**	28	15
3×3	16×16	16	26.24%	14	14
3×3	16×16	32	21.28%	**7**	14
4×4	8×8	8	29.79%	22	13
4×4	8×8	16	24.82%	11	12
4×4	8×8	32	19.86%	**7**	11
4×4	16×16	8	30.50%	30	14
4×4	16×16	16	24.11%	15	13
4×4	16×16	32	19.86%	10	13
5×5	8×8	8	30.49%	21	13
5×5	8×8	16	26.95%	11	13
5×5	8×8	32	21.28%	**6**	12
5×5	16×16	8	30.50%	29	14
5×5	16×16	16	25.53%	13	14
5×5	16×16	32	20.57%	**8**	14

A.3 Multi-view online experiment results

Table A.9: Recognition rates for online experiments using L_2 distance metric with multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Correct objects
3×3	8×8	8	**53.90%**	8
3×3	8×8	16	47.52%	**9**
3×3	8×8	32	43.97%	**9**
3×3	16×16	8	47.52%	**9**
3×3	16×16	16	41.13%	**9**
3×3	16×16	32	45.39%	**9**
4×4	8×8	8	14.18%	4
4×4	8×8	16	21.28%	4
4×4	8×8	32	26.95%	4
4×4	16×16	8	33.33%	5
4×4	16×16	16	32.62%	7
4×4	16×16	32	34.04%	7
5×5	8×8	8	29.08%	5
5×5	8×8	16	31.91%	4
5×5	8×8	32	29.79%	5
5×5	16×16	8	34.75%	5
5×5	16×16	16	29.08%	7
5×5	16×16	32	38.30%	7

Table A.10: Recognition rates for online experiments using χ^2 distance metric with multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Correct objects
3×3	8×8	8	36.17%	9
3×3	8×8	16	36.88%	9
3×3	8×8	32	40.43%	9
3×3	16×16	8	51.07%	10
3×3	16×16	16	**56.74%**	**12**
3×3	16×16	32	**58.87%**	**12**
4×4	8×8	8	32.62%	8
4×4	8×8	16	36.88%	10
4×4	8×8	32	36.17%	8
4×4	16×16	8	43.97%	9
4×4	16×16	16	41.13%	9
4×4	16×16	32	47.52%	9
5×5	8×8	8	26.95%	8
5×5	8×8	16	36.88%	10
5×5	8×8	32	46.10%	10
5×5	16×16	8	29.08%	8
5×5	16×16	16	41.13%	10
5×5	16×16	32	48.23%	11

Table A.11: Recognition rates for online experiments using Jeffreys distance metric with multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Correct objects
3×3	8×8	8	31.21%	5
3×3	8×8	16	47.52%	7
3×3	8×8	32	44.68%	8
3×3	16×16	8	46.81%	6
3×3	16×16	16	42.55%	8
3×3	16×16	32	51.77%	7
4×4	8×8	8	34.75%	7
4×4	8×8	16	53.19%	9
4×4	8×8	32	60.99%	**10**
4×4	16×16	8	52.48%	8
4×4	16×16	16	48.23%	7
4×4	16×16	32	51.77%	9
5×5	8×8	8	39.01%	8
5×5	8×8	16	**66.67%**	8
5×5	8×8	32	**66.67%**	9
5×5	16×16	8	60.28%	9
5×5	16×16	16	48.23%	8
5×5	16×16	32	51.77%	**10**

Table A.12: Recognition rates for online experiments using squared-chord distance metric with multi-view fusion (141 examples, 20 objects).

SPRH	OppCo	TBGR	Recall	Correct objects
3×3	8×8	8	41.84%	8
3×3	8×8	16	51.06%	9
3×3	8×8	32	55.32%	10
3×3	16×16	8	38.30%	6
3×3	16×16	16	45.39%	10
3×3	16×16	32	49.65%	11
4×4	8×8	8	41.84%	10
4×4	8×8	16	45.39%	8
4×4	8×8	32	45.39%	10
4×4	16×16	8	53.19%	7
4×4	16×16	16	53.90%	8
4×4	16×16	32	**58.16%**	10
5×5	8×8	8	41.84%	10
5×5	8×8	16	46.10%	9
5×5	8×8	32	46.10%	11
5×5	16×16	8	53.19%	6
5×5	16×16	16	54.61%	11
5×5	16×16	32	**58.87%**	**13**

Bibliography

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, November 2012.

[2] Rolf Addams and Leanne Bischof. Seed Region Growing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(6):641–647, 1994.

[3] Haider Ali, Faisal Shafait, Eirini Giannakidou, Athena Vakali, Nadia Figueroa, Theodoros Varvadoukas, and Nikolaos Mavridis. Contextual object category recognition for RGB-D scene labeling. *Robotics and Autonomous Systems*, 62(2):241–256, February 2014.

[4] Oscar Alonso-Ramirez, Antonio Marin-Hernandez, Michel Devy, and Fernando M. Montes-Gonzalez. Indoor home furniture detection with RGB-D data for service robots. *CONIELECOMP 2014 - 24th International Conference on Electronics, Communications and Computers*, pages 172–177, 2014.

[5] Relja Arandjelovi and Andrew Zisserman. Three things everyone should know to improve object retrieval. *IEEE Conference on computer vision and Pattern Recognition*, pages 2911–2918, 2012.

[6] Himanshu Arora, Nicolas Loeff, David a. Forsyth, and Narendra Ahuja. Unsupervised Segmentation of Objects using Efficient Learning. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, June 2007.

[7] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer VisionâĂŞECCV 2006*, pages 404–417, 2006.

[9] Tom Botterill, Steven Mills, and Richard Green. Speeded-up bag-of-words algorithm for robot localisation through scene recognition. *2008 23rd International Conference Image and Vision Computing New Zealand, IVCNZ*, 2008.

[10] Gary Bradski. The opencv library. *Doctor Dobb's Journal of Software Tools*, 25:120–126, 2000.

[11] Sung-hyuk Cha. Comprehensive Survey on Distance / Similarity Measures between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307, 2007.

[12] Siddharth Choudhary, Alexander J B Trevor, Henrik I Christensen, and Frank Dellaert. SLAM with Object Discovery, Modeling and Mapping. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1018–1025, 2014.

[13] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. *Proceedings of the ECCV International Workshop on Statistical Learning in Computer Vision*, pages 59–74, 2004.

[14] Hendrik Dahlkamp, Adrian Kaehler, David Stavens, Sebastian Thrun, and Gary Bradski. Self-supervised Monocular Road Detection in Desert Terrain. *Proc of Robotics Science and Systems RSS*, 2006.

[15] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 886–893, 2005.

[16] Bertram Drost, Markus Ulrich, Nassir Navab, and Slobodan Ilic. Model globally, match locally: Efficient and robust 3D object recognition. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 998–1005, June 2010.

[17] Mathieu Dubois, Paola K. Rozo, Alexander Gepperth, O. Fabio a Gonzalez, and David Filliat. A comparison of geometric and energy-based point cloud semantic segmentation methods. *2013 European Conference on Mobile Robots, ECMR 2013 - Conference Proceedings*, pages 88–93, 2013.

[18] Staffan Ekvall, Patric Jensfelt, and Danica Kragic. Integrating Active Mobile Robot Object Recognition and SLAM in Natural Environments. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5792–5797, October 2006.

[19] Jan Elseberg, Stéphane Magnenat, Roland Siegwart, and Nüchter Andreas. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics (JOSER)*, 3(February):2–12, 2012.

[20] Jianping Fan, David K Y Yau, Ahmed K. Elmagarmid, and Walid G. Aref. Automatic image segmentation by integrating color-edge extraction and seeded region growing. *IEEE Transactions on Image Processing*, 10(10):1454–1466, 2001.

[21] Charles M. Felps, Michael H. Fick, Keegan R. Kinkade, Jeremy Searock, and Jenelle Armstrong Piepmeier. Integration of semantic vision techniques for an autonomous robot platform. *Proceedings of the Annual Southeastern Symposium on System Theory*, pages 243–247, 2010.

[22] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, September 2004.

[23] Ross Finman, Thomas Whelan, Michael Kaess, and John J Leonard. Toward lifelong object segmentation from change detection in dense RGB-D maps. *Mobile Robots (ECMR)*, 2013.

[24] Martin a Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting with. *Communications of the ACM*, 24:381–395, 1981.

[25] Per Erik Forssén, David Meger, Kevin Lai, Scott Helmer, James J. Little, and David G. Lowe. Informed visual search: Combining attention and object recognition. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 935–942, 2008.

[26] David Freedman and Persi Diaconis. On the histogram as a density estimator: L2 theory. *Probability theory and related fields*, 57:453–476, 1981.

[27] Alexander Gepperth, Egor Sattarov, Bernd Heisele, and Sergio Alberto Rodriguez Florez. Robust visual pedestrian detection by tight coupling to tracking. In *IEEE International Conference On Intelligent Transportation Systems (ITSC)*, pages 1935—-1940, 2014.

[28] Lena Gorelick, Olga Veksler, Yuri Boykov, and Claudia Nieuwenhuis. Convexity Shape Prior for Segmentation. *Computer Vision–ECCV 2014*, pages 1–16, 2014.

[29] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall, 1999.

[30] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE transactions on pattern analysis and machine intelligence*, 34(5):876–88, May 2012.

[31] Dirk Holz, Stefan Holzer, RB Rusu, and Sven Behnke. Real-time plane segmentation using RGB-D cameras. *RoboCup 2011: Robot Soccer World Cup XV*, pages 306–317, 2012.

[32] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. *IEEE International Conference on Intelligent Robots and Systems*, pages 2684–2689, 2012.

[33] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[34] C Igel and Michael Hüsken. Empirical evaluation of the improved Rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003.

[35] Shahram Izadi, Andrew Davison, Andrew Fitzgibbon, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Dustin Freeman. Kinect Fusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, page 559, 2011.

[36] Andrej Karpathy, Stephen Miller, and Li Fei-Fei. Object discovery in 3D scenes via shape analysis. *Robotics and Automation (ICRA*, 2013.

[37] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable SLAM system with full 3D motion estimation. In *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, pages 155–160, 2011.

[38] Hemanth Korrapati and Youcef Mezouar. Vision-based sparse topological mapping. *Robotics and Autonomous Systems*, 62(9):1259–1270, 2014.

[39] Mathieu Labbé and François Michaud. Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *Robotics, IEEE Transactions on*, 29(3):734–745, 2013.

[40] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1817–1824. Ieee, May 2011.

[41] Xiang Li, Mohan Sridharan, and Shiqi Zhang. Autonomous learning of vision-based layered object models on mobile robots. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 6239–6244. IEEE, 2011.

[42] Zhe Lin, Sungho Kim, and I.S. Kweon. Robust invariant features for object recognition and mobile robot navigation. *Proceedings of IAPR Conference on Machine Vision Applications, Tsukuba Science City, Japan*, 2005.

[43] Haibin Ling and Kazunori Okada. Diffusion distance for histogram comparison. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:246–253, 2006.

[44] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[45] Natalia Lyubova, David Filliat, and Serena Ivaldi. Improving object learning through manipulation and self-identification. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 1365–1370, 2013.

[46] Rotem Mairon and Ohad Ben-shahar. A Closer Look at Context : From Coxels to the Contextual Emergence of Object Saliency. In *Computer Vision–ECCV 2014*, pages 708–724, 2014.

[47] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2308–2315, 2010.

[48] D Márquez-Gámez and M Devy. Active visual-based detection and tracking of moving objects from clustering and classification methods. *Advanced Concepts for Intelligent Vision Systems*, pages 361–373, 2012.

[49] J Matas, O Chum, M Urban, and T Pajdla. Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. In *In British Machine Vision Conference*, pages 384–393, 2002.

[50] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

[51] Florica Mindru, Tinne Tuytelaars, Luc Van Gool, and Theo Moons. Moment invariants for recognition under changing viewpoint and illumination. *Computer Vision and Image Understanding*, 94:3–27, 2004.

[52] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, Anna Petrovskaya, Mike Pflueger, Ganymed Stanek, David Stavens, Antone Vogt, and Sebastian Thrun. Junior: The stanford entry in the urban challenge. *Springer Tracts in Advanced Robotics*, 56(October 2005):91–123, 2009.

[53] CA Mueller, Kaustubh Pathak, and Andreas Birk. Object recognition in rgbd images of cluttered environments using graph-based categorization with unsupervised learning of shape parts. *Intelligent Robots and Systems*, pages 2248–2255, 2013.

[54] Thanh Phuong Nguyen, Antoine Manzanera, and Walter G Kropatsch. Impact of topology-related attributes from Local Binary Patterns on texture classification. In *ECCVWorkshop on Computer Vision with Local Binary Patterns Variants*, 2014.

[55] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:2161–2168, 2006.

[56] Andreas Nüchter and Joachim Hertzberg. Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, 56(11):915–926, 2008.

[57] Kei Okada, Mitsuharu Kojima, Satoru Tokutsu, Toshiaki Maki, Yuto Mori, and Masayuki Inaba. Multi-cue 3D Object recognition in knowledge-based vision-guided humanoid robot system. *IEEE International Conference on Intelligent Robots and Systems*, pages 3217–3222, 2007.

[58] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[59] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[60] Lutz Prechelt. Early Stopping – But When? In *Neural Networks: Tricks of the Trade*, pages 55–69. Springer, 1998.

[61] Morgan Quigley, Ken Conley, Brian Gerkey, Josh FAust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Mg. ROS: an open-source Robot Operating System. *Icra*, 3(Figure 1):5, 2009.

[62] Arnau Ramisa, Shrihari Vasudevan, Davide Scaramuzza, Ramón López de Mántaras, and Roland Siegwart. A tale of two object recognition methods for mobile robots. *Computer Vision Systems*, pages 353—-362, 2008.

[63] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. *IEEE International Conference on Neural Networks*, 1993.

[64] Hélène Roggeman, Julien Marzat, Martial Sanfourche, and Aurélien Plyer. Embedded vision-based localization and model predictive control for autonomous exploration. In *IROS Workshop on Visual Control of Mobile Robots (ViCoMoR 2014)*, pages 13—-20, 2014.

[65] Ed Rosten and Tom Drummond. Machine Learning for High Speed Corner Detection. *Computer Vision – ECCV 2006*, pages 430–443, 2004.

[66] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International journal of computer vision*, 40(2):99—-121, 2000.

[67] Daniel L. Ruderman, Thomas W. Cronin, and Chuan-Chin Chiao. Statistics of cone responses to natural images: implications for visual coding. *Journal of the Optical Society of America A*, 15(8):2036, 1998.

[68] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[69] RB Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 ...*, 2011.

[70] Renato F. Salas-Moreno, Richard a. Newcombe, Hauke Strasdat, Paul H.J. Kelly, and Andrew J. Davison. SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, June 2013.

[71] Nizar Sallem and Michel Devy. Modelisation d'objets 3D en vue de leur reconnaissance et leur manipulation par un robot personnel. In *ORASISâĂŹ09 - Congres des jeunes chercheurs en vision e par ordinateur*, 2009.

[72] T Schaul, J Bayer, D Wierstra, Y Sun, M Felder, F Sehnke, T Rückstieß, and J Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.

[73] D W Scott. On optimal and data-based histograms. *Biometrika*, 66:605–610, 1979.

[74] Ayet Shaiek. *3D object recognition with points of interest.* PhD thesis, École Nationale Supérieure des Mines de Paris, 2013.

[75] Ayet Shaiek and Fabien Moutarde. Fast 3D keypoints detector and descriptor for view-based 3D objects recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7854 LNCS:106–115, 2013.

[76] Hideaki Shimazaki and Shigeru Shinomoto. A method for selecting the bin size of a time histogram. *Neural computation*, 19:1503–1527, 2007.

[77] Arjun Singh, James Sha, Karthik S Narayan, Tudor Achim, and Pieter Abbeel. BigBIRD : A Large-Scale 3D Database of Object Instances. *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 509–516, 2014.

[78] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, pages 1–9, 2001.

[79] J Sivic and A Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, October 2003.

[80] Kristoffer Sjö, Dorian Gálvez López, Chandana Paul, Patric Jensfelt, and Danica Kragic. Object Search and Localization for an Indoor Mobile Robot. *Journal of Computing and Information Technology*, pages 67–80, 2009.

[81] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, July 2009.

[82] Mohan Sridharan and Peter Stone. Real-Time Vision on a Mobile Robot Platform. *Image (Rochester, N.Y.)*, pages 1–9, 2005.

[83] Frank Steinbrucker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense RGB-D images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 719—-722, 2011.

[84] M J Swain and D H Ballard. Color indexing. *International Journal of Computer Vision*, 7:11–32, 1991.

[85] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, and And Others. Stanley: The robot that won the DARPA Grand Challenge. *Journal of field Robotics*, 23(9):661—-692, 2006.

[86] M. Tkalcic and J.F. Tasic. Colour spaces: perceptual, historical and applicational background. *The IEEE Region 8 EUROCON 2003. Computer as a Tool.*, 1, 2003.

[87] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *Computer Vision–ECCV 2010*, pages 356–369. Springer, 2010.

[88] A Trevor, Suat Gedikli, Radu Bogdan Rusu, and Henrik I Christensen. Efficient organized point cloud segmentation with connected components. In *Proceedings of Semantic Perception Mapping and Exploration*, pages 1–6, 2013.

[89] Andre Uckermann, Robert Haschke, and Helge Ritter. Real-time 3D segmentation of cluttered scenes for robot grasping. *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 198–203, November 2012.

[90] Koen Erik Adriaan van de Sande. *Invariant color descriptors for efficient object recognition*. PhD thesis, FNWI: Informatics Institute, 2011.

[91] E Wahl, U Hillenbrand, and G Hirzinger. Surflet-pair-relation histograms: a statistical 3D-shape representation for rapid classification. In *Proceedings of the Fourth International Conference on 3-D Digital Imaging and Modeling (3DIM)*, 2003.

[92] Min Liang Wang and Huei Yung Lin. Object recognition from omnidirectional visual sensing for mobile robot applications. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pages 1941–1946, 2009.

[93] Tomáš Werner. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1165–1179, 2007.

[94] Jamie Westell and Parvaneh Saeedi. 3D object recognition via multi-view inspection in unknown environments. *11th International Conference on Control, Automation, Robotics and Vision, ICARCV 2010*, pages 2088–2095, 2010.

[95] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J Leonard, and John Mcdonald. Robust Real-Time Visual Odometry for Dense RGB-D Mapping. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5724–5731, 2013.

[96] Thomas Whelan, Michael Kaess, and Maurice Fallon. Kintinuous: Spatially extended kinectfusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[97] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction. In *pre-print*, pages 1–9, 2014.

[98] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):6–es, 2006.