Towards Unsupervised And Incremental Semantic Mapping

Guillaume Duceux

Towards Unsupervised And Incremental Semantic Mapping

Thèse présentée par

Guillaume Dominique Duceux

à l'Unité Informatique et Ingénierie des Systèmes
pour l'obtention du grade de Docteur de
l'ENSTA ParisTech
Spécialité : Informatique

Thèse soutenue le
27 Novembre 2015

devant le jury composé de

rapporteurs:	Simon Lacroix	CNRS
	Olivier Aycard	University of Grenoble 1
	Paul Checchin	Institut Pascal
examinateurs:	Patrick Rives	INRIA
	Eric Moline	DGA
directeur de thèse:	David Filliat	ENSTA-ParisTech

U2IS, ENSTA-ParisTech
Palaiseau, France

January 15, 2016

Abstract

Robots are slowly entering our houses. But for them to perform more difficult and interesting tasks, they need to have a richer knowledge of the different objects and places present in a particular household. Building and using such a complex knowledge is called semantic mapping and navigation, which requires many different capabilities such as mapping, localization and object recognition. Most of them have been thoroughly studied in the past but often separately and in a different context. We propose a complete solution to perform incremental and unsupervised semantic mapping working on a real robot. Each aspect of the solution is discussed from the software architecture to all the functionalities needed by the robot. Some already existing techniques have been derived in the effort of integration, but more importantly three different parts of the solution are original contributions. The first one is the use of 2D laser range finder to perform object recognition using multiple views of the objects. Numerous semantic mapping solutions are using this sensor for mapping and obstacle avoidance, but we fully leveraged it by developing techniques to use it in the entire process of semantic mapping, showing that it is effective at recognizing up to 20 different objects in an indoor scenario. Secondly we introduce the graph-of-views object modeling method. In our effort to extend bag-of-views techniques to perform multi-modal modeling of objects we derived an original formulation which is more adapted to partial perception of objects. We applied this approach to object recognition using both a laser range finder and a RGB-D camera, showing that it is able to take advantage of both sensor strengths. Finally, while most existing semantic mapping techniques rely on supervised learning of objects, we present an original incremental and unsupervised learning algorithm. Our technique was applied to the learning of multi-modal models of dynamic objects that are encountered as the robot navigate an indoor environment for an extended period of time, showing that it is able to produce consistent models of these objects without human intervention.

Résumé

Les robots entrent peu à peu dans nos maisons. Mais pour réaliser des tâches plus difficiles et intéressantes, ils doivent posséder une connaissance riche des différents objets et lieux présent dans une habitation particulière. Construire et utiliser une telle connaissance s'appelle la cartographie et navigation sémantique et nécessite de nombreuses capacités comme la cartographie, la localisation ou encore la reconnaissance d'objets. La plupart de celles-ci ont été longuement étudiées dans le passé mais souvent séparément et dans un contexte différent. Nous proposons une solution complète fonctionnant sur un robot capable d'effectuer progressivement et sans supervision la cartographie sémantique d'un lieu. Chaque aspect de cette solution est examiné depuis l'architecture logicielle jusqu'aux fonctionnalités requises par le robot. Certaines techniques déjà existantes ont été adaptées dans l'effort d'intégration et trois parties différentes de la solution sont des contributions originales. La première est l'utilisation de la télémétrie laser 2D pour effectuer la reconnaissance d'objets en utilisant des vues multiples des objets. De nombreuses solutions de cartographie sémantiques utilisent ce capteur pour la cartographie et l'évitement d'obstacle, mais nous en avons pleinement tiré profit en développant des techniques pour l'utiliser dans l'ensemble du processus de cartographie sémantique, montrant qu'il est efficace pour reconnaître jusqu'à 20 objets différents dans un scénario d'environnement intérieur. Deuxièmement, nous présentons une méthode de modélisation d'objets par graphes de vues. Dans notre effort pour étendre les techniques de sac-de-vues pour effectuer la modélisation multi-modale des objets, nous avons développé une formulation originale qui est plus adapté à la perception partielle des objets. Nous avons appliqué cette approche à la reconnaissance d'objets en utilisant à la fois un télémètre laser et une caméra RGB-D, montrant qu'elle est en mesure de tirer avantage des points forts des deux capteurs. Enfin, alors que la plupart des techniques sémantiques de cartographie existantes reposent sur l'apprentissage supervisé des objets, nous présentons un algorithme d'apprentissage incrémental et non supervisé original. Notre technique a été appliquée à l'apprentissage de modèles multi-modaux des objets dynamiques présents dans un environnement intérieur pour une période de temps prolongée, montrant qu'elle est capable de produire des modèles cohérents de ces objets sans intervention humaine.

Contents

Chapter 1

Introduction

1.1 Context: Semantic navigation

Figure 1.1: The robot Asimo bringing coffee to the table in a house. It illustrates how having a robot butler could look like.

We had long dreamed of building or having a robot butler. It would be fantastic to have one or more robots in our houses to take care of all the household chores. This kind of robots is referred to as domestic robots or personal service robots (fig.1.1).

According to the International Federation of Robotics, "in 2013, about 4 million service robots for personal and domestic use were sold, 28% more than in 2012. The value of sales increased to US$1.7 billion"[1]. So far the available personal robots are mostly floor cleaners, toys, social robots, surveillance or telepresence robots (fig.1.2). It is a really good start, but they are still fairly

[1]quote from: http://www.ifr.org/service-robots/statistics/

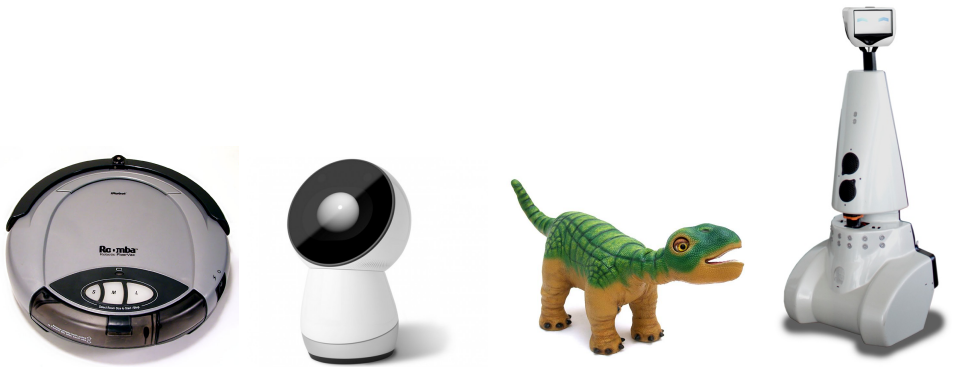


Figure 1.2: From left to right, Roomba a vacuum cleaner, Jibo a social robot, Pleo a toy robot and Jazz a telepresence robot.

limited and we are still far away from a robot butler.

Nevertheless, due to the progress made in this area and robotics in general, it has become more and more the concern of the scientific community, governments and industries. Each year, an increasing number of investments and fundings are made to support the development of service robots and growth of the industry. The main reason is that they could be a potential solution to many problems facing the modern society. For example, we imagine that a service robot could provide assistance and care for the elderly.

Such a service robot requires many capabilities and functionalities. Many of them have been addressed and solved with notable success since the beginning of robotics. From the navigation perspective, the robot capabilities include to be able to localize itself in the environment, to move smoothly while avoiding obstacles and to recognize objects and places. Those issues have resulted in many sub-domains addressing them such as SLAM, path planning, object recognition and machine learning. In numerous scenarios and applications, the techniques that have been developed in those domains have proven to be efficient. However, as new techniques are developed, new challenges appear. In general, we can say that the fundamental problem is to improve the robots' understanding of the environment. The more unconstrained and dynamic the environment is, the more difficult the task is. In order to progress in this direction, we will need richer and more abstract representation of the world, as well as more efficient solutions to process those representations.

A household is a very difficult environment to represent in details. It is challenging to have a robot capable of navigating safely and efficiently in such environment, or capable of performing orders like "go fetch my keys" or "clean the kitchen". It requires that the robot can navigate while knowing much about the different objects and places, and be able to reason about them. The robot needs to know what a particular object looks like, where it

is able to find it and so on.

Constructing a representation of the environment which holds the nature of objects and places as well as their positions is the purpose of semantic mapping. On the other hand, using such a representation to perform meaningful navigation tasks is the concern of semantic navigation. In many senses they are at the crossroad of classical navigation, object recognition and machine learning.

1.2 Target application

The question we asked ourselves at the beginning of this thesis was the following: what if I just bought a robot to help me in my house? What should I expect? Here is the scenario and the considerations we came up with.

Firstly, the robot has just been bought so it doesn't know anything about the owner or his house. What is the layout? What kind of objects and furnitures does he have? What are his habits? The robot needs to learn all of that. However the end user is not an expert, so training it should be as easy as possible. Also the robot cannot possibly know all of that from a previous training or learning because we don't all have the same objects or furnitures, and new ones are invented everyday. Finally, we could sometimes move furniture or even change the layout of the house and the robot would need to adapt smoothly without requiring to be extensively trained again. In other words, the robot needs advanced adaptation or learning capabilities. Those skills require to be unsupervised, incremental and real-time.

In this scenario, we can imagine that the robot would start by exploring its surroundings and immediately start building a rich model of the environment. It would then have to update and maintain this representation over time. Then the robot could ask the owner the name of the different places or objects it has identified, thus simplifying the teaching process. We could also imagine the robot fetching information from the Internet. The things that would typically be necessary to name are the places, like the kitchen or the bathroom, and the objects present in the house.

When a new object, a new information or a new situation is perceived by the robot while it is working, it should be able to gather that knowledge and incorporate it, and take a decision right away without having to stop its work. It would be better if the robot could adapt on the fly but it is possible to imagine that later on, when it is in standby, it would go over the newly acquired data and process it. Nevertheless, the user shouldn't have to train the robot for every new situation it encounters, and it shouldn't block the robot from doing its job.

Considering this, some of those problems already have efficient solutions, and some others have partial solutions. For instance, simultaneous

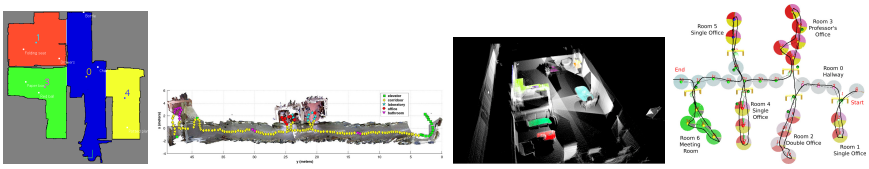


Figure 1.3: Different semantic map representations. From left to right, an occupancy grid where the room have been extracted and some object are localized, a colored 3D map where the positions of the robot have been annotated with the type of room it recognized, a 3D map where horizontal surface have been segmented, a topological map where the type of room and the positions of the doors have identified along the robot's trajectory.

localization and mapping (SLAM) works very well in a fairly static environment. Given an efficient SLAM, many techniques exists for safely navigating. However, the problem is that those techniques are based on representations containing little information. Most of the time, the only represented characteristic of a location is to be occupied or not. Thus one of the challenge is to have richer environment representation.

In conclusion, our research goal is to obtain such rich representation of the world. We want to be able to build it and maintain it over time, as much as possible without supervision, incrementally and in real-time.

1.3 Related work

Our goal is to build semantic maps automatically and with as less a priori knowledge as possible. We focused on system architecture, object recognition and learning. Here is a brief analysis of related researches in these three areas, aiming at positioning our contributions.

Semantic mapping is a rather new topic of research. A recent survey (Kostavelis and Gasteratos 2014) provides a detailed state of the art. The building of semantic maps is based on classical mapping and recognition techniques. So, it is natural that different combinations of popular techniques among those two subtopics were studied. We can therefore divide semantic mapping studies on whether they use metric maps (Iocchi and Pellegrini 2007; Biswas et al. 2002; Nüchter and Hertzberg 2008; Liu and Wichert 2013), topological ones (Ranganathan and Dellaert 2007), or both of them (Kostavelis and Gasteratos 2013; Pronobis, Sjöö, et al. 2010) (see fig.1.3). We can also divide those studies on whether they were focused on place recognition (Kostavelis and Gasteratos 2013), object (Ranganathan and Dellaert 2007) recognition or both (Aydemir et al. 2011). The main purpose of those studies being the finding of suitable representations to perform semantic navigation.

In the literature, the use of different sensors have been studied. Some

work only uses one modality, the most popular one being RGB-D sensors (Kostavelis and Gasteratos 2013), and other try to benefit from different modalities (Iocchi and Pellegrini 2007; Pronobis and Jensfelt 2011).

To recognize objects or places, popular sensors are the one giving rich information: color and RGB-D cameras, stereo vision, 2D and 3D laser scanner. For navigation purposes on the other hand, more choices are available: sonar range finder, 2D and 3D laser scanner, GPS, IMU, Wifi sensor, different radar technologies and of course cameras. The most popular ones are laser range finders and cameras.

For instance, in (Aydemir et al. 2011) they study the use of semantic maps and cues to perform active visual search. They use a chain-graph for representing object locations and relations. Using decision theory and heuristics on this model they show that they can perform efficient object search. The representation is adapted to reasoning purposes and can be built incrementally without supervision, but the appearances of rooms and objects categories are learned off-line with supervised techniques. In this study two different sensors are used: a 2D laser range finder for navigation purposes and a camera for room and object recognition.

In (Ranganathan and Dellaert 2007), they choose objects as the semantic unit. Visual and geometric features are extracted from a stereo camera, from which objects are recognized. Using those recognized objects, the places are then modeled and recognized. In other words, objects are inferred from visual features, and places are inferred from objects. The object models are learned off-line from a dataset.

The main limitation of those studies is that the robot's knowledge to recognize places or objects is principally acquired from a dataset in a supervised learning setup (Aydemir et al. 2011; Ranganathan and Dellaert 2007). It is a limitation because in spite of the powerful representations, the robot can only operate with previously learned objects or places, and the process of learning is rather laborious.

Some works were done on the acquisition of such knowledge with less supervision. For example, in (Mason and Marthi 2012) by comparing different runs of the robot and keeping track of the different planes in the environment they segment objects and discover change or novelty. The semantic map is represented as a 2D occupancy grid with 3D geometric models of the objects. So this work provides a way to perform unsupervised object discovery but doesn't deal with the modeling part. In (Biswas et al. 2002) they proposed an occupancy grid mapping algorithm to segment non-stationary objects and represent them as local occupancy grid maps. The work of (Endres et al. 2009) uses local shape descriptors with Latent Dirichlet Distribution on 3D range data for object discovery. Their method is unsupervised but not on-line and assumes knowledge of the number of objects in a scene.

However, in most of these works, the unsupervised learning is performed off-line and constrained to certain type of objects, places or scenarios. Never-

theless, incremental and unsupervised learning is the focus of many studies in the computer vision, machine learning and developmental robotics communities. The idea is to build rich knowledge step by step, during operation of the robot and without human intervention. Those studies are usually applied to object recognition with regard to manipulation by humanoid robots (Natalya Lyubova and Filliat 2012; Natalia Lyubova, Filliat, and Ivaldi 2013; Kemp and Edsinger 2006; Montesano et al. 2008). Still a lot of those studies could be applied to semantic navigation.

In conclusion, not much work has been done yet to apply incremental and unsupervised learning with multiple modalities to semantic mapping, which is an important practical goal (see section 1.2). We therefore focused our work on these aspects as detailed in the next section.

1.4 Contribution summary

We propose a complete solution to perform incremental and unsupervised semantic mapping working on a real robot. Each aspects of the solution are discussed: the software architecture and all the functionalities needed by the robot. Some already existing techniques have been derived in the effort of integration, others are used as is, but more importantly three different parts of the solution are original contributions:

1. **The use of 2D laser range finder to perform object recognition.** This type of sensor has some advantages over more popular ones for object recognition such as camera, RGB-D camera and 3D laser range finder. So far numerous semantic mapping solutions are using this sensor for mapping and obstacle avoidance, some work used it to characterize certain aspects of the environment, but we fully leveraged this sensor by developing techniques to use it in the entire process of semantic mapping.

2. **Introducing the graph-of-views modeling method.** In our effort to extend bag-of-views techniques to perform multi-modal modeling of objects we derived an original formulation which is more elegant and powerful than the former on several aspects.

3. **An original incremental and unsupervised learning algorithm.** While exploring existing clustering techniques we encountered several limitations. We came up with a new solution to the problem of unsupervised learning of object models.

Part of this work was published in three different publications:

- Jean-Christophe Baillie et al. (2011). "Software architecture for an exploration robot based on Urbi". In: *Proceedings of the 6th National Conference on Control Architectures of Robots*, pp. 1–12

- David Filliat et al. (2012). "RGBD object recognition and visual texture classification for indoor semantic mapping". In: *2012 IEEE Conference on Technologies for Practical Robot Applications, TePRA 2012*, pp. 127–132

- Guillaume Duceux and David Filliat (2014). "Unsupervised and online non-stationary obstacle discovery and modeling using a laser range finder". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 593–599.

1.5 Outline

This work is structured into seven chapters, including the introduction and is organized as follows.

Chapter 2 gives a thorough background on all the concepts and techniques needed to fully understand this work. For each item, a state of the art review is provided. The reading of this chapter is necessary for readers with limited knowledge about semantic mapping but can be skipped or be used as reference otherwise.

Chapter 3 will discuss the functional architecture of the robot. The choices made will be explained and discussed. Although it is not a crucial part of this work, it is an important aspect of the solution proposed and necessary to fully understand the other parts of the work. Our studies of the software and functional architecture of a robot for semantic mapping led to two publications: (J.-C. Baillie et al. 2011), (Filliat et al. 2012).

Chapter 4 will present our solution to perform object recognition using 2D laser range finder. Two new descriptors are presented as well as experimental results on recognition using them in bag-of-views models. Performing semantic mapping using only a 2D laser range finder have been studied and the work is presented in (Duceux and Filliat 2014).

Chapter 5 is about our original representation of object. It will discuss the related techniques and show why and how our method is interesting.

Chapter 6 will present the incremental and unsupervised learning part. It will explain the limitations we encountered using existing techniques and how our original solution can solve them.

Finally, chapter 7 will summarize our achievements and discuss the limitations and perspectives of this work.

Chapter 2

Background

This chapter gives a short state of the art in all the areas exploited in our work in order to provide context for the understanding of our architecture described in chapter 3. Reading of this chapter is useful for readers with limited knowledge about semantic mapping but can be skipped or be used as reference otherwise. More specific state of the art related to our contributions will be also presented in the following chapters.

2.1 Introduction

Semantic mapping and navigation have mainly one concern: integrating place and object recognition in the navigation process. The central question is how to represent the information so that it can be built, maintained and used efficiently. The navigation tasks that require rich knowledge are for example searching for a particular object, or adapting the behavior of the robot with regard to its context. So a good representation, or semantic map, is one that makes reasoning about places or objects easy. For instance, if the end-user asks the robot to fetch the car keys, the robot needs information such as the appearance of the object, where it has been seen last, where it usually is or what other objects can be found near the keys. Once the robot retrieves those information, it then needs to make an efficient plan to fetch the keys, which involves where to look first and how to get there. Having a smart autonomous robot therefore requires the ability to manipulate rich and heterogeneous information about the environment.

Most of the underlying computation needed to acquire and manipulate specific information about the robot and the environment have been already extensively studied. Perhaps the most important is the localization of the robot and the representation of its environment. This problem is known as simultaneous localization and mapping (SLAM) and will be reviewed briefly in the section 2.2. One solution to this problem provides an estimate of the location of the robot and a rudimentary representation of the environment

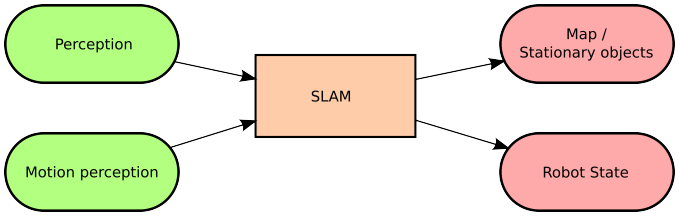


Figure 2.1: Diagram of the SLAM process. The inputs are proprioceptive sensors(such as wheels encoders or a inertial measurement unit) and exteroceptive sensors (such as a camera or a laser range finder). The outputs are a map of the environment and the location of the robot in the map.

suitable for basic navigation. On top of that, there is the problem of path planning and obstacle avoidance described in the section 2.3 which makes it possible to plan and adjust a safe trajectory towards a goal given a localization of the robot, a map of the environment and a target location. If the environment contains dynamic obstacles one might need a tracking module (section 2.4) to estimate displacement and position of those obstacles.

Those techniques have been studied for a long time and efficient solutions do exist, but they don't provide any information about the nature of places or objects. Those are the focus of object and place recognition described in section 2.6 which often involves the detection and segmentation (section 2.5) of the sensors output.

Semantic mapping aims to incorporate the nature of objects and places in the navigation techniques as discussed in the section 1.3. In section 2.7, we will discuss the informations stored in semantic mapping and how it is represented as well as how it is used for robot tasks.

Finally, how to bring all those functionalities together in one robot is difficult. It is the robot architecture problem which will be reviewed in section 2.8.

2.2 Environment representation, robot localization and mapping

Maybe the most fundamental problem in navigation is to know where the robot is. In order to do that, the robot needs to localize itself with regard to a certain representation of the world. But to build it the robot needs to keep track of its location. This chicken-and-egg problem is famously known has the "simultaneous localization and mapping (SLAM)" problem. It has been studied for a long time and several efficient solutions exist. The essential and classical ones are described with details in (Thrun, Burgard, and Fox 2005). The figure 2.1 illustrates this process. We will not have space for an in-depth review of SLAM, and we will therefore only detail a few approaches related

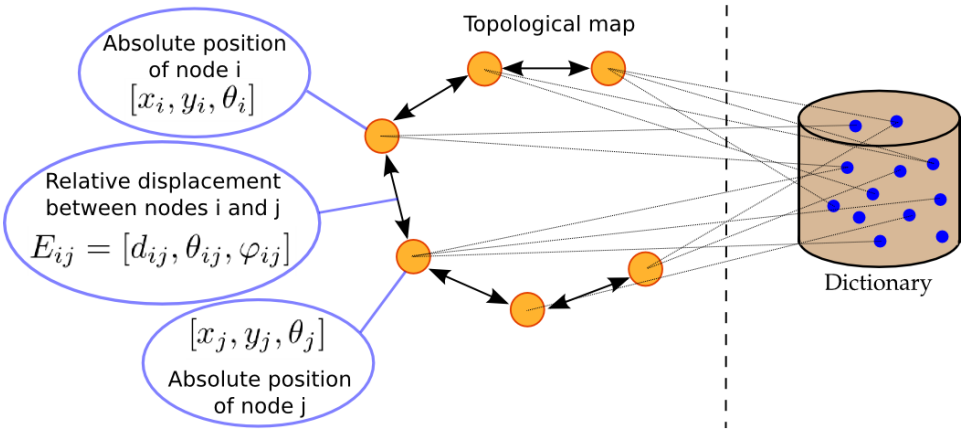


Figure 2.2: A topological map. Each node represents the position and appearance of a location. Each edge represents the relative displacement between nodes. In this work appearance is coded via bag-of-words method whose dictionary is represented on the right (Angeli et al. 2009).

to the map representation to introduce the simple metric representation that has been used in our work.

There are mainly two different paradigms when representing the environment: topological representations and metric representations. In the first case, the environment is represented by a graph of discrete locations. This can be very compact and distinctive, and it is easily embedded with semantics. However, this kind of representation requires a high level of abstraction of the raw data, and extracting those discrete locations reliably and robustly is difficult. Figure 2.2 shows an illustration of such a map from (Angeli et al. 2009). The metric representation on the other hand, represents the environment using raw data (or features with a low level of abstraction) in a metric space, which can result in a very precise representation. However the required volume of data often scales very badly with the size of the environment, and it is harder to embed with semantics. Recent works in this field have produced various hybrid combinations of those two representations (Tomatis, Nourbakhsh, and Siegwart 2003).

For this study, we used a metric representation called occupancy grid (Elfes 1989) which is common when using a 2D laser range finder. An occupancy grid (see figure 2.3) represents the environment as a grid of evenly spaced cells memorizing the presence or not of an obstacle. Generally a cell holds the probability of being occupied or not. Many SLAM algorithms use this representation. Popular techniques for localization and mapping using this representation are based on a particle filter (Hahnel et al. 2003) or expectation maximization as reported in (Burgard et al. 1999). In a recent popular SLAM (Kohlbrecher et al. 2011), they use an optimization algorithm based on a fast approximation of map gradients and a multi-resolution grid

Figure 2.3: An occupancy grid. White represents free cells, black are occupied cells and gray are unexplored. This map was produced by our perception pipeline using the HectorSlam algorithm (Kohlbrecher et al. 2011).

to perform fast localization. Overall, those techniques are efficient, robust and are used in commercial products. They are usually used with laser-based or sonar-based systems.

One limitation of this representation however is that it is based on the assumption that the environment is static. Some studies provide solutions to remove this limitation (Meyer-Delius, Beinhofer, and Burgard 2012; Mitsou and Tzafestas 2007; Arbuckle, Howard, and Mataric 2002). In the first one, their method represents both the occupancy grid and its changes in the corresponding area. The dynamics are characterized by the state transition probabilities of a Hidden Markov Process. Furthermore, they propose an on-line learning technique to estimate the model parameters from data. They have shown that their representation is more accurate and can be used to improve SLAM and path-planning. In the second one (Mitsou and Tzafestas 2007), for each cell of the occupancy grid, instead of only storing a probability, they store a time index which holds efficiently a history of the cells. They have shown that because they store all the information, it is possible to use this representation to retrieve the static and dynamic areas of the environment, as well as footprint of non-stationary objects. Finally in the last one (Arbuckle, Howard, and Mataric 2002) they consider occupancy over different time scales and show that it can be used as a tool for classifying motion in an area.

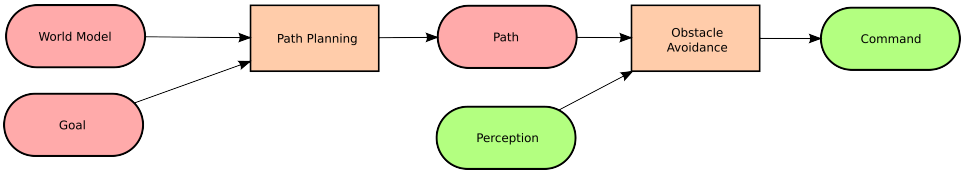


Figure 2.4: A possible diagram path planning and obstacles avoidance processes. Given a representation of the environment and a destination, the path planning process will compute a path that the robot will follow to reach its target (similarly to a Global Positioning System). However, usually this path doesn't account for dynamic obstacles. Thus another module called obstacle avoidance is used to follow the path while avoiding obstacles.

2.3 Obstacle avoidance and path planning

Path planning is the study of how to generate a series of actions so that a robot can go from a given configuration to a target one. Obstacle avoidance is concerned with how to ensure that a robot won't collide with an unforeseen obstacle. Those fields are closely related. The different techniques from these studies strongly depend on the representation of the environment and the available sensors. Figure 2.4 illustrates a commonly used diagram of such processes.

The navigation architecture is often divided into two layers (L. C. Wang, Yong, and Ang Jr 2002). The first one is a deliberative layer which compute a global plan to perform a certain task. This layer needs a complete and accurate knowledge of the world, and the computation involved might be time consuming. Because it is not possible to have a complete knowledge in advance, a reactive layer is there to prevent collisions in case of unforeseen obstacles. The reactive layer would control the robot until the deliberative layer takes into account the new information. Solutions from the obstacle avoidance field are used in the reactive layer, whereas path planning ones usually fall into the deliberative layer.

Reactive behaviors for robots have been studied early. There are two ideas behind the different techniques developed (Zohaib et al. 2013). The first one is to turn around the obstacle when encountered, until some criteria are met. This idea is the basis for the Bug algorithms family (Ng and Bräunl 2007). They tend to be time consuming and have different practical limitations. The second one is the potential field idea (Khatib 1986). The goal is modeled as generating an attractive force and the obstacles are generating repulsive ones. Applying the generated field to the robot makes it move toward the goal while avoiding obstacles. Those methods are less time consuming than the bug one but they get stuck in local minima. For instance U-shaped obstacles are often a problem for potential field based solutions.

Some more deliberative methods are also popular for obstacle avoidance.

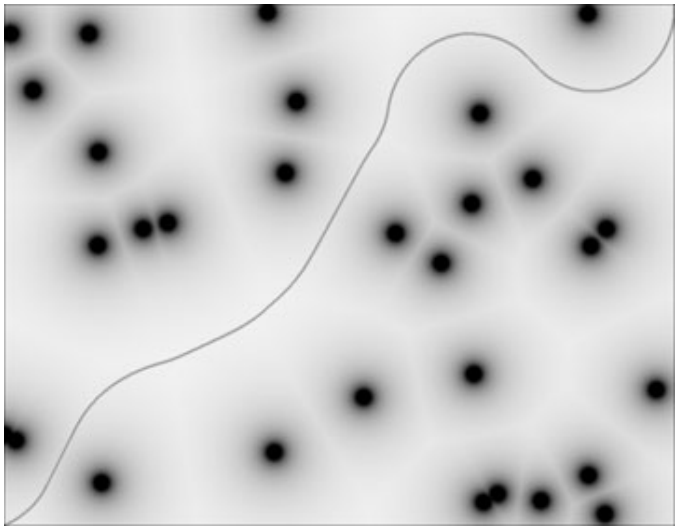


Figure 2.5: Field D* planning through a potential field of obstacles (Ferguson and Stentz 2007). Obstacles are represented in black. The curve line in the middle is the path produced by the algorithm.

Those have a short-term memory world model to plan better and avoid the shortcomings of reactive behaviors methods while being fast enough.

In the method called Vector Field Histogram (VFH) (Borenstein and Koren 1991), a histogram grid of fixed size centered around the robot position is maintained and updated with sensory information. This histogram grid which represents a short-term memory world model is processed in two steps to produce a velocity vector to command the robot. The first step consists in transforming the histogram grid into a polar histogram around the robot's momentary location. Each sector of this representation describe the density of obstacles in a certain direction. The second step compute a steering direction and a velocity taking into account the previously computed obstacle density the direction of the target.

Other popular approaches are derivatives of the Dynamic Window (Fox, Burgard, and Thrun 1997). In those, the kinematic constraints of the robot are taken into account by searching a solution into the velocity space of the robot. The dynamic window contains the admissible velocities achievable by the robot given the current acceleration and velocity. The admissible velocities are the one that allow the robot to come to a stop before collision with an obstacle. The command for the robot is selected by choosing the one that maximizes an objective function which takes into account the goal position and distances to obstacles. This method is improved in (Brock and Khatib 1999), where a small occupancy grid is maintained centered around the robot as in the VFH approach. This grid is used to compute a variant of Dijkstra's algorithm (Dijkstra 1959) which allows to favor direction towards

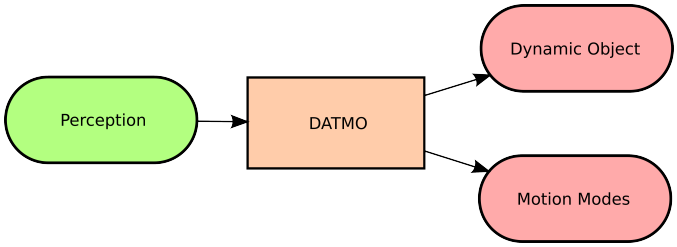


Figure 2.6: Diagram of the Detection And Tracking of Moving Objects (DATMO) process. Given an exteroceptive sensor, the DATMO system detects objects of interest and estimates their displacements.

the goal free of local-minimum.

Those solutions are not adapted to control the robot toward a far away goal, but they are efficient to avoid unforeseen obstacles while following a path. This path is given by a path planning solution.

For mobile robots in indoor environment, given a representation of the world, most of the time provided by a SLAM module, graph-based methods are popular to find a path from the robot to a destination. The study of those methods started with Dijkstra's algorithm (Dijkstra 1959) up to the current D*lite (Koenig and Likhachev 2002) and Field D* (Ferguson and Stentz 2007) through successive improvement. The figure 2.5 illustrates a path obtained from this method.

Dijkstra's algorithm is the simplest algorithm to compute the shortest path in a graph. It is usually sufficient when the search space is small, but is limited when the environment grows. The first improvement over Dijkstra's algorithm was to use a distance heuristic to improve the search. This is the A* algorithm (Hart, Nilsson, and Raphael 1968). The idea is that by searching the nodes closer to the goal first based on the heuristic used, the solution is found faster. Another improvement is to compute the graph from the destination to the robot position. This way it is faster to compute a new path when the robot moves or when newly perceived obstacles are added to the representation, because the path only needs to be updated locally. Finally, in the Field D* algorithm, interpolation is used to compute the traversal cost of cells so that the overall path is computed in the continuous space instead of the discretized one. This makes the resulting path smoother and more realistic for the robot to follow.

2.4 Moving object tracking

Dealing with dynamic objects is essential for mobile robots in almost every scenario. This problem is often referred to as Detection And Tracking of Moving Objects (DATMO) (Pancham, Tlale, and Bright 2011). In short, it

consists in estimating the position and the trajectory of detected dynamic objects from the perception stream. Figure 2.6 illustrates this process. This has been the major focus for video surveillance and automated car applications. Recent works (C.-C. Wang et al. 2007) are integrating this with SLAM (which is called SLAMMOT: Simultaneous Localisation And Mapping with Multiple Object Tracking) to improve overall performance. Here we will only discuss the tracking aspect since detection will be discussed later in section 2.5.

There are two problems in the tracking process. The first one is the data association problem: how to correctly associate the currently tracked object states with the incoming perception of multiple detections. The association has to account for newly perceived objects, those that are not visible anymore, those coming close together and those splitting apart. In the review (Pancham, Tlale, and Bright 2011) the popular data associations techniques reported for SLAMMOT application are the Global Nearest Neighborhood (GNN) (Blackman and Popoli 1999), the Joint Probabilistic Data Association Filter (JPDAF) (Fortmann, Bar-Shalom, and Scheffe 1983) and the Multiple Hypothesis Tracking (MHT) (Reid 1979). For a given number of tracks and current measurements, a set of possible associations is calculated for each track. This is referred to as gating. The GNN approach keeps only the best possible association, whereas the JPDAF combines all of the potential candidates for association to a track in a single statistically most probable one. Those methods only keep a single hypothesis about measurements received in the past. The MHT on the other hand consider multiple associations hypotheses over time.

The second problem is filtering to correctly estimate the trajectory and state of the objects. Popular techniques include Kalman Filters (KF) (Kalman 1960), Particle Filters (PF) (Gordon, Salmond, and Smith 1993) and Interacting Multiple Models (IMM) (Blom and Bar-Shalom 1988). The KF is a recursive method which keeps track of the estimated state of the system and its uncertainties through a Gaussian model. Those are updated using a state transition model and a new measurement. Particle filters use a sampling approach to estimate the posterior density of the system state by applying the Bayesian recursion equations. In their basic form they are susceptible to a mismatch between the state transition model and the real motion of the targets. A popular solution is the IMM which is an estimator using multiple filters with different transition models.

2.5 Object detection, segmentation and discovery

Detection of unknown objects in robotics context can be made using differences between maps or world models constructed at different times, assuming that the objects of interest will eventually move. The work in (Herbst et al.

2011) is based on 3D maps represented as point clouds. They compare two maps of the same environment by registering the 3D scenes together and use the map differences as possible objects. A similar approach is used by (Biswas et al. 2002) using 2D occupancy grid maps. They detect changes by a straightforward comparison of occupancy grids. Connected components of occupied cells in that difference are considered to be objects. Using the expectation maximization algorithm, they reconstruct the models of the different objects.

Others work directly with sensor data. For example, (Mason and Marthi 2012), (Filliat et al. 2012) and (Aydemir et al. 2011) use plane segmentation based detection on depth images to discover or search for objects. The objects are therefore supposed to stand out of flat surfaces such as the floor or a table.

Another approach is to compare the current observation to a world model. For instance, (Modayil and Kuipers 2004) uses a filtering method on range data localized by a SLAM technique to discover novelty. Once the laser reading has been registered, each endpoint either confirms or contradicts the occupancy grid. If a group of endpoints conflict with the occupancy grid, it is most likely caused by a non-stationary object.

Using those detections and segmentations, it is possible to perform object discovery which is reconstructing models of unknown objects from partial observations. In (Collet, Xiong, et al. 2013), the authors use a constraint similarity graph. The graph's nodes contain segmentations that represent candidate objects. To generate a candidate, the approach described in (Collet, Srinivasa, and Hebert 2011) is used. This method employs different cues from color and range information to form a well structured region based on a metric they developed. For each candidate they store color and geometric information as well as meta-data (position, time, is it planar or not, seen on a table or not, etc...). The edges of the graph are the similarities between nodes computed from these various data. By clustering nodes in the graph they can reconstruct the object models.

In computer vision, as the task of discovering objects without prior knowledge is difficult, statistical methods are often used over large datasets to find reoccurring patterns. A metric was proposed in (Tuytelaars et al. 2010), to compare different techniques. It was then used with popular methods applied to two different public datasets. The datasets are composed of images of objects from different categories. The images from the first dataset contain only one object, while there can be multiple objects per image in the second set. The images are represented as bag-of-words (Csurka et al. 2004; Y. Zhang, Jin, and Zhou 2010). Two categories of methods were tested: latent variable models and spectral clustering, showing that spectral clustering performs better when a single object exist in each image, but that latent variable models are better suited when objects appear in multiple instances. In (Russell et al. 2006), they use a technique to obtain multiple

segmentations from each image. Those segments are modeled as bag-of-words and categories are discovered using Probabilistic Latent Semantic Analysis and Latent Dirichlet Allocation following methods from the text analysis community. Once the categories are discovered the segments are filtered to extract the good ones. Thus they can jointly discover object categories and segment them in the corpus. Spectral clustering is also used in (Fu et al. 2008). In this work they represent an image as a set of unordered features. They produce a graph where each image is a node and where edges represent the partial matching between images. Using different refining methods on the graph they can extract meaningful set of features representing objects across images.

2.6 Object modeling and recognition

There are mainly three ways to model and recognize objects using range data, as will be done in our thesis.

The first way is to use registration or scan matching to generate and recognize geometrical models. In (Herbst et al. 2011) and (Modayil and Kuipers 2004), they align surfaces belonging to an object in 3D and 2D range data respectively. By aligning those surfaces, they obtain a model of the object consisting of a point cloud as it would be seen by the sensor if it could see the entire object. Those approaches are very susceptible to noise in sensor data, and are not well suited for modeling objects with changing shapes like people.

The second approach is to extract invariant local features from observations and to differentiate objects based on the set of features they possess. The work of (Endres et al. 2009) uses local shape descriptors with Latent Dirichlet Distribution on 3D range data. Their method is unsupervised but not on-line and assumes knowledge of the number of objects in a scene. An object is represented in this case as a distribution of local surface shapes. This kind of object model is more robust to noise in sensor data and change in object appearance. They are usually faster and more efficient than geometrical models to learn and to recognize. The use of local features has also been widely studied by the computer vision community as will be reviewed at the end of this section.

The third way is to create multi-view object models by regrouping the views of an object from different viewpoints or different times. This approach is used in (Natalya Lyubova and Filliat 2012), based on vision and takes advantage of object manipulation by a robot to gather different views of the objects. Using tracking techniques to put together different views have also been used in (Modayil and Kuipers 2004) using laser scans. Views are usually encoded using a descriptor or signature. Many of these exist in vision or 3D, but far fewer for 2D range data. Nevertheless, (Tipaldi and Arras 2010)

shows that it is possible to use descriptors with laser range finder, applied to place recognition in their case.

Among the numerous approaches existing in computer vision that we will not be able to review here, extracting local features from images as a first step to object recognition is very popular (Campbell and Flynn 2001). Most of the time, it requires the encoding of certain properties of the appearance into descriptors, and the clustering of those properties. SIFT (Lowe 2004) and SURF (Bay, Tuytelaars, and Van Gool 2006) are frequently used to describe points of interest. Using local features, a popular way to represent objects is as a set of local features. In particular the Bag-of-Word representation (Csurka et al. 2004), which as been developed originally in the text analysis community (Joachims 1998), is frequently used or derived. Recently, due to the apparition of cheap RGB-D sensors, this approach has gained in popularity and different descriptors and keypoint detectors have been developed to work on this kind of data (Filipe and Alexandre 2014; Rachmawati, Suwardi, and Khodra 2014).

2.7 Semantic mapping

Semantic Mapping studies are concerned with producing a world model not only representing the appearance and spatial layout of the environment, but also with places and objects identified, categorized and localized. The aim is to add semantic knowledge to the classical navigation techniques in order to overcome their shortcomings. Semantic mapping offers several advantages such as simplifying information sharing about the environment with a human operator. It also helps modeling possible interaction with objects and their dynamics which is interesting because most computation involved in navigation could be improved with such knowledge. For instance, knowing about the floor properties can help having better estimate of the robot displacement, or knowing about the location of different visible objects can help with the localization. Semantic mapping is a promising direction towards improving environment representations and robots autonomy.

Most of the proposed semantic map models in the literature are layered representations containing metric, topological and conceptual information (Pronobis and Jensfelt 2012; Galindo, Saffiotti, et al. 2005; Ranganathan and Dellaert 2007; Vasudevan and Siegwart 2008). One of the first model proposed by (Galindo, Saffiotti, et al. 2005) is a double parallel hierarchical structure. The first structure is a spatial representation where the lower level represents sensory readings, the second level contains a topological map of the environment and the third level is a single node representing the whole environment. The second structure represents the different concepts of place and object and their relationship. The two structures are linked by anchoring of concepts to spatial locations. One of the most complete

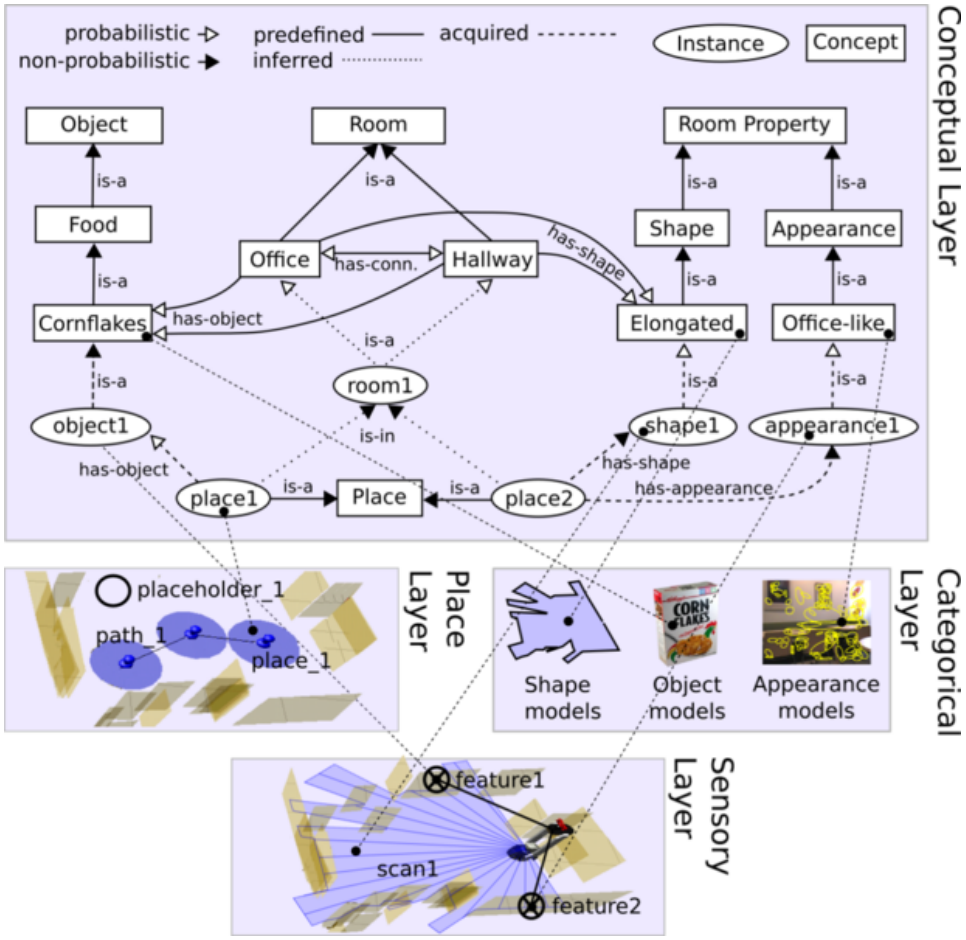


Figure 2.7: The layered structure of the semantic map representation proposed in (Pronobis and Jensfelt 2012). The conceptual layer comprises knowledge about concepts (rectangles), relations between those concepts and instances of spatial entities (ellipses). The categorical layer contains every information necessary to recognize the different places and objects present in the environment. The sensory layer contains an occupancy grid of the environment. Finally, the place layer contains a topological map which links the concepts to the different geometric locations.

model for indoor environments is presented in (Pronobis and Jensfelt 2012). It is a spatial representation with four layers (see figure 2.7). The first one called sensory layer contains a metric map of the environment and represents the lowest level of abstraction. Above, the place layer is a topological map containing locations, paths and placeholders. The categorical layer is the learned knowledge of the robot. It contains objects and places appearances as well as models for recognition purposes. Finally the conceptual layer represents the relationship between objects, places and their properties.

Another approach is to annotate metric maps with semantic information. In (Nüchter and Hertzberg 2008) they proposed to annotate point clouds produced with a 3D laser scanner. The solution contains three steps: 6D SLAM, scene interpretation which extracts the ground, ceiling, walls and doors from the point cloud and finally an object classifier to label the remaining parts of the point cloud. The classifier is previously trained and uses the ICP technique against a known 3D geometrical model to recognize the objects. The information about walls and the ground are used to improve the quality of the model by enforcing their perpendicularity.

Both approaches are used in the semantic map structure proposed in (Drouilly, Rives, and Morisset 2014). Their map called Metric-Topological-Semantic map (MTS-map) consists of 3-layered local sub-maps globally connected to each other into different graph structure. The bottom layer of the local sub-maps is an RGBD spherical view of the environment acquired with a multi-cameras system. This layer represents the sensor view and provides both metric and color information. The second layer is extracted from the spherical views by labeling it pixel wise using a supervised classification method. The top layer is a semantic-graph in which nodes are the different consistent regions in the labeled image and edges represent the connection between these regions. Those semantic sub-maps are used in three different structure. Firstly, they are connected into a graph structure with edges representing relative pose estimation between the sub-maps which forms an hybrid metric-topological map of the environment. Secondly, they are clustered according to their content similarity into a tree structure. With this structure they can perform efficient relocation using semantic information at a coarse level and sensor measurements at a fine level. Finally, they use a conceptual graph to characterize non-spatial relations between categories.

One of the concern about semantic representation is how it can be used to perform abstract tasks such as searching for objects. A popular approach is to plan tasks using logical reasoning about objects, places and their spatial relationship (Kunze et al. 2012; Galindo, Fernandez-Madrigal, et al. 2008; Aydemir et al. 2011). In (Kunze et al. 2012) they propose a decision-theoretic approach for fetch-and-delivery tasks. To do so, they extend the semantic representation of (Tenorth et al. 2010) to handle large-scale environment and multi-level building. The basic idea is to select the search location that maximizes an expected utility function which is determined by the probability

of success, travel costs and task context. The possible search location are obtained through logic calculus on the conceptual layer of their world model. They demonstrate their method with a robot system that successfully fetch a sandwich within a multi-level building.

Another concern is to use semantic information to outperform existing techniques solely based on metric information. Traditional path planning optimize the path length and obstacle clearance. In (Drouilly, Rives, and Morisset 2015), they use the MTS-map to perform path planning using higher-level constraints. For instance, by taking into account the visibility of the landmarks along the computed routes, they can compute a path which is short in length, while ensuring that the robot will not get lost. Furthermore, they propose a way to extract a high-level description of a computed route by using semantic information. Finally, in (Drouilly, Papadakis, et al. 2014), they propose to improve the mapping process by extrapolating the environment based on extracted semantic information.

2.8 Robot architecture

Building a robot and programming it to perform tasks as complex as semantic mapping and navigation is not an easy problem. There are many considerations to take into account. At the very beginning, there is the robot's purpose: what do we want to accomplish with the robot. Then comes the hardware: what kind of sensors or actuators can we use to accomplish such goals, what processing power do we require. Once a purpose for the robot is set, and the hardware has been decided, the robot needs to be programmed. There are two sides of the robot software. Firstly, there is the programming of the different functionalities and computations required by the system, which are referred as the software components. Secondly, there is the software architecture which is how the components are arranged and how they interact together. All of this forms the robot architecture.

In the last decades, a lot of work was dedicated to the design of robot architectures. There are mainly three paradigms for these architectures (Nakhaeinia et al. 2011): deliberative, reactive and hybrid. In the deliberative one, also called hierarchical, the robot gathers the data from its sensors into a global world model and plans at each step the next action. The advantage of this type of system is that it is possible to plan complicated tasks in a top-down and long term fashion. The main disadvantages are that it's usually heavy on computation and obtaining an accurate model of the world is difficult. In the reactive paradigm, the robot has multiple couples of sense-action called behaviors which are independent of each other. Given a certain situation, the robot will do a combination of behaviors following a set of rules. For example, in the subsumption architecture (Brooks et al. 1986) the behaviors are organized in layers with priorities depending on the situation.

The main advantage is that it's easier to obtain a fast reaction from the robot because there is no need to process the newly sensed information into a world model. Instead each component gets a small piece of more grounded information and sends an action to the robot. The disadvantage is that sometimes complex planning is required and designing a robot with this type of architecture is then very difficult. Naturally, most of the architectures fall into the third category, the hybrid paradigm, which is a combination of the two first and exists in many variants.

For instance, the work of (Rosenblatt 1997) presents an architecture where behaviors of the robot are distributed and all output votes for the possible commands. One of them is selected through an arbiter. The behaviors can be either of deliberative or reactive nature. This has the advantage of limiting the need for sensor fusion or a monolithic world model. By distributing behaviors and only coupling them in the command space, the architecture can be easily updated and maintained. By distributing behaviors through computers or independent processes, the whole system is more efficient and reactive. However, in their solution, commands issued to the arbiter have no semantic value which is limiting. Recent work on architecture tend to perform command or action selection using a task planning functionality (Scheutz et al. 2007; Hawes, Wyatt, and Sloman 2009). Another popular approach to hybrid architecture is using a hierarchy of sub-architectures. In (Connell 1992), they propose an architecture with three different layers called servo, subsumption and symbolic. The first layer servo is performing the initial sensor processing and the actuators control. The second layer is a subsumption architecture receiving detected situations form the servo layer and issuing commands to it. They are both reactive sub-architectures. The subsumption layer communicates failures and achievements to the symbolic layer through events which change the parametrization of the second layer to adapt the behavior of the robot. This symbolic layer is a deliberative one. The advantage of this solution is that it preserves the reactivity of the system with low level reactive sub-systems and allows longterm and complex reasoning with a deliberative layer. However, in this form the overall architecture is very constrained.

Nowadays, as robots become more complex and computers more powerful, the robot software architectures are often data-flows of concurrent and distributed modules, with less considerations for which paradigm they fall in.

In (Scheutz et al. 2007) a generic architecture called DIARC is presented and implemented on a real system. The system contains concurrent software components divided in three layers: perceptual, central and action processing. The perceptual layer is able to call for immediate action through high priority actions ensuring the reactivity of the system when facing danger. Those layers are further divided in 6 different modules running across three computers. In their work, the action selection is done using priorities based on a cost-benefit analysis and influenced by the perception of the environment. They stress the

need to have mechanisms to recover from failure in such a complex system, which is done in their work by monitoring the progress of on-going tasks and modulating the priorities of the different tasks.

To facilitate the implementation of such architecture, the robotic community has seen the emergence of efficient robot middleware. They simplify the integration and maintenance of the robot by taking care of inter-components communication and by providing tools to develop, debug, visualize and other helpful utilities. For instance in this work both the Urbi (J.-c. Baillie et al. 2008) and ROS (Quigley et al. 2009) framework were used. Nevertheless, theoretical and practical tools to efficiently engineer and design robots (Ramaswamy, Monsuez, and Tapus 2014) are still needed. Research on the matter is recently gaining in importance.

The next chapter will present the hybrid architecture developed during our work.

2.9 Conclusion

Performing semantic mapping is a complex task which involves computing different types of representations in order to capture a rich knowledge of the environment. In this chapter we have seen the different techniques and representations commonly found in a semantic mapping solution, and we briefly discussed the issue of integrating such techniques in a robot architecture. After a thorough study of semantic mapping's state of the art, we found mainly three aspects of existing techniques which could be improved. The solutions we came up with are tightly interconnected to solving the three issues we identified.

Firstly, most of the studies use a camera to recognize the nature of objects and places in the environment (Kostavelis and Gasteratos 2014). Recognizing objects with a camera has been studied to a large extent by the computer vision community and state of the art techniques can achieve high performances. However, in a domestic robot setup, several limitations remain. Cameras often have a limited field of view which can actually be used to recognize objects. Moreover, the information they provide is very rich and therefore hard to process in a timely manner, leading to processing images that are taken from far away position when moving the robot. As a result, a robot mostly takes arbitrary images of the environment to perceive it, and in the majority of those images, objects are ill positioned in the camera field of view which results in poor recognition results. To cope with that, a solution is to ensure that the robot takes good pictures by moving it in front of interesting objects. Popular solutions to that problem are visual saliency techniques (Harel, Koch, and Perona 2006). However, since many studies - including us initially - use a 2D horizontal laser range finder to perform navigation and geometric mapping of the environment, we decided

to solve this problem by using a laser range finder in order to enhance the robot's perception of objects. We were inspired by studies which show that 2D shape of objects are recognizable and that classical techniques can be applied to a laser range finder (Modayil and Kuipers 2004; Tipaldi and Arras 2010). In chapter 3, we will present how our laser range finder is used to detect objects and how it is integrated in our semantic mapping solution. In the chapter 4, we study how we achieved object recognition using this sensor.

Secondly, many studies use only one property or modality to recognize objects. However, an object can be described in terms of many properties such as its shape, color, texture, functionality and so on. A solution to recognize objects which would use different properties should achieve higher performance and robustness. Furthermore, we have discussed how a laser range finder could be used to improve a camera based object recognition system. We therefore need a solution that can use both sensors information to represent an object. Last but not least, as we will discuss in the next paragraph, we want to incrementally learn models of objects. To solve those three issues, we developed an original object representation which is detailed in the chapter 5. This work was inspired by (Mei, Sibley, and Newman 2010) which extends the bag-of-words technique in a meaningful way. The representation we came up is similar in nature but used in a very different way. We will show how we can use our representation to represent and recognize objects based on various sensors and modalities, and how it can be used to incrementally learn objects' models.

Finally, most of existing semantic mapping solutions use supervised machine learning techniques to learn and recognize objects. We believe for several reasons that it is not a pragmatic solution for domestic robots. Households have a limited number of objects but they come in many flavors. It is therefore impossible to store a dataset of objects' models on a robot which would cover every possible object it could encounter. Besides, if a robot is to stay in a particular environment, the most obvious and efficient solution is for the robot to store only representations of objects present in this particular environment. We can imagine the robot downloading from a cloud dataset the models it needs (Waibel et al. 2011). This is probably a good solution since most robot use cameras to recognize objects. However, sensors and households come in many different flavors and evolve over time. Maintaining such a database would be a great undertaking. On the other hand, adapting to an environment might be achieved through unsupervised learning techniques. If such learning were to be successful, robots would acquire models with their own sensors and only useful knowledge. This could lead to more reliability and efficiency in the robot's perception of its environment. We believe both solutions are complementary. However, little study have been made yet on using unsupervised techniques in a semantic mapping context. We pursued this idea and developed an original unsupervised learning technique which is detailed in the chapter 6.

Chapter 3

Robot architecture for semantic navigation

In this chapter, we will describe the two robot architectures that we developed during our work. While we do not propose theoretical contributions in the development of these architectures, we want to emphasize the importance of these architectures and the large amount of work represented by their development. We also think that some of the design choice made could be of interest for future developments.

3.1 Introduction

As mentioned in the introduction, we aim at making a robot able to fulfill the scenario presented in section 1.2. In order to do so, the robot must possess many capabilities, some of which have already been the focus of numerous studies with notable success. From the navigation perspective those capabilities are dealing with moving objects, appearance variations and other dynamic changes of the environment. Some of those needs to be solved using machine learning, which might add the difficulty of having to learn with minimum supervision from a non-expert human. Putting the different solutions together means integrating them in a coherent architecture. We should also consider that in practice we would need solutions that can be easily maintained and updated.

During our thesis, we elaborated two robot architectures for the integration of these functionalities. The first architecture was developed in order to participate to the Défi CAROTTE (J.-C. Baillie et al. 2011; Filliat et al. 2012). The goal of this challenge was to have a robot autonomously explore an unknown environment and produce a semantic map containing rich information usable by a human. The second architecture is an adaptation of it to support the contributions this work, in particular the capability to discover and model dynamic objects autonomously as described in the next chapters.

Figure 3.1: A photo of the robot used during Défi CAROTTE.

In this chapter, these two architectures will be presented and all the functionalities will be introduced at least briefly, while some of them which were the subject of our main contributions will be detailed in the following chapters.

3.2 Hardware

We participated to a challenge called "Défi CAROTTE" which stand for "Cartographie par Robot d'un Territoire" organized by the French Defense Procurement Agency (DGA) and financed by the french National Research Agency (ANR). The goal of the challenge was to explore an unknown environment and produce a semantic map of it. The map had to contain information about the different rooms, floors, walls and objects present in it. The environment included difficult obstacles like mirrors, windows and gravels.

The robot used during this work was a Pioneer 3 dx mounted with a camera looking down used for the floor classification, a RGB-D camera (Kinect) for object recognition, walls classification and obstacle avoidance, a sonar belt for obstacle avoidance (especially windows and mirrors) and a 2D laser scanner for the SLAM (see fig. 3.1). The laser scanner used is an hokuyo utm-30lx. It has a precision of 0.03m from 0.1m to 10m, an angular resolution of 0.25 degrees and an angle of view of 270 degrees. The kinect

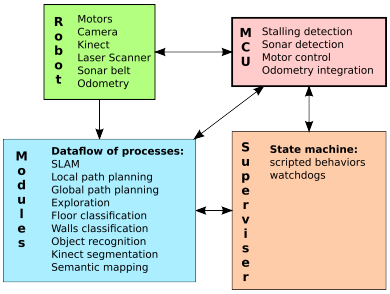


Figure 3.2: An illustration of the software architecture for the Défi CAROTTE.

is a RGB-D camera with a 57 degree horizontal field of view, a 43 degree vertical field of view and range sensing from 1.2m to 3.5m. After 3.5m the accuracy of the depth sensor degrades quickly. It has a resolution of 640*480 pixels with 30fps frame rate.

3.3 Participation to the Defi CAROTTE

3.3.1 Software Architecture

For the Défi CAROTTE, the task at hand was the exploration of an unknown environment in order to build a semantic map usable by a human. A software architecture was realized (Fig. 3.2) for this purpose, following the hybrid architecture paradigm (see section 2.8). This architecture was executed on a laptop computer and the low-level control of the actuators and emergency stop were performed by a dedicated micro-controller.

The many required functionalities were implemented as independent processes or modules and organized in a data-flow manner. For instance, the SLAM module would receive laser scanner readings and odometry and provide with a position and an occupancy grid, which would be then be received by other modules such as the global path planning. This is a paradigm popular in the robotic community as it as many advantages. However, since modules are designed as black boxes to perform specific tasks, it is not well suited to handle failures or to encode the strategies and behaviors of the robot because it often requires knowledge of the whole architecture. Thus, the software architecture contains a supervisor process.

The first assumption made is that all modules may fail under certain circumstances. For example, the global path-planner would fail if the target destination given as input is unreachable from the robot position, or if there is an error in the robot position or the map. Because all modules may fail, the supervisor is connected to all of them. A module would send event signal upon detected error. The behaviors responding to certain events are implemented as a state-machine with a set of goal-condition-action rules. For

instance one of the rule is: if the mission is in progress and the global path planning failed, then try to find another target in a different place.

Secondly, some errors may not be detected by the modules, such as disconnection of one of them, or a run-time error which would cause a process to crash. Thus, the supervisor is also monitoring the states of the module to detect such event. This is implemented as a set of watchdogs which are independent co-routines monitoring the modules and other states of the robot by checking periodically. For example a watchdog would check that a module is responding every second and if its not, restart it and trigger the appropriate behavior. Another watchdog would check the time remaining for the mission, and change the goal of the robot from exploring to going home when the time is up.

For safety reason, a special care was attributed to the module interacting with the micro-controller unit (MCU). The MCU contains the servoing of the actuators, monitoring of the current drawn to detect stalls, the battery status and the sonar belt readings. A stall situation could happen if the robot tried to cross an unseen obstacle. It is important for safety that danger can be detected and motors be stopped at once. So the module interacting with the MCU would discard new command under certain condition (stall or close obstacle in the direction asked) and report it to the supervisor along with the underlying cause. The behavior retained in this case was to make the robot backtrack a little and add the current area to the obstacles list.

The modules retained for the Défi CAROTTE were a metric SLAM, a local path planning, a module for global path planning and exploration target computation, floor classification, wall classification, kinect based object segmentation, object recognition, and the semantic map generation. The SLAM used was a 3rd party software call Karto from SRI international based on laserscan matching. The floor and walls classification was realized using the random forest classifier proposed in (Maree et al. 2005). The semantic map generation used a line detector to segment the occupancy grid into rooms and produce a topological map and a Kalman filter to estimate the position of the different objects. It also integrates the recognitions of wall and ground textures. The other modules, in which we were more specifically involved or which are used in our contributions, will be detailed in the following sections.

Concerning the behaviors of the robot during a mission, five global states or phase were retained: *initialization, idle, exploration, going-home* and *result generation*. During *initialization*, all sensors and actuators are started and software modules launched. Upon completion, the robot is in the *idle* state, waiting for the start signal which change it to the *exploration* state. During exploration, the general strategy adopted was to perform a loop which decide for a goal and try to reach it while detecting objects and mapping the environment. In case of problems reaching the goal, we stopped the robot when an unexpected obstacle was detected, backtrack a little if necessary, incorporate the obstacle in the world model, and try to carry on

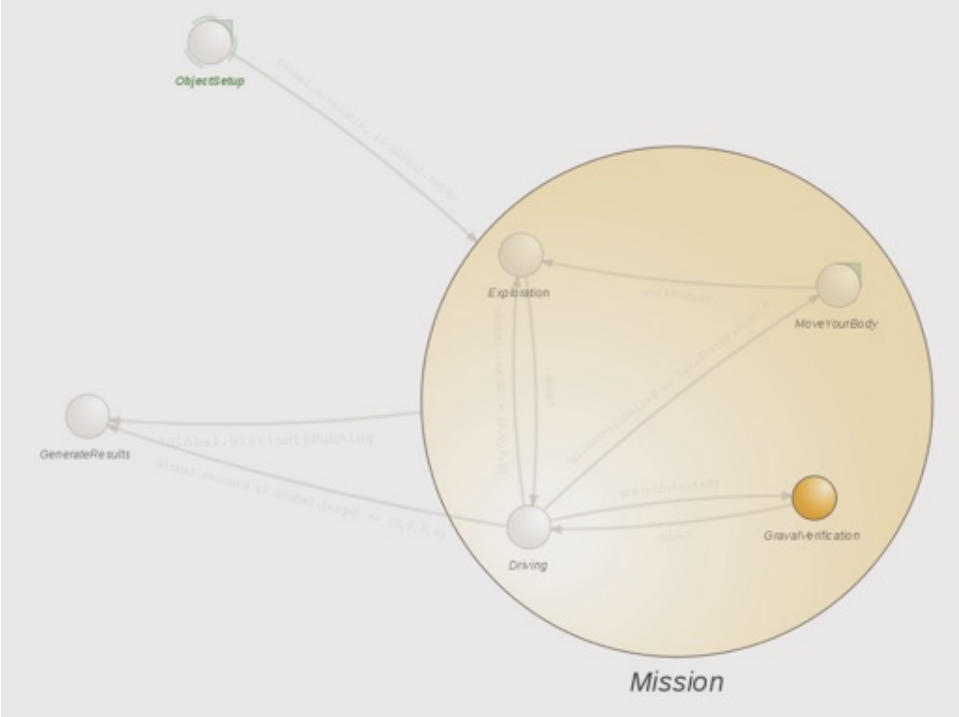


Figure 3.3: Example of an hierarchical state machine controlling our robot in the Gostai Studio software. The nodes displayed in yellow are the ones currently activated.

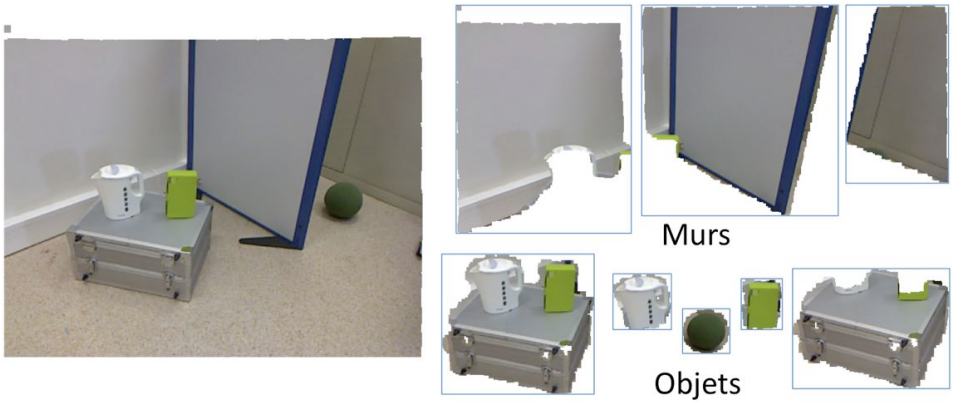


Figure 3.4: Exemple of kinect detection and segmentation.

by calculating a new exploration target and the path to reach it. When the exploration is completed, the robot goes to the *going-home* state. If the robot encounters difficulty to go back to the starting point, then it would try to get as close as possible until the situation was unlocked. When the starting point is reached, or if the robot remains blocked, the *result generation* is executed where the robot generates all the semantic information required by the competition. The software architecture was implemented using the Urbi framework (J.-c. Baillie et al. 2008). The modules are written in C++ and the supervisor in UrbiScript through a graphical user interface (Gostai Studio) designed to generate finite state machines (see fig. 3.3).

3.3.2 RGB-D object segmentation and recognition

The segmentation used in this work is the one described in (Caron, Filliat, and Gepperth 2014) and (Dubois et al. 2013). When a kinect reading comes in, the floor is removed by finding the biggest horizontal plane in the RGBD image. Then the perpendicular planes are removed if they meet some criterion in order to remove walls. Then by successive transformation and parallel plane detection the kinect reading is segmented into clusters of points (see figure 3.4). A cluster of points represent an object view.

The object recognition was realized with a neural-network based machine learning technique as proposed in (Caron, Song, et al. 2014). The neural-network was trained on a dataset composed of several segmented views of the different objects. Its inputs is a set of different descriptors extracted from an object segmentation. The descriptors used are a bag-of-words of SIFT keypoints (Lowe 2004), a color histogram called TBGR and a 3D shape descriptor called surflet (see details in (Caron, Song, et al. 2014)). The surflet is a concatenation of histograms of angles and distances considering pair of points and normal vectors in a point cloud. It therefore encodes the 3D

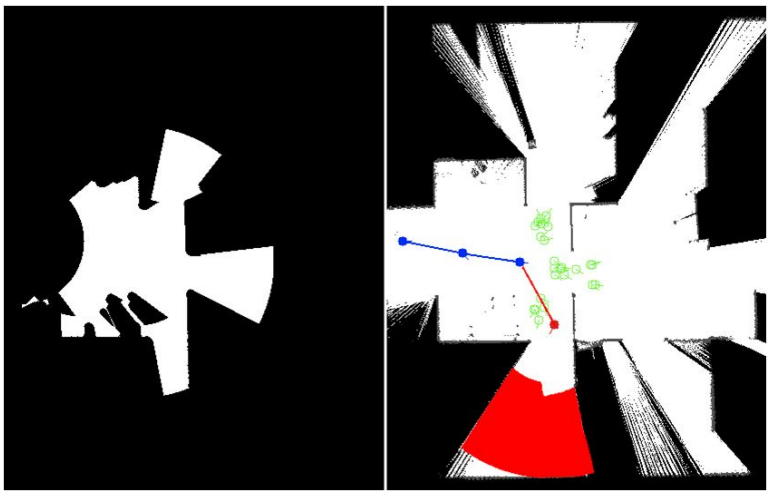


Figure 3.5: Example of an exploration target computation. The image on the left represents the area already observed by the kinect sensor. The image on the right shows the candidate targets in green, the former positions and path of the robot in blue, the selected target and the potentially discovered region in red.

shape of the object view. The TBGR which stands for Transformed Blue Green Red, is an histogram of the pixel value where each channel of the image is separately normalized to zero mean and unit variance. It therefore encodes the contrast in the three image channels, thus reducing the color information to enhance robustness. We will use the surflet and the TBGR features for our object recognition presented in chapter 5.

3.3.3 Exploration target and multi-modal path planning

For an exploration task, the robot needs to calculate a relevant target destination to explore the environment efficiently. Targets are relevant if they allow the robot to discover a lot of space, find new objects and if they do not make the robot waste time by going back and forth. This time can be estimated by the global path planner using the current map of the environment, but this path has little chance to be precisely executed because of unknown or dynamic obstacles. For this reason, we separated global path planning using the global map produced using the laser scanner and a local planner that take additional sensors (the kinect and sonars) into account for executing this path. This is important because, for example, mirrors and windows are difficult to detect using light based sensor but are easily detected using a sonar. Upon detection of such obstacles, they are integrated in the global map so as to plan a new global path around them.

In (Filliat et al. 2012) the global path and exploration targets are com-

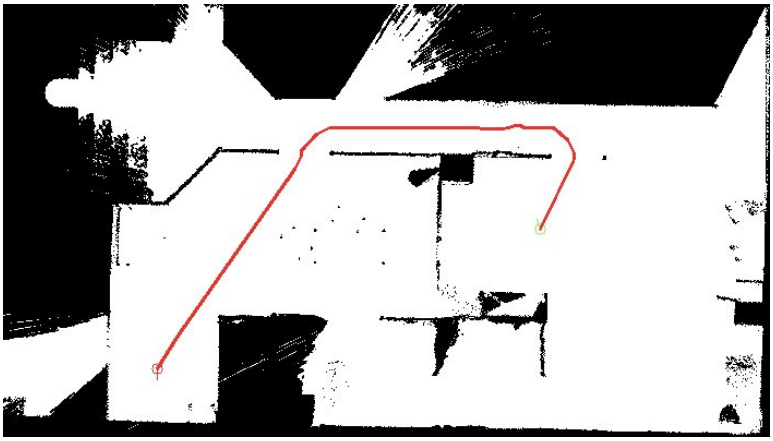


Figure 3.6: Example of a smoothed global path. The white represents traversable cells and black forbidden ones. The computed path is represented by the red line.

puted by the same software module. To do so, the detections from the different sensors are projected into a single occupancy grid, and the Dijkstra algorithm is used to compute the reachable areas by the robot. To calculate exploration targets, random positions are drawn from all the accessible positions and the best one according to a function cost is kept (see figure 3.5 and (Jebari, Bazeille, and Filliat 2011) for details). The function cost is a score based on how much space the robot might uncover from the target and how far it is from the robot. The first criteria favors positions from which the robot would have clear view with the kinect of an unseen place and discover new areas using its laser sensor. However, this alone is not enough because the robot might tend to go back and forth between too far apart areas of the environment, which is why the second criteria enforce the robot to explore close areas first. This way the robot will prefer to explore an entire area in an efficient way before moving on to the next one.

Once a target as been selected, a global path is computed to reach the goal using the same Dijkstra's algorithm output. The classical version of this algorithm compute the shortest path between two positions which tend to make the path go near obstacles when circling them. To reduce this limitation a common solution is to add a cost to cells in the graph near obstacles. The cost is correlated to how close the path is to the obstacle. Because the path is computed in a discretized representation of space, the result is often not smooth. Since the robot has a certain dynamic, following such a path would result in jitters. To solve this, a final step to the computation was added to render the path smoother. To filter the path, going from the robot to the goal, the path is iterated through to find parts of the path that can be replaced by a straight line. For this, we use the Dijkstra's graph cost

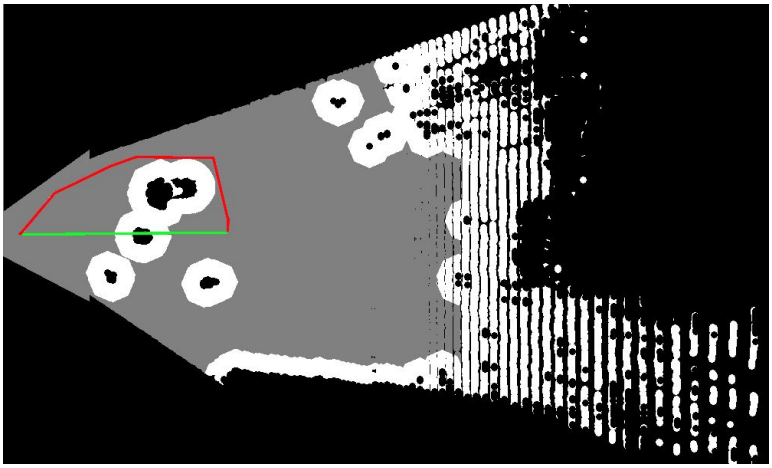


Figure 3.7: Example of a local path computation. The image represents a kinect frame projected on the floor plane. The black color are cells where there is no information or an obstacle. The white cells are the ones too close to an obstacle for the robot to go. The gray cells are traversable cells. The portion of the global path present in the field of view is drawn as a green line. In this example, the robot is about to reach the target. The red line represents the local path computed.

obtained previously to see if all the cells along lines approximating the path are unoccupied and sufficiently far from any obstacles. This way the path obtained tend to stay away from obstacles but in a smoother way (see figure 3.6).

Once a global path is obtained and the robot is moving towards the destination, for every kinect image received, a local path is computed. A local path needs to be computed firstly because the robot may drift away from the path due to actuators and perception noise, and secondly because unforeseen obstacles that have not been detected by the laser scanner might be present on the path. The local path tries to rejoin the global path from the current position while avoiding obstacles. To do so, the kinect point cloud is projected onto a 2D occupation grid as well as the relevant portion of the global path. Then using A* algorithm, a local path is computed to reach the end of the visible portion of the global path. We didn't use the laser scanner in this process because it wouldn't bring more information. In fact, the kinect perceives the same thing as the laser and additionally perceives below and above. Furthermore, since this process is repeated frequently, the narrow field of view of the kinect is sufficient in this context. Nevertheless, many obstacles could still be missed by this sensor and this local path planning solution, so in parallel the sonar belt and the downward camera are processed to detect obstacles on the floor and those not visible by the kinect. If an obstacle is detected this way, the robot is stopped, the obstacle is added to

the occupancy grid and a global planning is done to resume the exploration.

3.3.4 Limitations of the architecture

We got notable success using this architecture during the Défi CAROTTE. The experimental results are presented in the section 3.5. Nevertheless, in the context of this work different shortcomings have been identified and addressed. Firstly, this architecture only purpose is the exploration of an unknown environment but we want a domestic robot functioning day after day in the same environment. Secondly, the kinect sensor is suited for recognizing objects because of the rich information it provides, but it is inefficient for many tasks of an everyday life due to its narrow field of view. For instance, searching for a particular object is tedious with only this sensor because the robot would take an important amount of time to cover a large area. Finally, in this architecture, all objects present in the environment are learned off-line using a tedious process which is not satisfactory for a domestic robot. In the next section, we present how the architecture was modified to address those limitations.

3.4 Unsupervised and incremental semantic mapping

3.4.1 Software Architecture

The architecture proposed before was developed for the exploration of unknown environments. It contains many of the required functionalities for long-term semantic navigation in households, but it needs to be adapted and extended for this purpose. Furthermore certain shortcomings have been identified and addressed. Thus, we propose a different perception pipeline suitable for unsupervised semantic mapping.

The first thing we noticed is that the exploration of the environment is a good first step, but the robot needs to be able to localize itself later on in the world model obtained and update it overtime. To do so, the solution retained was to replace the SLAM module by what we called an incremental SLAM module. The purpose of this module is to localize the robot in the known environment and to produce incrementally an occupancy grid that contains only static information. The details of this module are given in the following section.

The second shortcoming identified is that objects are recognized based on a single view and a voting scheme is used to select an identity out of different recognition results. This solution has two drawbacks: it can not handle moving objects correctly and the recognition result is not always stable. In this architecture, we studied the use of a tracking module to accumulate the consecutive views of an object in order to perform the object recognition on

sequence of views, and therefore improve the temporal coherency of object detection results.

Thirdly, objects are only recognized using the RGB-D camera which has a narrow field of view, which makes it rather inefficient when searching for objects. We investigated how to improve that by adding the 2D laser scanner in the whole object recognition pipeline. To do so, we developed two laser-based detection algorithms, two laser descriptor extractors and an object representation suitable for multiple sensors.

Finally, in the exploration architecture, objects, walls and floors categories are learned off-line with a method that require a strong expertise. This is not suitable for a household environment since the objects present may vary overtime and it would be difficult to provide a robot with a model of all possible objects. The user cannot be expected to know how to train a classifier and it would be a burden for the company providing the robot to do the training anew every time a model needs to be added. Last but not least, data acquired by the robot in the target environment are often better learning example than those in datasets. For those reasons we have investigated the use of unsupervised and incremental learning of object models.

In the remainder of this chapter, the incremental SLAM, the laser-based object detections and the tracking approaches will be detailed. The rest of the perception pipeline is presented briefly and is the focus of the following chapters.

3.4.2 Incremental static occupancy grid

In order to improve performances of classical navigation techniques and to separate clearly the static and dynamic aspects of the environment, we investigated the use of what we call a static map. It is an occupancy grid which is constructed and maintained overtime to contain only static information of the environment.

In this occupancy grid, each cell can be occupied, free or unknown as in the standard approach. However, occupied cells in this context means locations that have always been seen as occupied (walls or heavy furnitures that never move). Free cells represent locations that have been seen at least one time as free. Unknown cells represent locations for which the robot has no information.

Assuming the pose of the robot is known, obtaining or updating this occupancy grid is similar to classical SLAM techniques with one slight adjustment. Cells that have been classified as free before, with a certain level of freeness, are locked and cannot change states (see figure 3.8). An exception is made for free cells that are close to an occupied cell. Without this exception, the errors in localization tend to make the walls in the map shrink or shift.

Localization on the other hand becomes more difficult than usual because

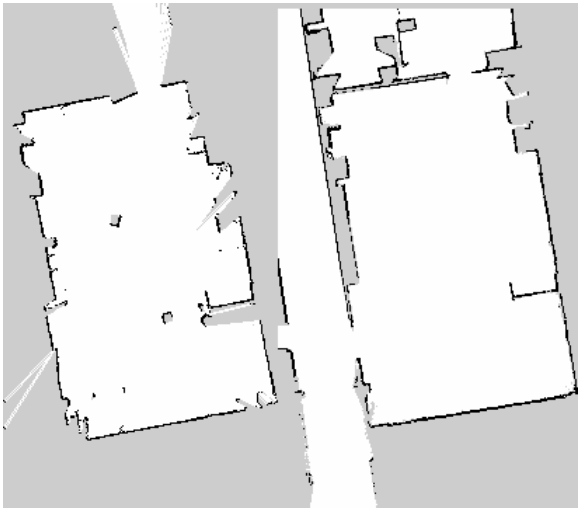


Figure 3.8: A map of a room after a first arbitrary run of the robot, the same static map after several runs.

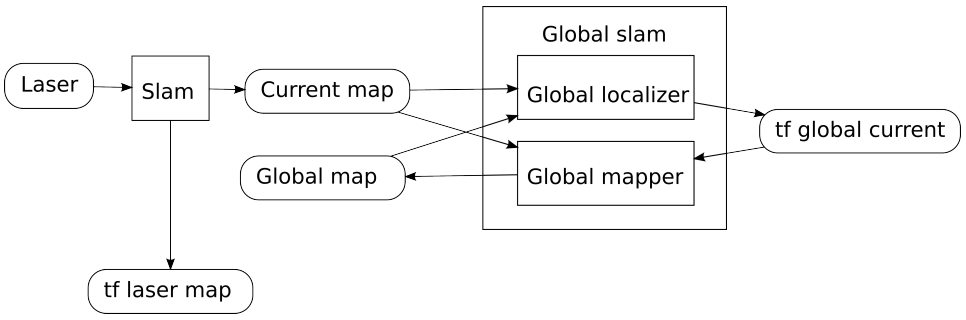


Figure 3.9: Flow chart of the global slam process. TF (in the ROS framework) represent the relative position of two elements

the map tends to contain only walls and heavy furnitures, so the part of a laser reading that can be matched to the map becomes smaller in comparison to the part caused by non-stationary obstacles. In order to solve this problem, a local[1] SLAM is performed to have a robust and reliable localization regarding the current state of the environment, and the output of this SLAM is localized in the static map by a slower technique using several sensors output. The figure 3.9 shows a detailed data-flow process diagram of this functionality. The SLAM used in this work as the local SLAM is called HectorSlam (Kohlbrecher et al. 2011). The global localization process takes as input the static map and several consecutive laser scanner readings localized by the local SLAM. The output is the transform between the static map and the local map origins. To do so, for each new laser readings, a fix amount of points is randomly picked in the set of past laser end-points and in the new ones. Doing this instead of simply using the last laser reading increase the performance for two reasons: more points are considered and since they are more distant (possibly), the localization is less susceptible to local minima. Once the laser points are picked, two set of random particles or position hypotheses are drawn. They represent the possible positions of the local map origin in the frame of static map, or said differently, the possible transforms from the static map to the local map. The first one is a set of uniformly distributed positions in the static map. The motivation of this set is to initialize the localization and eventually escape a local minima. The second one is a set of normally distributed positions around the current best one. For each transform hypothesis, using a likelihood field model (as in (Thrun, Burgard, and Fox 2005)), a matching score is computed and the best one is kept for the next iteration. In order to compute the score, for each laser points previously picked, we apply the transform hypothesis to obtain the point in the static map frame, and we compute a likelihood of this point based on how close it is to an occupied cell using a distance transform algorithm. Each point's likelihood is summed to obtain a score representing how well the local map and the static map match according to one hypothesis. Since the transform doesn't change because the two origins are fixed relative to each other (as long as the local SLAM is not restarted), once enough precision is achieved, the localization process can be stopped.

3.4.3 Novelty detection using laser readings

Given the static map of the environment, we are now interested in the detection of potential objects in the environment in order to ultimately model and recognize them using the laser sensor (see algo. 1 and figure 3.10). The localization provides a set of laser endpoints localized in the map reference frame that are noted l_i. An endpoint can either correspond

[1] Here, local refer to time, i.e. a SLAM limited to the information gathered by the robot during a particular run in the environment

Algorithm 1 Novelty detection algorithm

Input: *static_map* the occupancy grid representing the static obstacles of the environment
Input: L a set of laser endpoints
Output: C a set of clustered points

1: $outliers := \phi$, $inliers := \phi$, $C := \phi$
2: **for all** $l_i \in L$ **do**
3: $dist$ = distance to closest obstacle in *static_map* of l_i
4: **if** $dist > distance\_threshold$ **then**
5: add l_i to $outliers$
6: **else**
7: add l_i to $inliers$
8: **end if**
9: **end for**
10: **for all** $pt \in outliers$ **do**
11: $clustered = false$
12: **for all** $c_j \in C$ **do**
13: $dist$ = distance from pt to center of c_j
14: **if** $dist < radius$ **then**
15: add pt to c_j and update center of c_j
16: $clustered = true$
17: **break**
18: **end if**
19: **end for**
20: **if** not $clustered$ **then**
21: create cluster c with pt and add to C
22: **end if**
23: **end for**
24: **for all** $c_j \in C$ **do**
25: **for all** $l_i \in inliers$ **do**
26: $dist$ = distance from l_i to center of c_j
27: **if** $dist < radius$ **then**
28: add l_i to c_j
29: **end if**
30: **end for**
31: **if** $|c_j| < size\_threshold$ **then**
32: remove c_j from C
33: **end if**
34: **end for**
35: **return** C

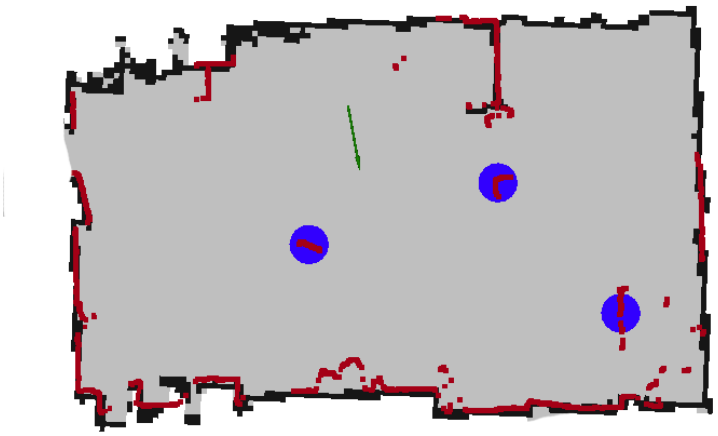


Figure 3.10: Example of novelty detection using our laser range finder. The black and grey colors represents respectively the occupied and free cells of the static map. The green arrow correspond to the robot's pose. The red dots are the laser range finder endpoints. The blue disks shows the detections obtained with our algorithm.

to something static (wall, static furniture), or to a moving object (chair, human, door). Points belonging to known static objects (mainly walls) should have a small distance to occupied cells, depending on the localization error. Whereas points belonging to dynamic objects usually have big distances to occupied cells. To separate them, the distance of each endpoint to the closest occupied cell in the map is computed using a distance transform algorithm. A static distance threshold (*distance_threshold*) is used on d_i to detect points belonging to a non-stationary object. The detected points are then clustered together given that they are at a certain distance *radius* of a cluster center. To do so, we iterate over all the detected points to compute the cluster they belong to and update the corresponding cluster center position. If no cluster is found for a particular point, we start a new cluster with it. This process is performed to group points belonging to different non-stationary objects. Once the clusters are formed, non-detected points are added to the them with the same *radius* criterion but without updating the center this time. The reason for this is that a non-stationary object near a wall might be ill detected using *distance_threshold* causing some of its border to be undetected and the remainder correctly detected. Thus, adding non detected point to the found clusters increase the chance to detect those correctly. Finally, some clusters might contain not many points because of noise in the sensor, occlusion or object seen from afar. They are filtered based on their size (*size_threshold*).

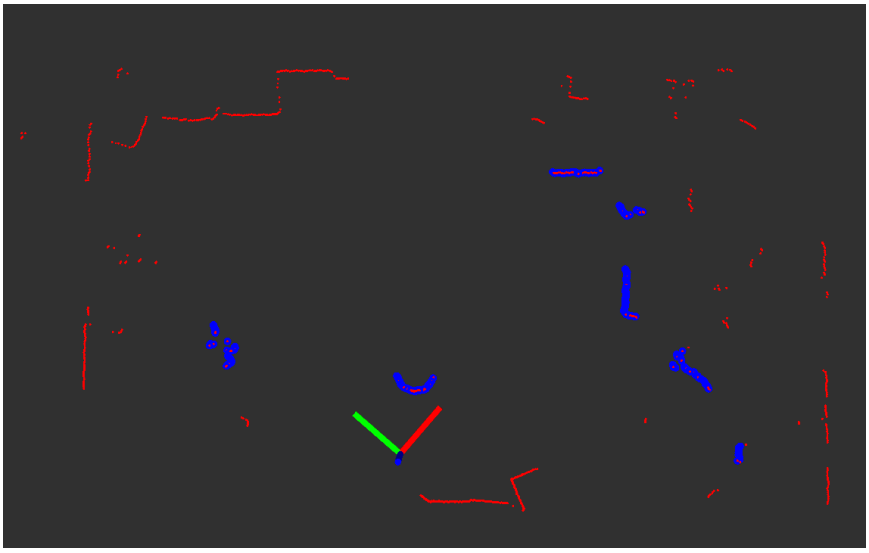


Figure 3.11: Example of a laser reading segmentation. The points in red represent the laser points, the points in blue represent the detections, the axes represent the robot position.

We set *radius* to $0.5m$ because most of the dynamic objects being considered are not wider than $1m$. The *distance_threshold* parameter is set according to the *static_map* resolution and the precision the localization. We set it to $0.20m$ We set *size_threshold* to 5 to have a chance of detecting thin objects up-close. However, laser descriptors (described in 4) extracted from clusters with few points are more noisy.

3.4.4 Laser readings segmentation

The previously presented novelty detection is used to detect non-stationary objects. However it fails to catch all the objects in a scene when the static map is not yet finished and it can not be used when exploring a new environment. For those reasons, another laser-based detection is used which can operate in every scenario (see figure 3.11). There are mainly three steps for this process, the first one is to split the laser reading based on discontinuity. Objects that are apart or corners will cause successive points in the reading to be more distant from each other than with other neighbors. So the reading is split using a threshold on the distance between successive points. The second step is to filter the split. Noise in the range reading tend to cause splits containing only few points. Again a threshold is used to remove the small splits. Finally, the third step is to filter the splits based on occlusion in order to keep only plain views of objects. The splits containing the end-points of the laser reading are first removed. Since they are perceived at the limit of the field of view they are clearly not plain views. Then all the splits that

are behind a successive or a previous split from the robot perspective are removed. To do so we compare the distance between the robot and the splits' end-points. Splits that are behind others are occluded views of an object or a wall.

Compared to novelty detection, this segmentation is able to detect more objects in more situations. However it gives also more false positives and the objects detected are not always non-stationary ones. For those reasons the novelty detection is a more reliable source for the object learning process, and the laser segmentation is better suited for active object search.

3.4.5 Object Tracking

Object tracking is important in a realistic navigation scenario as we mentioned before (see 2.4). In this study however, we focused on using object tracking as part of the object modeling process. In fact we see the tracking as a way to solve a temporal correspondence problem between new and past object detections. Thus, the object recognition we will describe in the next chapter doesn't operate on single views of objects as it is often the case, but on a stream of successive views. To do this, it is within the tracking process that views or detections of the same object from different sources are put together.

Because this study wasn't focused on solving the tracking problem in difficult situations, we didn't use a state of the art algorithm to effectively compute the tracking. We used a simple threshold based algorithm in which upon new detection input, the closest currently tracked object is retrieved and, depending on a time and distance threshold, it is viewed as from the same object or not. This solution was sufficient to track effectively one or two humans passing by and of course static objects, which was enough for this study. However, in a more realistic scenario, a more advanced tracking solution should obviously be used.

In the remaining of our work, we will call *tracker* the set of perceptions of an object gathered from its first perception to the point where the tracking of this object is lost.

3.4.6 Object modeling, recognition and learning

So far, we have presented a perception pipeline containing the detection of dynamic objects with a 2D laser scanner and a RGB-D camera and a tracking system to accumulate observations of one object. The sequence of views obtained out of the tracking system contains segmentations from our two sensors in temporal order of their perception. For each segmentations, different descriptors are extracted and sparse coded using dictionaries. The extraction of descriptors for the laser readings will be detailed in chapter 4. How we use dictionary learning to recognize the descriptors is also described in chapter 4. In chapter 5, we will explain how we model and recognize

Figure 3.12: View of our robot in the arena during the CAROTTE competition.

objects using the sequence of descriptors obtained after the sparse coding. The sequence of views thus obtained represent partial observations of objects. In fact, most of the time, the robot is only passing by objects and do not observe it from many different point-of-view. In chapter 6, we will explain how the robot can learn new objects incrementally and without supervision by grouping and filtering these partial observations.

In conclusion, our perception pipeline contains means to discover and learn new objects autonomously as well as to detect and recognize objects thus learned. By using two sensors and several descriptors representing different properties, it is strongly relying on multi-modal information and thus robust to many perception problems.

3.5 Experimental results

In this section, we report a few qualitative results illustrating the overall behavior of our robot architecture on the two addressed problems. More detailed and quantitative results concerning our main contributions on object modeling and recognition will be given in the next chapters.

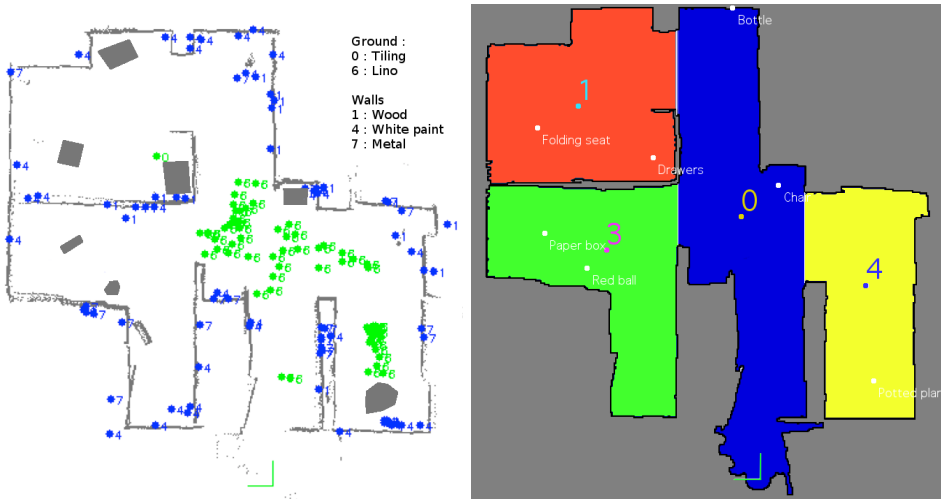


Figure 3.13: Map produced by our system (see text for details).

3.5.1 Défi CAROTTE

In the context of this work, we participated to the Défi CAROTTE of which the goal was to develop a robot capable of autonomously exploring an unknown environment and producing a semantic map containing rich information usable by a human (see figure 3.12). Figure 3.13 presents a map produced by our system in an $40m^2$ indoor environment that contains nine known objects (2 folding seat, drawers, 2 bottles, 1 chair, 1 red ball, 1 potted plant, 1 paper box). The top part shows the 2D map produced by the 2D laser scanner, along with the obstacles detected by Kinect and by the gravel detection algorithm (grey polygons). Green and blue numbers indicate the positions of the detected ground and wall type. In this environment, only lino is present and ground type is correctly identified in all image except one, recognized as tiling. The wall categories are mostly correct, with some confusions between the 3 categories that are filtered out when integrating detections in the rooms. The bottom part shows the result of the room segmentation and the mean position of the detected objects inside each room. Seven objects out of nine present in the environment have been detected. The two objects have been missed because they do not appear completely in any image taken during exploration. The positions of the detected objects are within 20 cm of their true positions. The error is larger for bigger objects as it is more difficult to estimate a correct object position from a partial view of these objects.

While our team was not successful during the competition, the proposed software architecture worked well on many different situations. In particular, the robot's behavior when facing difficult to perceive obstacles was satisfactory. The multi-modal navigation was efficient and failure often recovered from

successfully. With the architecture the robot wouldn't get stuck in a difficult situation. Finally, the robot was able to complete its mission most of the time (not during the final, though).

There was three main reasons for the robot to fail which the architecture couldn't handle. The worst of them was the overload of the CPU and memory resources. In case of lack of resources the system would start to lag and drop messages between modules. This would cause most of the modules to fail or perform poorly. Nevertheless, for safety reason, the reactive layer was implemented on a dedicated computer and would stop the robot when the last command received was too old. Because the reactive layer had is own resources, it wasn't affected by this problem and safety was ensured. Apart from improving each module computationally, this situation could have been managed with a mechanism of priority in the allocation of the resources.

The second issue was the failure of the SLAM system. The SLAM is the only module giving an estimation of the position of the robot, and most of the other computations involved require a good estimation. Thus, when the SLAM fails, the system is fairly compromised. A solution to this problem might be to detect the failure and reset the SLAM system.

Finally the last problem that the architecture has difficulty to handle is when an obstacle avoidance module is sending repetitive false positive obstacles messages. This was particularly problematic when the floor recognition system would have difficulty to recognize a certain type of floor and would randomly guess the correct category or mistake it for an obstacles. In this case the robot would start to back-out and discard this module messages to try to get out of the situation, but it wouldn't succeed all the time. A possible solution to this problem could be to use different recognition system to increase the robustness.

3.5.2 Unsupervised Semantic Mapping

The perception pipeline for unsupervised semantic mapping, including the modules for multi-modal object learning and recognition using a laser and a RGB-D camera that will be presented in the next chapters, was implemented on the robot and runs in real-time. The incremental SLAM module was validated by producing a static map of two rooms and a corridor and was then used in the other experiments (see figure 3.8). To evaluate the perception pipeline, we recorded the behavior of the system on 8 different trajectories in the same environment containing 8 objects that were moved in between. The figure 3.14 illustrates this experiment.

Concerning the long term localization and mapping, the system successfully build a static map of our environment and the localization took at most several seconds in our experiments. However, it could become very slow on larger environments.

Concerning the laser based object segmentation modules, the experiments

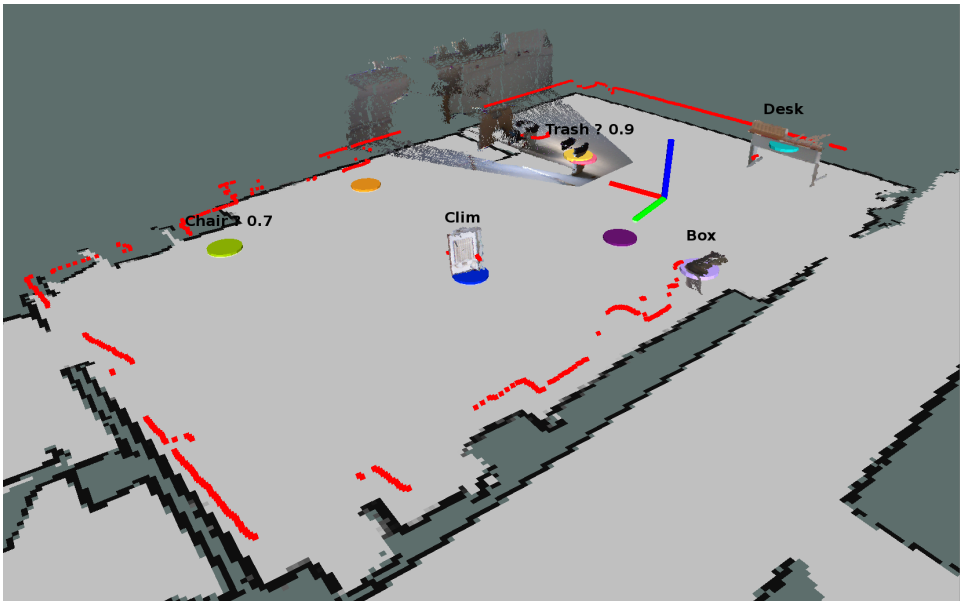


Figure 3.14: Capture of the experiment on one of the 8 trajectories. Three objects have been correctly classified: clim, box and desk. An object temporarily recognized as a chair is being tracked and evaluated even though no kinect information are available. A trash is being recognized as a trash. A definitive label is given to observations only when the tracking is over or when the recognition result is sufficiently high.

Localizer	50 ms
Detector	0.7 ms
Tracking	0.1 ms
Object learning and recognition	21.6 ms with 114 models
Total	72.4 ms after 10 runs

Table 3.1: Table of computation times.

indicate that they are able to detect objects as soon as the robot enters a room and largely before the objects are in the kinect field-of-view. However, there are several small limitations. Doors can be detected but only when open. Only big objects can be successfully detected when against a wall. Cluster of non-stationary objects are interpreted as one. Small or thin objects can only be detected when close to the robot because otherwise not enough laser points hit the object. However, the laser scanner field-of-view being wider than the kinect one, the perception of those objects is still slightly improved. Wide objects such as sofa or desks can be detected but not in their entirety.

The tracking and the detection modules used were sufficient to track and accumulate views of interesting objects (including humans) and to test our recognition system. However, they couldn't handle properly groups of detected objects. For instance, if a human and an object are detected, and if the human go to the object and push it around, the moment when the human and the object become close enough, they are treated by the system as a single different object.

For each implemented module, the mean computation time was recorded (table 3.1). The code was written in C++ without particular optimization. The computation times are bounded for each modules except the object learning and recognition ones (which are the focus of the following chapters and therefore are not detailed here).

In conclusion, using the laser scanner and tracking, the robot perceives objects sooner and during a longer time period than when using the kinect only. Thus, more information is gathered for the recognition and learning. The incremental mapping and localization system is able to build occupancy grids containing only static information for environments the size of common houses or lofts. Furthermore, computation time are reasonable for our application. Overall, the modification we made to the first architecture are an improvement for the domestic robots we envision.

3.6 Discussion and Conclusion

In this chapter, a robot architecture for semantic mapping of an unknown environment was proposed, followed by an extension of its perception pipeline to perform longterm unsupervised and incremental semantic mapping. Both the robot architecture and the perception pipeline proposed present a lot of advantages and some drawbacks.

First of all, at this level of complexity, which is far from a theoretical robot able to do house-chores, the robot seems to have a will to get in situations unexpected by the developers and to find all the bugs in the software. As such, considering possible failure is not an option but a requirement. Most of the difficulties encountered during this work was about the robot general robustness. Much work has been done regarding this, but a lot remains to be done.

From a practical point of view, having a data-flow of separate rather generic modules running in separate processes makes the development of the robot easier. Those modules represent most of the code running on the robot (maybe 80 %). Our team comprised eight people at the time of the competition. It was easy to separate the work and maintain or upgrade the system once the requirements for each module was set. This is the reason why this solution is so popular.

The behavior of the robot was coded using a script language as a hierarchical state machine. Using a script language made it easy to develop and try new scenarios very efficiently. Because this part is the less generic of all the system it makes sense to separate it from the rest. However, monitoring all the modules in one place means this process is really demanding in resources which in our case was not always a good thing. Perhaps, this layer would benefit from being separated into subsystems.

The separate reactive layer has proven necessary for safety reasons. It was designed to take precedence over the rest but communicates with the control system so that the robot could plan how to get out of a difficult situation. This proved very effective in avoiding some dangerous situations.

From the components perspective, the functionalities presented here are common across mobile robots. A special care was dedicated to multi-modal obstacle avoidance and path planning. The solution retained was satisfactory and the robot was able to take effectively into account different sensors and sources of information. Moreover, the perception pipeline extension improves the semantic information extraction. The robot field of view is enlarged by using the 2D laser scanner in the object detection and recognition processes. The quality of the recognition is improved by using different sensors and by performing it on an accumulation of view as will be shown in the following chapters.

Chapter 4

Object recognition using laser range finder

This chapter will present our first contribution on the use of a laser range finder to recognize objects while a robot is navigating an indoor environment.

4.1 Introduction

Many semantic navigation solutions use 2D laser range finder to perform SLAM, and rely on another sensor, generally a camera, to perform object recognition. Some studies use a 2D laser scanner to perform place recognition but few work has been done on object recognition. The main reason is that 2D laser scanners don't provide much information on objects compared to cameras. However, a laser range finder has a much bigger field of view which means that it starts perceiving objects before cameras can. Since it is used for navigation, the objects it senses are highly relevant in this context. Finally, because this sensor represents far less data to process and a high rate, we can suppose that it would be fast to have an object recognition response.

For those reasons, we believe that a laser range finder could be used to improve an object recognition system by giving a quick and good prior. In this chapter we present our solution to recognize objects using only this type of sensor. To do so, given a set of one object's laser segmentation taken at different point of view, we model the object by a bag-of-view (i.e., an unordered set of laser scans of the object) where a view is represented by a descriptor extracted from the segmentation. In our architecture, the input of this system is given by the laser detection and tracking algorithm presented previously in chapter 3.

This work is structured as follows: first we will present a review of related works in the section 4.2, then we will present two different laser descriptor computations and the object modeling process in the section 4.3, then a thorough evaluation is given in section 4.4, and finally we will conclude in

section 4.5.

4.2 Related work

In (Modayil and Kuipers 2004), object recognition using 2D laser range finder relies on geometric representations produced by accumulating localized segmentation of the objects. However, using this representation, it is not possible to model objects that might change shape overtime (such as humans), or objects that are ill perceived. Moreover, it heavily relies on a good localization of the robot, tracking of the object (if it is moving) and/or matching of few 2D points, which is a hard task.

Using techniques based on descriptors on the other hand reduces those difficulties. That is why we choose to study this solution to solve the problem of modeling objects with 2D range data. Moreover, since computer vision have had great success in recognizing objects based on points of interest and descriptors, it seems worth trying to do the same on 2D range data and thus taking benefits of the related techniques already developed.

Far less work has been done on describing segments of 2D laser range finder data than for images or 3D point clouds. Nevertheless, some work on describing binary images can be extended to describing 2D laser range data such as shape contexts (Belongie, Malik, and Puzicha 2002) and spin images (Johnson 1997). The shape context is a way of describing shapes in order to measure similarity and recover point correspondences. The idea is to pick n points of the shape and for each of them to encode their shape context using the relative distributions of the remaining $n-1$ points. This distribution is approximated by a uniform log-polar histogram of the relative positions. The spin image is a surface representation technique which is used for surface matching and object recognition in 3D scenes. For a given oriented point (3D point with a surface normal), a cylindrical coordinate system is defined and the surrounding surface is encoded using an histogram of the neighbor points expressed in that system. Note that both techniques use relative positions between points to encode geometrical information.

It has also been shown in (Tipaldi and Arras 2010) that it is possible to use descriptors with laser range finder, applied to place recognition in their case. They developed a technique to compute interest points and describing those using laser range finder. They have shown that it can be used to perform loop-closure, place recognition or global localization. However, this descriptor is meant for place recognition and so it is variant to point of view, which is not desirable for object recognition.

4.3 Contribution

We describe now two different descriptors that have been developed and our approach to multiple view object modeling using these descriptors. The first descriptor has been used in our work (Duceux and Filliat 2014). We call it pair-of-points based laser descriptor (PPLD for short). The second one is called triangle based laser descriptor (TLD) and was derived from the first one in order to improve its discriminative power.

4.3.1 Pair of points based descriptor

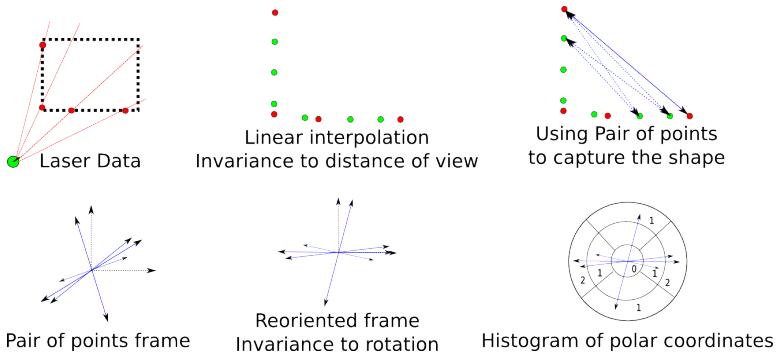


Figure 4.1: Illustration of the pair-of-points based laser descriptor building steps.

It is rather impractical to compare directly two sets of few points. This is why a descriptor is computed for every view in order to compare them. Using a descriptor provides a quicker way to compare views, and a certain level of robustness regarding noise. Its efficiency is a critical part of the system.

To be able to recognize non-stationary objects, invariance from point of view is required. This means that if the robot sees the same part of an object, the resulting descriptor should be the same, independent of where the robot sees it. To achieve that, several steps are involved in the construction of the descriptor. Fig. 4.1 illustrates those steps. To construct the descriptor, we followed ideas from (Belongie, Malik, and Puzicha 2002) and (Johnson 1997).

The detection provides a set of points representing a part of an object boundary. However, as the robot gets further from the objects, fewer laser points will hit the object, and they will be more separated. The first step is therefore to re-sample the points with a fixed inter-distance in order to be invariant to the object distance. For each pair of successive points, we use a linear interpolation to generate new points at regular intervals, which leads to having almost the same amount of points when the object is seen from afar than up close.

For each pair of points in this set, the vector that goes from one point to

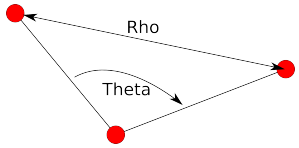


Figure 4.2: Illustration of the couple (rho, theta) computed for each triangle considered.

the other is computed in polar coordinates (r, θ). The θ coordinate of those vectors is dependent on the rotation of the object in the map reference frame. In order to achieve rotational invariance, a reference angle is computed as the maximum argument of the histogram of the θ coordinates distribution. This technique has been used in (Rofer 2002) to match laser scan and gave slightly better results than computing an axis of reference with Principal Component Analysis, or by using the endpoints of the detection. For every vector, a new θ coordinate is computed relative to that angle of reference.

Finally, the descriptor is computed as an histogram of the polar coordinates of these vectors normalized by the number of points. The histogram is parametrized by the number of divisions of both the angular coordinate (in $[-\pi, \pi]$) and the distance coordinate (in $[0, 1]$m). Those parameters have an important role in the performance of the system.

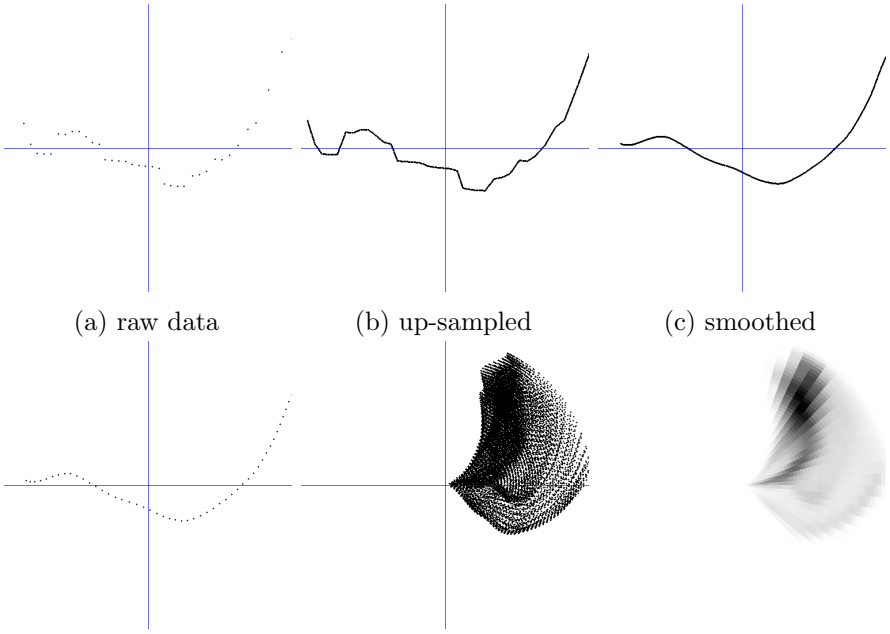
To compare two descriptors, the Symmetric Chi-Square metric is used. A comparison of popular metrics (Cha 2007) has shown slightly better results in our case with this one. The distance is expressed as follows:

$$d_{\chi^2}(I, J) = \frac{1}{2} \sum_i \frac{(I_i - J_i)^2}{I_i + J_i} \tag{4.1}$$

with I and J two descriptors, I_i and J_i the i-th element of the descriptor I and J respectively.

4.3.2 Triangle based descriptor

We propose a second solution to construct descriptors of laser data. Compared to the previous one, this solution is an improvement for our application. The problem with the previous one is that it is dependent on an angle of reference, which is difficult to obtain for some shape and can cause variations. To avoid those variations, some symmetries are added to the descriptor which makes it less discriminative. In this solution, triples of points are considered. For a given point belonging to a triangle, the angle at this point and the norm of the opposed side are computed (see figure 4.2). Both are invariant to point of view and don't depend on a point or angle of reference. However, as the number of points grows, computing all the possible triangles doesn't scale well because the complexity is too high ($O(n^3)$)). That is why a way to select relevant points is necessary.



(a) raw data (b) up-sampled (c) smoothed

(d) down-sampled (e) triangle space (f) polar histogram

Figure 4.3: Illustration of the descriptor building steps. From top-left to bottom-right: the input laser points, the signal up-sampled, then smoothed, down-sampled, the triangles in polar representation and finally the polar histogram which is the descriptor.

Two solutions are possible: draw a desired number of points randomly, or sub-sample the signal. We retain the sub-sampling solution by taking evenly spaced points.

The figure 4.3 illustrates the construction steps of this descriptor. As before, the building process starts by up-sampling the input signal (fig. 4.3b). This is a way to take into account the distance of perception. Since less points are going to be selected than in the previous descriptor, this solution is more susceptible to noise in the sensor. That is why the up-sampled signal is smoothed (fig. 4.3c). To smooth the signal, we compute the average point with a sliding window of $2 * n$ neighboring points. During this process of smoothing, the borders are cut which is acceptable because they are often noisy. Then the signal is down-sampled to pick a desired number of points that are uniformly distributed (fig. 4.3d). For all the triples from left to right, the angle at the middle point and the norm of the opposed side are computed, which gives a couple (rho, theta)(see figure 4.2 and (fig. 4.3e)). Those values are then transformed into a polar histogram (fig. 4.3f). We use a spreading scheme to add the points in the histogram, which means when a cell is hit, the neighboring cells are incremented too to lessen the effect of the quantization.

The resulting descriptor is a slight variation of the previous one, which tends to be more discriminative and robust. We used the same distance 4.1 to compare two descriptors.

4.3.3 Descriptor clustering

Algorithm 2 On-line dictionary

Input: D a descriptor
Input: W the dictionary
Output: a view label

Begin
  find closest view w_i to D in W
  d_min = distance w_i to D
  if $d_{min} < threshold$ **then**
    return label of w_i
  else
    add D to W
    return label of D
  end if
End

In order to have a compact representation of the objects, we follow the bag-of-views approach as described in the next section. For this, we need to compute a dictionary of descriptors obtained by clustering the perceived descriptors. We used the incremental method presented in (Filliat 2007). In this method (algorithm 2), a distance threshold is fixed to decide whether to create a new view in the dictionary or not, when a new descriptor is perceived. If the descriptor is far enough from all the views, it is used as the center of a new view.

4.3.4 Object modeling

Objects are represented as bag-of-views, i.e., as histograms of occurrences of the different views in a tracker (see section 3.4.5). An important problem is that the sampling of the views around the object will depend on the robot trajectory around it. As we want to construct the models on-line and be able to recognize objects with partial information, i.e., seen from only one side, we need to enforce a sampling of the views that will limit the dependency on the particular robot trajectory. To do so we filter descriptors during the construction of the model to increase chances of having similar models.

The filter comprises a condition on the relative position between the object and the robot and on the view being perceived. Indeed, since some objects might change shape, we can't filter only on the position. Therefore,

Figure 4.4: The 22 different objects in the database and their associated label for supervised tests. Two spiral trajectories around an object have been recorded for each one.

we only add a view to the model if it is different from the previous one or if the relative position of the object has moved more than a given distance (we use $10cm$).

Two objects descriptions are compared using histogram intersection:

$$d_{\cap}(I, J) = \sum_i min(I_i, J_i) \tag{4.2}$$

with I, J two histograms (bag-of-views) being compared. Note that an object histogram is normalized by its number of elements. A comparison between popular similarities and distances metrics (Cha 2007) has shown that although the difference is slight, the intersection gave the best results.

4.4 Experimental results

In order to assess the quality of our object representation, we built a dataset consisting of 22 objects (see figure 4.4). They have been chosen following several critera. Firstly, most of the objects are commonly laid on the ground and can influence the navigation (chairs, stool, human, wicker basket, vacuum, fan, trashes, shopping cart) which makes them interesting for our work. Secondly, since ours descriptors are encoding the shape and the size of the objects, we selected a few with similar ones (blue and black chairs, boxes) in order to assess their performance. Thirdly, our sensor is based

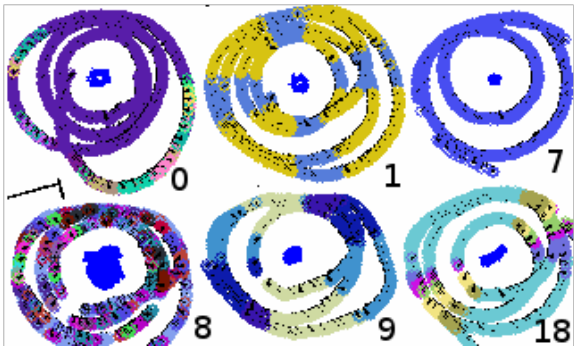


Figure 4.5: Regions with the same color represent where the robot has seen the same view in the dictionary using PPLD. The number represents the object label. The regions formed are consistent with the invariances expected from the descriptor, except when laser data is too noisy for certain objects.

on infrared light emission which is susceptible to dark color and reflective areas, therefore some objects are purposely hard to perceive (black screen, armchair and luggage). Finally, some object have a dynamic shape which can make them difficult to describe accurately (human, shopping cart).

To construct the database we moved the robot around the objects and recorded the trajectory and the laser data. Two trajectories were recorded by objects to ensure a separate training and test set. With this database, we performed experiments to set the different parameters using grid search and to evaluate the performances in an ideal case. The parameters were selected by optimizing the recognition precision with partial models and are given in the sections 4.4.1 and 4.4.4.

4.4.1 Pair of points based laser descriptor evaluation (PPLD)

The first experiment was to control the efficiency of the descriptor regarding the invariance we were expecting. In order to do that, we generate a map of the views obtained as a function of the position of the robot.

Fig. 4.5 has been made with a dictionary threshold of 0.03. The descriptors have 6 bin on distances and 11 on angles. The up-sampling step is done with a resolution of $3mm$. The figure shows that the expected invariances are achieved on objects with good response to laser sensor: objects 1, 7, 9, 18. Problems arise when far from an object, as too few points are obtained from it, which limits the distance at which we can perceive it. Also, certain objects are not well perceived by laser range data, such as black colored objects and objects having hard edges. In the latter case, a smaller variation in the position of the robot produce different views, so a descriptor can still be computed and used in the recognition process but with less robustness. For this type of object, the model needs to contain views coming from several

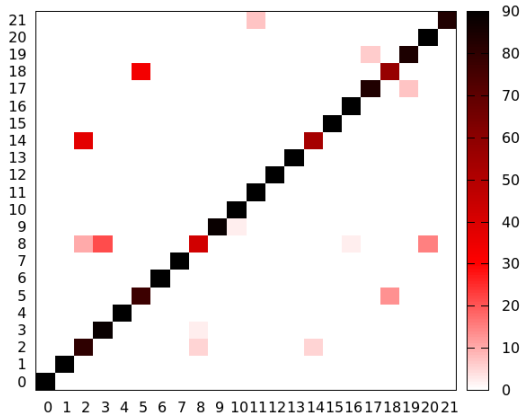


Figure 4.6: Confusion matrix for object recognition with complete models.

readings at the same place. That is why filtering of the repartition of views are made based on both relative position and the value of the view obtained. Finally, some objects are really noisy, such as object 8 in figure 4.5 which is a moving human. But even in this case we will see that the corresponding bag-of-views is specific enough to recognize it.

With this set of parameters, computing the PPLD on our robot's laptop took on average $28ms$ per view. This calculation time, given a fix set of parameters, varies mainly depending on the number of points to process which depends directly on the size of the object.

4.4.2 Recognition with complete models using the PPLD

In a second experiment, we evaluated the performance of the recognition when seeing the objects completely, i.e., from all possible viewpoints. We constructed a set consisting of eight complete models of each object. In order to perform cross-validation, the set was randomly divided ten times into a training and a validation set. Each time, one model by object was randomly picked to go in the training set. The remaining models were put in the validation set. Each time a confusion matrix was computed. All the results were accumulated in a final confusion matrix shown by Fig. 4.6. The global recognition rate obtained was 89% .

Results show that the method works well with complete models. The false recognitions are explained by the fact that some objects are perceived as having very similar shape and size with a laser range finder. For instance, the two chairs are more often confused and so are the box 18 with the box 5. However, most of the time the differences in size and shape are sufficient to avoid confusion. Lastly, the most misinterpreted object was the moving person. In fact, when moving, people's legs appearance is highly variable for the laser sensor. This in turn causes high variation in the resulting models,

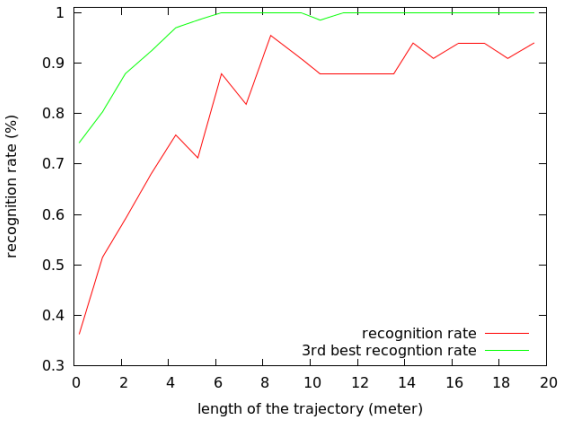


Figure 4.7: Recognition rate as a function of the length of the trajectory.

hence more confusion.

4.4.3 Recognition with partial models using the PPLD

In a real application though, the robot should be able to recognize objects with partial models without performing a full circle around the object. In order to assess this in a controlled setup, we computed the recognition rate as a function of the length of the trajectory sampled from the same database. In this experiment, the training set still consists of one complete model for each object. The test set consists of randomly generated trajectories of varying length.

Fig. 4.7 shows the recognition results with two different criteria. For the first one (in red), we have considered an object as being recognized if the most similar object is the correct one. As expected, the more an object is perceived, the better it is recognized. Note that around 4 meters the recognition rate is already strong, which correspond to seeing about half of the object. This corresponds to trajectories that the robot would have when avoiding an obstacle or passing by it. It suggests that recognition during the robot motion for another task could perform well.

In the second criteria (in green), we considered a recognition being successful if the right answer was in the three best score. The performances are clearly improved with a perfect recognition above 6 meters. This suggests that when the system is wrong on the identity of an object, it is not far off. For instance, when recognizing the black chair, we have seen that the system often confuses it with the blue one, but the similarity with the black chair would still be high. This result indicates that we could rely on this recognition as a good prior for mixing it with another algorithm using a different modality.

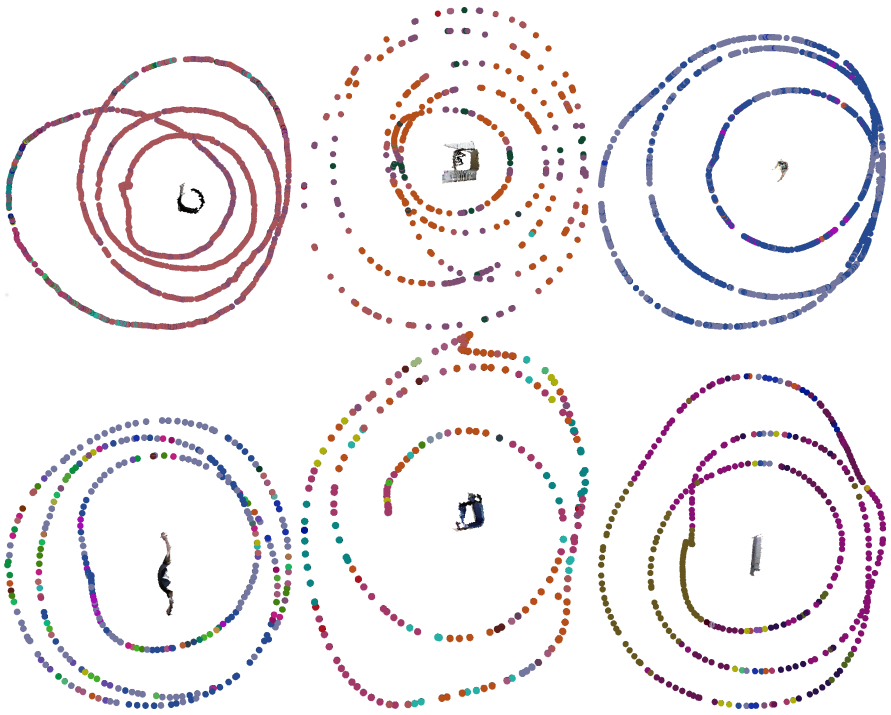


Figure 4.8: Regions with the same color represent where the robot has seen the same view in the dictionary using the TLD descriptor. The regions formed are consistent with the invariance expected from the descriptor, except when laser data is too noisy for certain objects.

4.4.4 Triangle based laser descriptor evaluation (TLD)

As before, to evaluate the use of this descriptor, we mapped the views obtained when perceiving the object with the position of the robot.

Figure 4.8 has been made with a dictionary threshold of 0.2. The resolution for up-sampling is $1mm$. The size of the window for the filtering is 100 points. The smoothed points are down-sampled to 50 points. The number of bin on the angle is 12 and the size of the bin for the norm is $0.1m$. The trajectories selected are the same than in section 4.4.1. We can see that the behavior of this descriptor is similar to the previous one. The classification results will show that it is more robust.

With this set of parameters, computing the TLD on our robot's laptop took on average $5ms$ per view. This is faster than the PPLD as reported on section 4.4.1. However, this difference in computation time depends heavily on the parameters chosen. In fact, the TLD's computation time grows rapidly with the increase of the selected points during the down-sampling step. Nonetheless, in our application using TLD is faster for a higher rate of recognition (see the next section) which makes it an improvement over

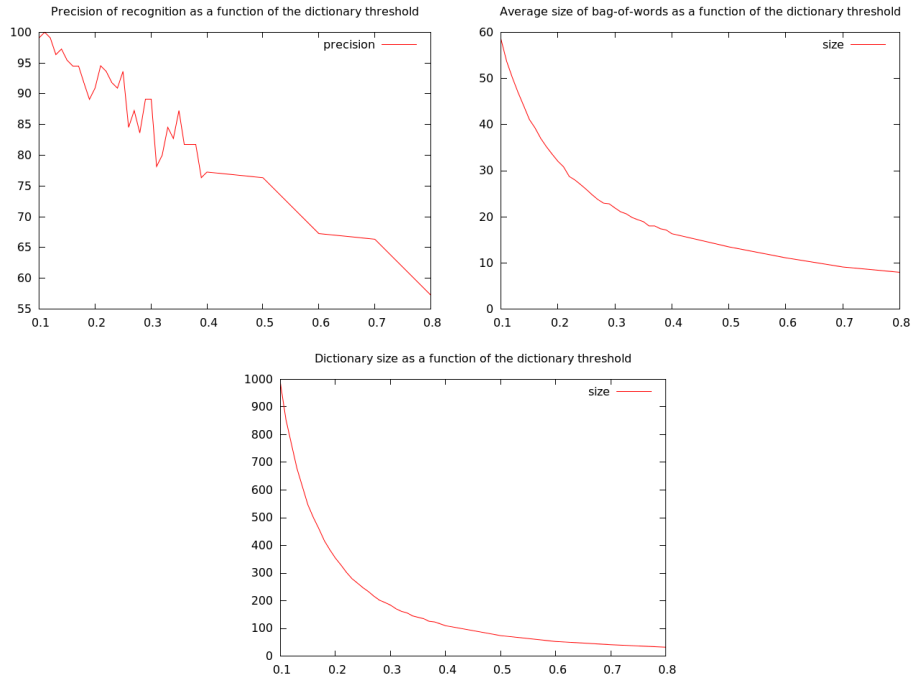


Figure 4.9: Results obtained by averaging over 5 different trials for each threshold setup. First row: from left to right: the recognition precision, the average number of views by model and second row: the size of the resulting dictionary.

PPLD.

4.4.5 Recognition with complete models using the TLD

The figure 4.9 shows the result obtained during the recognition using partial models in the same conditions as in section 4.4.2. We can see that the descriptor performs much better than the PPLD when the dictionary threshold is around 0.1. With this setup, the average precision obtained is around 97%, the average number of views by model is about 55 and the average size of the dictionary is near 980. With a threshold of 0.1 it takes about twenty seconds to process all the data (which represent 54012 descriptors) and to obtain the results (see table 4.1). We can see that a dictionary threshold around 0.2 gives approximately the same results than in section 4.4.2. With this threshold the resulting bag-of-views and dictionary are about the same size but since the TLD is faster to extract, the whole process is quicker. So, in conclusion we have shown that the TLD descriptor is an improvement over the PPLD, being able to achieve higher recognition rate, and being faster at the same level of recognition.

The figure 4.10 shows some confusion matrices obtained during this

TLD			PPLD		
dictionary threshold	dictionary size	recognition rate	dictionary threshold	dictionary size	recognition rate
0.1	982	97	0.01	2119	92
0.2	351	88	0.02	678	91
0.3	184	82	0.03	331	89
0.4	111	75	0.04	196	87

Table 4.1: A comparative table of the results obtained using the parameters given in 4.4.1 and 4.4.4. The first column represent the dictionary threshold, the second one is the number of views (or size) in the dictionary and the third one correspond to the recognition rate obtained. The results given for the size and recognition rate are average over ten trials. PPLD with a threshold of 0.03 correspond to the results given in 4.4.2.

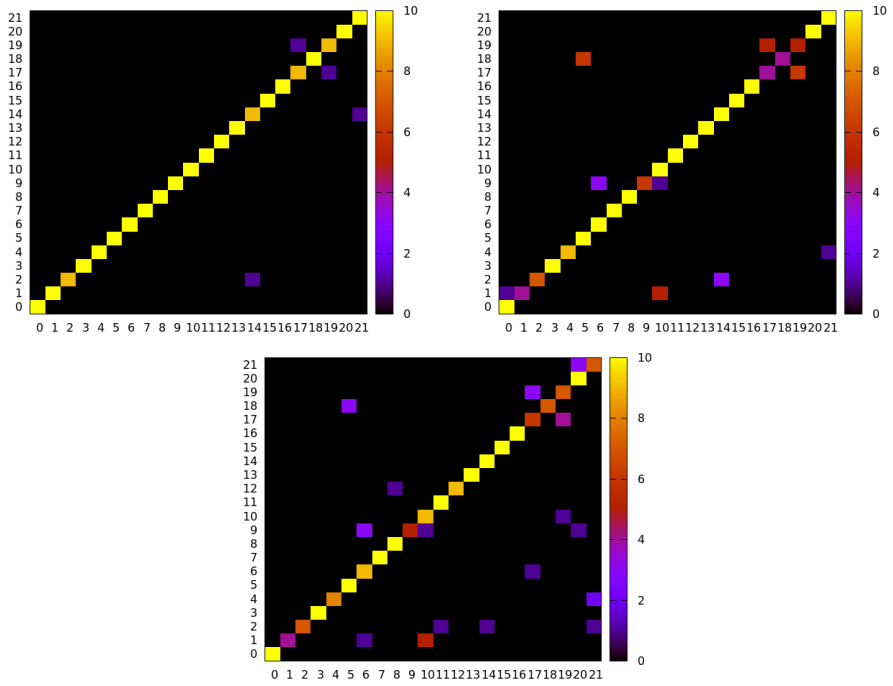


Figure 4.10: From left to right, the confusion matrices obtained with a threshold of 0.1, 0.2 (first row) and 0.3 (second row) respectively by accumulating the scores over 10 trials.

experiment for several dictionary thresholds. We can see that there is a strong confusion between the object 17 and 19 (as it was also the case with PPLD), which can be easily explained by the fact that they have the same shape and a small difference in size. The other confusions are less obvious and might be caused by noise in the sensor readings, especially for black or small objects. However, it has to be noted that during this test only the dictionary is a cause for variation in the precision result because the descriptors and tracks are not recomputed. The dictionary is rebuilt every time by feeding all the descriptors from all objects in a random order.

4.5 Conclusion

In this chapter we presented two novel 2D range data descriptors and we used them in bag-of-views models to recognize objects in a supervised manner. The results show good performance in the recognition tasks, and that the descriptors have the intended invariance. Since 2D range data are far less complex than images or point cloud, the recognition power is also lessened. But such data are easier and faster to process, and a laser range finder usually has a larger field of view than a camera. Moreover, in a navigation task, most of the objects involved are perceived by this sensor, and that is why it has been used intensively in the past. With this limitation, we have shown that it is possible to distinguish between a reasonable number of objects sufficient for common household setups and to perform recognition.

Those reasons make the use of 2D laser range finder for object recognition in a navigation context worth studying furthermore.

Chapter 5

Multi-modal object modeling

This chapter describes our contribution on multi-modal object modeling using the graph-of-views approach in order to extend the methods of the previous chapter to object recognition with multiple sensors.

5.1 Introduction

In chapter 4, objects were modeled by histograms of views obtained with the laser readings following the bag-of-views approach. Those models represented a distribution of possible views for an object. In the case of an unsupervised and on-line learning, the models obtained would be partial and highly dependent on the trajectory of the robot. So, in order to obtain a complete model from the partial ones, it would be necessary to merge the distributions which is difficult considering that they do not represent the same aspects of the object. The figure 5.1 illustrates this problem.

One possible approach to reduce this problem is to use a binary histogram, meaning it's not the number of times an element as been seen that is stored but just whether it has been seen or not. In this case, it is possible to easily merge the partial models. However, using this method with the laser descriptor is not very efficient because its discriminative power is too limited.

To improve the discriminative power of local descriptions, a common solution is to use pairs or even lists of descriptors as histogram elements. The combination of views being more rare it gives more discriminative power.

Another direction of work in order to be more discriminative and robust in our situation is to use different sensors and/or types of descriptors. This raises the question of how to combine those inputs of different nature. They could be processed independently and their recognition results could be merged, but a better solution would be to take advantage of the co-occurrences of all those data.

Studying these problems, we came up with an original formulation to represent the objects that we call graph-of-views. This is the focus of this

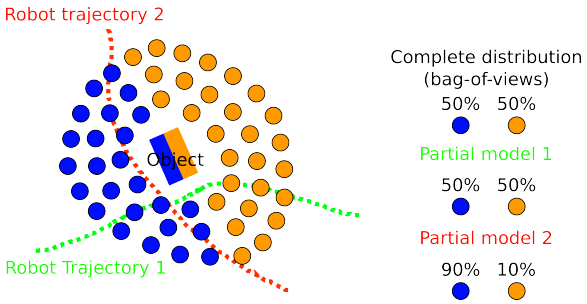


Figure 5.1: An object with a blue side and an orange side. If the robot were to see the object from all possible point-of-views, it would observe a distribution of color of 50% blue and 50% orange. However, when the robot perceive the object partially (only on trajectory 1 and 2), the distribution observed can be very different.

chapter.

5.2 Related work

Using graphs to represent a scene or an object in a recognition tasks has already been studied many times. However, there are many different ways to represent something with a graph which leads to representation with very different properties.

Our graph-of-view model presented in the next section resembles aspect graphs (Cyr and Kimia 2004). Given a certain viewpoint of a camera, an object is perceived as having a certain 2D appearance. An aspect is a partition of the viewpoint space where the 2D appearance is invariant. The aspects are the nodes of the graph and the edges represent the visual events of changing aspect when rotating around an object. This representation is efficient to perform object recognition but is very expensive to construct and to store for complex objects. Also in the formalization, there is no study on how to handle various kind of appearances coming from different sensors.

In (Mei, Sibley, and Newman 2010) they replace bag-of-views scene representation by co-visibility graphs for loop closure. The graphs represent the different features that are visible in the same frame or at the same robot location. By exploiting this co-visibility explicitly they improved loop closure solutions because it depends less on the discretization of space. If our work resembles this in spirit by replacing bag-of-views by a graph-of-views representation, the semantic of their graphs is slightly different from ours and they are used very differently.

A graph of view representation is used in (Liang et al. 2014). They propose a method to learn and model object simultaneously. Their method however is based on Visual SLAM techniques and the purpose is to reconstruct RGB-D

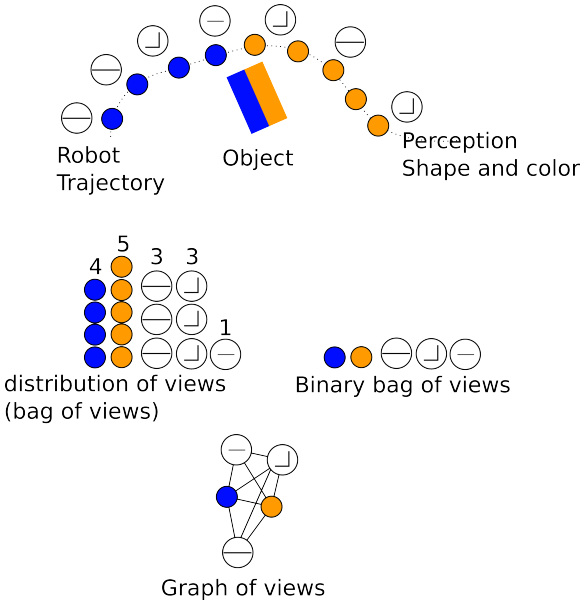


Figure 5.2: Different modeling of an object. See text for details.

point clouds for objects and their results are shown using only three different objects that are fairly different. Our representation is not aiming at such complete and precise metric object models and is therefore computationally more efficient. It also handles different appearance properties.

Concerning the fusion of different appearance properties, (Caron, Song, et al. 2014) propose to use a Neural Network to learn models of objects with different features extracted. This representation can not be updated incrementally and the learning can not be unsupervised. However, they do propose to fuse different features and show an increase in generalization and performance when recognizing objects in everyday scene. The features they used were tested with our model as will be described in this chapter.

Finally, in computer vision, some studies see the different features from a single image as a graph and try to recognize objects from the image by matching the graphs obtained with different images. This is known as the graph matching problem (Caetano et al. 2009). This is a difficult problem when there is no unambiguous correspondence between nodes and edges of the graph which we didn't study in depth. Instead, the correspondence between our features is learned by the dictionaries, and we use different similarity metrics to compare the graph structures.

5.3 Graph-of-view object model

5.3.1 Representation

The graph-of-views is an extension of the bag-of-views representation where we consider relations between the perception of different views. Figure 5.2 illustrates the graph-of-views model and the difference with the bag-of-views. The top of the figure represents a given object which is passed by a robot. On its course, the robot perceives different aspects of the object depending on its point of view. The problem is then how to represent those observations of objects so that the robot can compare and identify them. The bag-of-views approach idea is to represent the object by the distribution of descriptors perceived. In other words, a bag-of-views model stores how many times the different descriptors have been perceived. A variant of this representation is the binary bag-of-views in which only the presence of a certain descriptors is retained. In the graph-of-views approach, each node of the graph represents the presence of a view descriptor. The edges represent spatial and temporal relations. It means that when the robot perceives one node of the graph, the edges represent what other nodes it could perceive next or at the same time. So, the graph-of-views representation extends the binary bag-of-views by adding information about the relationship between the perception of the different descriptors.

For example, if an object has one side blue and the other one orange as in figure 5.2, when the robot turns around the object, it is going to perceive the change. In this case, the graph would be a node ”blue”, a node ”orange” and an edge between ”blue” and ”orange” because the robot perceived that particular transition. In this case, the edge is a temporal relation between perception of the same kind. But edges can also represent spatial relations between perceptions of different kind. Suppose the robot has a shape sensor and a color sensor and the object perceived is rectangle-shaped, then the resulting graph would contain a node ”rectangle”, a node ”blue” and an edge between ”rectangle” and ”blue” because it has perceived both of them at the same time and location.

Figure 5.3 shows a real observation of a box taken by our robot using a laser and an RGDB camera and several features for each sensor. Figure 5.1 and 5.4 show the corresponding bag-of-view and graph-of-view model respectively. A model derived from such observation is a partial model of the object because the robot didn’t perceive it from all possible points of view. When two partial models have been identified as being from the same object, we would like to derive a more complete model. As we said earlier, since the bag-of-views are distributions, deriving another model from two partial observations would be difficult. However, using binary bag-of-views or graph-of-views, merging two observations is straightforward: it suffices to take the union of the two models. In the section 5.4.5, experiments will

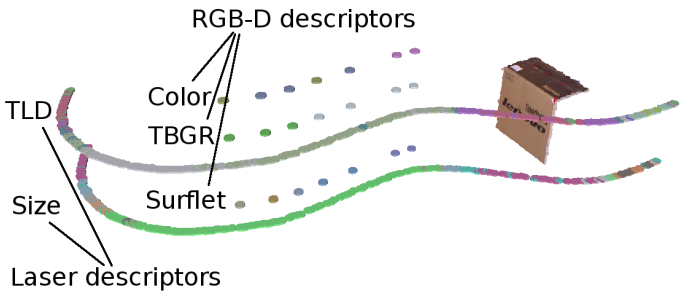


Figure 5.3: A sequence of views perceived by the robot on a particular trajectory around a box. Each disk represents a robot position where a view was extracted. Disks of same color are the same view perceived at different places, disks at the same height are views of the same descriptor type. Note that descriptors extracted from the laser scanner are more numerous due to larger angle of view and higher sensor frequency.

type	id	frequency	type	id	frequency	type	id	frequency
TLD	0	131	TLD	1	36	TLD	2	56
TLD	3	7	TLD	4	15	TLD	5	16
TLD	6	3	TLD	7	1	TLD	8	1
TLD	9	1	size	0	62	size	1	110
size	2	40	size	3	15	size	4	17
size	5	12	size	7	1	size	8	1
size	9	1	size	10	1	size	11	1
color	0	2	color	2	1	color	1	4
surflet	0	4	surflet	1	1	surflet	2	1
surflet	3	1	tbgr	0	3	tbgr	1	4

Table 5.1: The bag-of-view model corresponding to the sequence of figure 5.3

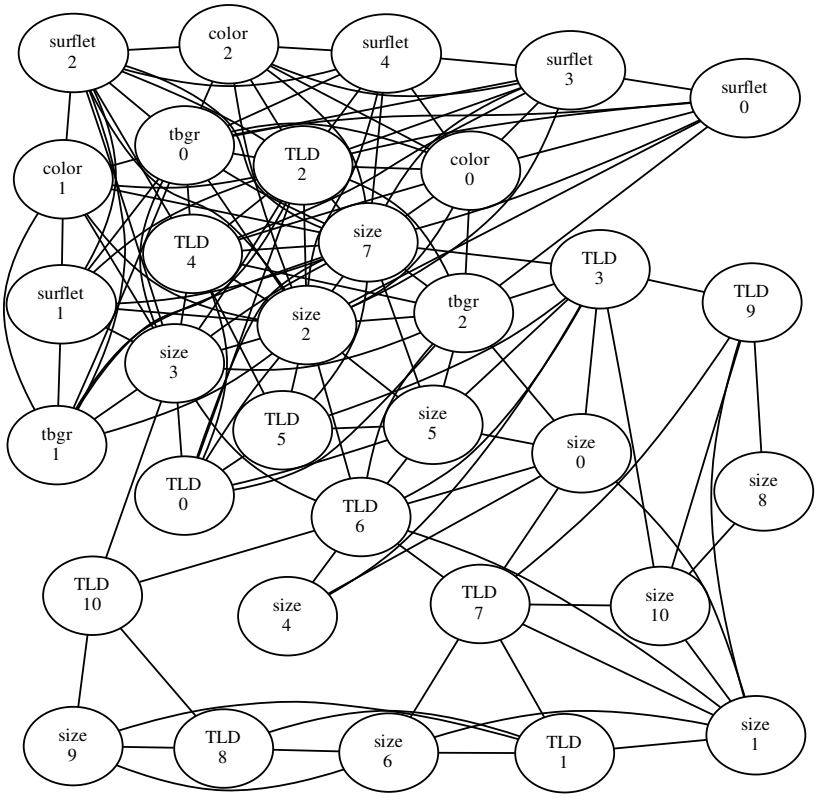


Figure 5.4: The graph-of-view model corresponding to the sequence of figure 5.3

show the utility of merging models to improve performances.

5.3.2 Definition

In this section we will define more formally our graph-of-view representation and introduce the notations used for defining graph comparison metrics.

A graph-of-view I is a pair $I = (N_I, E_I)$ comprised of a set N_I of nodes together with a set E_I of edges.

A node $n \in N_I$ is a word taken from one descriptor dictionary among the various modalities used for object recognition (e.g. colors from a RGB-D camera, TLD from a laser sensor). Two nodes are equal if and only if they are the same word from the same dictionary. An edge $e \in E_I$ is an unordered pair of nodes ($e = (n_1, n_2)$). Two edges are equals if they contain the same two nodes.

The intersection of two graph-of-views is the graph-of-view defined as: $I \cap J = (N_I \cap N_J, E_I \cap E_J)$, i.e., the set of nodes and edges that are common to the two graphs. Similarly, the union of two graph-of-views is a graph-of-views defined as: $I \cup J = (N_I \cup N_J, E_I \cup E_J)$. Note that in this last operation, there are no considerations about the origin or nature of the visual words contained in the graphs. Finally, we define the size of a graph-of-view as: $|I| = |N_I| + |E_J|$.

5.3.3 Comparison one to one

In order to perform object recognition using our graph-of-view representation, different similarity functions have been tested to compare two graphs.

The first one and the most obvious is to calculate the size of the intersection. It represents the number of nodes and edges that two graphs have in common. Let $I = N_I, E_I$ be a graph-of-views constituted of N_I its set of nodes, and E_I its set of edges. Let $J = N_J, E_J$ be another graph-of-views. Then, the intersection size similarity function is written as follows:

$$intersection\_size(I, J) = |I \cap J| = |N_I \cap N_J| + |E_I \cap E_J| \tag{5.1}$$

This similarity function is fast to compute and gave good results so it was often used in the following studies. Since it doesn't take into account the type of view (i.e. the particular sensors and feature they are derived from) or connectivity constraints, it is available as soon as the robot gets information on an object. Note that when counting the edges in common, the nodes attached by it are also counted. This means that $|N_I \cap N_J|$ and $|E_I \cap E_J|$ are not independent. It is tempting to think that one only need to count the edges in common. However, we will see in section 5.4.2 that it doesn't result in better performance. This is explained by the fact that in our problem and in practice, two graphs of the same object can have the same nodes but

with different edge structures. Which means that the nodes intersection is a relevant information and is not the same as the edges intersection.

Derivatives of the intersection have been tested, namely the degree of inclusion and the intersection divided by the union. The degree of inclusion is written as follows:

$$inclusion\_degree(I, J) = \frac{|I \cap J|}{|I|} \tag{5.2}$$

It is an interesting similarity function because if $inclusion\_degree(I, J)$ is equal to 1, we know that I is completely explained by J. This means that I doesn't provide any new information, which can be useful when learning objects as we will see in chapter 6. It is a normalized measure which is helpful when classifying models in a setup where the different labels are represented by models with different sizes.

The intersection divided by the union measure is written as follows:

$$intersection\_over\_union(I, J) = \frac{|I \cap J|}{|I \cup J|} = \frac{|I \cap J|}{|N_I \cup N_J| + |E_I \cup E_J|} \tag{5.3}$$

This measure is interesting because it is normalized and that it is equal to 1 when the two models are exactly the same. This is useful when comparing complete models to each other to measure the similarity between the objects.

Sometimes the intersection between two models would be composed of different connected components, so we tried to use the size of the biggest connected components in the intersection as a measure. The idea behind it is that in certain conditions the intersection of models of similar objects can result in a graph of big size but not very connected. In this case it might be interesting to measure how well the intersection is connected to obtain better results. We define the function $Comp(G)$ which gives the set of connected components of G, then the measure can be written:

$$max\_component(I, J) = \max_{C \in Comp(|I \cap J|)} |C| \tag{5.4}$$

Until now, all the similarity measures proposed did not explicitly take into account the different modalities. The drawback is that the measure do not exploit multi-modality information that could be useful to differentiate objects that are similar in some modalities. For example, imagine the robot has three observations to compare. The first observation is of an object with a circle shape and the color blue, the second is an object with a circle shape and the color red and the third is an object with a circle shape but the color is not available. In this example, the first and second observation have the same intersection than between the first and third observation even though they contain different colors and thus cannot be the same objects, while the first and third do not have conflicting information. In this case, the measure

could be improved by taking into account explicitly the different modalities present in the models.

To do so, we define the function

$$number\_of(t, G) = |\{e \in G : \text{type of } e \text{ is } t\}|$$

which gives the number of times a modality or feature type occurs in a graph-of-view. Similarly we define the function $modalities\_number(G)$ which returns how many different modalities are present in G. For example, let E be a graph composed of one shape (noted s) view and two color (noted c) views as follows:

$$E = ((s, 0) - (c, 0) - (c, 1))$$

In this case we would obtain:

$$\begin{aligned} number\_of(s, E) &= 1 \\ number\_of(c, E) &= 2 \\ number\_of(s - c, E) &= 1 \\ number\_of(c - c, E) &= 1 \\ modalities\_number(G) &= |\{s,\ c,\ s - c,\ c - c\}| \\ &= 4 \end{aligned}$$

Using those functions, different measures were found interesting. The first measure is the number of modalities for which there is at least one node or edge in common between two models and can be written as follows:

$$modalities\_match(I, J) = modalities\_number(I \cap J) \tag{5.5}$$

We found that this measure is a very relevant information because it tells how many modalities are involved in a similarity between two objects. However, on its own it doesn't yield good results because in practice dissimilar objects can have few nodes or edges of different types in common. To solve this, we tested measures that combine both information of how much two models match and across how many modalities.

In the *joint_modalities_intersection*, the number of modalities is multiplied by the intersection size:

$$joint\_modalities\_intersection(I, J) = modalities\_match(I, J) * |I \cap J| \tag{5.6}$$

The normalized version of this measure is written:

$$normalized\_jmi(I, J) = \frac{joint\_modalities\_intersection(I, J)}{max\_modalities * |I|} \tag{5.7}$$

Note that the number of modalities is normalized by $max\_modalities$ which is the maximum number of modalities considered in the system. The reason for that is because otherwise the measure would lose its power to distinguish between models with different modalities and models with fewer modalities available, which is recurring when comparing partial models against complete ones.

Finally, the function $per\_modalities\_inclusion(I, J)$ measures per modalities how much is included. With it, we want to have a more quantitative measure of how many modalities are in common than with $modalities\_match(I, J)$. It is written as follows:

$$per\_modalities\_inclusion(I, J) = \sum_{t \in types} \frac{number\_of(t, I \cap J)}{number\_of(t, I)} \tag{5.8}$$

Imagine we use a system with three modalities: color, texture and shape. Object A and B both have 15 descriptors of shape, 10 of color and 5 of texture. A is equals to B in shape, has half of its color descriptors in common with B and is completely different in texture. Then they have 1.5 modalities in common out of 3 ($per\_modalities\_inclusion(I, J) = 0.5$). Note that this measure is different from the $inclusion\_degree(I, J) = 0.66$ which would give 20 elements in common out of 30.

In the following section 5.4.4 a comparison of all those measures is given along with the measures one to many presented in the next section.

5.3.4 Comparison one to many

In the previous section we presented the different similarity measure studied to compare graph-of-views against one another. However, knowing several objects brings additional information like the frequency of each feature across the different type of object. It is possible to exploit that additional knowledge by using a measure that takes into account all the known models. In this work, we tested two different solution: the first one is known as TF-IDF, which stands for term frequency and invert document frequency, and the second one is an original measure that we developed and called modalities consensus.

TF-IDF at the origin is a measure used in text retrieval which have been popularized by (Sivic and Zisserman 2003) for object matching using visual features. It has been used in numerous studies with success. The idea behind this method is to give different weight to a view, or visual feature, depending on their frequency in the document, or model, being compared, and their frequency across the labeled documents. The goal is to reduce the importance of views that are too common because they are less informative. In our case, a model is a graph-of-view and we apply this technique to both nodes and edges. We write it as follows:

$$tf\_idf(I, J, M) = \sum_{e \in I} tf(e, J).idf(e, M) \tag{5.9}$$

In this equation I represent the queried model, the one being classified, M represent the set of known models and J the one being considered for comparison such that $J \in M$. $tf(e, J)$ is the term frequency of e in J, e being an element of I either node or edge. Since we don't considerer views frequencies in the construction of our models, we can simply write:

$$tf(e, J) = 1 \text{ if } e \in J \text{ or } 0 \text{ otherwise} \tag{5.10}$$

We write the *idf* term as follows:

$$idf(e, M) = log \frac{|M|}{|\{m \in M : e \in m\}|} \tag{5.11}$$

This means that the *idf* term is the logarithm of the size of M divided by the number of models m in M that the element e appears in. So the more a node or edge is present in different categories, or common, the less weight it gets.

The idea behind modalities consensus is to measure how well modalities agree on a category given a queried model. It is computed as the sum of the intersection per modalities normalized over the existing models. It is written as follows:

$$modalities\_consensus(I, J, M) = modalities\_number(M)^{-1} * \sum_{t \in types} \frac{number\_of(t, I \cap J)}{\max_{m \in M}(number\_of(t, I \cap m))} \tag{5.12}$$

In this equation I, J and M represent the same things as previously. The function *number_of* was defined in the previous section 5.3.3 which returns the number of times a certain type occurs in a graph-of-views. So $number\_of(t, I \cap J)$ is the size of the intersection when considering only the descriptor or pair-of-descriptor of type t, and $\max_{m \in M}(number\_of(t, I \cap))$ is the maximum obtained across all labeled models. This means that the fraction is equal to 1 when J is the best match in M according to the modality t. By summing over the types we compute how much modalities agree on J being the best match. The sum is normalized over the total number of modalities in M to have a similarity measure comprised between 0 and 1.

In the following section 5.4.4 a comparison of those measures is given along with the measure one to one.

Figure 5.5: Images of the 25 objects of the dataset extracted from the recorded point-clouds.

5.4 Experimental results

5.4.1 Dataset

To evaluate this representation we performed similar experiments as in chapter 4 using another dataset (see figure 5.5) which is more realistic in what the robot would experience in our scenario (see section 1.2). In this setup, two sensors are used and recorded: a 2D laser scanner (hokuyo utm-30lx) and a RGB-D camera (kinect). For 25 different objects, 6 different trajectories are recorded. Each trajectory represents the robot going toward an object from a different angle and then getting around it. The goal is to record realistic trajectories that correspond to what the robot would do when avoiding the object or when going towards it to recognize it. As an example, figure 5.6 shows the trajectories recorded for the black armchair.

For the experiments on this dataset, five type of descriptors were used. Two of them were computed from the laser: the TLD as described in chapter 4 and a size descriptor which is the measure of the most distant two points in a laser segmentation. It is approximately the perceived diameter of the object. Three descriptors were computed from the RGB-D images: a 3D shape descriptor called surflet (see section 3.3.2), a contrast descriptor called TBGR (see section 3.3.2) and a color opponent histogram. The color

Figure 5.6: Sample of 4 trajectories for the black armchair from the dataset. Regions with the same color represent the same view from the TLD dictionary. They are placed where the robot has perceived them. We can see that the trajectories share resemblances, and that it could be possible to recognize the object with a few of them as references.

opponent descriptor is an histogram of the pixels of an image where the RGB values have been transformed to the color opponent color space.

Some of the objects are purposely difficult to perceive or differentiate using only one sensor or type of descriptor. The black armchair and the screen are difficult to perceive by the laser because they have a reflecting black color which adds a lot of noise to the sensor reading. The others objects (the chairs, the luggage, the metal fan, the stool, the tripod and the turtlebot) are difficult to perceive because the portion seen by the laser scanner is very thin. The objects difficult to perceive by the RGB-D camera are the metal fan and the turtlebot because their surface are reflective and thin. This makes the metal fan the hardest object in the dataset to recognize because both sensors have difficulties to sense it. The pair black armchair and red armchair have exactly the same shape, so they are difficult to recognize using only this type of sensor. The black chair and the blue chair have a similar shape and the same color in the back. The human has been recorded while moving which makes it difficult for shape descriptors. Finally, the color descriptors can be disturbed by the variance in illumination. Thus, the objects in the dataset are difficult to recognize using a single sensor and/or a single descriptor.

The extracted models for the experiments were all obtained using the pipeline presented previously (see section 3.4): the global slam, the 2D laser and RGB-D segmentation, the tracking, the descriptors extractors and the dictionaries presented in section 4.3.3. No intervention were performed to correct the output. The experiments were repeated ten times with a dictionary initialized using a random subsample of the descriptors in the dataset. Reported values are the mean over these 10 experiments.

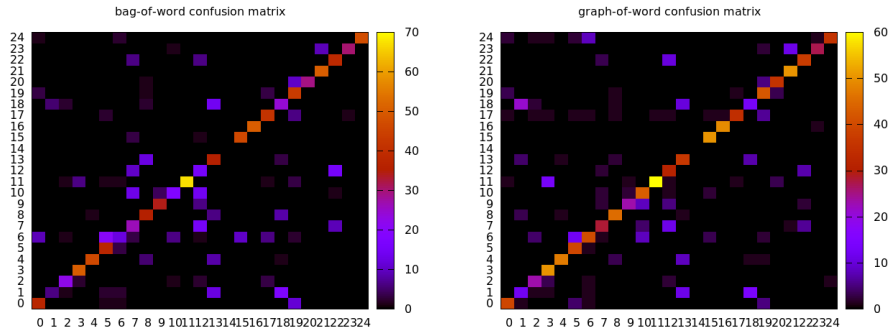


Figure 5.7: Confusion matrix using bag-of-views (left) and graph-of-views (right) representation based only on the Triangle Laser Descriptor. The graph-of-views one yields slightly better results.

5.4.2 Graph-of-Views versus Bag-of-Views

With laser descriptor only

Using this dataset we compared the precision obtained using bag-of-views and graph-of-views modeling. In this experiment only the TLD descriptor was used. The TLD dictionary threshold was set to 0.1. For each object, a model of each representation was built for each different trajectories. A truth set was selected by choosing for each object the model containing the most views, the models left were use as test set. The precision obtained are 71% using bag-of-views and 75% using graph-of-views. The figure 5.7 shows the corresponding confusion matrices. Graph-of-Views performs slightly better because it is more stable across the trajectories. This is partly because the distribution of views depends on the robot path and thus bag-of-views are less robust when comparing partial models. This experiment was done using the *intersection* metric (given in section 4.3 by equation 4.2 for bag-of-views, and in section 5.3.3 by equation 5.1 for graph-of-views) for both types of representations.

Influence of the dictionaries

To further investigate the difference between bag-of-views and graph-of-views, we performed the same experiment as previously (still using the intersection measure) but with all the descriptors described in the previous section and with two different dictionaries setup. Indeed, while working on the laser descriptor using bag-of-views (see chapter 4), we found it was difficult to fine tune the dictionary and that the system was very susceptible to its parameters. With this experiment we investigate how sensitive both representations are by performing the recognition using two setup of dictionaries. The first setup that we call coarse dictionaries is using high threshold for each descriptor types

Dictionary thresholds	Precise	Coarse
TLD	0.05	0.1
Surflet	0.04	0.05
TBGR	0.05	0.1
Color opponent	0.5	1.5
Size	0.0025	0.01

Table 5.2: Table of the dictionaries thresholds.

Setup	Bag-of-Views	Graph-of-Views
Using only TLD	71%	75%
Coarse dictionaries	59%	72%
Precise dictionaries	72%	81%

Table 5.3: Comparison of the recognition rate using bag-of-views or graph-of-views in different situations.

meaning a low number of views in the dictionaries and less discriminative power. The other setup called precise dictionaries uses a fine tuned threshold for each descriptor.

To obtain those setups, we recorded the average precision score and average number of views in the models given a dictionary threshold and one descriptor. This was done over a large range of thresholds and for each descriptor. We then selected the best possible thresholds while keeping a similar average number of views across the descriptors to obtain the precise dictionaries setup. The coarse dictionaries were obtained with the same procedure but selecting thresholds with worse precision results (see table 5.2).

The table 5.3 shows the resulting performances. We can see that the graph-of-views outperforms the bag-of-views in every setup, and furthermore it degrades less when using coarse dictionaries.

Several things are worth noting here. First of all, the bag-of-views precision doesn't improve much between using only the TLD and using all the descriptors. This is due to the fact that laser readings are much more frequent than the RGBD camera ones. This results in distributions dominated by laser descriptors in the representation and so other descriptors don't have much influence with the measure used. To overcome this would mean using a weighting-based measure which would be less flexible and difficult to fine tune. Since graph-of-views only count the presence or the

Setup	Bag-of-view	Graph-of-view	nodes only	edges only
Coarse dictionaries	59%	72%	62%	72.5%
Precise dictionaries	72%	81%	80%	76%

Table 5.4: Comparison of performances using the bag-of-views, the full graph-of-views, only nodes and only edges of this graph with the intersection metrics.

absence of the different views, it is not susceptible to this and thus using different sources of information bring more improvement to the precision.

Influence of edges

We now look at the contribution of the edges and nodes in the intersection similarity measure (see table 5.4). It is worth reminding that counting only the nodes is equivalent of using a binary bag-of-views. We can see that the nodes only perform better than the classic bag-of-views which confirms that using the distribution of views perceived in the trajectories is not efficient. We can also note that the nodes are more affected by the precision of the dictionary, but that in the case of a precise one they constitute most of the final precision score. However, using a coarse dictionary, the edges which are less affected become a more reliable source of information. In conclusion, both information are relevant in a practical case, and the graph-of-views model captures more than bag-of-views and is more suited to our application.

5.4.3 Modalities contributions

Another question we wanted to answer is how much does each descriptor contribute. Using the same experiment setup as previously, we computed the recognition precision with different rules. For each descriptor, we computed the precision when considering only this type of descriptor and when considering all of them without it. However, sometimes because models are partial, no intersection was present between the model being recognized and the set of labeled models for a particular descriptor. In this case the classification is not possible. In order to best assess the contribution of each descriptor, we computed two different precisions. The first one called real precision is the total precision including when classification is not possible. The second one called apparent precision is computed only when classification was possible.

We can see from the table (see table 5.5) that none of the modalities can perform better than the complete model on their own, especially with the coarse dictionaries where the individual precision are low. Removing one of the camera descriptor doesn't affect the total precision in the precise dictionaries setup, but the precision is always affected in the other setup.

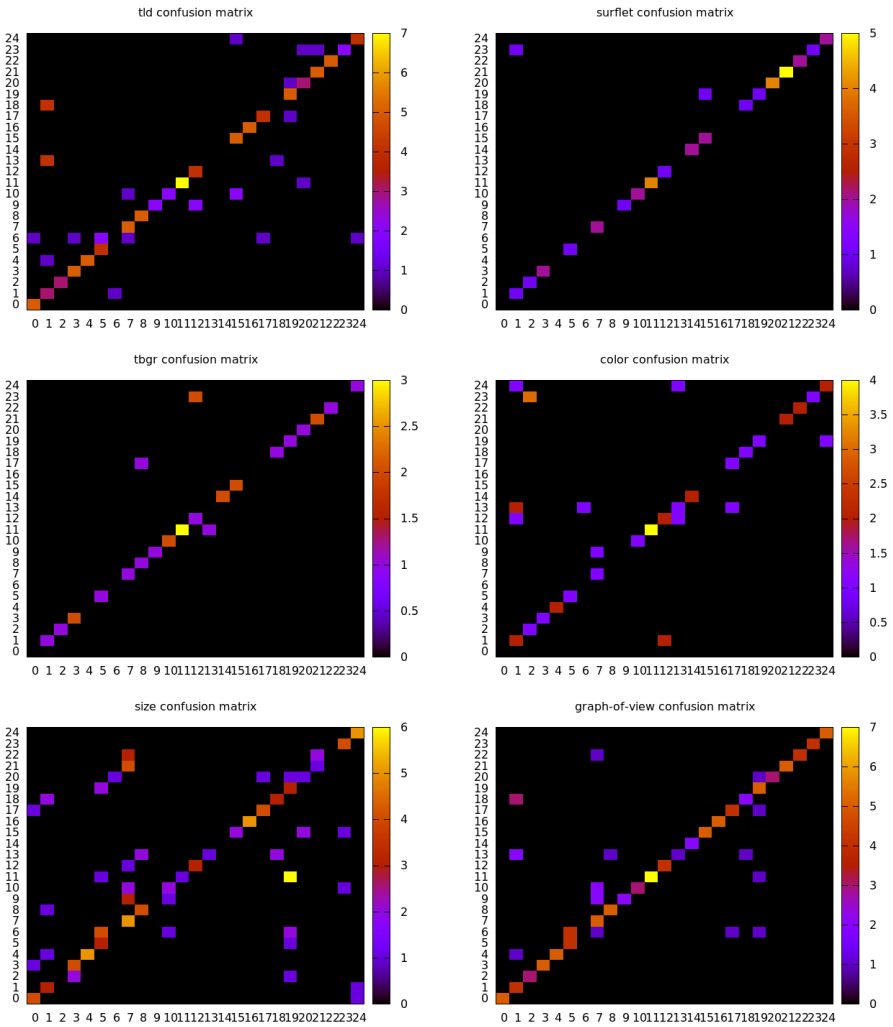


Figure 5.8: The confusion matrices obtained with different descriptors using the precise dictionaries.

	Precise Dictionaries		Coarse Dictionaries	
Rule	real precision	apparent precision	real precision	apparent precision
complete graph-of-views	80.8	80.8%	71.7	71.7%
only node	80%	80%	61.7	61.7%
only edge	75.8%	75.8%	72.5	72.5%
only TLD	73.3%	75.2%	57.5	58.5%
only surflet	29.2%	94.6%	25	30%
only tbgr	20.8%	86.2%	22.5	26.2%
only size	52.5%	53.4%	37.5	38.1%
only color	23.3%	65.1%	25.8	34.1%
without TLD	65.8%	65.8%	63.3	63.3%
without surflet	80.8%	80.8%	70.8	70.8%
without tbgr	80.8%	80.8%	70	70%
without size	79.2%	79.8%	63.3	63.3%
without color	80.8%	80.8%	68.3	68.3%

Table 5.5: The precision obtained when considering only one descriptor or when removing it. Two setup are shown, one with a precise dictionary for each descriptor and another with a coarse dictionary. Real precision is the total objective one, apparent precision is when cases with no match at all are discarded (see text for details).

Finally some camera descriptors' apparent precisions are higher than the complete model in the precise dictionaries setup. For these reasons we can see that if the descriptor and dictionaries are already very good, the graph-of-views doesn't add much, however, it can cope very well with the loss of precision in the dictionaries and the loss of information, which makes it a strong representation for multi-modal learning.

We see also from table 5.5 that the surflet is very discriminating when using a precise dictionary. It is often not possible to classify using only this one (29.2% of real precision), but when it is possible the classification is almost always correct (94.6% of apparent precision). The reason for this low real precision score can mostly be explained by the lack of overlap between the trajectories of an object and the small frequency of our RGB-D camera perception pipeline. The TLD descriptors is the most important descriptor in our system with the highest score in real precision when used alone (73.3%) and the highest loss when removed (loss of 15%). The less powerful descriptor appears to be the size computed from laser. However, even thought it is a very simple descriptor compared to the others, and that it doesn't capture much information about the objects, it is still useful to the system since there is a loss of precision without it. Alone it has a honorable score all things considered. Concerning our two color based descriptors, they show good results on their own and contribute to the recognition precision especially

Similarity function	Precision (%)
modalities_consensus	83.5
tf_idf	83.3
joint_modalities_intersection	83.2
normalized_jmi	83.2
intersection_size	82.2
inclusion_degree	82.2
per_modalities_inclusion	80.8
intersection_over_union	78.3
max_component	67.8
modalities_match	53.6

Table 5.6: Comparison of the similarity functions.

using the coarse dictionary.

Looking at the confusion matrices (see figure 5.8), we see that the confusions caused by our descriptors are very different which would explain why they all contribute to the recognition. However, some of the objects not classified by some descriptors are also confused by the other descriptors. For instance, it is the case for object 6 (cart) and 13 (black crate). This appears to be where the recognition results are the worst.

5.4.4 Comparison of different similarity measures

In this experiment we looked at the difference between the similarity measures described earlier in the sections 5.3.3 and 5.3.4. We use the precise dictionaries, and perform a similar experiment than previously only using different comparison functions.

From the table 5.6, we can see that our two one to many similarity functions performs better than the one to one similarity functions. This is the expected result since those functions take into account the additional information of knowing several objects. They are well suited for a classification context. The measure with the highest score is the *modalities_consensus* which is slightly better than the tf_idf.

The best one to one measure is the *joint_modalities_intersection* and is only slightly below the tf_idf. Note that *joint_modalities_intersection* and *normalized_jmi* are exactly the same in this classification context, only the second one is normalized. This also true for *intersection_size* and *inclusion_degree*.

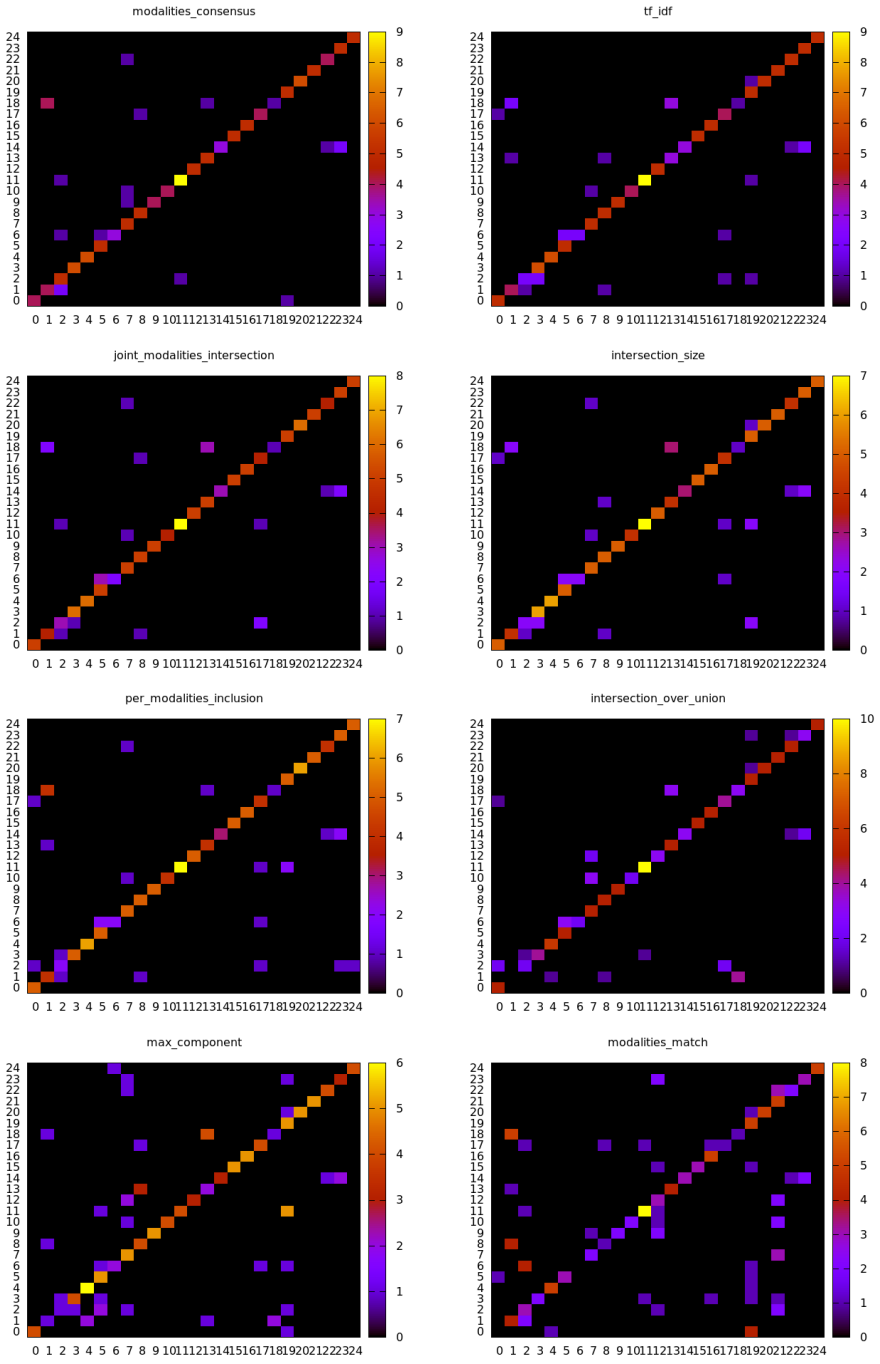


Figure 5.9: The confusion matrices obtained with the different similarity measures.

Number of models per label	1	2		3		4		5	
Setup		Merged	Not	Merged	Not	Merged	Not	Merged	Not
normalized_jmi	**79.0**	89.2	88.1	91.6	**89.3**	**95.6**	93.0	**96.8**	93.2
inclusion_degree	78.4	**89.4**	**89.4**	91.4	88.9	94.6	93.3	**96.8**	93.7
tf_idf	78.0	89.2	89.2	**91.8**	89.1	**95.6**	**94.3**	96.3	**94.2**
modalities_consensus	78.4	87.6	86.7	89.7	**89.3**	94.6	92.4	96.3	93.2

Table 5.7: The precision obtained with different measures when merging models or not when more than one model is present per label in the truth set.

We can see in the confusion matrices (see fig. 5.9) that some similarity functions have similar confusions and some others very different. In particular the functions *intersection_size* and *modalities_match* don't confuse many of the same objects and combining both of them in *joint_modalities_intersection* yields better results.

5.4.5 Merging study

In the previous experiment, there were only one model per type of object being considered when performing classification. In this experiment we study how the classification improve when more models are given per real object. We computed a classification score when one model per label is given, then two and so on up to five. Each time the labeled models are selected randomly. The results given are an average over ten iterations. When more than one model per label is present, we computed a classification score both when the models were merged and not. In both cases, to perform the classification we simply find the most similar labeled model to the candidate one and take this label as the candidate category. The results are reported in the table 5.7.

As expected, the more there are labeled models per real object, the better the results are whether the models are merged or not. We can see that the classification using merged models is always better or equal to using separate models. With two labeled models per real object, the different is slight (up to 1.1% with *normalized_jmi*), but is significant with five labeled models (up to 3.6% with *normalized_jmi*). Those results indicate that merging partial models to obtain a complete one is beneficial for the object recognition performances.

Finally, in this experiment, there is no similarity function that always performs better than the others and the one to many functions don't appear to be the best anymore. In this experiment, with five merged labeled models (which correspond to the best classification), the difference between the similarity function's results is at most 0.5%. In the previous experiment, the difference between the same four functions was at most 1.3% (see table

5.6). Those differences are not important and it is not clear which of those similarity function is in fact better.

5.5 Conclusion

In this chapter we presented an original representation that we call graph-of-views to perform multi-modal object recognition. To assess its performance we created a dataset which contains 6 different trajectories of the robot avoiding 25 different objects. Some objects need both shape and color information to be differentiated and some are difficult to perceive with the sensors. So, from each trajectory in the dataset an object model or representation can be extracted but the information is only partial.

Using this dataset we have shown that our representation performs better than bag-of-views to recognize the objects from partial information, which means it generalizes more across the trajectories. This is very important in a robot because sometimes datasets are not available for supervised learning and perception of objects through sensor during a navigation tasks is almost always partial.

Furthermore, we have shown that using our representation it is possible to perform multi-modal object recognition in a very easy and robust way. To do so we perform different experiments on the dataset using 5 descriptors extracted from two different sensors.

Adding a descriptor in our representation only means choosing a threshold and a distance to produce a dictionary (assuming the one presented in chapter 4 is used), which is not much overhead compared to other methods, showing its simplicity. Also, we have shown in our experiment that the recognition can benefit from a new descriptor (such as the size) even if its dictionary or itself are not very precise, demonstrating the robustness of our representation. Finally, we have shown that the graph-of-views works very well even when one of the sensors or descriptors are not available, which means not only the representation works well when only one side of an object has been perceived, it also works well when it is ill-perceived by one of the sensors in use.

Chapter 6

Unsupervised and incremental object learning

This chapter describes our contribution to the unsupervised and incremental object learning using the graph-of-views approach presented in the previous chapter.

6.1 Introduction

In chapter 1, we argued that a domestic robot should have the capacity to learn and adapt to the environment continuously without expert human supervision. It entails that a new situation should not prevent the robot from performing its tasks and that the robot has to handle the situation on the fly without waiting for human intervention. A simple solution could be to record the new events for further processing, but, as the robot will operate continually, it can not simply record everything because the amount of data would eventually be too high. Therefore, the robot should have the capacity to only record relevant events, i.e., events that bring new information about the objects of the environment.

In order to illustrate this, let's say that the owner bought a completely new object. The robot should understand that it is a new object, record its particular perception of it and memorize its model. Later observations of the object should also be recorded and incorporated to its model if necessary in the case the initial model is not complete. This scenario can be solved by a fast unsupervised and incremental learning process that we are going to define more precisely.

Firstly, unsupervised learning (specifically clustering in our case) can be seen as finding a hidden structure in unlabeled data. It can also be defined as categorizing data without ground truth or human given examples. It is therefore distinct from supervised learning because there is no labeled examples and from reinforcement learning because there is no reward to

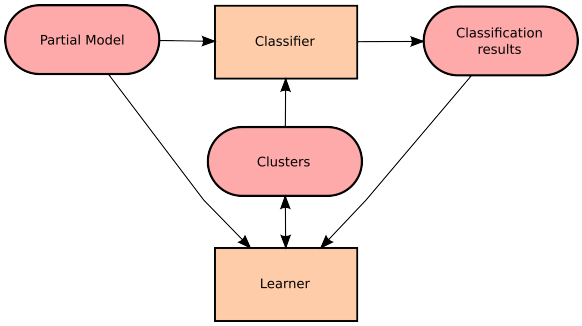


Figure 6.1: Flow chart of the learning process. The classification takes as input the model of newly perceived object and the categories present in memory, and outputs how well the object is recognized. The learner use the recognition results to incorporate the new model in its memory.

guide the learning. Secondly, incremental learning is the idea of gradually learning when samples are perceived while adjusting what has been learned according to this new observation. Finally, by fast we mean compatible with the operation time scale of the robot, i.e., that the robot should take at most a few seconds to issue a response when encountering a new situation.

In order to perform fast, unsupervised and incremental learning, our approach can be split into two distinct functionalities and be represented by the flow chart shown with figure 6.1. The first functionality is classification: it takes as input a partial model of an object being perceived, and a set of representatives models of the different already known categories. The output is the possible category of the object being perceived with a score of confidence. The second functionality is learning: it takes as input the perceived model and the result of the classification. It refines the different categories if needed by incorporating the information if it is considered new, and returns a new set of representatives models. An important feature is that our model should be capable of adding new categories automatically when a new object appear in the environment, without requiring the information that this is indeed a new object.

6.2 Related work

The clustering problem has been addressed in many contexts and many different techniques exist. An extensive review is given in (Jain, Murty, and Flynn 1999) in where they present a survey of the different families of clustering methods and gives a clear definition of the most popular ones. When confronted with a clustering problem, different aspects of the data and the problem itself will decide which kind of clustering can be used. Those aspects or constraints are principally the observation representation, the

similarity measure, the cluster representation and of course the amount of data to be processed. When those constraints are defined, the remaining concerns when choosing a clustering algorithm are its stability, its plasticity, its complexity and its order-dependence (in case of incremental clustering). Since our problem is to perform incremental clustering, in the following we will only review this type of clustering algorithms.

The data mining community has to deal with huge datasets that don't fit in a computer memory so incremental algorithm is an important focus in this community. A popular algorithm is Birch (T. Zhang, Ramakrishnan, and Livny 1996) which can summarize the dataset in a tree incidentally with a linear complexity. It is efficient but the entire clustering process supposes a finite dataset with some refinement steps, so it can not be used continuously in an open-ended fashion, which is not suitable for us. In (Young et al. 2010), they proposed a clustering algorithm that employs the "winner-take-all" paradigm and that can continuously update centroids. The algorithm is modest in resources and can as such deal with large amounts of data efficiently. However, the number of centroids is fixed which is an important limitation for us.

The most closely related work is (Paul, Rus, and Newman 2012). Their purpose is to have a robot capable of summarizing long periods of visual perception. They use the star clustering algorithm (Aslam, Pelekhov, and Rus 2004) to compute a topic-driven organization of the robot's image collection. This paper and the star clustering algorithm met most of our criteria. It doesn't assume the number of clusters is known, it is incremental and efficient enough to be performed on-line. However, some differences made us study another solution. They want to summarize the whole perception when we only want to record observation of new experiences. They compare images and cluster them into topics for further processing but no output is readily available for the robot to take decisions. In our problem, the robot perceives objects partially, and needs to group those partial observations to form models so that the objects can be recognized. Not only do we want to extract the most relevant information during the robot working cycle, we also want it to learn from this information directly.

6.3 Unsupervised object learning

As explained above, there are two different functionalities in our unsupervised object learning approach: classification and learning. The classification takes as input the model of newly perceived object and the categories present in memory, and outputs how well the object is recognized. The learner use the recognition results to incorporate the new model in its memory.

In order to formalize these algorithms, let O be the set of real objects being observed by the robot. Let M be the set of all partial models m_i in

memory. In this context m_i is a bag-of-views or a graph-of-views as described in chapter 5. Let note C a set of clusters c_j with a cluster being a set of models: $c_j = \{m_0, ..., m_k\}$. Finally, let note g a newly perceived partial model which we will be referred as the candidate.

Given that, the goal of learning is to produce C such as each $c_j \in C$ is only composed of models from a single real object o_i, and such that there is only one cluster per real object. On the other hand the goal of classifying is to find c_j such as g and c_j are both representing the same real object o_i, or to detect that g comes from a newly observed object if no cluster is representing it. We will now detail our proposition for these two tasks.

6.3.1 Classification

Our approach to perform classification is quite simple and general and rely on solving the following equation:

$$\begin{cases} class\ of\ g = arg\ \max_j(similarity(g, c_j)) \\ score = \max_j(similarity(g, c_j)) \end{cases} \tag{6.1}$$

The similarity measure that we used is the inclusion of g with the union of all m_i of c_j, which we can write as follows:

$$inclusion(g, c_j) = \frac{|g \cap (\bigcup_{m_i \in c_j} m_i)|}{|g|} \tag{6.2}$$

In the case where the known object models are complete and not ambiguous, this approach will recognize the current object or determine that the object is new if the score is low. However, since in our scenario all the models are considered partial and the amount of knowledge on the objects is unknown, there isn't much that can be said from this score alone. If the score is low it doesn't necessarily means the robot is perceiving a new object it hadn't seen yet, it could be that it is perceiving new aspects of an already known object. It is also possible to obtain several objects with high score in the case that several objects share a similar appearance from a given point of view and that their models are not complete.

These two examples show that it is not efficient to base the learning on this classification result alone. We however tested a simple approach based on this idea (see next section), but we developed a second approach using more information (see section 6.3.3). So in our classification algorithm, instead of computing only the most similar model, we return the ordered set of models based on their similarity to the candidate (see algorithm 3). This result will be used in the learning algorithm described later, but, at a given point in time, the best classification we can obtain on a particular object being perceived remains the identity of the most similar model in memory as defined previously.

Algorithm 3 Classification

Input: g the candidate partial model
Input: C the current set of clusters
Output: the class of g
Output: the score of the classification

```
1: function CLASSIFY(g, C)
2:     result := φ
3:     for all c_j ∈ C do
4:         append {c_j, inclusion(g, c_j)} to result
5:     end for
6:     sort result based on inclusion
7:     return result
8: end function
```

6.3.2 Sequential Clustering Algorithms

Our first attempt to solve the problem of incremental unsupervised learning was to use a simple sequential clustering algorithm (see algorithm 4). In this algorithm, when a new candidate model is acquired, the classification score is computed. If the score is high, then the model is recognized and is used to update the corresponding cluster, otherwise the model is new information and lead to the creation of a new cluster.

This method was used with some success in (Duceux and Filliat 2014) with the bag-of-views and the PPLD laser descriptor (see chapter 4). The details of the experiments conducted are given in the section 6.4. However, some limitations were encountered that lead us to develop a new approach.

Algorithm 4 Sequential Clustering Algorithm

Input: g the candidate partial model
Input: C the current set of clusters
Output: the updated set of clusters C

```
1: res := CLASSIFY(g, C)
2: if res > recognition_threshold then
3:     add g to res.cluster
4: else
5:     create a new cluster c_g with g
6:     add c_g to C
7: end if
8: return C
```

This kind of algorithm usually works well when the purpose is to separate simple samples into categories. However, in our case, we have partial observations that we want to regroup into complete models of object. In this

situation, as mentioned in section 6.3.1, a very low score (near 0) doesn't necessarily occur when the perceived object was never seen before, and a very high score does not only happen for the correct object if the models are incomplete or ambiguous. This lead to two types of errors.

We named the first type of error *duplicate clusters* when two clusters represent the same objet. For example, let o be an object with a complex shape. Imagine the first time the robot perceives the object o, it only perceives the front of the object. This leads to a model $front$ which is clustered into a cluster c_{front}. Then, imagine the robot sees the same object o but from the back this time, giving it a model $back$. Since the object has a complex shape, it could happen that there is no similarity between $front$ and $back$, in which case another cluster c_{back} would be created, leading to two different clusters representing the same object. Thus, a low classification score can mean either the robot sees a new object, or a new side of a previously seen object. There is no way to know which one it is using only an appearance similarity measurement.

The second kind of error is called *corrupted clusters* and happens when models of two different objects are grouped together. Because of noise, a threshold is set to tolerate a certain difference between a model and a cluster to allow for recognition (line 2 of algorithm 4). However, some objects are fairly similar in nature, like a red chair and a blue chair of the same model and their similarity could be above the selected threshold. Thus, a high classification score doesn't ensure that we can safely add a model to a cluster.

In sight of that, given this kind of algorithm, the *recognition_threshold* must be carefully set to avoid any errors and to not limit the learning capacities. If the threshold is too high, then the algorithm will create many duplicate clusters and fail to update correctly the ones in memory.On the other hand, if the threshold is too low then the algorithm will corrupt clusters in memory and fail to create new clusters when new objects are encountered (see results in section 6.4).

However, there is a silver lining. Duplicate clusters are not a big problem because usually one of them eventually gets bigger than the others and take over. This means that it should be possible to filter the memory after a while to remove the its small duplicates. Moreover, using this method with active perception could ensure better results and could diminish some of the limitations. For instance if the robot gets a low classification score, instead of creating a possibly duplicate cluster, a possible behavior would be to get a closer look at the object and build a stronger model of it. Then the classification score of this richer model would have more meaning.

Regarding the corruption of clusters, we envisaged to develop an off-line method to post-process the clusters. Since an off-line method wouldn't have to be incremental and fast, maybe it would be possible to perform a better clustering and thus ensuring the robustness of the robot overtime. However, instead of developing such an algorithm, we came up with another line of

reasoning and a better incremental algorithm, which is presented in the next section.

6.3.3 Graph clustering approach

We have seen in the previous section the difficulties related to our problem of clustering partial models into complete ones. In the development of a new solution to this problem based on a graph clustering approach, we considered two general ideas and made two assumptions.

Firstly, since it is difficult to disambiguate between observing a new object or perceiving a new aspect of an already seen one, duplication errors are inevitable. Likewise, corruption errors are also inevitable because different objects can share strong similarities between themselves, and often enough objects are ill-perceived. Thus, the solution needs to be flexible and take only soft decisions that can be changed when new information appears. In other words, it necessitates the capacity not only to update clusters but also to merge, split or modify them. The assumption made here is that eventually, as the robot observes many times the different objects and increase the models completeness, observed similarities between models from same object will become stronger than the similarities between models of different objects.

Secondly, given a long time period, some of the partial models acquired would contain fairly complete information about the objects, especially if the robot performs active perception. Those larger models would be better than models obtained from a clustering of smaller ones, because less prone to errors and noise. Moreover, as it would be inefficient to keep in memory all observations ever made of one object, those larger models could be preferred over the smaller ones. So our idea is, instead of focusing on the clustering, to concentrate on selecting the good models and removing the bad ones. We choose to implement such a solution by constraints on its memory footprint and by filtering models as new one comes. We therefore make the assumption that at some point in time, the robot will be able to represent all its knowledge about an object by a small set of large (even if still partial) models.

Based on these two assumptions, our idea is to maintain and update a graph of strong similarities between observations where clusters are formed by the disconnected part of the graph (see figure 6.2). To update the similarity graph, we compare the new observation to a subset of selected [1] models in memory, update the links between models and filter them (see algorithm 5 and figure 6.3).

Each time the similarity between a new observation and a model is sufficiently high, a potential link is added between the two of them (lines 16 and 17). Then, the links of the two models are filtered to keep only the *max_edges_threshold* best ones (lines 18 and 19). This way, a new

[1] How we select this subset will be explained later

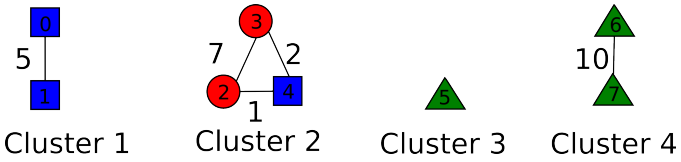


Figure 6.2: Illustration of a graph of similarity. The different drawings (the blue squares, red circles and green triangles) represent observations of different real objects. They constitute the nodes of the graph. The number in the nodes identify them for clarity. The robot don't know which real object each observation represent, it needs to cluster them using the similarity between the models. A link between two drawings represent an edge of the graph and the similarity between the models. The number attached to it represent how much they are similar. In our graph of similarity, only the strongest similarities are kept. The disconnected parts of the graph constitute the clusters. In this example there are four clusters. Cluster 1 and 4 are correct. The cluster 2 is a corrupted one since it contains models of two different real objects and cluster 3 is a singleton since it contains only one observation. Cluster 3 and 4 are a duplication because they represent the same real object.

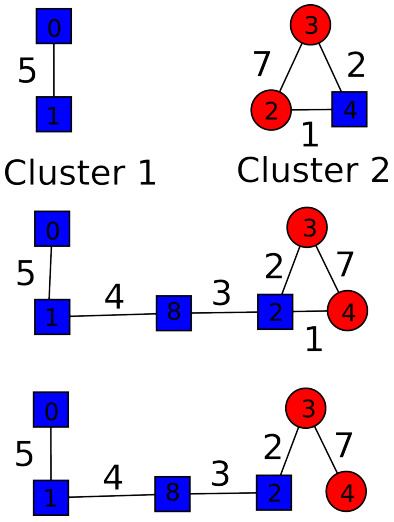


Figure 6.3: Illustration of the graph of similarity updating algorithm. A new observation is made (model 8) which is compared against cluster 1 and 2. As a result two links are created with model 1 and 2. The links of the affected nodes are then filtered according to the threshold *max_edges_threshold* (which is two in this example) by removing the worst one when the limit is reached. In this example the link between model 2 and 4 is thusly removed.

Algorithm 5 Similarity graph updating

Input: g the candidate partial model
Input: $G = (N, E)$ the similarity graph
Input: M the subset of models to be compared
Output: an updated similarity graph

function UPDATE_SIMILARITY_GRAPH(g, G, M)
    add g to G
    for all $m \in M$ **do**
        if $inclusion(g, m) > high\_inclusion\_threshold$ **then**
            discard g
            attach all models linked to g to m
            FILTER_EDGES(m, G, $max\_edges\_threshold$)
            return G
        end if
        if $inclusion(m, g) > high\_inclusion\_threshold$ **then**
            discard m
            attach all models linked to m to g
            FILTER_EDGES(g, G, $max\_edges\_threshold$)
            continue
        end if
        if $similarity(g, m) > low\_similarity\_threshold$ **then**
            add edge $g - m$ with weight $similarity(g, m)$ to E
            FILTER_EDGES(g, G, $max\_edges\_threshold$)
            FILTER_EDGES(m, G, $max\_edges\_threshold$)
        end if
    end for
end function

function FILTER_EDGES(m, G, $threshold$)
    $edges := \{\forall e \in E, \forall m_i \in N | e = m - m_i\}$
    while $|edges| > max\_edges\_threshold$ **do**
        remove $\underset{e \in edges}{arg\,min}(e.weight)$ from E and $edges$
    end while
end function

observation can bind two clusters and remove a duplication or "steal away" a model from a cluster and reduce corruption. By limiting the amount of links between models and the number of models per cluster to *max_models_per_cluster_threshold* (as will be explained later), the similarity graph will self organize into stable clusters that contains large models that are strongly similar. After different trials we found that the best limit of edges per node is the maximum number of models per cluster minus one: $max\_edges\_threshold = max\_models\_per\_cluster\_threshold - 1$. This number of edges is exactly the required number to form fully connected components (or cluster) of the similarity graph. Using this threshold makes the clusters more stable and hence the graph too. This graph representation offers the kind of flexibility we mention earlier as a first requirement. By adapting the structure of the graph locally with each new observation, we modify the clusters in memory by merging, splitting or modifying them.

When updating the similarity graph, an interesting special case may occur: we could find an observation completely included in another (line 4 in algorithm 5). In this case there is no doubt that such an observation provide no added information to the graph and can be discarded. When doing so, all the edges attached to the node being discarded are added to the other node (line 6) and then filtered (line 7) to avoid loosing valuable information. This is actually helpful to remove short observations containing a small amount of views or with simple objects. It is more rare otherwise to have a model in memory included in a new perception (line 10) due to noise and the unlikeliness of observing exactly a portion of a sequence already in memory with complex objects and long trajectories.

Algorithm 6 Cluster reduction

Input: G the similarity graph
Input: $c = (M_C, E_C)$ a cluster and subgraph of G
1: **if** $|M_C| > max\_models\_per\_cluster\_threshold$ **then**
2: remove $\underset{e \in E_C}{arg\,min}(e.weight)$ from G and c
3: res = compute resulting clusters
4: **for all** $c \in res$ **do**
5: reduce cluster c
6: **end for**
7: **end if**

As said previously, it would be inefficient to keep in memory all observations ever made of one object. Instead we want to keep only the best of them. To do so, we limit the number of models contained in one cluster to *max_models_per_cluster_threshold*. When the graph of similarity has been updated, the disconnected components of the graph form our clusters. For each cluster that have been updated with the arrival of a new observation, if

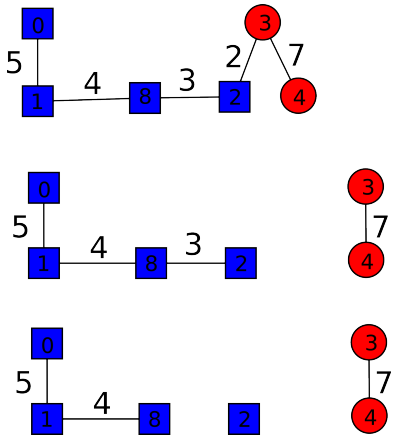


Figure 6.4: Illustration of the cluster reduction algorithm. In this example a cluster contains more model than the limit which is 3. We successively remove the weakest link until all resulting clusters are small enough. In this case, first the link between the model 2 and 3 is removed producing two clusters. Then the link between 2 and 8 is removed. The result is three smaller clusters. In this example, the mechanism removed a corruption but created a duplicate singleton.

the limit of models is reached, the cluster have to be reduced.

Our initial idea was to remove a selected model. To do so, different approach were envisaged. We tried removing the model the least included in the cluster (i.e., the one with the weakest edges to the other models) but we found that, even though it is a good criterion to remove models causing corruption, if the cluster is valid it tends to remove model providing much valuable information (i.e., information different from the other models) about the object thus ”weakening” the cluster. The opposite is also true: removing the model the most included in the cluster is a good criterion to reduce a correct cluster but tends to fail on corruption. The other idea we tried was to remove the shortest observation, i.e. the one made along the shortest trajectory and thus containing less information than the others. This solution is actually quite good at the beginning of the learning to retain interesting models. However, this solution is not efficient to choose between models of good quality and tends to discriminate objects that are difficult to perceive.

Finally, the solution which gave the best results was simply to remove the weakest links in the cluster until it splits and that all resulting clusters where under the limit (see algorithm 6 and figure 6.4). This solution was found better than the others mainly because it tends to make the learning process more stable. Most of the time, reducing cluster like that will produce singletons (models with no links) by rejecting one model of the cluster. We do not remove the new singletons right away because it would prevent newly

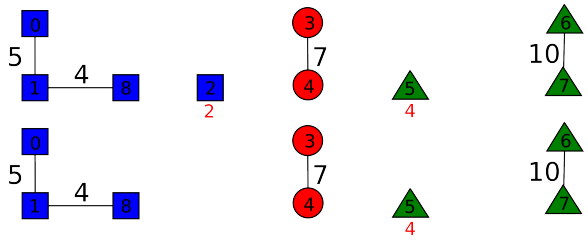


Figure 6.5: Illustration of the singletons reduction algorithm. In this example the graph contains two singletons: model 2 and 5. The maximum number of singletons is reached (which is 1 in our example), so a singleton is removed. We remove the one containing the less information (noted in red in the figure): model 2. To measure how much information is contained in a model, we record the length of sequence originally observed. In this case the model 2 was observed over a trajectory of 2 meters, and model 5 over 4 meters which means model 5 contains more information than model 2.

observed objects to be learn correctly. Instead we developed a mechanism to handle singletons which we will detail after. Reducing the cluster size answers partially the second requirement of constraining the size of the memory because it prevent clusters from growing arbitrarily large and because most singletons will eventually be discarded. However, the number of cluster itself is not limited so that the number of objects our algorithm can learn is theoretically not limited.

There are two reasons why limiting the number of model per cluster is a valid choice. Firstly, our unsupervised learning system is supposed to function on a domestic robot days after days. So, eventually a new object in the environment will be observed correctly many times, especially if the robot can use active perception and a subset of these correct observation will correctly represent the object. Secondly, we have shown in chapter 5 good recognition results with only five models merged per object which were extracted from common trajectories. Thus, it is reasonable to think that a small number of good observations per object is enough to model them completely.

Algorithm 7 Singletons reduction

Input: G the similarity graph
Input: S the set of singletons
1: **while** $|S| > max\_singletons\_threshold$ **do**
2: remove $\underset{s \in S}{arg\,min}$(length of observation) from G and S
3: **end while**

As explained before, when reducing a cluster, our algorithm can produce a singleton. Noisy observations may also result in an unbounded amount

of singletons which needs to be filtered. However, they can not be immediately discarded because they might be the first observation of a newly seen object. So a mechanism to handle the number of singletons overtime is required. The solution we found is to tolerate a certain amount of singletons (*max_singletons_threshold*), and when the limit is reached, select one for removal (see algorithm 7 and figure 6.5). Remember that, at the origin of our models, we have sequences of perceptions. A simple measure of the importance of this perception can be derived from the length of this sequence, the longer sequences being potentially more informative. We therefore choose to remove the singleton corresponding to the shortest sequence, which are more likely to be noisy observations or to contain less information.

Algorithm 8 Unsupervised and incremental object learning

1: Find the k closest clusters to candidate model m (classify)
2: **if** highest score near 1 **then**
3: add m to the corresponding cluster, update it and reduce it
4: **else if** highest score near 0 **then**
5: add m to the graph as a singleton
6: **else**
7: For those k clusters, update the graph structure with the new model
8: For all singletons, update the graph structure with the new model
9: Perform a new clustering on the models concerned
10: Reduce new clusters if necessary
11: **end if**
12: Reduce the number of singletons if necessary

Let's now detail the overall algorithm that integrate a new model in the graph using the previously defined algorithms. When a new observation is made, three cases are possible (see algorithm 8). The simplest one (line 4) is if there is no similarity between the new observation and any of the current cluster. In that case there is nothing else to do but to create a new cluster with the observation by adding it in the similarity graph with no edges.

The second case (line 2) is when the new observation is totally included in a cluster. A naive idea would be to consider this observation of no value since it brings no new descriptors into the cluster and simply discard it. However, there are two good reasons for not doing that. Either the new observation could be of better quality than one contained in the cluster, or, the cluster could be a corrupted one and the new observation could help solve the problem. So, in this case the best choice to add the observation to the cluster (see algorithm 5) and then remove the worst one from the cluster (see algorithm 6).

The last case (line 6) is when none of the above applies. Then the observation is used to update the similarity graph (see algorithm 5), the resulting new clusters are computed (see algorithm 9) and reduced (see

algorithm 6). As the number of models in the graph is not limited, this computation can become expensive as the graph grows. That is why the new observation is only compared to the k most similar clusters and all the singletons.

Using the classification result to select which models are to be compared significantly improve the computational performance of our approach. In the worst case, a new observation is compared against all cluster (which are merged models so the comparison is the same cost as a model), then against the maximum number of model per cluster multiplied by k, and finally against the maximum number of singletons allowed.

Algorithm 9 Cluster extraction

Input: M a set of partial models
Input: E the set of weighted edges between models
Output: a clustering C of models

```
1:  function COMPUTE_CLUSTERS(M, E)
2:      C := {}
3:      queue := create queue from M
4:      while |queue| > 0 do
5:          c := create new cluster
6:          m := first of queue
7:          RECURSIVE_ADD_TO_CLUSTER(m, c, E, queue)
8:          add c to C
9:      end while
10:     return C
11: end function

12: function RECURSIVE_ADD_TO_CLUSTER(m, c, E, queue)
13:     if m ∈ c then
14:         return
15:     end if
16:     add m to c
17:     remove m from queue
18:     for all e|m − m_j ∈ E do
19:         RECURSIVE_ADD_TO_CLUSTER(m_j, c, E, queue)
20:     end for
21: end function
```

The role of the last algorithm required by our approach is to retrieve a cluster from the graph or a subgraph. To do so we use a recursive depth-first-search algorithm (see algorithm 9). At the beginning, all models (belonging to the updated clusters) are in a queue to be processed. For each model we create a new cluster and add recursively all model linked to it, until all model are assigned a cluster. Then for all the newly created cluster, we merge all

models contained in it and output it for the classification algorithm (see algorithm 3).

To summarize (see figure 6.6), when a new model is acquired, we first compute the classification score. If the score is good, then we only update the most similar cluster. If the score is low, we add the model to the graph as a singleton and then filter them. If the score is in-between, then we update the graph by comparing the new model to the k most similar cluster. Then each affected cluster is reduced if necessary and the singletons are filtered. Finally, the clusters a recomputed and merged into a single model for the next classification .

If our hypotheses are right, our algorithm deals with corruption (as illustrated in figure 6.6) and duplication errors, it doesn't require to know how many real objects have been observed, it doesn't limit the number of real objects that can be learned while keeping only a few best observations for each real object. Furthermore we tried to make the algorithm efficient and its complexity is linear in the number of cluster. Experimental results are given in section 6.4.2 and a comparison with the sequential algorithm is given in section 6.4.3.

6.4 Experimental results

6.4.1 Sequential clustering on Bag-of-views using the PPLD descriptor

Sequential clustering on Bag-of-views

In this section, we will first present the results we obtained with the simple sequential clustering algorithm presented in section 6.3.2 that have been published in (Duceux and Filliat 2014).

Incremental learning

For this first experiment, we tested incremental learning using trajectories sampled from the database presented in section 4.4 in order to have a ground truth on the object identity and be able to assess the quality of the resulting clusters. In order to set the threshold for integration in a cluster, we studied the behavior of the clusters in memory when varying this threshold. Figure 6.7 shows that when the threshold is low, few clusters are created and they are mostly corrupted, i.e., they contain models from different real objects. On the other hand, when the threshold is too high, every tracking results into a cluster being added to the memory, and few clusters are updated. From these results, we set the threshold to a value of 0.8 to optimize the number of correct cluster while minimizing the number of corrupted ones.

In order to see the resulting distribution of clusters in memory, we constructed figure 6.8. The database was split into 85 trackers with varying

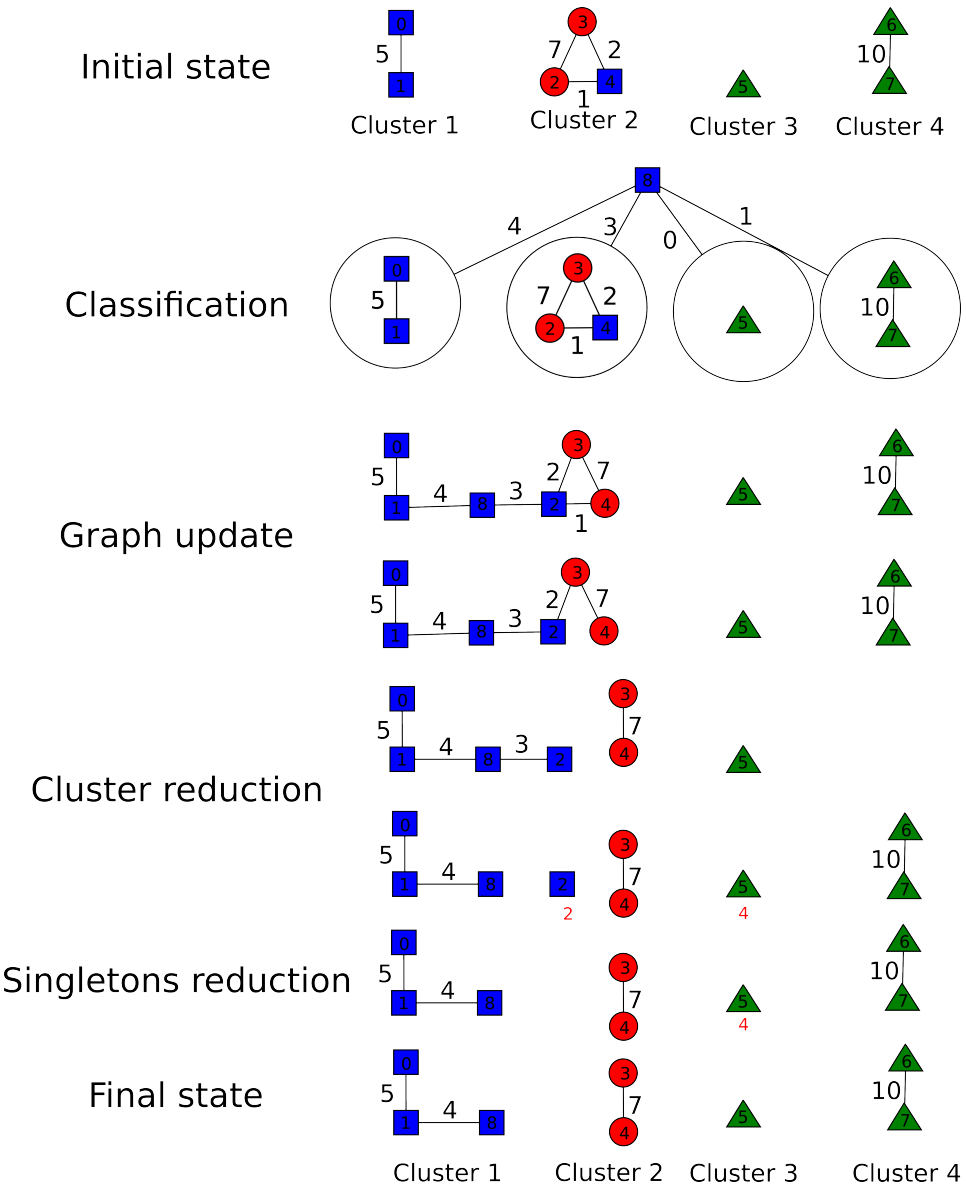


Figure 6.6: Illustration of a complete learning step. In this example, a new observation (in blue) added to the graph is able to remove the corruption from cluster 2 and improve the quality of cluster 1.

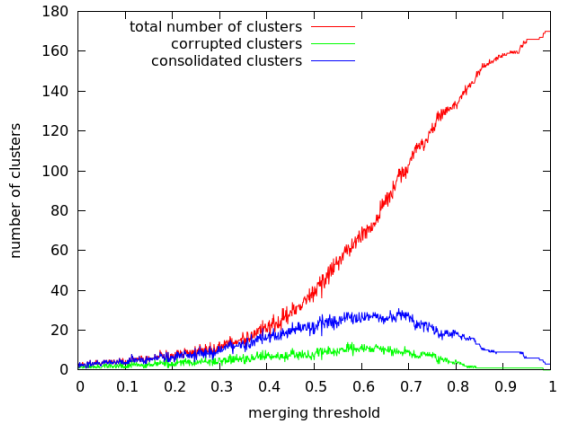


Figure 6.7: Number of clusters as a function of the merging threshold. In red the total number of clusters in the memory. In blue, the number of clusters that have been updated successfully. In green, the number of corrupted clusters (cluster containing models coming from different objects).

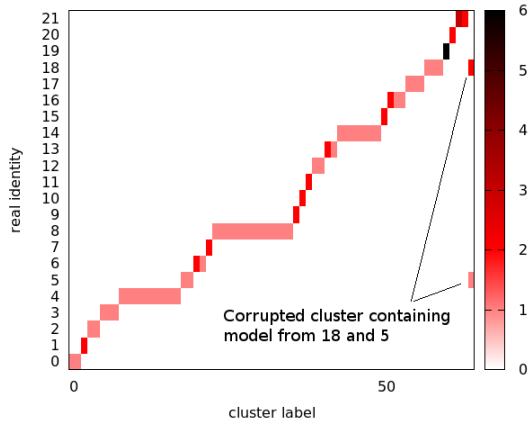


Figure 6.8: Number of clusters in the memory by real objects identity and their size.

size (between 1 and 80 according to the live experiments, see section 6.4.1). The matrix was built by picking randomly one tracker from the dataset, and clustering it as explained in section 6.3.2, until the dataset was empty. For clarity reasons, the resulting set of clusters was ordered. In this case, we obtained 51 clusters with a single model and 13 clusters with multiple models, with 1 corrupted. Some objects resulted in few clusters in the memory (objects 1, 7, 9, 10, 11, 15, 19 and 20). Which means that the first time the object was seen, the resulting model was a good representation and that the object is easy to recognize. Other objects are more difficult to recognize from partial models and result in several clusters in the memory.

Live experiment

As explained in the section 3.5.2, the system was implemented on the robot in real-time. To evaluate it, we recorded the behavior of the system on 8 trajectories in a room containing 8 different objects that were moved between the robot trajectories. The system resulted in 125 different models, resulting in 16 clusters, among which 2 were corrupted. This result shows that our algorithm is efficient at grouping together models of the same object, dividing by almost 8 the total number of models, while limiting the number of corruptions.

On the trajectories that we studied we obtained a maximum of 81 views and an average of 16 views in each model, depending on the duration of the tracking of the objects. Figure 6.9 shows an example of a trajectory with the associated views recorded with two different objects.

Figure 6.10 show two pure clusters of models constructed for two different objects. The views of each objects are plotted on the trajectory of the robot during its creation in order to show the diversity of the trajectories that make it possible to recognize an object.

6.4.2 Graph clustering on multi-modal Graph-of-views

In this section, we now report experiments performed with the graph clustering approach described in section 6.3.3.

Preliminary results

One of the hypothesis for our graph clustering to work is that even though it is not possible to find a suitable threshold to separate all models, most observations of the same objects will tend to be more similar than observations from different objects. In order to confirm this hypothesis, we constructed the maximum spanning tree of all the models of the second dataset (see section 5.4.1) using the joint modalities intersection similarity (see section 5.3.3). Figure 6.11 shows the obtained graph. In the ideal case, this graph should contain a minimum of 24 edges between different labels since there

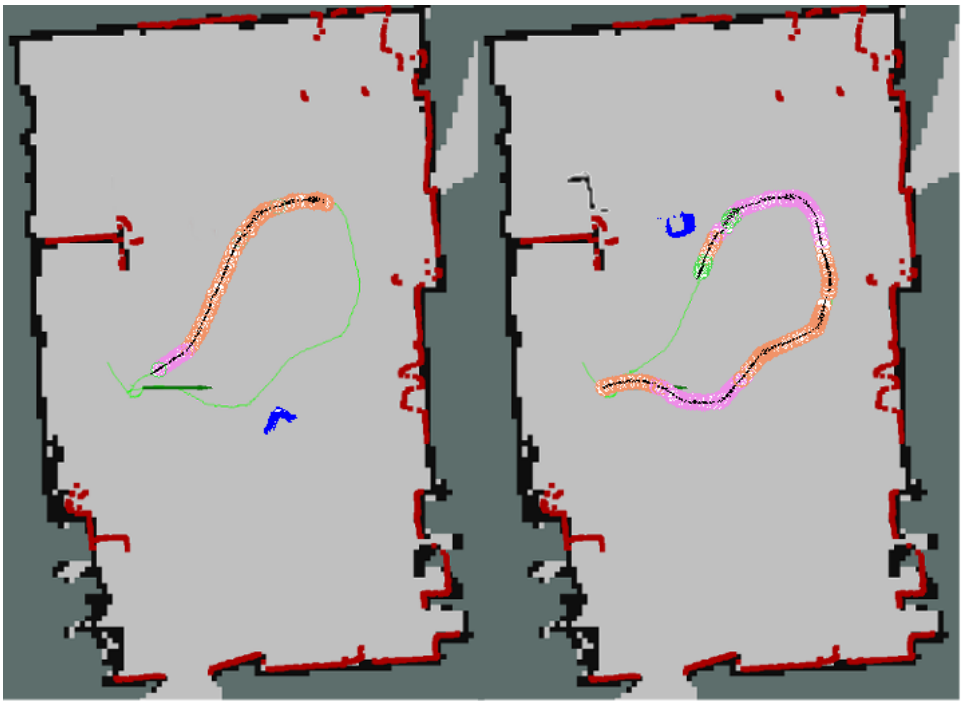


Figure 6.9: Example of trajectory and models obtained. The green line represents the trajectory, the blue points are the laser readings on the considered object, the circles represent where the robot registered a view in the model for the object, the colors represent which view it is.

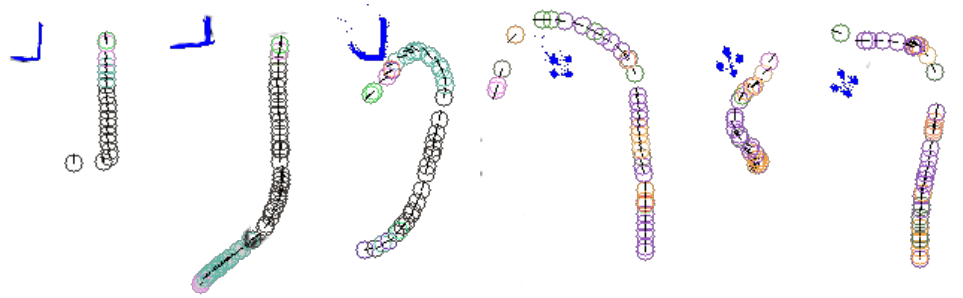


Figure 6.10: Example of clusters obtained. On the left, three models coming from an armchair, on the right three models coming from a stool.

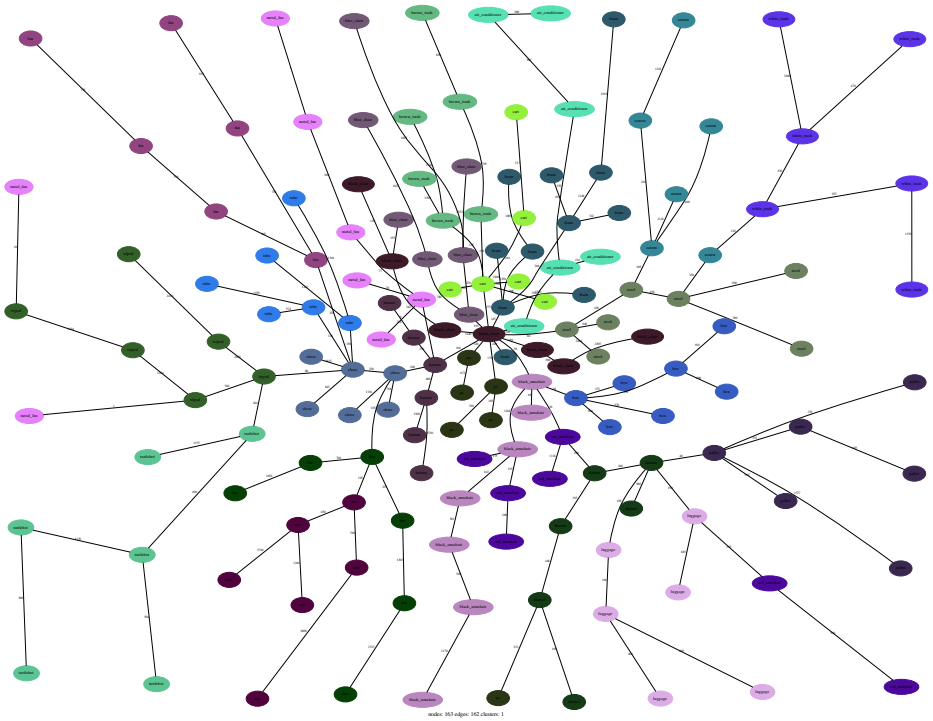


Figure 6.11: A maximum spanning tree where nodes are models from the second dataset and edges are their similarity. The color indicates which real object the models represent for more visibility. We can see with this tree that our assumption about observations being more similar when representing the same real object is correct because nodes of the same color are in majority linked together and form branches of the tree.

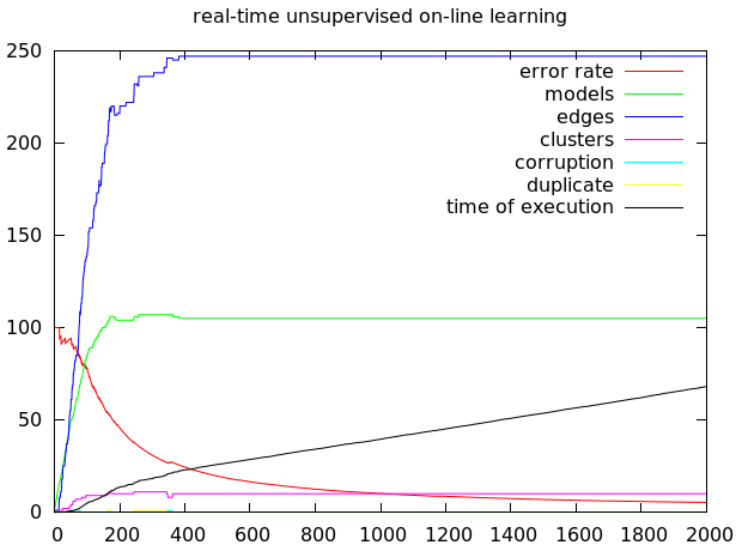


Figure 6.12: Evolution of the learning for a simulated experiment (see text for details).

are 25 different labels in the dataset. Furthermore, edges between the same labels should all have a higher value than edges between different labels if we were to separate them based on a threshold. In this graph there are in fact 37 edges between different labels out of 162 which roughly rounds up to 8% of incorrect edges. The values of edges between different labels are in the range 1 to 328 with a mean of 151. The values of edges between equal labels are in the range 1 to 1292 with a mean of 787. Those values confirm that it is not possible to find a threshold to cluster correctly the labels on similarity between models but that our hypothesis is validated: observations representing the same real object do tend to be more similar than observations representing different objects.

Simulation

In order to develop our graph clustering method, we conducted experiments on simulated datasets. The datasets were generated by producing random sequences of integer in order to create the ground truth models. Each sequence represent an object, while each integer represent a view identity from all the viewpoints around the object. Those sequences were generated such as they would share some parts and some views to varying degrees so we can simulate the resemblance between objects. In order to produce partial models, sub-sequences where randomly picked with varying length. On those sequences random noise is added to simulate errors in the perception of the robot. Finally graph-of-views were constructed from those and processed by

the clustering system to assess its performance.

Figure 6.12 shows the evolution of the learning during one of the simulations. In this simulation, the system has to learn ten different objects with at most fifty percent of resemblance between the source sequences. Two thousands partial models were generated with a maximum length of thirty percent of the original sequence. For each partial model, the noise added to the sub-sequence was five percent of its total length. We can see on this experiment that the system perfectly learns the ten objects by producing ten non-corrupted clusters. What is interesting in this experiment is that we show the potential of this method to overcome the shortcomings observed in the sequential clustering algorithm. Firstly, we can see in the early stage of the learning the presence of duplicates and corrupted models that are overcame afterwards. Secondly, the system stops learning at some point by not modifying clusters anymore, and so the execution time per queries becomes constant and proportional to the number of clusters.

Two main source of error were observed during these experiments. First, the system could not learn an object if the partial models were too small. The limit was found around twenty percent but varies with different sources of noise. This limitation is not surprising and is not a real issue in practice if the robot performs long enough trajectories, and even less if it uses active perception to learn objects. Second, if the true objects are very similar or the noise in the partial models is too important, the system has difficulty to distinguish between the objects. Again this limitation is comprehensible. Using multiple modalities should in practice lessen this limitation.

Real Dataset

In order to assess the performance of the graph clustering algorithm on real data, we experimented it on the second dataset described in section 5.4.1. To do so, the partial models are computed using the precise dictionaries and the descriptors used in the experiments of chapter 5. The partial models are given in a random order to the graph clustering system and the evolution of the learning is recorded. The *low_similarity_threshold* signifying models don't belong together was set to 0.01 and the *high_inclusion_threshold* to 0.95. The limit of edges per node was set to 4. The maximum number of models per cluster was set to 5. The two most similar cluster given by the classification and the singleton were updated for each new model. The maximum number of singletons allowed was set to 30.

The figures 6.13 and 6.14 represent the final clustering. There are 4 corrupted clusters and 12 duplications which are mostly singletons. All categories of object have at least one consolidated cluster. The figure 6.15 represent the evolution of the graph clustering system while learning. As we can see, at the early stage several clusters are corrupted and duplicated but the number is reduced afterwards as new models are processed. However,

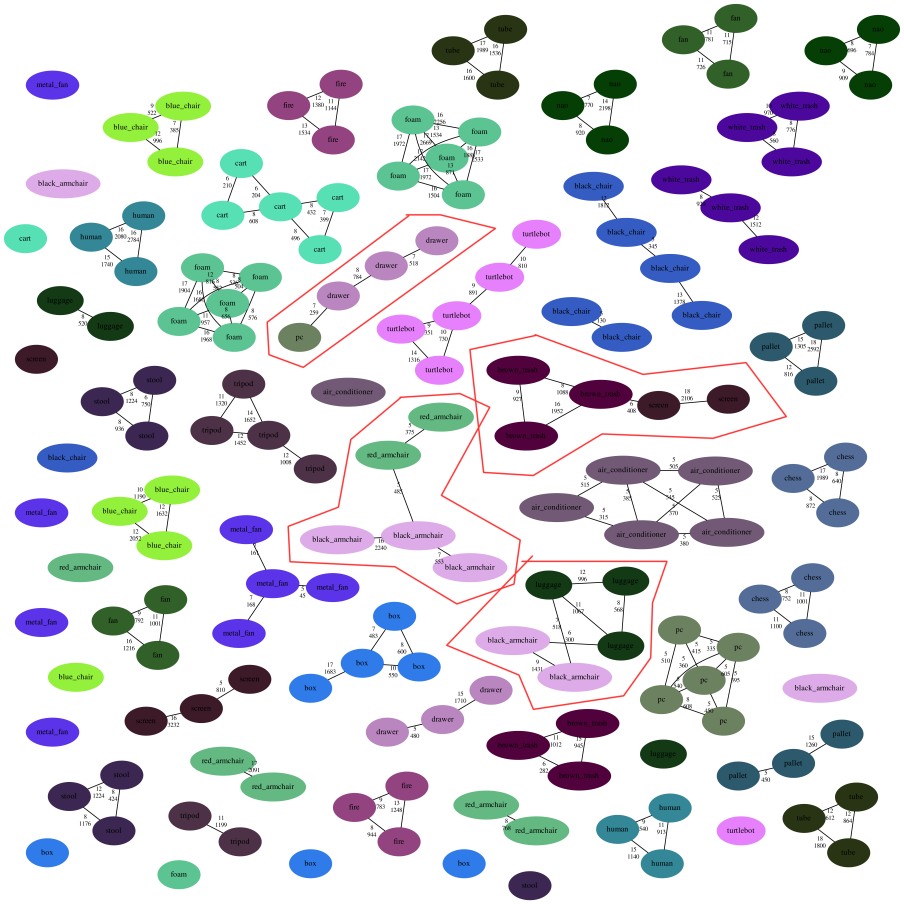


Figure 6.13: Final clustering obtained on the second dataset using graph-of-views models and the graph-clustering approach. The figure shows the final state of the similarity graph. Each color represents a real object. The corresponding names are written on the nodes. The edges between nodes represent their similarity. The first number is the *modalities_match*, and the second is the *joint_modalities_intersection*. The corrupted clusters are circled in red.

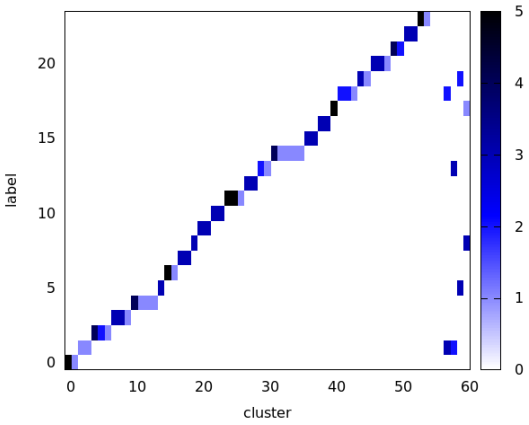


Figure 6.14: Number of clusters in the memory by real objects identity and their size using the graph-clustering approach on our second dataset.

the number of models in the dataset is too small to see a convergence of the learning like in the simulation.

In practice, for each category of objects, the dataset contains 6 trajectories which means at least 6 observations[2]. Those trajectories were processed by our perception pipeline without our intervention other than labeling the observations as a ground truth for evaluation. This way we could ensure that our dataset was processed by the graph learning system as closely as possible as in a real situation. In light of this, having clusters of 5 observations plus a singleton in the resulting graph for some of the objects is a good result, as it is the best that can be achieved given the constraints on the cluster size. The singletons were not discarded because the limit was not reached in this experiment.

6.4.3 Comparison of sequential and graph clusterings

In order to compare both clustering approaches, we conducted the same experiment than in section 6.4.1, but using the second dataset with graph-of-views models as in the previous section.

We first had to select the threshold for the incremental clustering on this dataset. The figure 6.16 represents the result of the learning with the incremental clustering approach on this dataset, depending on the threshold set. We compared all similarity functions and the one with the best result was the joint modalities intersection. The threshold was set to 0.2 in the system, as a good tradeoff between consolidated and corrupted clusters. We then recorded the evolution of the sequential clustering and the final label/cluster matrix obtained for comparison with the graph clustering.

[2]For some objects, the trajectories resulted in more than 6 observations due to momen-

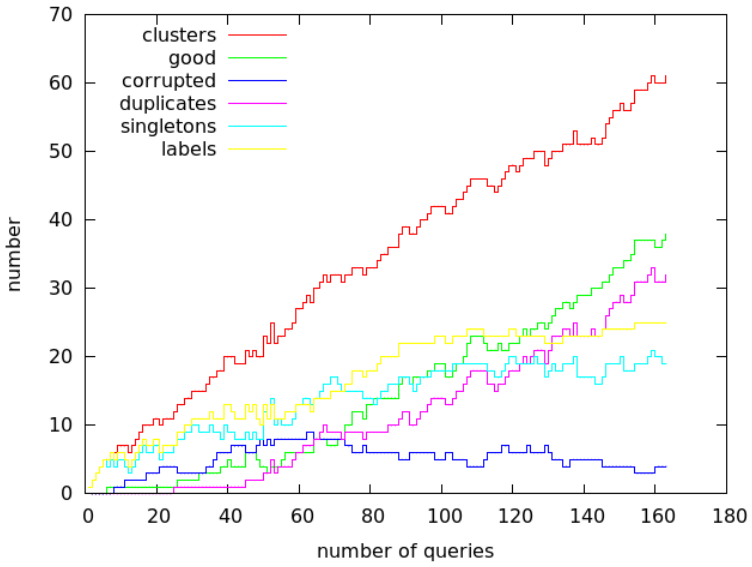


Figure 6.15: Evolution of the learning. In red the number of clusters, in green the ones containing more than one observation and not corrupted, in blue the corrupted clusters, in magenta the number of duplicates (cluster representing an object already represented by another one), in cyan the singletons (clusters containing only one observation), in yellow the number of objects represented by at least one non corrupted cluster.

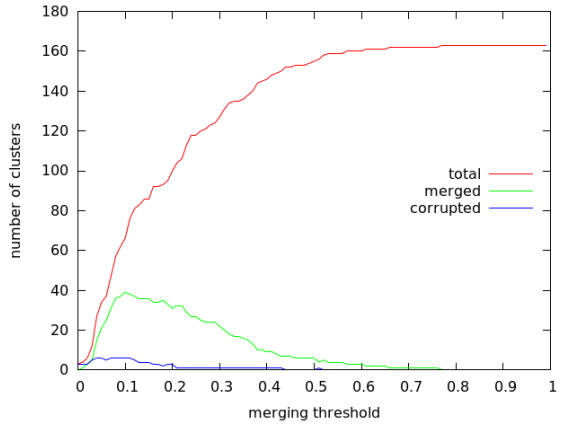


Figure 6.16: Number of clusters as a function of the merging threshold with the incremental clustering. In red the total number of clusters in the memory. In blue, the number of clusters that have been updated successfully. In green, the number of corrupted clusters (cluster containing models coming from different objects). The similarity function used is the normalized joint modalities intersection.

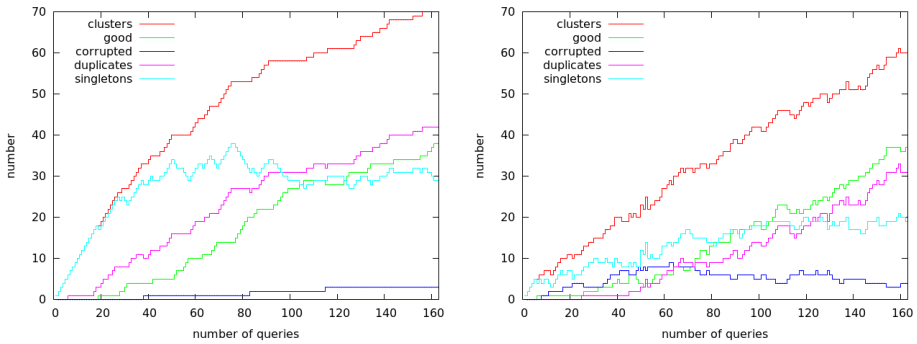


Figure 6.17: Evolution of the learnings for comparison of the sequential clustering (left) and of the graph clustering (right). We can see that early on, the graph clustering produce less clusters but more corrupted ones. That is because it tolerates less similarity between models inside a cluster. Nevertheless, the corrupted clusters are slowly corrected to finish with a number of 4 (3 for the sequential algorithm). Furthermore, the final result for the graph clustering contains less clusters, less duplicates, less singletons and approximately the same amount of correct clusters.

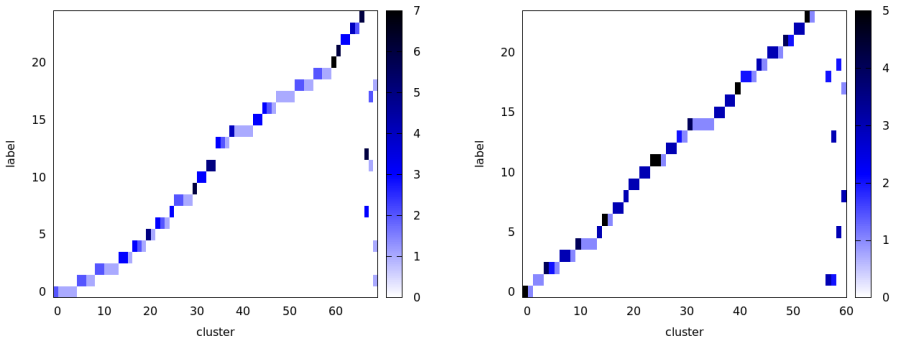


Figure 6.18: Number of clusters in the memory by real objects identity and their size for the sequential clustering (left) and for the graph clustering (right). We can see that the sequential algorithm failed to learn the object 12 and produced many duplicates. On the other hand, the graph clustering produced less clusters (10 less) and less duplicates.

Figures 6.17 and 6.18 show the results from both clustering methods. We can see that the graph-clustering approach produces a better clustering than the incremental clustering: all but one categories have at least one consolidated cluster, there is approximately the same amount of corruption and correct clusters and less clusters, duplications and singletons are present. More precisely, for the incremental clustering, seven object resulted only in singletons or cluster containing two models, showing that these objects have been badly learned. However, eleven objects resulted in strong clusters with at most one duplicate. For the graph clustering, almost all real objects resulted in a correct cluster merging several models without corruption, with the exception of the object 1 which only resulted in singletons. Sixteen objects resulted in a correct cluster with at most one duplicate. Note also that since the sequential algorithm have no mechanism to remove duplicates or corruption, this result also indicates that the graph clustering should perform much better on a longer period of time.

6.5 Conclusion

In this chapter we presented two unsupervised clustering method to learn models of objects from partial observations. The first one is a simple sequential clustering, but the study of this method revealed that when a new partial observation is made by the robot, certainty about its true identity is rare if the object is unfamiliar. So when learning a new object, even though the method will eventually produce a good consolidated model of the object, it has a tendency to produce duplications and some corrupted models. Our conclusion is that in those cases, the clustering method should take soft decision in order to minimize corruption and duplication of models. It should also minimize the amount of data retained so that the memory can scale to a day long of processing. In fact we proposed that during the robot normal functioning, a clustering method should be more about filtering the perception to gather only information with high value and quality. For this reason, we presented an original graph-clustering method which takes soft decision when there is no certainty and constraints its underlying memory structure so that only the best information is kept.

Using simulation, we performed experiments to ensure the scalability and behavior of the method. We showed very encouraging results on this regard. We then tested it on our dataset to assess the quality of the clustering. We showed that the algorithm is able to learn models of all the objects even in presence of ill-perceived and similar ones. Its performance are very encouraging since it was able to form models from only a small number of partial observations with low overlap. Furthermore, we showed an improvement over

tary ill-perceptions and interruption of the tracking.

the first method on our dataset, but we didn't had enough time to show the graph-clustering working in a real situation over a long time period.

Going from there, three different directions could be interesting to investigate. Firstly, we would like to see how using active perception could benefit the learning and the behavior of the robot. We could imagine the robot trying to gather pertinent observation when an object is difficult to classify, thus improving its learning capabilities. Secondly, we would like to effectively process long hours of perception and see if an off-line method could then refine the learning. We believe such dual mechanism could yield very good results. Finally, we would like to extend the algorithm so that it can be used in a semi-supervised fashion. Thus, using human cues or on-line resources to group together observations of the same object could improve further the learning process without losing its flexibility.

Chapter 7

Discussion, future work and conclusion

In this final chapter, we will first discuss several aspects of our work, on which our approaches have proposed contributions and in which future work could be envisaged, before summarizing our work and concluding.

7.1 Long term semantic mapping and navigation

We presented a complete software architecture for semantic mapping. It is composed of many modules to perform specific computations such as SLAM, path planning or object recognition, and of a state machine communicating with the different modules to plan robustly the exploration of an unknown environment.

In order to have a robust and safe solution, existing techniques have been used to perform the navigation using different types of sensors. We have shown that we are able to explore an environment and produce a semantic maps of it representing the layout of the rooms and the objects inside. However, the architecture could be improved in many different ways. In particular, an important weakness of this architecture is that only one technique is used per task. For instance, there is only one SLAM module present in the architecture and many other computations rely on its output. Upon its failure, it is very difficult for the robot to recover from it. It would be interesting to see how robustness and safety could be improved by using concurrent techniques for the same sub tasks along with a process selection mechanism, as proposed for example in (Dalgalarrondo and Luzeaux 1999).

In order to have a long term semantic representation, we proposed that it should be split into two problems: maintaining a representation containing only static information of the environment, and a representation containing the dynamic aspects. In this work, a SLAM technique was derived to incrementally produce an occupancy grid containing only static information.

This is an interesting idea because many existing navigation techniques use occupancy grids but make the assumption of a static world. It is adapted for a global path planning technique whereas the dynamic part could be handled more naturally by a local obstacle avoidance. Furthermore, once this static representation is fully converged, detecting the dynamic objects in the environment is easier and we have shown that it then makes it possible to learn and recognize them. The results shown are promising, however a more thorough study is required to fully understand the implication of such representations: for example how to properly handle the influence of the dynamic part on the global path, when an unexpected obstacle forces to completely change the plan.

Using our system we have shown that non-stationary obstacles could be detected and recognized from afar using sensors that provide little information as the laser scanner. We have also shown that the robot can to a certain extent learn new objects incrementally and without supervision. However, we didn't study how to use the recognition and how to represent properties of the object such as possible locations to perform active object search. Nevertheless existing techniques could be integrated to our system.

Future work on the problem of long term semantic mapping and navigation could be also to extend our architecture so that it can perform other tasks like active object search. A possible solution to this problem could be the use of a task planning technique along with a representation holding the possible location of objects, as was proposed for example in (Pronobis, Jensfelt, et al. 2010).

7.2 Fast saliency and prior recognition

In this work, we proposed to use a 2D laser scanner in the entire process of object recognition. On the one hand, we argued that popular sensors for object recognition such as RGB-D cameras have a narrow field of view and involve demanding computation. This would eventually lead to poor and inefficient behavior from the robot when accomplishing certain tasks such as search for a particular object in a large environment. A possible improvement is to have a saliency and prior recognition mechanism which can perceive objects afar in order to prune the search space. On the other hand, using 2D laser scanner has shown its efficiency in the past to perform mapping and navigation for mobile robots. Many existing solutions use this sensor as such. Its advantages are a wide field of view, a good precision over a long distance and a small amount of data to process. Furthermore, this sensor is not affected very much by illumination and can work in the dark. For those reasons it is a very good choice to perform saliency or some level of object recognition.

To do so, we have developed two different object detection algorithms.

The first one is based on the comparison of the registered scan with the static occupancy grid. This algorithm is interesting for the learning process because it ensures the robot is perceiving a non-stationary obstacle if the localization is correct. The second one is based on the segmentation of the sensor output directly. It is interesting because it can detect objects when no localization is available or when the static occupancy grid hasn't converged yet. However it is more prone to false positive. Using both detections, we have shown that the robot can detect the objects when entering a room long before our camera could sense them. The computations involved are not demanding which means we can perform fast saliency with the 2D laser scanner. The main drawback of this process is that it can not handle cluttered objects.

To be able to recognize the objects detected by the 2D laser scanner, we developed two different shape descriptor extractors. Their input is a set of points belonging to one object and representing a fragment of its 2D contour as perceived by the sensor. To ensure that the robot can recognize an object correctly, the descriptors must be invariant to point-of-view. To achieve that, both extractor require an oversampling step of the initial set of points in order to remove the variance regarding the distance between the robot and the object. To encode the shape, the first extractor uses information about pairs of points. The results obtained show that this extractor has low computational costs and can be effectively used to recognize different objects. However, the step of its computation that extract a reference angle was found lacking robustness and reliability on certain category of objects, which lessens its discriminative power. A slightly different extractor was developed which uses information about points triplets. The reason is that the triangles formed by the points triplets are completely invariant to the robot point-of-view and do not require the computation of a reference angle. The results obtained show that this second extractor is more robust and discriminative on a wider range of objects, but that it is slightly more costly to compute.

Using a tracking system, we accumulate the descriptors extracted from one object into a bag-of-views model. With this process, we have shown that we could recognize up to 22 different objects. We have also shown good recognition results even when the robot just started to see an object. This is interesting for two reasons.

First, we have shown that a 2D laser scanner can be used in the whole perception process: from mapping to object recognition. Existing works have already shown that this sensor could also be used to perform room or place recognition (Tipaldi and Arras 2010). It means that using this sensor alone, it should be possible to improve existing service robot solution by adding semantic information about the rooms and the objects.

The second reason is that this process could be used to perform prior recognition to improve a camera based system, for example by using active perception where the camera could be directed toward difficult to recognize

objects for the laser sensor. It can also be useful to reduce sensibility to ill-perception and appearance variation as will be discussed more in detail in the next section.

7.3 Robustness to appearance variations and ill perception

Recognizing objects and dealing with the problem of appearance variations is still an open problem and the focus of many studies from different domains (computer vision, machine learning, robotics). Many difficult challenges surround this problem.

Firstly, the appearance of objects, either geometric or visual, are subject to many variations. Secondly, depending on the context, an object can be occluded or in a clutter. It can also be ill perceived by a sensor. It means that multi-modal perception is a necessity to reach higher level of robustness. Finally, in a robot, the memory and computation resources are limited, so the perception of objects needs to be fast and efficient. In the perception pipeline proposed in this work, some of these issues have been addressed.

For the detection of objects, we used two different sensors and three different algorithms. We have shown that our system was able to detect and recognize to a certain extent objects that are not well perceived by one of the sensors. To do so we performed our experiments on different objects (38 in total) which vary in difficulty to be perceived or to be differentiated. We believe we made our case for a large amount of possible obstacles in a household, with the exception of two categories: very small objects (tennis ball) and very large ones (furnitures). Another limitation of our system is that we discard detected objects that are occluded and we can not segment clutter of objects. The main reason for those limitations is that our descriptors work on plain views of objects. A possible partial solution would be to incorporate the use of local visual features in the whole pipeline. Indeed, our detection is made in the geometric space only, considering also the color space would be interesting and could result in better performance. Furthermore, adding local visual features in our modeling and recognition processes should be possible.

When an object has been detected, the robot tracks it. Having a tracking system in a service robot is very important in order to perform robust obstacle avoidance and navigation. However, in this particular case, the tracking is used as a robust way to identify and group detections as being from the same object. In other words, using tracking techniques, the robot constructs the sequence of views that it sees of an object from the moment it detects it until it disappears from its field of view, even if the object is moving, if its appearance is changing or if multiple sensors are used. Those sequence of views obtained by the robot are in fact the raw information about the

objects that we need to compare. Comparing sequence of views rather than single views has in theory many benefits. This improves temporal coherence while operating the robot, and this should be more robust and give better recognition results since more information is taken into account. However, one difficulty is that the content of the sequence varies depending on the robot trajectory relative to the object.

In order to compare those sequences, we extracted different descriptors from the views. Using a dictionary learning technique, the sequences are sparse coded into sequence of discrete views. They are then transformed into a graph-of-views and compared using a similarity function. This object modeling process has several advantages.

Firstly, using descriptors of views allows us to use different sensors and to capture different properties of the object in a single model. This is important for having a robust object recognition system because different sensors have different limitations. For instance, a color camera can't sense in the dark but an infrared based sensor can. Furthermore, many objects share properties whether it is the same color, or shape or texture. So it is also important to capture as many properties as possible of an object in order to have a precise and robust object recognition. In this work we used two different sensors and five different descriptors. It would be very interesting to see how this system behaves as more sensors and descriptors are added.

Secondly, because the objects are modeled with view-descriptors and temporal events, the process offers a certain robustness to appearance variations. For instance, it doesn't require a precise localization or an accurate perception as long as the descriptor is stable to small variations in the point of view. It also doesn't require that the object be static when perceived. However, in certain cases a model of an object could grow very large. For instance a soft object like a curtain observed in different lighting and wind conditions would probably result in a huge graph-of-views model. How our system would cope to very large models is unclear and should be investigated.

Finally, our modeling process allows us to compare models containing different amounts of information. It is possible to have a recognition result at the first glimpse of an object, or when a sensor or descriptor is not available. Furthermore, since the comparison is not based on a single view but on the integration of a sequence of views, the recognition result is improved as the object is perceived longer. This should be very helpful in an active perception scenario. For instance, it could be used by a top-down system to filter what objects the robot should focus on depending on its tasks.

As we said before, the principal drawback of our modeling process is that partial occlusions and clutter of objects are not processed. We suggested that a possible solution would be to use local visual features. This would mean some modification to the graph-of-views model. In its current form, the nodes represent properties extracted from a view of an object, and the edges represent some relations between those properties. Two are considered:

an edge exists between properties extracted from the same frame of a same sensor, and an edge exists between consecutive properties. In the case of using local features, a node could represent a local property and edges could represent close properties in space and time. Unfortunately, our first attempt to do so showed that it is not straightforward. It results in more computation time, in larger models and in an unbalanced amount of the different properties. Another possible direction could be to add a step in the modeling process and recognize object views not based on a single view-descriptor but on a set of local-descriptors.

7.4 Unsupervised and incremental objects learning

On the path of having robots able to adapt to a new environment smoothly, we believe a requirement is to achieve efficient learning with the minimum supervision possible. To do so, the robot should be able to gather pertinent information while it is functioning and to learn from it as much as possible. When inactive, the robot could use for example an on-line database to improve its learning or perhaps ask the user for cues.

In this line of thinking, we have researched pure unsupervised object learning without prior knowledge. Details of the contributions are presented in chapter 6. Specifically, we studied how to recognize objects seen in the past from a current partial perception, and if possible incorporate the newly seen information in the global knowledge of the robot. This functionality is to be used while the robot is performing other tasks and thus should be efficient.

We started our study by using a basic sequential algorithm. Upon perceiving an object, the robot would decide if it is confident it recognized the object, and either consolidate its model or start a new one. This study revealed several aspects and requirements of unsupervised learning in presence of partial knowledge.

Firstly, few decisions are certain at the early stage of the learning. For instance, observing new information can either mean the robot discovered a new object, or just a new aspect of an object already observed before. Secondly, because the quality of the perception varies with the situation or the object being observed, it is difficult to find absolute criterion about the similarity between object. Finally, in a realistic scenario, the robot would be active for long hours at once observing multiple times the same object but with great variation in the quality of the observation. Sometimes the robot could perceive an object just in glimpses, sometimes it could take the time to observe it from all angles.

To solve this problem, the requirements for the learning system is not to cluster and make use of every observation, but rather extract the best

observations in terms of quality and take hard decisions only when absolutely certain to preserve the memory.

With this line of thinking, we developed an original graph clustering method which maintains a constrained graph of observations and their similarities. We designed the method so that only the best observations of an object are kept and so that it scales gracefully with the number of objects and time. We obtained promising results. Using a simulation we have shown that the method scales correctly with the number of objects, that the memory converges towards a stable state and that the method presents good resilience to noise in the observations. Using our dataset, we have shown that the method can indeed learn difficult objects without prior knowledge. However, the method should be further tested on real data for longer periods of time and with many more objects.

Once the robot has an efficient unsupervised learning system, it would be interesting to study the use of active perception to improve the learning. When the robot observes an object which it can not classify, using active perception, the robot could try to obtain a richer information in order to accelerate its learning or recognition. For example a strategy could be to first get closer and make sure the robot gets an unobstructed view of the object with its camera, and perhaps then turn around the object to get different point of views. This kind of behavior could be very useful and adequate when the robot explores its new environment for instance. Another interesting direction is to combine the robot unsupervised learning capabilities with external supervised classifier. For instance the robot could take picture of the objects it sees and try to label its observation using an on-line tool to perform object recognition. How those information should be handled to improve the robot knowledge should be an interesting study.

7.5 Conclusion and future work

The presented work addresses the semantic mapping problem as a whole and in a pragmatic spirit. The envisioned scenario is that upon buying a service robot, it would have to first explore the environment exhaustively and with some prior knowledge in order to be operational, and then continuously adapt to the environment without requiring expert intervention (see section 1.2). The purpose of this work was to produce a solution to this scenario from a navigation perspective. In order to do so, four main difficulties were identified and addressed in this work.

The first problem is how to integrate all the existing techniques into a service robot. We presented in chapter 3 a full system able to explore the environment and produce a semantic representation of the environment. Then an extension of this system was given so that the robot could maintain and improve the semantic representation over time. This system represents

an interesting starting point toward achieving a more accomplished service robot solution.

Secondly, we identified that many existing solutions based their object recognition solely on cameras and that it is not satisfactory. The reason is that those type of sensors often have a narrow field of view and require an expensive computation before producing a valid result. This would cause the robot to be inefficient while searching for an object for example. To address that, we proposed to use a 2D laser scanner in the entire process of object recognition. We presented in chapter 4 how we use such sensor to perform object recognition. The results obtained showed that using a 2D laser scanner is a valid option to perform object detection and fast prior recognition which could be used to improve existing systems.

The third problem addressed is that of robust object recognition. Considering the possible objects present in a household it is obvious that it is not possible to differentiate them easily. Many objects are similar in terms of size, shape, color or texture. Many of them can have variation of appearance over time. And finally in many situations it is not possible to perceive them fully. As such, to obtain a robust and efficient recognition system, it is natural to consider using different sensors and multiple kind of attributes, and a system able to cope with partial information. In the chapter 5, we propose an original solution to combining different sensors and modalities into a single object representation. Doing so is often cumbersome and we tried to keep the solution as simple and easy to deploy as possible. The results show that this solution is suitable to multi-modal object recognition even in case of partial observation.

Finally, most existing semantic mapping solutions use supervised learning and don't address the problem of continuously adapting to the environment. Other studies offer solutions to discover new objects but they can't be applied on the fly while the robot is working. For the robot to react gracefully to a new situation like encountering a new object, it needs to take an appropriate decision, which could be processed more thoroughly later on. Regarding this problem, specifically in the case of a new object, we studied in chapter 6 the use of unsupervised and incremental techniques to give the robot the capabilities of gradually learning new object.

In conclusion this work proposes the foundations toward a semantic mapping system which can adapt gradually and without supervision to its environment. It opens interesting research perspectives regarding the robots capability to apprehend their environment. The first direction is in the integration of active perception not only for performing complex tasks that require a rich knowledge of the environment, but in the robot learning process as well so that it can adapt to its changing environment efficiently. Secondly, another interesting study would be to combine unsupervised and supervised learning techniques. The robot could be gathering and filtering information while active using an unsupervised approach and improving its knowledge

while unused using on-line tools. Finally, perceiving objects through several sensors to capture their different aspects and properties raise the question of how to exploit the information hidden in their similarities and dissemblance. We could for example imagine a system that could produce a hierarchical clustering of the objects, separating the most dissimilar objects first in order to be able to concentrate on more subtle but important differences if required.

Bibliography

Angeli, Adrien et al. (2009). "Visual topological slam and global localization". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4300–4305 (see p. 11).

Arbuckle, Daniel, Andrew Howard, and Maja Mataric (2002). "Temporal occupancy grids: a method for classifying the spatio-temporal properties of the environment". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* 1, pp. 409–414 (see p. 12).

Aslam, Javed A, Ekaterina Pelekhov, and Daniela Rus (2004). "The star clustering algorithm for static and dynamic information organization". eng. In: *Journal of Graph Algorithms and Applications* 8.1, pp. 95–129 (see p. 89).

Aydemir, Alper et al. (2011). "Plan-based Object Search and Exploration Using Semantic Spatial Knowledge in the Real World". In: *Proceedings of the European Conference on Mobile Robotics (ECMR 2011)* (see pp. 4, 5, 17, 21).

Baillie, Jean-Christophe et al. (2011). "Software architecture for an exploration robot based on Urbi". In: *Proceedings of the 6th National Conference on Control Architectures of Robots*, pp. 1–12 (see pp. 6, 7, 27).

Baillie, Jean-christophe et al. (2008). "The Urbi Universal Platform for Robotics". In: *International Conference on Simulation, Modeling and Programming for Autonomous Robots*, pp. 580–591 (see pp. 24, 32).

Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool (2006). "SURF: Speeded up robust features". In: *Computer vision (ECCV)*. Springer, pp. 404–417 (see p. 19).

Belongie, S., J. Malik, and J. Puzicha (2002). "Shape matching and object recognition using shape contexts". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.4, pp. 509–522 (see pp. 52, 53).

Biswas, Rahul et al. (2002). "Towards object mapping in non-stationary environments with mobile robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* 1, pp. 1014–1019 (see pp. 4, 5, 17).

Blackman, S.S. and R. Popoli (1999). *Design and Analysis of Modern Tracking Systems*. Artech House radar library. Artech House (see p. 16).

Blom, Henk A.P. and Yaakov Bar-Shalom (1988). "The interacting multiple model algorithm for systems with Markovian switching coefficients". In: *IEEE Transactions on Automatic Control* 33.8, pp. 780–783 (see p. 16).

Borenstein, Johann and Yoram Koren (1991). "The vector field histogram-fast obstacle avoidance for mobile robots". In: *IEEE Transactions on Robotics and Automation* 7.3, pp. 278–288 (see p. 14).

Brock, Oliver and Oussama Khatib (1999). "High-speed navigation using the global dynamic window approach". In: *Proceedings of IEEE International Conference on Robotics and Automation*. Vol. 1. IEEE, pp. 341–346 (see p. 14).

Brooks, Rodney et al. (1986). "A robust layered control system for a mobile robot". In: *Robotics and Automation, IEEE Journal of* 2.1, pp. 14–23 (see p. 22).

Burgard, Wolfram et al. (1999). "Sonar-based mapping with mobile robots using EM". In: *Proceedings 16th International Conf. on Machine Learning*, pp. 67–76 (see p. 11).

Caetano, Tibério S et al. (2009). "Learning graph matching". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31.6, pp. 1048–1058 (see p. 67).

Campbell, Richard J. and Patrick J. Flynn (2001). "A Survey Of Free-Form Object Representation and Recognition Techniques". In: *Computer Vision and Image Understanding* 81.2, pp. 166–210 (see p. 19).

Caron, Louis-Charles, David Filliat, and Alexander Gepperth (2014). "Neural Network Fusion of Color, Depth and Location for Object Instance Recognition on a Mobile Robot". In: *Second Workshop on Assistive Computer Vision and Robotics (ACVR), in conjunction with European Conference on Computer Vision*. Zurich, Switzerland (see p. 32).

Caron, Louis-Charles, Yang Song, et al. (2014). "Neural network based 2D/3D fusion for robotic object recognition". In: *ESANN*. Bruges, Belgium, pp. 127–132 (see pp. 32, 67).

Cha, Sung-hyuk (2007). "Comprehensive Survey on Distance / Similarity Measures between Probability Density Functions". In: *International Journal of Mathematical Models and Methods in Applied Sciences* 1.4, pp. 300–307 (see pp. 54, 57).

Collet, Alvaro, Siddhartha S Srinivasa, and Martial Hebert (2011). "Structure discovery in multi-modal data: a region-based approach". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, pp. 5695–5702 (see p. 17).

Collet, Alvaro, Bo Xiong, et al. (2013). "Exploiting domain knowledge for Object Discovery". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2118–2125 (see p. 17).

Connell, Jonathan H. (1992). "SSS: a hybrid architecture applied to robot navigation". In: *Proceedings 1992 IEEE International Conference on Robotics and Automation* (see p. 23).

Csurka, Gabriella et al. (2004). “Visual categorization with bags of keypoints”. In: *Proceedings of the ECCV International Workshop on Statistical Learning in Computer Vision*, pp. 59–74 (see pp. 17, 19).

Cyr, Christopher M and Benjamin B Kimia (2004). “A Similarity-based Aspect-Graph Approach to 3D Object Recognition”. In: *IJCV* (see p. 66).

Dalgalarrondo, Andre and Dominique Luzeaux (1999). “Dynamic selection of perception processes on an autonomous robot”. In: *AeroSense’99*. International Society for Optics and Photonics, pp. 216–224 (see p. 115).

Dijkstra, Edsger W. (1959). “A note on two problems in connexion with graphs”. English. In: *Numerische Mathematik* 1.1, pp. 269–271 (see pp. 14, 15).

Drouilly, Romain, Panagiotis Papadakis, et al. (2014). “Local map extrapolation in dynamic environments”. In: *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pp. 3109–3114 (see p. 22).

Drouilly, Romain, Patrick Rives, and Benoit Morisset (2014). “Fast hybrid relocation in large scale metric-topologic-semantic map”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 1839–1845 (see p. 21).

— (2015). “Semantic Representation For Navigation In Large-Scale Environments”. In: *IEEE Int. Conf. on Robotics and Automation, ICRA’15*. Seattle, United States (see p. 22).

Dubois, Mathieu et al. (2013). “A Comparison of Geometric and Energy-Based Point Cloud Semantic Segmentation Methods”. In: *ECMR’13 - 6th European Conference on Mobile Robotics*. Proceedings of the 6th European Conference on Mobile Robotics (ECMR). IEEE. Barcelona, Spain, pp. 88–93 (see p. 32).

Duceux, Guillaume and David Filliat (2014). “Unsupervised and online non-stationary obstacle discovery and modeling using a laser range finder”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 593–599 (see pp. 7, 53, 91, 101).

Elfes, Alberto (1989). “Using occupancy grids for mobile robot perception and navigation”. In: *Computer* 22.6, pp. 46–57 (see p. 11).

Endres, Felix et al. (2009). “Unsupervised Discovery of Object Classes from Range Data using Latent Dirichlet Allocation”. In: *Proc. of Robotics: Science and System (RSS)* (see pp. 5, 18).

Ferguson, Dave and Anthony Stentz (2007). “Field D*: An Interpolation-Based Path Planner and Replanner”. In: *International Symposium on Robotics Research* 28, pp. 239–253 (see pp. 14, 15).

Filipe, Silvio and Luis A. Alexandre (2014). “A Comparative Evaluation of 3D Keypoint Detectors in a RGB-D Object Dataset”. In: *9th International Conference on Computer Vision Theory and Applications* (see p. 19).

Filliat, David (2007). “A visual bag of words method for interactive qualitative localization and mapping”. In: *International Conference on Robotics and Automation*. Italy, pp. 3921–3926 (see p. 56).

Filliat, David et al. (2012). "RGBD object recognition and visual texture classification for indoor semantic mapping". In: *2012 IEEE Conference on Technologies for Practical Robot Applications, TePRA 2012*, pp. 127–132 (see pp. 7, 17, 27, 33).

Fortmann, Thomas E., Yaakov Bar-Shalom, and Molly Scheffe (1983). "Sonar tracking of multiple targets using joint probabilistic data association". In: *Oceanic Engineering, IEEE Journal of* 8.3, pp. 173–184 (see p. 16).

Fox, Dieter, Wolfram Burgard, and Sebastian Thrun (1997). "The dynamic window approach to collision avoidance". In: *Robotics Automation Magazine, IEEE* 4.1, pp. 23–33 (see p. 14).

Fu, Siyao et al. (2008). "Unsupervised learning of categories from sets of partially matching image features for power line inspection robot". In: *Proceedings of the International Joint Conference on Neural Networks* June, pp. 2596–2603 (see p. 18).

Galindo, Cipriano, Juan-Antonio Fernandez-Madrigal, et al. (2008). "Robot task planning using semantic maps". In: *Robotics and Autonomous Systems* 56.11. Semantic Knowledge in Robotics, pp. 955–966 (see p. 21).

Galindo, Cipriano, Alessandro Saffiotti, et al. (2005). "Multi-hierarchical semantic maps for mobile robotics". In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 3492–3497 (see p. 19).

Gordon, Neil J., David J. Salmond, and Adrian F. M. Smith (1993). "Novel-Approach to Nonlinear Non-Gaussian Bayesian State Estimation". In: *Iee Proceedings-F Radar and Signal Processing* 140, pp. 107–113 (see p. 16).

Hahnel, Daniel et al. (2003). "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS)*. Vol. 1. IEEE, pp. 206–211 (see p. 11).

Harel, Jonathan, Christof Koch, and Pietro Perona (2006). "Graph-based visual saliency". In: *Advances in neural information processing systems*, pp. 545–552 (see p. 24).

Hart, P.E., N.J. Nilsson, and B. Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2, pp. 100–107 (see p. 15).

Hawes, Nick, Jeremy Wyatt, and Aaron Sloman (2009). "Exploring design space for an integrated intelligent system". In: *Knowledge-Based Systems* 22.7, pp. 509–515 (see p. 23).

Herbst, Evan et al. (2011). "Toward object discovery and modeling via 3-D scene comparison". In: *Proceedings - IEEE International Conference on Robotics and Automation* Section II, pp. 2623–2629 (see pp. 16, 18).

Iocchi, Luca and Stefano Pellegrini (2007). "Building 3d Maps With Semantic Elements Integrating 2d Laser, Stereo Vision And IMU On A Mobile Robot". In: *Proc. 2nd ISPRS Intl. Workshop 3D-ARCH* (see pp. 4, 5).

Jain, Anil Kumar, M. Narasimha Murty, and Patrick J. Flynn (1999). "Data clustering: a review". In: *ACM Computing Surveys*. CSUR 31.3, pp. 264–323 (see p. 88).

Jebari, Islem, Stephane Bazeille, and David Filliat (2011). "Combined Vision and Frontier-Based Exploration Strategies for Semantic Mapping". In: *3rd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2011)*. China, pp. 237–244 (see p. 34).

Joachims, Thorsten (1998). "Text categorization with support vector machines: Learning with many relevant features". In: *Machine Learning: ECML-98* 1398.2, pp. 137–142 (see p. 19).

Johnson, Andrew (1997). "Spin-images: a representation for 3-D surface matching". In: *Technology*, p. 138 (see pp. 52, 53).

Kalman, Rudolf E. (1960). "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME-Journal of Basic Engineering* 82.Series D, pp. 35–45 (see p. 16).

Kemp, Charles C. and Aaron Edsinger (2006). "What Can I Control ?: The Development of Visual Categories for a Robot's Body and the World that it Influences". In: *Proceedings of the International Conference on Development and Learning (Special Session on Autonomous Mental Development), Bloomington, IN* (see p. 6).

Khatib, Oussama (1986). "Real-time obstacle avoidance for manipulators and mobile robots". In: *The international journal of robotics research* 5.1, pp. 90–98 (see p. 13).

Koenig, Sven and Maxim Likhachev (2002). "Improved Fast Replanning for Robot Navigation in Unknown Terrain *". In: *Technology*, pp. 968–975 (see p. 15).

Kohlbrecher, Stefan et al. (2011). "A flexible and scalable SLAM system with full 3D motion estimation". In: *9th IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 155–160 (see pp. 11, 12, 39).

Kostavelis, Ioannis and Antonios Gasteratos (2013). "Learning spatially semantic representations for cognitive robot navigation". In: *Robotics and Autonomous Systems* 61.12, pp. 1460–1475 (see pp. 4, 5).

— (2014). "Semantic mapping for mobile robotics tasks: A survey". In: *Robotics and Autonomous Systems* 66, pp. 86–103 (see pp. 4, 24).

Kunze, Lars et al. (2012). "Searching objects in large-scale indoor environments: A decision-theoretic approach". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4385–4390 (see p. 21).

Liang, Mingjie et al. (2014). "Simultaneous Recognition and Modeling for Learning 3-D Object Models From Everyday Scenes". In: *IEEE Transactions on Cybernetics* (see p. 66).

Liu, Ziyuan and Georg von Wichert (2013). "Extracting semantic indoor maps from occupancy grids". In: *Robotics and Autonomous Systems* 62.5, pp. 663–674 (see p. 4).

Lowe, David G. (2004). "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision*. Int. J. Comput. Vis. (Netherlands) 60.2, pp. 91–110 (see pp. 19, 32).

Lyubova, Natalia, David Filliat, and Serena Ivaldi (2013). "Improving object learning through manipulation and robot self-identification". In: *ROBIO*. China (see p. 6).

Lyubova, Natalya and David Filliat (2012). "Developmental Approach for Interactive Object Discovery". In: *Neural Networks (IJCNN), The 2012 International Joint Conference on*. Australia, pp. 1–7 (see pp. 6, 18).

Maree, Raphael et al. (2005). "Random subwindows for robust image classification". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE, pp. 34–40 (see p. 30).

Mason, Julian and Bhaskara Marthi (2012). "An object-based semantic world model for long-term change detection and semantic querying". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3851–3858 (see pp. 5, 17).

Mei, Christopher, Gabe Sibley, and Paul Newman (2010). "Closing loops without places". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 3738–3744 (see pp. 25, 66).

Meyer-Delius, Daniel, Maximilian Beinhofer, and Wolfram Burgard (2012). "Occupancy Grid Models for Robot Mapping in Changing Environments". In: *Proc. of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 2024–2030 (see p. 12).

Mitsou, Nikos C. and Costas S. Tzafestas (2007). "Temporal Occupancy Grid for mobile robot dynamic environment mapping". In: *2007 Mediterranean Conference on Control and Automation, MED* (see p. 12).

Modayil, Joseph and Benjamin Kuipers (2004). "Bootstrap learning for object discovery". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 1, pp. 742–747 (see pp. 17, 18, 25, 52).

Montesano, Luis et al. (2008). "Learning object affordances: From sensory - Motor coordination to imitation". In: *IEEE Transactions on Robotics* 24.1, pp. 15–26 (see p. 6).

Nakhaeinia, Danial et al. (2011). "A review of control architectures for autonomous navigation of mobile robots". In: *International Journal of the Physical Sciences* 6.2, pp. 169–174 (see p. 22).

Ng, James and Thomas Bräunl (2007). "Performance comparison of bug navigation algorithms". In: *Journal of Intelligent and Robotic Systems* 50.1, pp. 73–84 (see p. 13).

Nüchter, Andreas and Joachim Hertzberg (2008). "Towards semantic maps for mobile robots". In: *Robotics and Autonomous Systems* 56.11, pp. 915–926 (see pp. 4, 21).

Pancham, Ardhisha, Nkgatho Tlale, and Glen Bright (2011). "Literature Review of SLAM and DATMO". In: *Proceedings of the 4th Robotics and Mechatronics Conference of South Africa (ROBMECH 2011)*. Vol. 23, p. 25 (see pp. 15, 16).

Paul, Rohan, Daniela Rus, and Paul Newman (2012). "How was your day? Online visual workspace summaries using incremental clustering in topic space". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, pp. 4058–4065 (see p. 89).

Pronobis, Andrzej and Patric Jensfelt (2011). "Hierarchical Multi-Modal Place Categorization". In: *Proceedings of the European Conference on Mobile Robots* (see p. 5).

— (2012). "Large-scale semantic mapping and reasoning with heterogeneous modalities". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3515–3522 (see pp. 19–21).

Pronobis, Andrzej, Patric Jensfelt, et al. (2010). "Semantic modelling of space". In: *Cognitive Systems* 8, pp. 165–221 (see p. 116).

Pronobis, Andrzej, Kristoffer Sjöö, et al. (2010). "Representing spatial knowledge in mobile cognitive systems". In: *Intelligent Autonomous Systems 11, IAS 2010*, pp. 133–142 (see p. 4).

Quigley, Morgan et al. (2009). "ROS: an open-source Robot Operating System". In: *ICRA Workshop on Open Source Software* 3, p. 5 (see p. 24).

Rachmawati, Ema, Iping Supriana Suwardi, and Masayu Leylia Khodra (2014). "Review of Local Descriptor in RGB-D Object Recognition". In: *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 12.4, pp. 1132–1141 (see p. 19).

Ramaswamy, Arunkumar, Bruno Monsuez, and Adriana Tapus (2014). "Model-driven software development approaches in robotics research". In: *Proceedings of the 6th International Workshop on Modeling in Software Engineering - MiSE 2014*, pp. 43–48 (see p. 24).

Ranganathan, Ananth and Frank Dellaert (2007). "Semantic Modeling of Places using Objects". In: *Proceedings of Robotics: Science and Systems*. Atlanta, GA, USA (see pp. 4, 5, 19).

Reid, Donald B. (1979). "An algorithm for tracking multiple targets". In: *IEEE Transactions on Automatic Control* 24.6, pp. 843–854 (see p. 16).

Rofer, Thomas (2002). "Using histogram correlation to create consistent laser scan maps". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* 1, 625–630 vol.1 (see p. 54).

Rosenblatt, Julio K. (1997). "DAMN: a distributed architecture for mobile navigation". In: *Journal of Experimental & Theoretical Artificial Intelligence* 9.2-3, pp. 339–360 (see p. 23).

Russell, Bryan C. et al. (2006). "Using multiple segmentations to discover objects and their extent in image collections". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2, pp. 1605–1612 (see p. 17).

Scheutz, Matthias et al. (2007). "First steps toward natural human-like HRI". In: *Autonomous Robots* 22.4, pp. 411–423 (see p. 23).

Sivic, Josef and Andrew Zisserman (2003). "Video Google: Text Retrieval Approach to Object Matching in Videos". In: *Int. Conf. on Computer Vision*. Vol. 2, pp. 1470–1477 (see p. 74).

Tenorth, Moritz et al. (2010). "KNOWROB-MAP - knowledge-linked semantic object maps". In: *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pp. 430–435 (see p. 21).

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox (2005). *Probabilistic Robotics*. The MIT Press (see pp. 10, 39).

Tipaldi, Gian Diego and Kai O. Arras (2010). "FLIRT - Interest regions for 2D range data". In: *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3619–3622 (see pp. 18, 25, 52, 117).

Tomatis, Nicola, Illah Nourbakhsh, and Roland Siegwart (2003). "Hybrid simultaneous localization and map building: a natural integration of topological and metric". In: *Robotics and Autonomous systems* 44.1, pp. 3–14 (see p. 11).

Tuytelaars, Tinne et al. (2010). "Unsupervised object discovery: A comparison". In: *International Journal of Computer Vision* 88.2, pp. 284–302 (see p. 17).

Vasudevan, Shrihari and Roland Siegwart (2008). "Bayesian space conceptualization and place classification for semantic maps in mobile robotics". In: *Robotics and Autonomous Systems* 56.6, pp. 522–537 (see p. 19).

Waibel, M. et al. (2011). "RoboEarth". In: *Robotics Automation Magazine, IEEE* 18.2, pp. 69–82 (see p. 25).

Wang, Chieh-Chih et al. (2007). "Simultaneous localization, mapping and moving object tracking". In: *The International Journal of Robotics Research* 26.9, pp. 889–916 (see p. 16).

Wang, Lim Chee, Lim Ser Yong, and Marcelo H Ang Jr (2002). "Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment". In: *Intelligent Control, 2002. Proceedings of the 2002 IEEE International Symposium on*. IEEE, pp. 821–826 (see p. 13).

Young, Steven et al. (2010). "A fast and stable incremental clustering algorithm". In: *ITNG2010 - 7th International Conference on Information Technology: New Generations*, pp. 204–209 (see p. 89).

Zhang, Tian, Raghu Ramakrishnan, and Miron Livny (1996). "BIRCH: an efficient data clustering method for very large databases". In: *ACM SIGMOD Record*. Vol. 25. 2. ACM, pp. 103–114 (see p. 89).

Zhang, Yin, Rong Jin, and Zhi-Hua Zhou (2010). "Understanding bag-of-words model: a statistical framework". In: *International Journal of Machine Learning and Cybernetics* 1.1-4, pp. 43–52 (see p. 17).

Zohaib, Muhammad et al. (2013). "Control Strategies for Mobile Robot With Obstacle Avoidance". In: *International Journal of Basic and Applied Sciences (IJAMR)*, pp. 1027–1036 (see p. 13).