Universidad
Carlos III de Madrid

Interactive and Active Learning in Social Robots

Víctor González Pacheco

Department of Systems Engineering and Automation

Universidad Carlos III de Madrid

A thesis submitted for the degree of

Doctor of Philosophy

27th November 2015

TESIS DOCTORAL (DOCTORAL THESIS)

Interactive and Active Learning in Social Robots

Autor (Candidate): Víctor González Pacheco

Director (Adviser): Prof. Miguel A. Salichs
Directora (Adviser): Dr. María Malfaz
(Universidad Carlos III de Madrid)

Tribunal (Review Committee)

Firma (Signature)

Chair: Dr. José María Sebastián Zúñiga ______________

Member: Dr. Rodrigo Ventura ______________

Secretary: Dr. Concepcón Alicia Monje Micharet ______________

Título (Degree): Doctorado en Ingeniería Eléctrica, Electrónica y Automática
Calificación (Grade): __________________

Leganés, de de

This thesis wouldn’t have been possible without the love of my three girls.

Acknowledgements

A project that spans along four years cannot be finished without the aid, collaboration and support of many people. Although a PhD thesis is a personal manuscript, I could not emphasize more how much help I received from colleagues, friends and family. This thesis wouldn't have been possible without their support.

First of all, I want to thank to my supervisors, Miguel Ángel and María. Miguel, thanks for believing in me, and for giving me the opportunity and freedom to explore research areas that were new to the group. María, you have been my daily guide in this project and a great editor. You have had the patience to read this manuscript -and the papers that it has produced- almost as many times as I did, and your thorough comments and suggestions have made this thesis a much better document.

I also would like to acknowledge the effort and the help from all the members of the Roboticslab's Social Robotics Group, not only for their support in this thesis, but also for their aid in many other projects. Specially to Irene, José Carlos, Raul, Arnaud, David, and Fernando. Irene, thanks for being always willing to help, not only me but everyone. Jose Carlos, working with you couldn't be easier nor more rewarding. No matter it's coding or writing a proposal, you bring out the best of anyone near you. Raul, keep that drive to make things better, it will bring you wherever you want. Arnaud, you're a natural leader. It's really difficult for everyone to follow your pace, but anyone who does gets the reward of learning a lot with you. David and Fernando, thanks for being there when I landed to the group. At that time it was really hard to anyone who was entering to the group to learn the ropes quickly. And you two took the time to introduce me all the nuances of the hardware and software of the group.

I am really grateful with all the members of the Roboticslab. Specially with some of them who have already embarked in new adventures, leaving a friend here. We have shared ideas, great discussions, and moments. Javi, we have shared more than two masters and one PhD. But beyond that, you've the only person who has been there every single time I needed your help. Thanks.

Silvia, the kindest person I know. Thanks for your support. Alberto, an invaluable friend. I always learn something new with you. Miguel, thanks for your passion. It's really inspiring and contagious. Keep pushing. Álvaro, thanks for those great coffees and their conversations. One can talk about anything with you. Martin, thanks for those great ideas, and for listening to me so many times. Every time I did told you anything about my thesis, you gave me dozens of new ideas on how to improve it!

Finally, I would thank to the person who has been with me more than half of my life. Raquel, you've suffered with this thesis as much as I did, and you've been always supportive and encouraging. There is a quote from Samwise Gamgee in the Lord of the Rings that reflects perfectly your role in this thesis:

> *'Come, Mr. Frodo!' he cried. 'I can't carry it for you, but I can carry you.'*

Thank you for carrying that weight with me these years.

Abstract

The aim of this thesis is to study how social robots should interact with people when these people are teaching them new concepts. This objective is motivated by the need of having robots operating in real, unstructured environments where they have to cooperate or to socialize with humans. From a robot's point of view, people have the –almost obsessive– tendency to be unpredictable. This unpredictability makes the job of programming the robot's social behaviours a daunting task, especially if the robot's programmer has to take into account all the possible situations that the robot may encounter during its life cycle. A common solution to this problem is to let the robot to learn continuously from its environment and from the people it interacts with. Our inspiration comes from the way children learn from their teachers, parents, or from other people. Generally, they ask for the concepts they do not understand (e.g. unknown objects, places, etc.) to adults. Our approach is to mix techniques from Supervised Machine Learning and Human Robot Interaction to enable a natural, interactive learning where the robot plays the role of a child asking to a human teacher. To achieve this, we study and apply techniques from Active Learning literature to enable the robot to decide which questions are more appropriate to ask and when its better to ask them. We apply these techniques in two different areas, human pose detection and object recognition, to enable the robot to detect the situations in which it needs to expand its knowledge so it can ask for it to its human partner.

Resumen

El objetivo de esta tesis es estudiar como los robots sociales deben interaccionar con personas que les están enseñando nuevos conceptos. Este objetivo viene motivado por la necesidad de disponer de robots capaces de operar en entornos reales y desestructurados en los cuales tienen que cooperar o convivir con humanos. Desde el punto de vista de un robot, los seres humanos tenemos la tendencia, casi obsesiva, de ser impredecibles. Esto supone una seria dificultad para los ingenieros encargados de desarrollar sus comportamientos sociales, ya que es muy costoso tener en cuenta todas las posibles situaciones en las que el robot se puede encontrar durante su ciclo de vida. Una solución a este problema es permitir que el robot aprenda por sí mismo tanto de su entorno como de las personas con las que interacciona. Esta idea se inspira en cómo los niños aprenden de sus profesores, padres o de otras personas. Generalmente, ellos preguntan a los adultos por aquellas cosas que no entienden o no conocen: objetos desconocidos, lugares, personas, etc. Nuestra aproximación es mezclar técnicas de los campos de Aprendizaje Automático Supervisado y de Interacción Humano-Robot para establecer interacciones naturales e interactivas donde el robot aprende adoptando el rol de un niño que pregunta a un adulto. Para ello aplicamos técnicas presentes en la literatura de Aprendizaje Activo que permitirán al robot decidir qué preguntas son más apropiadas en cada momento. En esta tesis aplicamos dichas técnicas en dos distintas áreas, detección de las poses adoptadas por una persona y reconocimiento de objetos, para permitir que el robot detecte las situaciones en las que éste necesita aumentar su conocimiento y así poder preguntar a su compañero humano.

Contents

List of Figures

List of Tables

Part I

Introduction and Overview

Chapter 1

Introduction

This chapter introduces this thesis, its motivation, its challenges and objectives, and describes the organization of the manuscript. The aim of this thesis is to contribute to the field of interactive and natural robot learning by combining techniques from the fields of Machine Learning, Active Learning, Natural Language Processing, and Human Robot Interaction, among other fields. These techniques will be applied in two different learning problems: first to detect human poses and, second, to detect different objects. The focus of the thesis is on the interaction aspects by describing the different systems and techniques that enable a robot to learn from a human in a natural way.

1.1 Motivation

The scenario envisioned by this thesis is the following. Imagine a person that, while interacting with a robot, ends up in a situation in which the robot does not understand some indications given by this user since the robot's programmer did not foresee such situation. The concept or the indications that led the robot to a misunderstanding might have been an unknown gesture made by the user, some object that he/she was holding, etc.

In a situation like this, the user has two possibilities. The first one is to ask the robot's programmer or maintainer to enhance the robot capabilities so it can to understand him/her. The second one is to teach directly the robot the concept that it did not understood. In terms of effort and time, the first solution is the least desirable because the user has to, first, explain what was the problem to the robot's maintainer, and second, wait until this new feature is delivered (if it is at all). Therefore, the second approach, teaching the robot in real time, enables the robot to adapt to new situations faster and potentially better in terms of user requirements. That is, the robot would learn exactly what the user needs the robot to learn.

But, how should a person teach a robot? This topic has already been extensively covered in the literature [Argall et al., 2009]. However, in our research group we believe that a robot should exploit extensively its interaction capabilities while learning. Hence, in a social robot, like our robot Maggie [Salichs et al., 2006], the learning interface should be as natural as possible. Our definition of natural interaction for learning is inspired in how children learn when interacting with their parents or teachers. For instance, when a child sees for the first time an unknown object, he/she asks questions like "*What is this?*". Normally, the child's parent or teacher, would respond something similar to "*This is a ball*", "*A car*", etc.

As we can see, the child "observes" an example of an unknown concept and gets the label naming this concept. This type of learning resembles to the branch of Machine Learning called *Supervised Learning*. In such field, a learning algorithm is fed with training data that consists of the data itself (e.g the pixels corresponding to the picture of a ball) and a label indicating the name of the data being observed.

The main objective of this thesis is to apply supervised learning techniques to endow a social robot with the capability to learn from people in a way that is similar to the previous example. In such learning system, the robot has to interact with the user in the most natural way as possible. I.e. we aim for a natural conversation in which the robot learns asking questions to the user, and where the user can reply to these questions verbally, allowing him/her to express the same answer in many different ways. Moreover, we want the robot to ask the most meaningful questions—that is, the ones with the highest impact in the robot's learning. For that purpose we will use Active Learning, which has been widely adopted in the machine learning literature and has recently attracted the attention of the robotics community.

Therefore, to achieve a natural interaction during learning, we will need to combine both the interaction capabilities of the robot with the Active Learning strategies that will enable it ask questions when needed. However, as the next section describes, there are many unsolved challenges related to these two objectives that hinder the capabilities of robots to learn interactively.

1.2 Open Challenges

There are many challenges in the field of robot interactive learning. Some of the most important are addressed in this thesis:

- The interaction during learning, and in Human-Robot Interaction (HRI) in general, is an unsolved problem. Currently, the Human-Robot Interaction is usually not rich and is mostly pre-scripted. E.g. in many scenarios, the robot has pre-written answers to user commands. The problem here lies in the unpredictable nature of humans, which might lead to unexpected situations that the interaction designer may have not foreseen.

- It is costly, in terms of time and human effort, to acquire a label for training, and this cost can be higher if the human has to provide the training labels interactively to a robot. In machine learning problems, it is common to have a human oracle to manually tag a dataset. For instance, in a text classification task the human oracle might provide tags in a point-and-click interface. However, a Human-Robot verbal interface might have a lower throughput than a typical point-and-click task. Even more, an error in the detection of the human speech might imply a higher notification and recovery cost. For instance, if the robot does not understand what the human says, it has to tell him/her, usually by voice, which might be a longer process that simply notifying an error in a screen.

- In traditional supervised learning is the human trainer, not the robot, who decides when the learning session ends, that is, the human decides whether the robot has enough data or not. We will refer to this kind learning as **passive learning** to differentiate it from **active learning**. The main problem of *passive learning* is that the human can only know indirectly and qualitatively how well the robot has learnt. Even more, the human has not any means to know how well the robot would perform until he/she ends the training session and tests it. Hence, it can be much more adequate to let the robot to decide whether it needs more training data or not since it is the only one who can directly estimate, quantitatively, how well it will perform with its current data. Besides, this performance estimation can be done in real time during the training session.

- In many active learning cases, the robot does not have the opportunity to keep learning after the learning session has finished. However, in reality, there are many different unexpected situations that might occur during the robot's normal operation and that might require adding more data to the robot's learning algorithms. In this case, the usual set up, is to let the human supervisor to decide whether the robot needs more examples or not. This presents the problem that the human only can infer the robot's need for more training data from the robot's performance in a task. That is, the human only would know that the robot needs more training examples if the robot is not performing its task well (e.g. failing to detect an object, etc.). Unfortunately, there might be many reasons for a robot to fail in a task and the human might misinterpret those reasons, especially if he or she is not an expert. A way to mitigate this problem is to let the robot to decide whether it needs more data or not and ask the human for them when this situation occurs.

1.3 Objectives of this Thesis

This thesis aims to solve or to mitigate the challenges stated in the previous section. The focus of this thesis is on the interaction aspects of learning and on the use of active learning to improve robot learning. It is out of the scope of the thesis to research in new active learning algorithms nor to develop new computer vision algorithms. As it will be shown in the different chapters of this dissertation, the focus of this research is on integrating pre-existing components, algorithms, and systems; and on applying them in an innovative form to the field of robot learning.

From the challenges presented in the previous section, we can define the main objective of this thesis:

> *To* ***develop a system that enables a social robot to learn interactively in a natural way****. That is, similarly to how a person would learn from another person.*

To achieve this objective, some research aspects need to be addressed:

1. To **develop the active learning mechanisms** that enable this interactive learning system **to learn faster and better** by deciding to ask the most significant questions to the user.

2. To study how active learning can be applied for feature selection in the context of interactive learning and **to analyse how introducing domain knowledge from the trainer** in such active learning system **might impact the learning performance**.
3. To demonstrate that **these learning capabilities** can be **applied in other domains such as object recognition**.
4. To demonstrate that **our active learning approach** can also **improve the learning performance in such other domains**.
5. To **develop a multimodal interaction system** tailored **to enable robots to learn interactively from people**.
6. To **demonstrate that a robot can** be intrinsically motivated to **learn actively even beyond the training session has finished**.

All these objectives, combined, aim to introduce novel aspects in the way active learning is used for interactive, natural robot learning. Therefore, the novelty of this thesis is to study how active learning can be incorporated into a social robot so it can learn using natural interaction and lead its own training.

1.4 Organization of the Document

This thesis is divided in four parts, each one trying to address a different set of objectives or aspects of the manuscript. Part I introduces the thesis and gives an overview of Active Learning. Part II deals with the interaction aspects of robot learning and with supervised robot learning by itself. Part III extends these approaches to robot learning and introduces the active learning. Finally, Part IV, concludes and closes the document.

The chapters of Parts II and III, which are the core of the document, have been organized as self-contained papers. Some of them have been already published, while the rest have been already submitted or are in preparation. Chapter by chapter, the contents of this thesis are organized as follows:

Part I:

Chapter 2 gives an overview of the literature of Active Learning, the common learning framework followed in this thesis. Its main contribution is to summarize the contents of [Settles, 2012]—currently the most relevant, extensive and recent literature reviews of the field—, giving emphasis to the aspects that are relevant for the work presented in this document. The topics covered in the chapter will lay the foundation for the ideas that will be later presented in chapters 5, 6, and 7.

Part II:

Chapter 3 presents the first approach to interactive learning of this thesis. The chapter describes the system that has been developed to enable a social robot to learn poses interactively by fusing information captured by a depth camera. The presented system takes advantage of the robot's interaction capabilities (mainly speech recognition and synthesis) to enable the human teacher to lead the training session in a natural way, similar to what it would be if he/she would be training another person instead of a robot.

Chapter 4 presents another example of interactive, real-time learning. This time, the learning is in the domain of in-hand object detection and tracking. The chapter introduces a system that has been developed to teach the robot a different number of objects just by simply holding them in the user's hand. The system is based, again, in the Kinect, depth sensor but this time it uses both RGB (Red, Green, Blue) images together with the depth information that the sensor provides. The chapter's main contributions are describing the system and evaluating its learning performance and it serves as an entry point for chapter 6, which extends the system by adding active-learning and better interaction capabilities.

Part III:

Chapter 5 introduces the first approach of the thesis to Active Learning by extending the work presented in Chapter 3. The chapter analyses the impact of using active learning for feature selection in the domain of pose learning. The major contribution of the chapter are two. First it demonstrates that the user can introduce domain knowledge to the system but that this domain knowledge can be biased or incomplete potentially worsening the performance of the active learner. The second contribution consists of some initial ideas of how this issue can be mitigated by reducing the confidence in the user's answers.

Chapter 6 extends the work and the ideas of Chapter 4 by adding to the system active learning and a much more richer interaction to the learning process. The chapter main contribution are twofold. First, it describes the interaction capabilities of the system, which rely on a Dialogue Manager System that is complemented with several Natural Language Processing (NLP) modules. Together, these modules enable the system to, first establish a natural interaction with the user and, second, to virtually support an unconstrained number of different training labels. The second contribution of the chapter is the description and evaluation of the active learning subsystem that enables the robot to take the learning initiative after the first training session ends.

Chapter 7 introduces a different view to the active learning approach. The chapter describes a system that enables a robot to be intrinsically motivated to continuously learn from people. This system extends the active learning capabilities of the robot beyond the learning phase allowing it to actively seek for new, unknown, and interesting training instances. The approach is based on novelty detection techniques, but adapted to robotics and the domain of pose learning. The main contribution of this chapter is the use of novelty detection for interactive learning in robotics.

Part IV:

Chapter 8 concludes the document by summarizing the key contributions of this thesis and by offering a glimpse of the research lines that remain and that have been opened thanks to the findings and contributions this project.

Chapter 2

Active Learning

*Active Learning is a branch of machine mearning that has recently attracted the attention of the robotics community. The purpose of Active Learning is to improve the robot's learning performance—both in terms of accuracy and training speed—by enabling the robot to decide which training examples are the most useful for its own training. In our approach, as we will see in later chapters, we use these techniques to improve the robot's learning speed and performance by letting it to decide which questions it should ask and when it should ask them. This is specially relevant since, given our approach in which our robot learns by asking questions to the user, we want to avoid overwhelming the user with too many queries, so we want to make the robot's queries as most effective as possible. This chapter gives an overview of the Active Learning techniques that are used in traditional machine learning by introducing the concepts and the heuristics that have been more widely adopted by the machine learning community. A big part of the chapter consists in a summary of [*Settles, 2012*], which is currently one of the most relevant introductory texts and literature reviews in the field of Active Learning. We opted for using his work, instead of creating a new literature review, because the book is recent, relevant, and exhaustive. Therefore, our approach in this chapter is to present a summary of the parts of the book which we consider more relevant for the work in this thesis.*

2.1 Introduction

Active Learning is a branch of machine learning whose aim is to use an *oracle*—usually a human—, to provide some sort of knowledge to the learning algorithm, generally, in form of labels for unlabelled instances[1]. Active Learning can potentially reduce the system's learning time and increase its learning accuracy. The *oracle* provides this knowledge by labelling an instance whose label was unknown by the classifier. The key idea is that, to maximize learning performance, the classifier should ask for the most uncertain instances. That is, the instances are the ones that can potentially have the highest impact in the learning process.

AL is especially useful when unlabelled data is abundant and getting labels for such data is expensive, two conditions that are common in many robot learning scenarios. This fact has attracted the robotics community to the field and some techniques of AL have been incorporated in many robot learning workflows.

Due the nature of this thesis, whose core chapters take the form of self-contained papers, we considered that it was necessary to add a chapter introducing the basic concepts of Active Learning—specially their Machine Learning aspects—, so the reader can get a broader view of the field. For that reason, this chapter introduces the main approaches to the AL and describes the most adopted heuristics to decide which queries should be asked. The approach that we have followed in this chapter is to summarize and adapt the work of Settles [Settles, 2012] to the contents of this thesis instead of creating a new introductory text to the topic. Settle's book is one of the most relevant literature surveys and introductory texts to the field while, at the same time, is recent enough to discourage us of creating a new competing work to the topic.

2.1.1 Active Learning Scenarios

We can consider two main different scenarios where Active Learning can be applied. These scenarios depend on how the learning dataset is acquired.

Stream-Based Selective Sampling We choose an unlabelled sample from a data stream and we decide whether to ask the *oracle* or not.

Pool-Based Sampling We choose a sample from a data pool and ask for that sample to the *oracle*. The same label is associated to similar data points in the learning space.

In short, we could say that *Stream-Based Selective Sample* consists in deciding whether or not ask for each received instance in the data stream while *Pool-Based Sampling* consists in choosing the instances that might be most useful for learning among a pool of

[1] Although it it can also consist in other forms of knowledge such as new training examples, as information regarding features, etc.

data. Choosing between one or another method depends on how the learning problem is approached and how the data is acquired. For instance, as we will see in later chapters, the usual approach in this thesis will be *Stream-Based Sampling*. However, regardless the scenario, the heuristics to decide whether to ask the *oracle* are similar. In this chapter we describe three common approaches of querying heuristics:

1. **Uncertainty sampling**, described in section 2.2. This is one of the most popular techniques due its simplicity, good results, and because it is used, often, as a basis for more complex heuristics. It consists in asking the *oracle* for the most uncertain instances.

2. **Using multiple hypotheses**, described in section 2.3. Similar to *uncertainty sampling*, it consists in a committee (a set of hypotheses), in which each member contributes to the decision of querying for an instance.

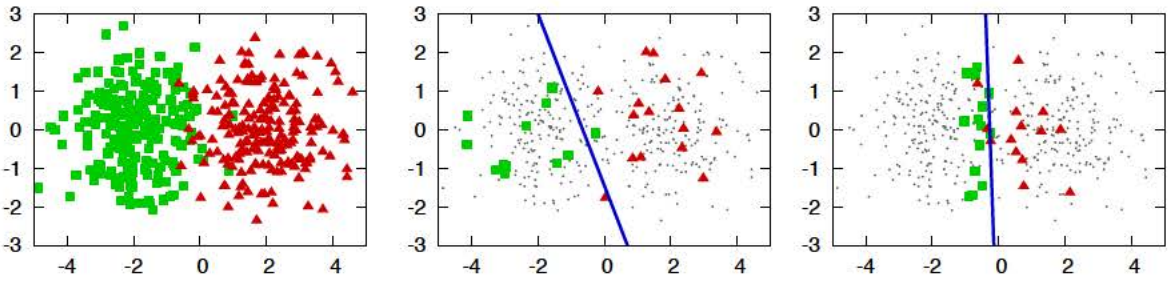
3. **Exploiting the structure in data**, described in section 2.4. In this case, the heuristic consists in complementing one of the previous heuristics with information related to the structure of the dataset (clusters, outliers, etc.).

Although some of the techniques presented in this chapter may not be directly used in this thesis, the methods used in later chapters, share some similarities or belong to the same categories of the methods that are presented in this chapter. For more detail on this, section 2.5 presents a mapping between the concepts presented here and the rest of this thesis.

2.2 Uncertainty Sampling

The key idea of Uncertainty Sampling is that, if you learn from data using a decision boundary and a *Max-margin* approach[1], the instances that are situated closer to the decision boundary are more informative than the other ones. For instance, imagine a dataset containing some labelled data with two different classes and some unlabelled instances. With the labelled data, the learning algorithm can produce a decision function that defines a boundary separating the two classes. Now, if we want to improve this decision boundary, the algorithm has to ask the oracle for the label of one instance. But which label has to choose? It is logical to think that the instances that lie far from the decision boundary are more certain to belong to a specific class. Conversely, the instances located near the decision boundary are more uncertain about their possible class, and they are, therefore, more informative and useful for the learning algorithm. Figure 2.1 gives

[1] Max-margin classifiers are the ones that establish their decision boundary in the region of the learning space that maximizes the separation between classes.



(a) A 2D toy dataset (b) Random Sampling (c) Uncertainty Sampling

Figure 2.1: Uncertainty sampling with a toy data set. (a) 400 instances, evenly sampled from two class Gaussians. Instances are represented as points in a 2D input space. (b) A logistic regression model trained with 30 labelled instances randomly drawn from the problem domain. The line represents the decision boundary of the classifier. (c) A logistic regression model trained with 30 actively queried instances using uncertainty sampling. Image retrieved from [Settles, 2012] with permission from the author. ©Burr Settles.

a visual intuition of this. The figure shows different decision boundaries produced from the same input distribution (Figure 2.1(a)) after 30 instances have been labelled. As we can see, random sampling (Figure 2.1(b)) produced a less descriptive decision boundary compared to a uncertainty sampling heuristic (Figure 2.1(c)).

In summary, the most uncertain instances are, generally, the most informative. Hence, if we want to ask to the *oracle* for labels of the data, it makes sense to ask first for the labels of the instances that are closest to the decision boundary. More formally, we are looking for the instances whose label distribution probabilities $P_\theta(Y|x)$ is as close as possible to a uniform distribution, where Y corresponds to the set of possible labels of the decision function θ given the instance x. For example, in a binary decision problem, we should ask for the labels of the instances x_i whose $P_\theta(\hat{y}|x) \approx [0.5, 0.5]$. But this raises the question of how do we measure the *decision boundary*?, and which uncertainty metrics shall we use to decide for which labels we ask for?.

2.2.1 Measures of Uncertainty

There are many ways of calculating the uncertainty of an unlabelled instance. Settles [Settles, 2012] lists three measures of uncertainty as the most popular ones.

Least Confident. This is the most basic strategy which consists in querying the instance whose predicted value is the least confident.

$$\begin{aligned} x^*_{LC} &= \operatorname*{argmin}_x P_\theta(\hat{y}|x) \\ &= \operatorname*{argmax}_x 1 - P_\theta(\hat{y}|x) \end{aligned} \tag{2.1}$$

where $\hat{y} = \operatorname{argmin}_x P_\theta(\hat{y}|x)$ is the prediction with highest posterior probability under model θ. Therefore, the strategy chooses the instance whose labelling is the most uncertain amongst all the unlabelled instances. One way to understand this strategy is that it asks for the instances that the model thinks that it has mislabelled the instance. The main drawback of this strategy is that, since it is considering only the least likely label, it is also discarding the rest of the posterior distribution.

Margin. Another strategy is to consider the margin of two predictions.

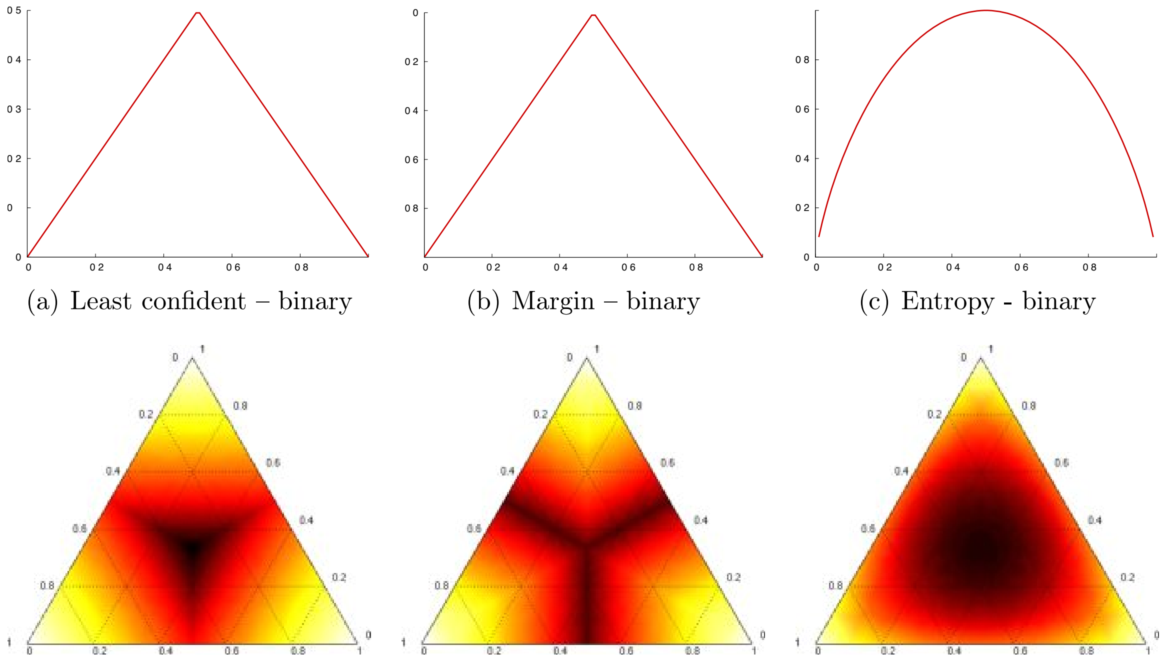
$$\begin{aligned} x^*_M &= \operatorname*{argmin}_x [P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)] \\ &= \operatorname*{argmax}_x [P_\theta(\hat{y}_2|x) - P_\theta(\hat{y}_1|x)] \end{aligned} \tag{2.2}$$

where $\hat{y}_1$ and $\hat{y}_2$ are the first and the second most likely predictions under the model θ. This strategy partially addresses the shortcoming of the *Least Confident* approach by incorporating the second most likely label. The intuition of this strategy is that if the margin is high, then the labelling is certain. On the contrary, if the margin is small, then there is more much more ambiguity of which should be the correct labelling for instance x and, therefore, the instance is a potential query. However, the margin strategy still ignores an important part of the posterior distribution that might be necessary in case the number of alternative labels is high.

Entropy. Finally, the most common strategy is to use the Shannon's *Entropy* [Shannon, 1948], denoted as H, as a utility measure:

$$\begin{aligned} x^*_H &= \operatorname*{argmax}_x H_\theta(Y|x) \\ &= \operatorname*{argmax}_x - \sum_y P_\theta(y|x) \log P_\theta(y|x) \end{aligned} \tag{2.3}$$

where y ranges over all possible labellings for x. Shannon's Entropy measures the average information content of a variable and it is also considered as a uncertainty measure by the machine learning community.



(a) Least confident – binary (b) Margin – binary (c) Entropy - binary

(d) Least confident – ternary (e) Margin confident – ternary (f) Entropy - ternary

Figure 2.2: Comparison of three uncertainty measures for binary classification tasks (a-c) and ternary classification tasks (three labels) (d-f). Each plot shows the utility score as a function of $P\theta(\oplus|x)$, which is the posterior probability of the positive class. In the heat map corresponding to plots (d-f), the corners represent very likely labels, their opposite edges represent the probability range over the other two labels, and the centre corresponds to the uniform distribution.Uncertainty sampling with a toy data set. (a) 400 instances, evenly sampled from two class Gaussians. Instances are represented as points in a 2D input space. (b) A logistic regression model trained with 30 labelled instances randomly drawn from the problem domain. The line represents the decision boundary of the classifier. (c) A logistic regression model trained with 30 actively queried instances using uncertainty sampling. Image retrieved from [Settles, 2012] with permission from the author. ©Burr Settles.

Figure 2.2 shows the relationship between the three measures. In binary classification tasks, *(a-c)* row, the three strategies are monotonic functions of one another. They are symmetric with respect to a peak at $P\theta(\oplus|x) = 0.5$, and they query the instances that lie closest to the decision boundary. In the case of ternary classification tasks[1], *(d-f)* row, the strategies begin to differ. For the three of them, the most informative instances lie in the centre of the triangles. At this point, the posterior label distribution is most uniform and, therefore, the model is most uncertain about the label. On the contrary, the least informative instances lie at the three vertices, where one single class has the most high probability and, therefore, least uncertainty. Is in the rest of the probability space where the strategies differ. For instance, the Entropy measure does not favour instances for which only one of the labels is highly unlikely (i.e, along the edges of the triangle) because the model is fairly certain that this is not the true label. On the contrary, the

[1] 3-label classification tasks

Least Confident and Margin approaches consider useful to query those instances if the model is not certain distinguishing the other two classes (at the midpoint of the triangle edges). These differences occur because Entropy tries to minimize the log-loss, while the other two measures, especially Margin, aim to reduce the classification error. I.e. Least Confident and Margin, prefer to query instances that help the model to discriminate better among specific classes.

The discussion so far on Uncertainty Sampling has focused on *pool-based* scenarios where the strategy is choose the "best" query from the unlabelled dataset $\mathcal{U}$. However, the three measures discussed above are also applicable to *stream-based* scenarios. In *stream-based* scenarios, where unlabelled instances come from an input distribution $x \sim \mathcal{D}$ one at a time, the strategy is to selectively sample them by deciding which incoming instances need to be queried. In such settings, the easiest strategy is to establish a threshold that defines a *region of uncertainty*. Unlabelled instances that fall instance falls inside that region are queried, and discarded otherwise. In this approach, each queried instance is added to the labelled dataset $\mathcal{L}$ and the learner can be trained after a new instance or a batch of instances are added to $\mathcal{L}$.

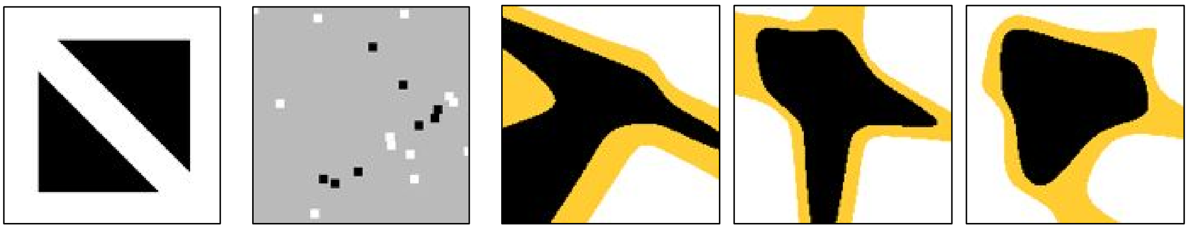
2.2.2 Common Problems of Uncertainty Sampling

Uncertainty sampling is one of the most popular Active Learning techniques. This approach to Active Learning has gained lots of traction in the community because is intuitive and relatively easy to implement. The measures described in section 2.2.1 require very little engineering overhead, specially if the learner provides an estimation of the confidence of its predictions along with the predictions themselves.

However, uncertainty sampling is not without problems. Settles [Settles, 2012] lists three issues that are common to uncertainty sampling and to, in some degree, other active learning strategies. First of all, uncertainty sampling uses only one hypothesis for the utility score. Also, since the nature of AL is to explore datasets with little or no available data, the hypothesis is trained with very little data. Finally, and due the sampling strategy, this data is biased. Therefore, in some situations active learners might omit important areas of the learning space, which might lead to a potential over-generalization.

For instance, in Figure 2.3, Settles illustrates a rather extreme but feasible case of overly-confident generalization from an active learner. The figure shows two black triangles that represent the objective function to be learnt. If an active learner starts asking queries from instances inside and outside the triangles but not from the small region between them, it might end up believing that the target function is similar to the one in Figure 2.3(c).

There are several techniques that tackle these problems, but all of them come at a cost of needing an extra engineering overhead compared to uncertainty sampling. For



(a) Target Function (b) Initial Sampling (c) Uncertainty Sampling

Figure 2.3: Example of an active learner overgeneralizing. (a) Positive instances are located inside the black triangles. (b) If the initial random samples fails to draw training instances from the white space between triangles, the uncertainty sampling stragegies might become very confident, mistakenly, that the region between triangles falls inside the positive class. This situation might lead to avoid querying instances of this area and ending up with a learned space like in (c). Image retrieved from [Settles, 2012] with permission from the author. ©Burr Settles.

that reason, the AL practitioner should hold some domain knowledge that would allow him/her to decide when to use one or another of these alternatives.

2.3 Sampling in the Hypothesis Space

This section takes another approach to *uncertainty sampling* while addresses some of its problems. The approaches shown here, will use different hypotheses to decide whether to query for an instance or not.

2.3.1 Hypothesis Space and Version Space

In machine learning an *hypothesis* is a model that attempts to explain some training data and make predictions on new data instances. Following Settles naming convention, we denote $\mathcal{H}$ as the *hypothesis space*, which is the set of all hypothesis under consideration. Examples of $\mathcal{H}$ are a given neural network structure, or all possible trees that a decision tree can build for that dataset. Therefore a single hypothesis in $\mathcal{H}$ would be a single neural network configuration (from this structure) or a particular decision tree that attempts to predict accurately new data.

From $\mathcal{H}$, it is possible to define the *version space* $\mathcal{V} \subseteq \mathcal{H}$, introduced by [Mitchell, 1982], as the subset of hypotheses which are *consistent* with the training data. $\mathcal{V}$ contains the hypotheses that make correct predictions for the labelled training instances in $\mathcal{L}$. Up to a certain point, the *version space* can be considered as the space that represents the candidate hypotheses equally well. The learning algorithm task during the training phase is to chose one of these hypotheses so it can make future predictions in the exploitation phase. Assuming that the target function is "separable"[1], then obtaining more labelled

[1] In machine learning, a function is separable if it can be expressed by one of the hypotheses in the

data reduces the area of version space $|\mathcal{V}|$. This occurs because, as we obtain more and more data, the hypotheses that remain in $\mathcal{V}$ approximate the target function more and more accurately. Therefore, the objective of a active learning algorithm is to obtain new training instances that minimize $|\mathcal{V}|$ as fast as possible.

2.3.2 Query by Disagreement

Query by Disagreement (QBD) [Cohn et al., 1994] is one of the first active learning algorithms that aimed to reduce the version space. QBD consists in a stream-based scenario in which the learner decides whether to query or discard every incoming instance. To do so, the algorithm evaluates the data instance x against all hypotheses in $\mathcal{V}$. If all hypotheses agree on the predicted label, then the label of x can be confidently inferred and thus x can be safely ignored. On the contrary, if any pair of hypotheses $(h_1, h_2) \in \mathcal{V}$ disagree predicting the label $\hat{y}$ for the data instance x, then the algorithm queries the oracle for the true label of y and incorporates the pair (x, y) to $\mathcal{L}$. An intuition of QBD is that the algorithm tries to reduce the version space by only querying for the instances that fall in a region of disagreement $\text{DIS}(\mathcal{V})$.

However, keeping track of all $h \in \mathcal{V}$ can be infeasible in practice since the version space may be infinite or too big to fit in memory. Also, comparing every pair of hypotheses in $\mathcal{V}$ may suppose a high computational cost. A way to overcome these shortcomings is to use two hypotheses $h_S, h_G \subseteq \mathcal{V}$, where h_S is approximately the most "specific hypothesis" $\mathcal{V}$ and h_G is roughly the most "general hypothesis" of the version space. In short, h_S and h_G are conservative and liberal hypotheses, respectively, that aim to be consistent with the data in $\mathcal{L}$. h_S and h_G present a region of disagreement $\text{DIS}(\mathcal{V}) \approx \{x \in \mathcal{D}_X : h_S(x) \neq h_G(x)\}$ that can be used to decide when to query incoming instances. When the learner receives an instance x from the input distribution $\mathcal{D}_X$, the instance is queried if it falls in the region of disagreement $\text{DIS}(\mathcal{V})$, and otherwise is discarded. A method to encourage the specific and general behaviour of h_S and h_G is to feed both models with artificial "background" data points $\mathcal{B} \sim \mathcal{D}_X$ and add them to L with artificial labels. To the specific model h_S these labels are set to the $\ominus$ negative label, making it more conservative in its predictions. On the other hand, the general model is trained with $\oplus$ labels, which would make it more likely to label the unknown regions as positive. This alternative form of querying is called $\mathcal{SG}$-QBD and it can perform better than Uncertainty Sampling in tasks such as the one shown in Figure 2.3 (see [Settles, 2012]).

version space $\mathcal{V}$.

2.3.3 Query by Committee

Despite QBD is a more advanced strategy than Uncertainty Sampling, it poses two assumptions that are too strict:

- The disagreement is measured among all hypotheses $h \in \mathcal{V}$, or between two approximate extremes h_S and h_G.
- It is a binary measure. That is, every controversial instance has the same importance.

The first problem with these assumptions is that QBD requires to measure the disagreement among every hypothesis in $\mathcal{V}$, which can be intractable or between to imperfect approximate extremes. This can be problematic if the data is noisy because then, $\mathcal{V}$ might not be separable. Second, QBD works only in *stream-based* scenarios (see section 2.1.1). Therefore, it is not possible to use disagreement-based heuristics in *pool-based* scenarios to query the most informative instance $x \in \mathcal{U}$.

The *Query By Committee* (QBC) algorithm [Seung et al., 1992] addresses these shortcomings and makes disagreement-based heuristics more broadly applicable. In the original formulation Seung samples 2 random hypotheses and uses the notion of binary disagreement to decide whether to query or not. Since then, QBC has evolved to be considered to any disagreement-based approach that uses a "committee" or ensemble of hypotheses denoted as $\mathcal{C}$. Therefore, in practice QBC is any method to obtain hypotheses from a committee and a heuristic for measure the disagreement amongst them.

Given this definition of QBC, it is natural to think of generic ensemble learning algorithms such as *boosting* [Freund and Schapire, 1997] or *bagging* [Breiman, 1996] to form the committee. The application of the former to active learning is called *query by boosting* and uses a *boosting* algorithm to iteratively form a set of hypotheses that become more and more specialized in each iteration by increasingly focusing them on the erroneous instances in $\mathcal{L}$. The latter uses a *bagging* algorithm which aims to smooth out high-variance predictions by training a committee of ensembles based on resamples of $\mathcal{L}$ with replacement Both methods were first proposed by [Abe and Mamitsuka, 1998] and have been extensively used since then. From all the literature using these methods Settles highlights two works. First, [Melville and Mooney, 2004], which proposes an ensemble-based method that explicitly encourages for diversity in the committee. Second, [Muslea et al., 2000], which proposes a method to construct a committee of hypotheses by partitioning the feature space into conditionally independent committee-specific subsets, that are used in conjunction with semi-supervised learning (see section 2.4.2).

Settles did not found in the literature a general rule to decide which is the optimal number of members for a committee. As it seems, it is common to have committees of five to fifteen members (hypotheses), but also smaller committee sizes have proven to work well in certain cases.

A key aspect of QBC is to decide how to measure the disagreement between committee members. Settles points out that the literature is quite diverse on heuristics for that matter, but points out two dominant trends, an entropy-based disagreement heuristic and a *Kullback-Leibler* divergence-based. The first is essentially a committee-based generalization of the uncertainty measures presented in section 2.2.1. This approach is called *vote entropy* and it is defined as follows:

$$x^*_{VE} = \underset{x}{\operatorname{argmax}} - \sum_{y} \frac{\text{vote}_{\mathcal{C}}(y,x)}{|\mathcal{C}|} \log \frac{\text{vote}_{\mathcal{C}}(y,x)}{|\mathcal{C}|} \tag{2.4}$$

where y ranges over all possible labellings of x, $\text{vote}_{\mathcal{C}}(y,x) = \sum_{\theta \in \mathcal{C}} 1_{\{h_\theta(x)=y\}}$ is the number of votes for y given the instance x among all the hypotheses $h_\theta \in \mathcal{C}$, and $|\mathcal{C}|$ is the committee size. This heuristic is considered to be a "hard" vote entropy measure and it is complementary to a "soft" vote entropy that includes the committee member's confidence in their predictions:

$$x^*_{SVE} = \underset{x}{\operatorname{argmax}} - \sum_{y} P_{\mathcal{C}}(y|x) \log P_{\mathcal{C}}(y|x) \tag{2.5}$$

where $P_{\mathcal{C}}(y|x) = \frac{1}{|\mathcal{C}|} \sum_{\theta \in \mathcal{C}} P_\theta(y|x)$ is the average –also called "consensus"– probability that y is the correct label according to the committee. In essence these measures are Bayesian versions of the entropy-based uncertainty sampling from Equation 2.3, but instead of using a point estimate –that is, a single hypothesis–, using $\text{vote}(y,x)/|\mathcal{C}|$ or $P_{\mathcal{C}}(y|x)$ as the ensemble's posterior label estimate. Intuitively, the ensemble approach can smooth out any hard over-generalizations made by a single hypothesis. It is possible to build analogous ensemble-based generalizations for the least-confident and margin heuristics from Equations 2.1 and 2.2 respectively.

The second QBC disagreement measure is based on the *Kullback-Leibler (KL) divergence* [Kullback and Leibler, 1951], which is an information-theoretic metric that measures the difference between two probability distributions. Here, the KL divergence quantifies the disagreement as the average divergence of the prediction θ of each committee member from that of the consensus $\mathcal{C}$:

$$x^*_{KL} = \underset{x}{\operatorname{argmax}} \frac{1}{|\mathcal{C}|} \sum_{\theta \in \mathcal{C}} KL\big(P_\theta(Y|x) \parallel P_{\mathcal{C}}(Y|x)\big) \tag{2.6}$$

where KL divergence is defined as:

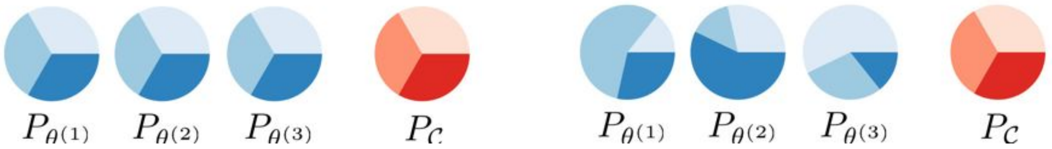
$$KL\big(P_\theta(Y|x) \parallel P_{\mathcal{C}}(Y|x)\big) = \sum_{y} P_\theta(y|x) \log \frac{P_\theta(y|x)}{P_{\mathcal{C}}(y|x)} \tag{2.7}$$

The main difference between the *vote entropy* and the *KL divergence* lays on how they quantify the disagreement. This difference can be exemplified in Figure 2.4. In 2.4(a), the predictions of the committee members are distributed uniformly, while in Figure 2.4(b) the individual hypothesis distributions are non-uniform and each member prefers a different label. However, in both cases, the consensus ends up with a similar distribution. Vote entropy (Eq. 2.5) only considers the consensus $P_{\mathcal{C}}$, and since in both subfigures the consensus ends up with high entropy, the vote entropy is incapable of distinguishing between both cases. In a way vote entropy can be considered as a Bayesian generalization to multiple hypotheses of the entropy-based uncertainty sampling. Settles points out that querying instances with predictions like in Figure 2.4(a) does not comply with the disagreement-based spirit of QBC. Although the consensus label is uncertain, the committee members *agree* that it is uncertain. The KL divergence (Eq. 2.6), on the contrary, favours instances with the distribution of predictions such as in Figure 2.4(b). Here, the consensus is uncertain, but only because the committee members differ in their predictions, which represents more accurately the original notion of disagreement.

Despite that Settles only describes these two metrics, he points to a couple of other common disagreement measures for QBC in the literature for classification tasks. The first one is the Jensen-Shannon divergence [Melville et al., 2005], which is a smoothed, symmetric version of the KL divergence. The second one is a measure called F-compliment [Ngai and Yarowsky, 2000], which uses the F_1 measure [Manning and Schütze, 1999] to quantify the disagreement among the committee members.

2.4 Exploiting Structure in Data

Both *uncertainty sampling* and *Query By Committee* (QBC) tend to sample the most informative instances in a data pool, These are, usually, the ones closer to the decision boundary but, in some situations, those instances might be outliers in the input distribution. Consider, for instance, the example shown in Figure 2.5. Here, a binary classifier using uncertainty sampling would likely to query the data instance A. However, A is



(a) Uncertain but in agreement (b) Uncertain and in Disagreement

Figure 2.4: Examples of consensus of the committee members. $P_{\theta^{(i)}}$ is the output distribution of the ith member's hypothesis, while $P_{\mathcal{C}}$ refers to the consensus across all the members of the committee. Image retrieved from [Settles, 2012] with permission from the author. ©Burr Settles.

likely an outlier and, therefore, is not representing properly the data distribution, so choosing such query it is unlikely to improve the classifier's accuracy on the data. The same behaviour is likely to occur with QBC since these kind of instances are uncertain or controversial precisely because they are outliers.

This section addresses this problem by using heuristics that are able to exploit the structure of data to select the instances to query. These approaches are usually called *Density-weighted* heuristics and work by looking for instances that are not only highly informative, but also representative of the underlying distribution in some sense—for instance, laying in dense regions of the input space. A generic heuristic that represents this concept is the *information density* heuristic, which it is defined as:

$$x^*_{ID} = \underset{x}{\operatorname{argmax}} \left(\phi_A(x) \times \left(\frac{1}{U} \sum_{x' \in \mathcal{U}} \operatorname{sim}(x, x') \right)^{\beta} \right) \tag{2.8}$$

where U is the size of the unlabelled data pool and $\phi_A()$ represents the utility of x according to some "base" query strategy A, such as *uncertainty sampling* or *QBC*. For example, the uncertainty sampling utility measure would be defined as $\phi_H(x) = H_\theta(Y|x)$. Note that while the first term returns the informativeness of x, the second term weights this informativeness by taking into account the similarity of x to all other instances in $\mathcal{U}$. The parameter β controls the relative importance of this density term. $\operatorname{sim}(x, x')$ can take several forms such as *cosine similarity*, *Euclidean distance*, *Pearson's correlation coefficient*, *Spearman's rank correlation*, or any other similarity measure.

Settles suggests that density-weighted approaches can outperform simpler methods that do not include density or representativeness in their calculations. This claim is backed up by the literature [Fujii et al., 1998; McCallum and Nigam, 1998; Settles and

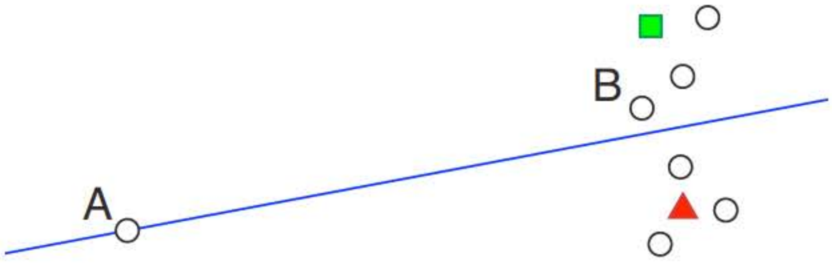


Figure 2.5: An example of uncertainty sampling not choosing the best data instance. The shaded polygons are labelled instances in $\mathcal{L}$, while circles represent unlabelled instances from $\mathcal{U}$. Note that A is on de decision boundary so, following uncertainty sampling, it would be queried for being the most uncertain. However, B can potentially provide more information about the input distribution as a whole. Image retrieved from [Settles, 2012] with permission from the author. ©Burr Settles.

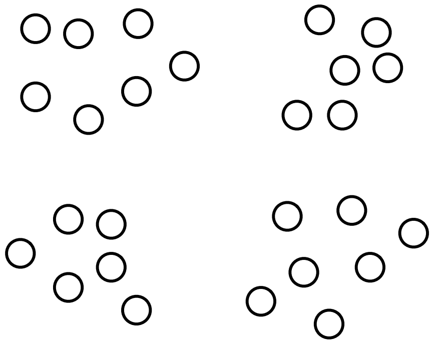


Figure 2.6: An example of cluster-based active learning. This input distribution seems to indicate that only four labels are needed. Image retrieved from [Settles, 2012] with permission from the author. ©Burr Settles.

Craven, 2008; Xu et al., 2007]. A major drawback of the density-weighted approaches might come from the time required to compute the similarity between instances. However, as Settles points out, this problem can be mitigated by pre-computing and caching the similarity for efficient lookup. This can reduce the computing time of density-weighted approaches to almost the same requirements as the based informativeness measure, e.g. uncertainty sampling [Settles and Craven, 2008]. This enables to use these methods in real-time scenarios with interactive *oracles*.

2.4.1 Cluster-Based Approaches to Active Learning

The *density-weighting* approach presented is a simple method to exploit the structure of the input distribution. An interesting alternative to exploit this structure is to apply other unsupervised learning techniques like clustering (see Figure 2.6). For instance, variants of *density-weighting* might consist in clustering the pool $\mathcal{U}$ and then compute the average similarities of each instance x to the other instances of its cluster. This approach has a complexity of $O(N^2)$ or faster for lager data sets, depending on the data and the used clustering algorithm. Another approach is to use clustering to "warm-start" $\mathcal{L}$ by pre-clustering the data and query the cluster centroids instead of a random sample [Kang et al., 2004]. Also, it is possible to cluster the most informative instances every iteration and then query the instances that are more representative of these clusters [Nguyen and Smeulders, 2004].

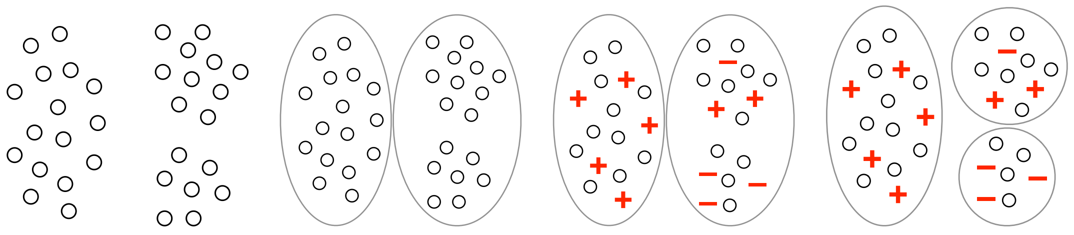
However, these approaches are not free of problems. First, it might not exist a clear clustering for $\mathcal{U}$ or neither a good similarity metric for separating the data into clusters. In addition, choosing the right number of clusterings is not an obvious task, specially if the data can be clustered at different levels of granularity. Finally, the clusters in $\mathcal{U}$ may not correspond to the given labels. To solve these problems [Dasgupta and Hsu, 2008] proposes the *hierarchical sampling* method. As Settles points out, their approach is interesting because "it takes the best of both worlds": it does not make hard assumptions between the input and label distributions, while is able to take advantage of the data structure when they are informative. The algorithm is shown in Figure 2.7. In short, it first takes the data (Figure 2.7(a)) and finds an initial coarse clustering (Figure 2.7(b)). Then it queries instances from these clusters (Figure 2.7(c)) and iteratively refines the clustering making them more pure by focusing its queries on the most impure clusters (Figure 2.7(d)).

Note that the algorithm queries *clusters* to obtain labels for instances that are randomly drawn from these clusters. Settles stresses that this is a key difference between *hierarchical sampling* and other active learning algorithms presented so far because, this enables the algorithm to keep valid estimates for the error induced by the current pruning regardless of how the clusters are formed.

An interesting feature of the algorithm is that, if the clusters are highly impure—that is, there is a low correlation between the cluster and the label structure—the algorithm gracefully degrades to random sampling. However, if the cluster structure and the hidden label distribution are related, it takes advantage of this relation to sample more often from the impure clusters. In this case, if the clustering contains that are ϵ-pure, sampling algorithm presents a complexity of $O(|\mathcal{P}|d(\mathcal{P})\epsilon)$, where $d(\mathcal{P})$ is the maximum depth of the pruning $\mathcal{P}$. More details of the algorithm can be found in [Settles, 2012] and in [Dasgupta and Hsu, 2008].

2.4.2 Relation of Active Learning and Semi-Supervised Learning

Semi-supervised learning [Zhu and Goldberg, 2009] and active learning share the same objective of improving supervised learning in contexts where labelled data is scarce or difficult to obtain. However, they approach the problem from different but complementary perspectives. In one hand, active learning, focus on minimizing the training effort by querying the most informative instances. On the other hand, semi-supervised learning objective is to improve the quality of the already learned model by exploiting the latent structure of data. In short, semi-supervised learning models try to "teach" themselves by



(a) Data (b) Clustering (c) Queries (d) Refinement

Figure 2.7: The basic stages of active learning by hierarchical sampling. Image retrieved from [Settles, 2012] with permission from the author. ©Burr Settles.

extrapolating what they think they have learned onto unlabelled instances. As [Settles, 2012] points out, semi-supervised techniques have usually some complementary counterpart active learning method. Following, we present two semi-supervised learning methods with complementary active learning methods that have been presented previously: uncertainty sampling and QBC.

The first technique is *self-traning* [Yarowsky, 1995], which consists in the learner being trained with a small amount of labelled data, and then letting the learned model to classify the unlabelled data to re-train itself. This is typically an iterative process where, in each iteration, the model adds the pairs $\langle x, y' \rangle$ with the most confident predictions y' to the training set $\mathcal{L}$. However, note that this process assumes that the learner can trust its most confident predictions. The active learning complementary technique to self-training is uncertainty sampling. Whereas self-training uses the most confident predictions, uncertainty sampling queries the instances in which the model is least confident assuming that the learner requires most guidance from the oracle on these instances.

The second technique is *co-training* [Blum and Mitchell, 1998], which uses ensembles for semi-supervised-learning. It consists in training separate models with the labelled data (typically using separate, conditionally independent feature sets) and letting these models to classify the unlabelled data. Then, each model's most confident predictions are used as training data of the other model. Co-training forces the different views of the data to agree not only on $\mathcal{L}$ but on $\mathcal{U}$ as well, which leads to a reduction of the version space. Query by committee is complementary to *co-training*, as it uses its committee to approximate different parts of the version space, and to query the the unlabelled instances where the competing hypotheses do not agree.

There is a third semi-supervised technique mentioned by Settles, *entropy regularization* [Grandvalet and Bengio, 2005], which is based on the intuition that the best models make the most confident predictions on the unlabelled data. In short, this method consists in adding a second regularization term to the model that penalizes parameter settings with the highest risk of making mistakes on the unlabelled data. The active learning complementary method to *entropy regularization* is the *log-loss expected error reduction* [Roy and McCallum, 2001], which we omitted in this thesis since it is not relevant to our work.

These semi-supervised methods are examples of how many active and semi-supervised learning algorithms address the same problem—taking as much profit as possible of the unlabelled data—but from opposite perspectives. On one hand, semi-supervised learning techniques exploit what the model thinks about the unlabelled data, while, on the other hand, active learning techniques explore the unknown aspects of these data. The complementary nature of both approaches make them easy to combine them to address different problems [McCallum and Nigam, 1998; Muslea et al., 2002; Tomanek and Hahn, 2009;

Tur et al., 2005; Yu et al., 2006; Zhou et al., 2004; Zhu et al., 2003].

2.5 Conclusions

This chapter has presented an introductory overview to Active Learning. The aim of the chapter has been to establish a common ground for the chapters in Part III, and to enable the reader who is not familiar with the field a minimum understanding of its core concepts.

Most of the topics covered in this chapter will be addressed, either directly or indirectly, in Part III of this thesis. In Chapter 5 Active Learning is used in a robot that learn to recognise human poses. Here the *oracle* answers a series of questions whose responses are used as feature filters for learning. The ideas presented in this chapter, will be related on taking profit of the structure of the data to reduce possible inaccuracies of the user response to the robot's queries. In Chapter 6 AL is used in an object recognition scenario. The robot asks queries when new unlabelled instances are uncertain and receives its responses in form labels. Here, a small committee decides whether to ask a query to the user. Finally, Chapter 7 uses techniques related to Active Learning to understand if the pose of a human is unknown to the system and if it would be useful for the system to learn it. The ideas presented in this chapter propose the use of novelty detection algorithms to analyse the structure in the data to decide whether the robot should ask for the label of a new instance or not.

Part II

Interactive Learning for Social Robots

Chapter 3

Learning Poses Interactively

This chapter begins a series of chapters that correspond with papers that either are already published in Journals or that are in the process of being published. Concretely, this chapter corresponds to [Gonzalez-Pacheco et al., 2013], which is our first approach to a social robot with interactive and natural learning.

The main activity of social robots is to interact with people. In order to do that, the robot must be able to understand what the user is saying or doing. Typically, this capability consists of pre-programmed behaviors or is acquired through controlled learning processes, which are executed before the social interaction begins. This paper presents a software architecture that enables a robot to learn poses in a similar way as people do. That is, by hearing its teacher's explanations and acquiring new knowledge in real time. The architecture leans on two main components: an RGB-D (Red, Green, Blue, Depth) based visual system, which gathers the user examples, and an Automatic Speech Recognition (ASR) system, which processes the speech describing those examples. The robot is able to learn the poses the teacher is showing to it by maintaining a natural interaction with the teacher. We evaluate our system with 24 users who teach the robot a predetermined set of poses. The experimental results show that, with a few training examples, the system reaches high accuracy and robustness. This method shows how to combine data from the visual and auditory systems for the acquisition of new knowledge in a natural manner. Such a natural way of training enables robots to learn from users, even if they are not experts in robotics.

3.1 Introduction

Human Robot Interaction (HRI) is the field of research that studies how humans and robots should interact and collaborate. Humans expect robots to understand them as other people do. In this aspect, a robot must understand natural language and should be capable of establishing complex dialogues with its human partners.

However, dialogue is not only a matter of words. Most of the information exchanged in a conversation comes from non-verbal cues like poses and arms or face gestures.

Gesture and pose recognition systems have been an active research field in recent years [Mitra and Acharya, 2007]. However, traditional image capture systems require the use of complex statistical models to recognize the body, making them difficult to be used in practical applications [Jaume-i Capó and Varona, 2009].

Recent technological developments are making new types of vision sensors more suitable for interactive scenarios [Foix et al., 2011]. These devices are depth cameras [Foix et al., 2011; Freedman et al., 2008; Scharstein and Szeliski, 2003], which make the extraction of the human body easier than with traditional cameras. Therefore, since extracting the body is much less CPU-consuming now, it is possible to execute in real time algorithms that actually process the user's gestures or poses.

Specially relevant is the case of the Microsoft Kinect sensor[1]. The kinect is a low-cost depth camera whose precision and performance is similar to high-end depth cameras, but at a cost several times lower. Together with the Kinect, several drivers and frameworks to control it have appeared. These drivers and frameworks provide direct access to a model of the user's skeleton. This model is precise enough to track the pose of the user and to recognize his/her gestures in real time.

Using these new vision sensors with Automatic Speech Recognition (ASR) systems enables robots to learn interactively from human examples. In this chapter, we present a software architecture that enables the social robot, Maggie [Salichs et al., 2006], to learn human poses using depth information coming from a Kinect camera and to process the user's voice explanations in real time. We take advantage of our robot's interaction capabilities to let it learn poses by interacting with its human teacher. In short, the user acts as a teacher, telling it in which pose she is standing. The robot is able to understand what the user is saying and fuses that information with the depth data to learn the current user pose. In such an interactive way, the robot is able to learn new knowledge through interaction with the users, and new context knowledge can be acquired incrementally in the long-term.

This document is organized as follows. Section 3.2 presents an overview of the related work in pose and gesture classification with depth cameras. Section 3.3 introduces an

[1] https://en.wikipedia.org/wiki/Kinect

overview of the hardware and software systems that act as the building blocks of the developed architecture. This section describes hardware components, such as the robot, Maggie, and the Kinect camera, as well as the software modules that act as the scaffold of the project. Section 3.4 describes the developed software architecture and is followed by the description of the experimental validation we have carried out to validate our architecture in Section 3.5. Finally, Section 3.6 closes the chapter, presenting the main contributions, broader issues and future remarks that are still open with our approach.

3.2 Related Work

3.2.1 Pose Recognition Using Depth Cameras

Depth cameras are systems that can build a 3D depth map of a scene by projecting light to that scene. The principle is similar to that of LIDAR (Laser Interferometry Detection and Ranging) scanners, with the difference being that the latter are only capable of performing a 2D scan of the scene, while depth cameras scan the whole scene at once.

Depth cameras are an attractive tool in several fields that require intense analysis of the 3D environment. Two surveys thoroughly describe the field. The first one [Kolb et al., 2009] dates from 2009 and surveys the technologies and applications prior to the release of the Kinect Sensor. This sensor revolutionized the field by making available a high-resolution and high-precision technology at consumer prices. A more recent survey (2013), but more focused on algorithms for body-motion analysis, is presented in [Chen et al., 2013].

However, the idea of using depth cameras for body analysis is not recent. For example, in references [Fujimura, 2004; Gokturk and Tomasi, 2004] their use to locate body parts is proposed. Since then, many other works have researched gesture recognition with depth cameras [Breuer et al., 2007; Droeschel et al., 2011; Haubner et al., 2010; Lahamy and Litchi, 2010; Nickel and Stiefelhagen, 2007]. Some of these works rely on kinematic models to track human gestures once the body is detected [Boulic et al., 2006; Ramey et al., 2011; Zhu et al., 2010].

Most of these works rely on capturing only one or few parts of the body. However, recent kinematic approaches, like the one in [Zhu et al., 2010], make possible the tracking of the whole body without a significant increase in CPU consumption. Schwarz *et al.* [Schwarz et al., 2011] propose a method to estimate the full body by transforming the foreground depth image into a point cloud. Then, they determine the centroid of this point cloud and find the primary landmarks by calculating the geodesic distance along the 3D body mesh. Shotton *et al.* [Shotton et al., 2011] are the authors of the human pose estimation technology used in the Xbox. They proposed a skeleton model, where the joints are fitted to previously labeled body parts using mean shift.

Our approach focuses on teaching concepts interactively to a social robot. Therefore, rather than extracting the body and tracking it directly, it uses the available technologies and algorithms as data-sources, which will be used to enable the grounding of high-level concepts, such as the name of a certain pose. Concretely, our vision system relies on the OpenNI[1] (NI stands for Natural Interaction) libraries for body extraction and tracking. OpenNI's g tracking is similar to the ones mentioned above.

3.2.2 Machine Learning in Human-Robot Interactions

Fong *et al.* present a survey [Fong et al., 2003] of the interactions between humans and social robots in which the authors stress that the main purpose of learning in social robotics is to improve the interaction experience. At the time of the survey (2003), most of the learning applications were used in robot-robot interaction. Some works addressed the issue of learning in human-robot interaction, mostly focusing on imitating human behaviors, such as motor primitives. According to the authors, learning in social robots is used for transferring skills, tasks and information to the robot. However, the authors do not mention the use of learning for transferring concepts, such as poses, that enable the robot to understand the user better.

Later, Goodrich and Schultz [Goodrich and Schultz, 2007] stressed the need of robots with learning capabilities, because of the complexity and unpredictability of human behaviors. They pointed out the need of a continuous learning process, where the human can teach the robot in an *ad hoc* and incremental manner to improve the robot's perceptual ability, autonomy and its interaction capabilities. They called this process *interactive learning*, and it is carried out by natural interaction. Again, their survey only reports works that referred to learning as an instrument to improve abilities, behavior, perception and multi-robot interaction. No explicit mention was made to use learning to provide the robot with high level concepts, such as the user's pose. The same occurs in [Argall et al., 2009], which presents a survey of several Learning from Demonstration (LfD) approaches and categorizes them depending on how robots collect the learning examples and how they learn a policy from these examples.

A few works use learning to teach concepts to the robot. This is the case of [Mahadevan et al., 1998], where the authors train a mobile robotic platform to understand concepts related to the environment in which it has to navigate. The authors use a feed forward neural network (NN) to train the robot to understand concepts, like doors or walls. They train the NN by showing it numerous images of a trash can (its destination point), labeling each photo with the distance and the orientation of the can. However, the work presented

[1] http://structure.io/openni

some limitations, such as the learning process lacked enough flexibility to generalize the work to other areas.

More importantly, there are some works where interactive teaching relies on the teacher's voice to acquire knowledge [de Greeff et al., 2009; Nicolescu and Mataric, 2001], but to the extent of our knowledge, most of these works rely on simple interactions, often using one or a few voice commands to describe the examples. Our work is closer to Rybski *et al.* [Rybski et al., 2007], where the authors created a dialogue-based system used to teach a robot different tasks. This chapter presents an approach similar to Rybski's and colleagues, in the sense of dialogue complexity for teaching, but, in our approach, instead of only using dialogue for describing the concept to be learned, we use it to describe what the robot is seeing. We also include visual information in the learning process.

Other similar approaches are [Goerick et al., 2009; Heckmann et al., 2009]. Both consist in the integration of visual and aural systems to teach a humanoid robot different spatial concepts. Despite that their ideas are similar to our work, there are some differences. First, the robot in [Goerick et al., 2009] learns spatial concepts related to objects, while our focus is on human-pose learning. Second, the authors in [Goerick et al., 2009] focus on the description of the cognitive architecture. Our approach, however, focuses on the HRI point of view. In this way, while the interaction in [Goerick et al., 2009] leans on simple commands issued by the tutor, in our work, the teacher has a wide range of utterances that can be used to teach the same pose to the robot.

On the other hand, Heckmann *et al.* [Heckmann et al., 2009] focus on the aural system of the robot to enable it to understand the user without the need for an external microphone. Compared to our work, in [Heckmann et al., 2009], the robot not only learns the visual representation, but also the auditive labels. However, each new auditive label needs five to eight repetitions from the tutor to be learned by the robot. Our approach prefers to ease the interaction during the learning session. Although our grammars must be pre-written by the robot programmer, they facilitate the task of the tutor by allowing the user to use many different utterances to refer to the same semantic label. This is an advantage to the tutor, who can express himself/herself in a much more natural way.

3.3 Hardware and Software Platform

Before entering into the design of our proposed system, we introduce the different building blocks that are necessary to build our system. In this section, we describe the different hardware and software components that enable our robot to learn the poses while interacting naturally with the user.

3.3.1 The Robot, Maggie

Maggie is a robotic platform developed by the RoboticsLab team at the Carlos III University of Madrid aimed at research in social robotics and HRI.

The hardware components of the robot are intended to allow the robot to interact with its environment and with humans. The most relevant ones are depicted in Figure 3.1. Except the webcam, all of them are used in this work. A complete description of the robot can be found in [Salichs et al., 2006].

3.3.2 The Kinect Vision System

The Microsoft Kinect RGB-D sensor is a peripheral that, was initially designed as a videogame controlling device for the Microsoft's X-Box Console. The sensor provides a depth resolution similar to the high-end Time-of-Flight (ToF) cameras, but at a cost several times lower. This fact has made the Kinect a widely adopted sensor in the robotics community.

This sensor retrieves the depth information using light coding technology, which consists of emitting a dot pattern onto the scene [Freedman et al., 2008; Zalevsky et al., 2007]. The Kinect is composed of one IR (Infrared) emitter, responsible for emitting the light pattern onto the scene, a depth sensor, responsible for capturing the emitted pattern, and a standard RGB sensor that records the scene in visible light. The system has a spatial resolution of 3 mm in the x/y axes and of 1 cm in the z axis at a distance of two meters. This spatial resolution is sufficient to accomplish pose detection tasks.

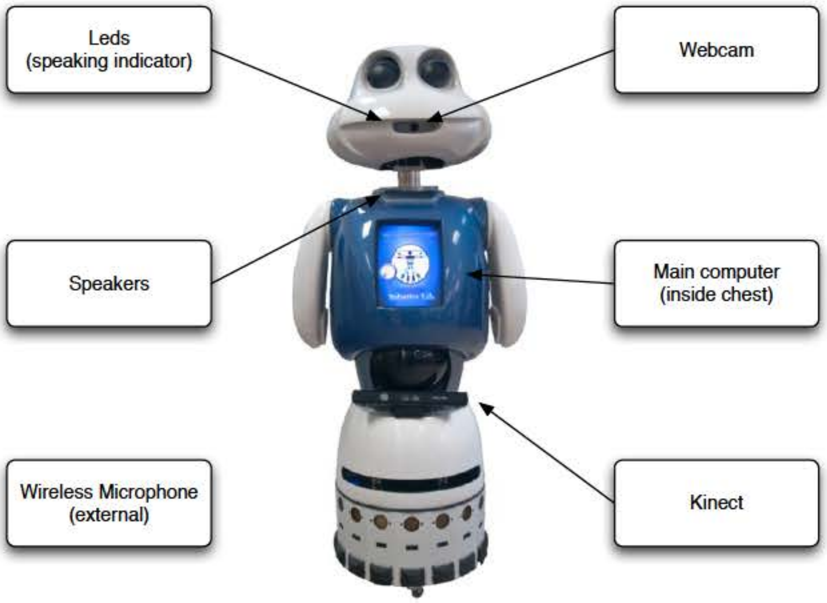


Figure 3.1: Interaction elements of the robot, Maggie. Among all of them, the learning system relies on the data captured by the Kinect and the wireless microphone.

3.3.3 The AD Software Architecture

Maggie's main software architecture is an implementation of the Automatic-Deliberative (AD) Architecture [Barber, 2000]. The basic component of the AD architecture is the *skill* [Rivas et al., 2007]. A skill is the minimum module that allows the robot to execute an action, such as moving through the environment, reading the output from a laser sensor or communicating with a human.

In essence, a skill is a process that carries out computing operations and shares the results of these operations with other skills. For example, imagine a skill that is in charge of detecting obstacles using the laser readings. In this case, the main operations of the skill are: reading the laser data, deciding whether there is an obstacle or not and making this information available to other skills. The sharing mechanisms used by the skills are events (a communication mechanism that follows the *publisher/subscriber* paradigm described by Gamma *et al.* in [Gamma et al., 1995]) or a shared memory system.

AD is implemented in the ROS (Robot Operating System) framework. ROS [Quigley et al., 2009] is an open-source, meta-operating system for robots. It provides services similar to the ones provided by an operating system (OS), including hardware abstraction, low-level device control, implementation of commonly-used functionalities, inter-process communication and packet management. Additionally, it also provides tools and libraries for obtaining, building, writing and running code in a multi-computer environment.

The main concepts of ROS that are relevant to this chapter are nodes and topics. The former are the minimum unit structure of the ROS architecture. They are processes that perform computation. Essentially, each AD skill is implemented as an ROS node. The latter, topics, are the communication system that enables the information exchange between nodes. They are, in fact, an implementation of the AD's events. Since AD is a conceptual architecture and ROS its implementation, in the rest of the chapter, we might refer to the terms skill/node and event/topic indistinctly.

3.3.4 Description of the Voice System of the Robot

To enable the robot to communicate by voice with the user, two software tools are needed: a Text To Speech (TTS) tool to talk with the user and an Automatic Speech Recognition (ASR) tool to hear and understand what the teacher says. The first tool, TTS, is a technology that transforms written information into spoken words, that is, a TTS says any text that it receives as an input. On the contrary, an ASR transforms any human utterance, captured by the microphone of the robot, to written text, which can be understood by a computer.

AD uses commercial TTS and ASR tools provided by Loquendo[1]. This vendor provides

[1] https://en.wikipedia.org/wiki/Loquendo. At the beginning of this thesis, Loquendo was ac-

APIs (Application Interfaces) for both TTS and ASR. These APIs are wrapped in the form of two skills: the *ETTS Skill* (Emotional Text to Speech) and the *ASR Skill* [Alonso-Martín and Salichs, 2011]. These are wrapped in the form of skills, so they permit other skills to send utterances to the *ETTS skill* and to retrieve what the user has said from the *ASR Skill* by simply using the communication mechanisms provided by AD.

3.3.4.1 Enabling the Robot to Talk: The ETTS Skill

Maggie's speaking capabilities lean on the ETTS Skill. The ETTS Skill wraps and adds some features to the Loquendo TTS API. We chose Loquendo's TTS over others because of its superior voice synthesis quality and its greater configuration possibilities [Alonso-Martin et al., 2011].

It is possible to configure several utterance parameters, such as the tone or the speed of the locution. This allowed us to create four predefined emotional states that the robot can express: calmed, nervous, happy and sad. Thanks to that, the robot produces a high quality synthetic voice that is pleasant to humans and that improves the likability of the robot. A grammar establishes a series of words or a combination of words and links them to a semantic meaning. The semantics of those words are coded into labels that can be codified as variables in a computer program. When a skill wants to make the robot talk, it simply sends an event to the ETTS Skill with the utterance to say and with the emotion parameters to express. The ETTS Skill manages Loquendo's API to produce the utterance with the appropriate desired parameters.

3.3.4.2 Speech Recognition: The ASR Skill

To maintain dialogue with the user, the robot not only needs to speak, but also to understand what he/she is saying. This task is carried out by the ASR Skill, which transforms the utterances of a person into text.

The ASR skill uses predefined grammars to detect and process the user's speech [Alonso-Martín and Salichs, 2011]. An ASR grammar is a set of words and phrases linked to a semantic meaning. When the user is speaking, the robot tries to match the user's utterances to the sets of the loaded grammar. When a match is produced, the robot returns the semantic meaning of that phrase. For instance, a grammar to understand when the user is saluting the robot could be defined by the phrases: "*Hello!*", "*Good morning, Maggie*", "*Hi Maggie, how are you today?*", *etc.* If the robot detects an utterance similar to these ones, it will interpret that the user is greeting it, so it can act accordingly.

quired by Nuance Communications Ltd. However, Loquendo's TTS and ASR technologies have been supported by Nuance for several years after the acquisition. Now, most of Loquendo's technologies have been incorporated into Nuance's TTS and ASR.

A more formal definition of a grammar is the following: A grammar, G, is defined by the tuple, $G = (U, s)$, where U is a set of utterances and s is the semantic meaning associated with U. The robot can load and unload different grammars in real time, to adapt itself to different HRI contexts.

Note that there are several utterances $U = \{u_1, u_2, \ldots, u_n\}$ that can be associated with a single semantic meaning, s_i. Each utterance, u_i, might be composed of single words, phrases or any combination of them. For instance, the semantic meaning $s_1 = raining$ might be composed of the utterances:

```
u_1 = [*] raining [*]
u_2 = [*] [take the] umbrella [*]
```

Listing 3.1: Example of utterances that provide the same semantic meaning: *raining*

where the asterisk, "$*$", acts as a wildcard to indicate any word or set of words and the text between square brackets "[ ]" indicate that the elements enclosed by them are optional. Therefore, utterances like "*It seems it is raining.*" and "*Take the umbrella before going out.*" trigger s_1.

In the case of pose learning, we have defined two grammars, G_p, which defines the different poses the users can adopt while teaching the robot, and G_c, which defines the interaction commands during the learning process. These grammars are further defined, later, in Section 3.4.1.2

3.4 Learning Architecture

This section describes all the modules that have been built to enable the robot to learn poses from the human by interacting with one. Figure 3.2 depicts the general scheme of the built architecture. The diagram separates the training and the exploitation phases. The upper part represents the training phase, where the user teaches the system to recognize certain poses. The lower part of the figure represents the exploitation phase, in which the robot uses what it has learned to discern in which pose the user is standing.

In the training phase, the robot uses two sensory systems. The first one is its Kinect-based RGB-D vision system. With it, the robot acquires the figure of the user separated from the background and processes it to extract a kinematic model of the user's skeleton using OpenNI's algorithms. The second input is the Automatic Speech Recognition Skill (ASR), which allows the robot to process the words said by the user and converts them into text strings.

The data of these sensors is fused to create a learning instance defined by:

- **The pose of the user**, defined by the configuration of the joints of the kinematic model of the user.

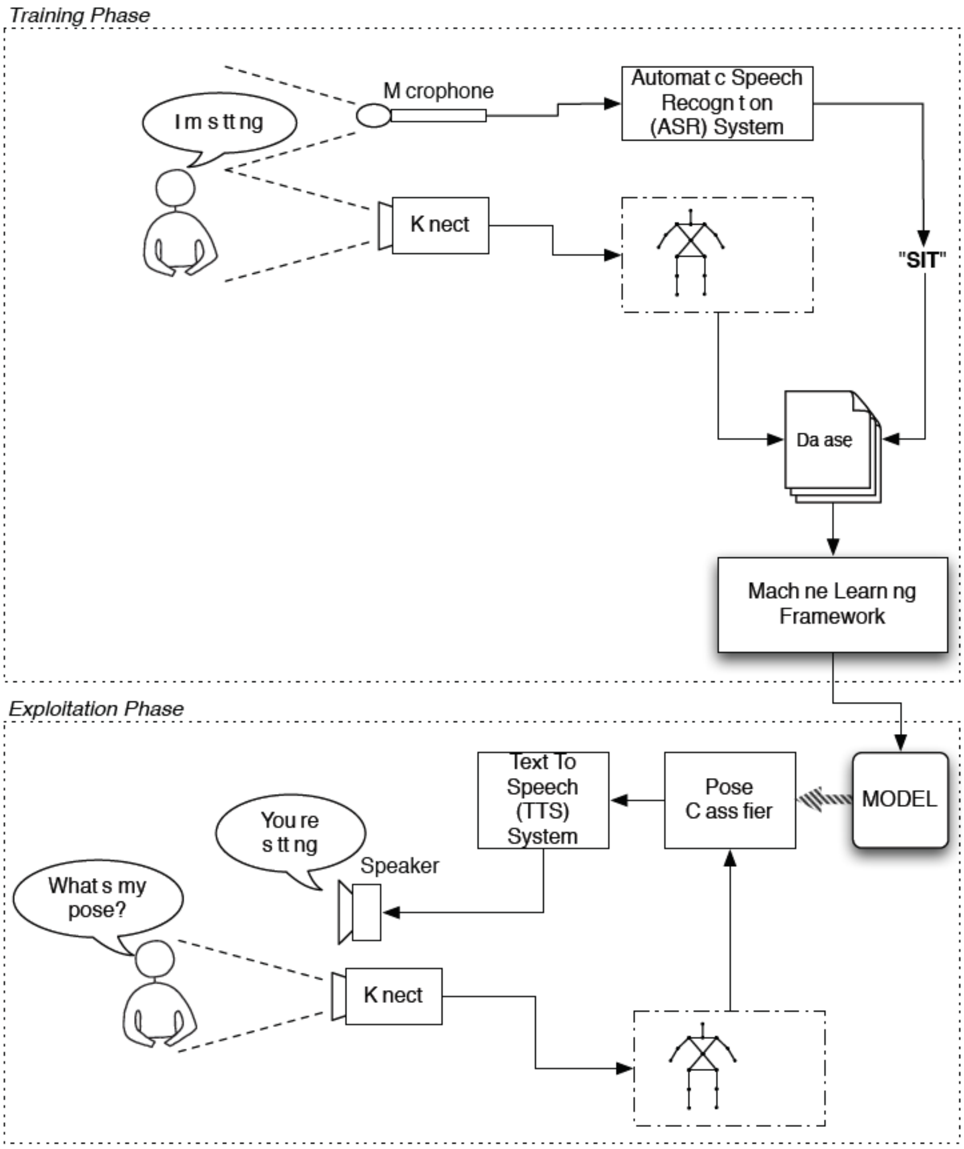


Figure 3.2: Learning System Overview. The upper part of the diagram shows the training phase, where the user teaches the robot, by verbal commands, which are the poses that the robot must learn. The lower part of the diagram depicts the exploitation phase, in which the robot loads the learned model and tells the user's current pose by its voice system.

- **A label identifying this pose**, defined by the text string captured by the auditive system of the robot.

While the user is training the robot, the Kinect keeps acquiring data from the user at 30 Frames Per Second (FPS). These data are labeled according to the label given by the user and stored in a dataset. Once the training session has finished, the dataset is processed by the Weka machine learning framework [Hall et al., 2009], which builds a model establishing the relations between the poses and their associated labels. That is, the learned model establishes the rules that define when a determined pose is associated with a certain label.

In the exploitation phase, the robot continues receiving snapshots of the skeleton model at every frame. However, this time, it does not receive the auditive input telling it what the user's pose is. Instead, the robot has to predict it using what it has learned from the user. For this purpose, it loads the incoming Kinect's data to the learned model, which evaluates it and returns the guessed label corresponding to that pose.

To provide feedback, the robot says the pose in which it believes the user is standing.

This is done by providing the guessed pose label to the ETTS skill of the robot, which is in charge of transforming this label to an utterance that the user can understand.

3.4.1 Data Acquisition and Preparation

This section enters more detail into how the data is captured and processed before it is fed to the learning system. First, it describes the robot's vision system and, later, its auditive system.

3.4.1.1 Processing Visual Data

The Kinect data provides raw depth data in the form of a 3D point cloud. This point cloud has to be processed before it is fed to the learning system. The preprocessing is carried out by an external library, named OpenNI.

OpenNI provides tools to extract the user's body from the background and to build a kinematic model from it. This kinematic model consists of a skeleton, as shown in Figure 3.3. OpenNI's algorithms provide the positions and orientations of these joints at a frame rate of up to 30 FPS (frames per second).

The skeleton model is the model used in our system to feed our pose detection system. In other words, the information that is provided to the learning framework comes from the output of OpenNI's skeleton extraction algorithms.

This model contains the data that is going to be used in our learning system. The data of each skeleton instance (S) is composed of 15 joints represented as:

$$S = (t, u, J) \tag{3.1}$$

where t is the time-stamp of the data frame, u is the user identification (Here, the user identification refers to the user being identified by the openNI framework. It is a value between one and four, and it serves only in the case that more than one user is being tracked by the openNI's skeletonization algorithm.) and J represents the joint set from the user's skeletonized model depicted in Figure 3.3:

$$J = (j_1, j_2, \ldots, j_{15}) \tag{3.2}$$

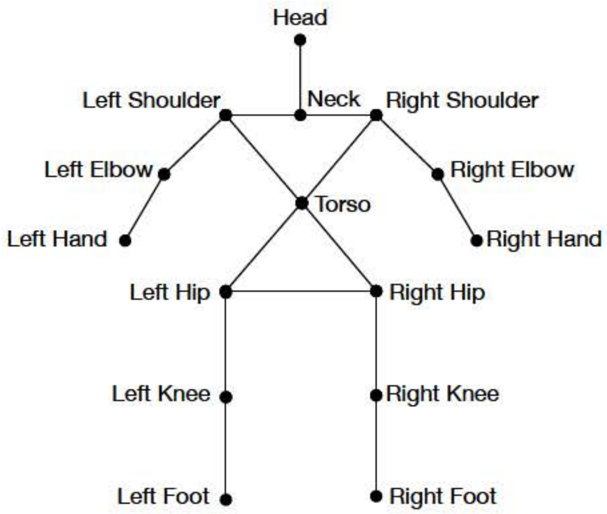


Figure 3.3: OpenNI's kinematic model of the human body. OpenNI[1] algorithms are able to create and track a kinematic model of the human body.

These joints have the following parameters:

$$j_i = (x, y, z, q_x, q_y, q_z, q_w, C) \tag{3.3}$$

where x, y, z represent the position of the joint in $\mathbb{R}^3$, q_x, q_y, q_z, q_w represent the orientation of the joint as a quaternion, and C is the binary confidence of the values of both position and orientation of the joint as provided by openNI. The parameters t, u and C were not used to learn. However, they provide useful control and state information that help to detect and recover from errors or to maintain lively interaction. For instance, if the robot detects that the user has been lost from its line of sight for a long period of time, it can ask for assistance or notify the user that it is not seeing her.

3.4.1.2 Processing Verbal Data

When the user starts training the robot, he/she executes two tasks. First, he/she stands in the pose that she wants to show the robot, and second, he/she tells it the name of that pose. From the robot's point of view, firstly, it has to "see" the teacher's pose, and secondly, it has to understand what she is telling it. Since the user indicates her pose by voice, the robot needs to process these pose names or labels using an auditive module. This module is the ASR Skill, previously described.

As described in Section 3.3.4.2, the ASR skill needs a grammar to process the user speech. In the case of pose learning, the robot needs a grammar to understand what the user says during the training session. Therefore, we have built a grammar, G_p, that enables the robot to detect up to 18 pose definitions by combining different semantics in three different categories (for simplicity, we omit u_i from the grammar):

$$G_p = (s_p, s_a, s_d) \tag{3.4}$$

where s_p, s_a, s_d are three semantic meanings of the user's speech:

1. **Posture Semantics, $\mathbf{s_p}$**: This can take one of the following values:

   $$s_p = \{sit\ ,\ stand\}$$

   where:

   (a) *sit* defines that the user is sitting on a chair.

   (b) *stand.* defines that the user is standing in front of the robot.

2. **Action Semantics, $\mathbf{s_a}$**: This can take one of the following values:

   $$s_a = \{turned\ ,\ looking\ ,\ pointing\}$$

   where:

   (a) *turned* defines that the user is turned (oriented her body) to the direction specified in s_d.

   (b) *looking* defines that the user has oriented her head to the direction specified in s_d.

   (c) *pointing* defines that the user is pointing in the direction specified in s_d.

3. **Direction Semantics, $\mathbf{s_d}$**: This can take one of the following values:

   $$s_d = \{left\ ,\ forward\ ,\ right\}$$

   where:

   (a) *left* defines that the action defined in s_a is carried out to the user's left side. For example, if the trainer is pointing (s_a = *pointing*), she is doing it to her own left.

   (b) *forward* defines that the action defined in s_a is carried out toward the user's front. For instance, if the trainer is pointing (s_a = *pointing*), she is doing it toward her own front.

   (c) *right* defines that the action defined in s_a is carried out to the user's right. For instance, if the trainer is pointing (s_a = *pointing*), she is doing it to her own right.

A complete list of the semantic meanings that G_p can understand is shown in Table 3.1.

$\mathbf{s_p}$	$\mathbf{s_a}$	$\mathbf{s_d}$
Sit	Turned	Left
Stand	Looking	Front
	Pointing	Right

Table 3.1: Semantic values of the pose grammar G_p. In order to be considered a valid sentence of G_p, the sentence must include one semantic value from each column. If the received sentence is valid for G_p, then the sentence is defining a pose. Note that G_p defines the 18 poses ($2 \times 3 \times 3$) that the robot can understand and, therefore, learn.

When speech is detected, the ASR Skill evaluates this speech against G_p. If the evaluation produces a valid result, the ASR tags the speech with a label, $l_p^{(i)} \in G_p$, where i is the i^{th} label. Note that $i \in (1, 18)$, since the cardinality of G_p is defined by all the possible combinations of its semantics. A grammar only is considered valid if the three semantics, s_p, s_a and s_d, have been detected in the speech. Some label examples might be $l_p^{(1)} = (sit, looking, left)$, which indicates that the user is sitting and looking to her left; or $l_p^{(2)} = (stand, turned, forward)$, which indicates that the user is standing and turned toward the robot.

Note that it is possible to arrive at these semantics with different words or phrases. That is, each semantic meaning of G_p has between three and five utterances associated with it, as described in Section 3.3.4.2 For instance, phrases, such as "*I'm sitting looking to the right*" or "*I'm in a chair, looking towards my right*" would produce the same semantic meaning: $l_p^{(1)} = (sit, looking, right)$

The output of the ASR Skill is sent to another skill called the *Pose Labeler Skill.* This skill first processes the results from the ASR to detect if the label told by the user is valid and, then, formats these data properly, so it can be passed to a third skill, the *Pose Trainer Skill*, which is in charge of the learning system itself.

In addition to the grammar, G_p, shown above, the *Pose Labeler Skill* uses an additional control grammar, G_c, which has no direct relation with the poses. This grammar acts as a control layer that allows the user to control some aspects of the training phase. The grammar, G_c, is defined as $G_c = \{s_c\}$, where s_c is its only semantic content defined by only two values: $s_c = \{change\ ,\ stop\}$. These values have the following meanings:

- *change*: used to allow the classifier to discriminate the transitions between two poses. It is used before she changes her pose. Examples of sentences that trigger this value can be: "Let's learn the next pose", or "I'm going to teach you another pose, Maggie".

- *stop*: used to end the training process. When the user says she wants to finish the training process, the ASR builds this semantic to allow the *Pose Labeler Skill* to end. Some examples of sentences that trigger this semantic value are: “So we have finished.” and “Let’s stop for a while.”

As can be seen, the ASR Skill of the robot enables it to understand natural language. This makes the learning session much more natural and pleasant for the user, who can train the robot even without being a robotics expert.

3.4.2 Learning from Gathered Data

Once the visual and verbal data have been preprocessed, they are delivered to the *Pose Trainer* Skill, which is in charge of learning from these data. To train our system, the *Pose Trainer* skill stores each joint set, J, that has been received from the vision system in a training instance, I, and it tags it with a label, $l^i \in G_p$, described in Section 3.4.1.2:

$$I^j = (J^j, l_p^i) \tag{3.5}$$

where $j \in (1, m)$ and m are the number of training examples of the session. Each training instance, I^j, represents a user pose and is defined by a joint set, J^j, and a label, l_p^i. Note that $j \in (1, m)$, while $i \in (1, n)$, which, in the case of G_p, is 18 (see Section 3.4.1.2). In short, this means that the user can show different examples, J^j, of the same pose, l_p^i.

During the training process, the user shows m different training instances to the robot, which are stored in the dataset, D:

$$D = \{I^{(1)}, \dots, I^{(m)}\}. \tag{3.6}$$

The dataset size (the value of m), can vary because the training process continues until the user decides to stop it by telling it to the robot. In that case, the ASR skill will return a label, $l_c = stop$, indicating to the system that the user has stopped the training process.

Our learning system is built on top of the Weka Framework [Hall et al., 2009], a widely used open-source software, which allows us to use several algorithms to build our model. Therefore, with the dataset already completed, the *Pose Trainer* Skill calls the Weka API in order to build a model from the dataset. This model can be represented by the set of associations of joint sets, J, and its corresponding pose labels, l^i:

$$M = \{J \longrightarrow l^i \in G_p\}. \tag{3.7}$$

This model represents the poses that the robot has learned from the user. The quality of the learned poses, *i.e.*, how well they are able to generalize to other situations depends, mainly, on the number and the quality of the examples that the user has provided to the system.

3.4.3 Using What the Robot Has Learned

The model, M, is what the robot loads during the exploitation phase in order to guess the user's pose. To do so, a skill called the *Pose Classifier* Skill loads the learned model and starts to feed it with data coming from the Kinect. For each received joint set, J, M returns the label, l^i, which it believes better corresponds to that pose.

We have created an interactive test that allows us to know how well the robot has learned. In this test, the robot tells which pose the user is standing in when it is commanded to do so. This skill is called the *Pose Teller* Skill. The *Pose Teller*'s main functionality is to receive the estimated label from the *Pose Classifier* Skill, to translate it to a human-readable text string and to send this string to the ETTS Skill, which is in charge of saying this text to the user.

The translation from a label, l^i, to a string that a person can understand is the inverse process that was carried out in the ASR Skill, where the user speech was processed according to a grammar and its semantic meanings extracted. For instance, the label, $l = \{sit, looking, right\}$, may be transformed into a string, like: "*You're sitting looking to your right*".

3.5 Experiments

3.5.1 Scenario Description

To test the system, we prepared a scenario in which 24 students taught the robot 3 different sets of poses:

- First, P_1 consisted of teaching the robot if the trainer was turned to his/her own left, right or if he/she was turned toward the robot.

- Second, P_2 consisted of teaching the robot if the trainer was looking to his/her own left, right or forward.

- Third, P_3 consisted of teaching the robot if the trainer was pointing at his/her own left, right or forward

An example of the poses that were taught to the robot is depicted in Figure 3.4. We used our grammar, G_p, to evaluate if the robot was able to learn interactively by carrying out a natural conversation with the users.

The experiment scenario is depicted in Figure 3.5. The colored rectangle shows the area in which the users had to remain during the training phase. It was drawn in the ground to enable the users to see whether they were inside of it. The motivation for setting a limit to the experimental area was to ensure that the user stayed inside the Kinect's field of view during the training process. However, the users were allowed to move wherever they preferred, as long as they stayed inside the rectangle while recording the poses. Finally, they were told to warn the robot before changing their pose, to prevent the recording of transitional poses.

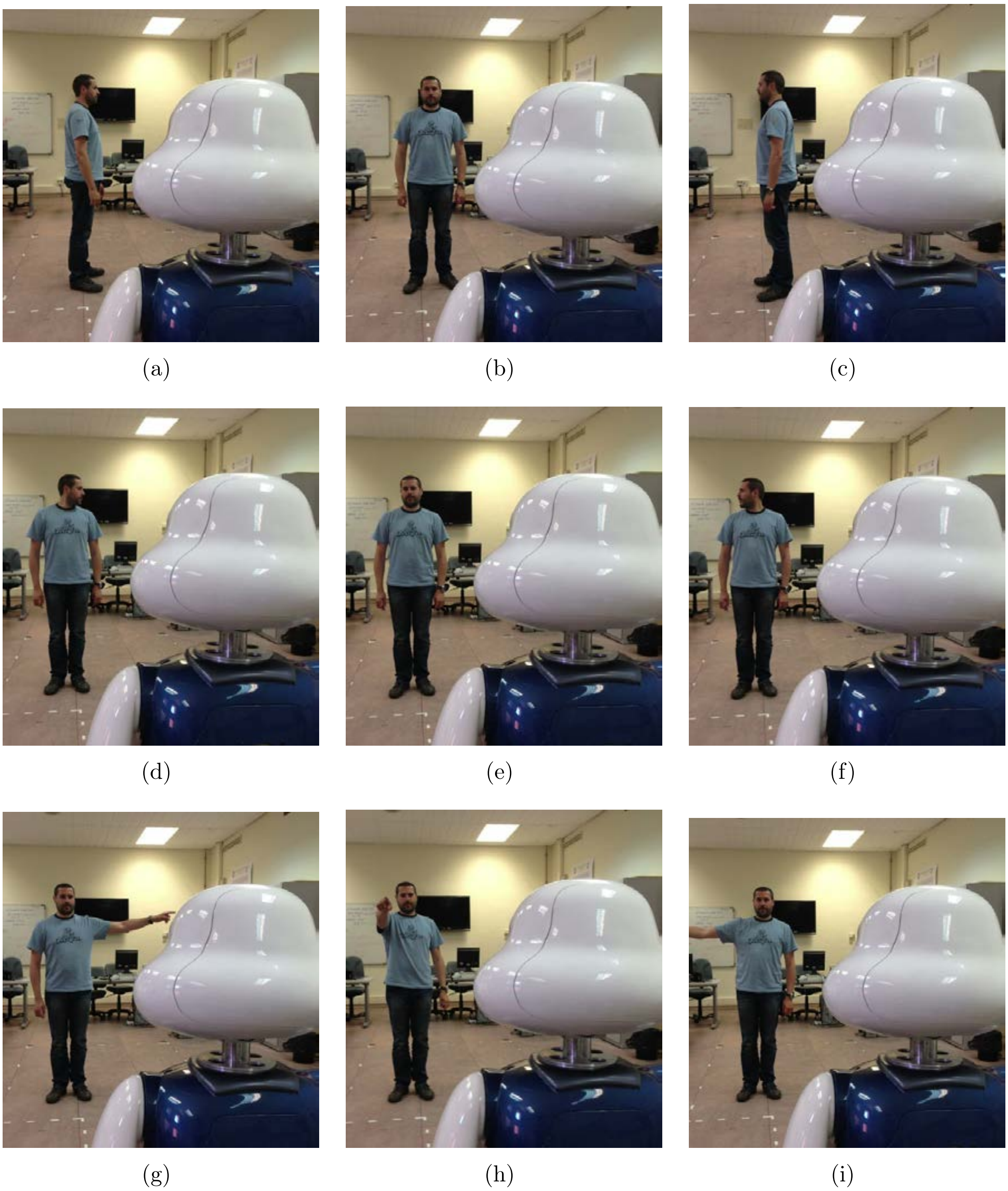
To avoid possible confusion between the user's utterances, we raised the ASR minimum confidence to 50%. That is, the confidence of an utterance belonging to a certain semantic value, s_i, had to be higher than this minimum value. This threshold was enough to avoid misclassification of semantic concepts. In the cases when the user's speech did not obtain this minimum confidence, the robot asked the user to repeat the utterance. However, these situations were occasional, and in the worst case, the user only needed a few attempts until she was able to make the robot understand her.

3.5.2 Method

Before the experiment started, the experimenter explained to the users the experimental procedure. It was indicated to them that they could ask the experimenter any questions related to the poses or the grammar commands whenever they wanted. The user has the initiative during the training process, being able to start, pause and finish the process at any moment. The experimental procedure consisted of the following steps: First, the user stands in front of the robot. Once the robot tells him/her that it is ready to start, she can begin the training when she considers it appropriate. For the recording of each pose, the user was told to, first, stand at a particular pose and, then, tell the robot which label defines this pose. Prior to changing to another pose, the user had to ask the robot to stop recording this pose. Once the robot tells the user that it is ready for recording the next pose, he/she is free to move to the next pose and start the process again. The user finishes the teaching session by issuing the "*stop*" command of the control grammar.

The users were encouraged to start with the left poses, to continue with the front poses and to finish each round with the right poses. The reason for keeping the order is because, despite the order in which the robot learns not being relevant, this helped us to analyze the data after the training. For instance, we found that three users tagged their pose to the left when they were looking (one case) and pointing (two cases) to their right. Since they were told to produce the poses in order from left to right, we assumed that they made a mistake when labeling.

We stored three datasets ($D_{1,2,3}$) per user, each one corresponding to the training of a set of poses, P_i, as described before. Each dataset consisted of a set of training instances,



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 3.4: Examples of poses that the users taught to the robot. (**a**) Turned Left; (**b**) turned forward; (**c**) turned right; (**d**) looking left; (**e**) looking forward; (**f**) looking right; (**g**) pointing left; (**h**) pointing forward; (**i**) pointing right.

as described in Equation (3.5). Although these poses represent a reduced pool of poses, we believe that they are representative of how the robot can learn by interacting with the user. Note that our focus here is not on the learning problem itself, but on the capability of the robot to learn by using natural interaction with the user.

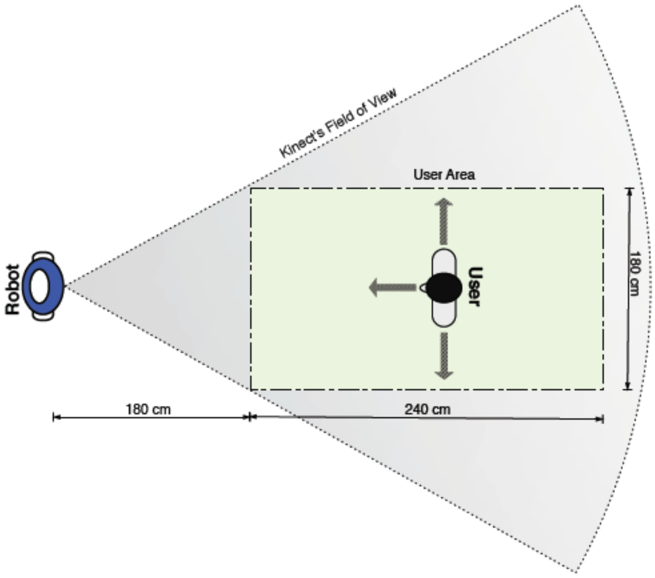


Figure 3.5: Scenario of the experiment — The cone represents the field of view of the Kinect sensor. The user was allowed to move inside the rectangle, as long as she always kept in the same pose while moving.

3.5.3 Results

To train the system, all the datasets corresponding to the same pose P_i, were combined into a bigger dataset, D_i, that was used to create a model of the learned concept. We evaluated our system using a modification of the cross validation (CV) method. Concretely, we used a 10 times twelve-fold CV, where each fold corresponded to all the instances of two users. This evaluation method is a slight variation of the CV evaluation method [Witten et al., 2011].

We performed the evaluation in such a way because we noticed that most of the examples shown by the same user had a strong correlation. On average, each user recorded 179 examples for D_1, 156 for D_2 and 160 for D_3. This led to having several instances of the same pose that were very similar to each other. Because of that correlation, in case we had performed a typical CV rather than our modified one, the evaluation of the system might have led us, erroneously, to conclude that the system generalizes better than it actually does. This effect is caused because, in a normal CV, the dataset is split into several folds of the same size. The problem comes if one of these folds divides the examples provided by the same user, putting some of them in different folds. When one of these folds is used to test the training set, the results will be far better than expected, because we are evaluating our training dataset with similar data to the one we used to train the system. Therefore, to avoid having some instances of the same user in training and testing sets, we decided to force the CV to create the foldings by the number of users instead of splitting the dataset into a fixed percentage of instances.

In summary, we realized that the difficulty of generalization comes from between different users, not among learning instances of the same user. Thus, to evaluate how capable our system is at generalizing what it has learned, we have to perform the evaluation against examples coming from other users who have not trained the system.

We evaluated our system using four different algorithms: J48 [Quinlan, 1993], Naive Bayes [John and Langley, 1995], Random Forests [Breiman, 2001] and SMOs (Sequential Minimum Optimization) [Shevade et al., 2000]. The detailed performance of each algorithm is shown in Table 3.2. Table 3.2a shows the performance of the four algorithms on dataset D_1 (*Turned*), Table 3.2b depicts the performance on dataset D_2 (*Looking*) and Table 3.2c depicts the performance on dataset D_3 (*Pointing*).

The tables show different performance metrics for the four algorithms that we used to test our system: Accuracy (Acc), True Positive (TP) rate, False Positive (FP) rate, precision, recall and F-measure The results are an average of all the CV runs described above. We provide, as well, the standard deviation and the 95% confidence interval (95% CI) for each one of the metrics.

The 95% CI is obtained from:

$$CI_{95} = M \pm 1.96SE \tag{3.8}$$

where M is the mean and SE is the standard error obtained from:

$$SE = \frac{SD}{\sqrt{N}} \tag{3.9}$$

where SD is the standard deviation and N is the number of runs of the cross validation process, which is, in this case $N = 120$.

Figure 3.6 shows the learning curves of the algorithms on datasets D_1 (*Turned*), D_2 (*Looking*) and D_3 (*Pointing*), respectively. The curves represent the average accuracy and the 95% CI of the system when trained with $2, 4, 6, \ldots, 22$ users. The values of the training with 22 users are the same as the ones reflected in Table 3.2. Note that the maximum number of training users is 22 instead of 24, because the CV forces that in each run there will be two users for the evaluation of the training set of that run.

- In dataset D_1 (*turned*, Figure 3.6a), all the algorithms scored nearly 100% accuracy with relatively few training users. The only exception to this is the J48 algorithm, which needed 12 users to reach the performance of the other algorithms.

- In dataset D_2 (*Looking*, Figure 3.6b), all the algorithms showed the worst performance. We believe that this might have been because the orientation of the head is

Dataset D1	J48	Naive Bayes	Random Forests	SMO
Accuracy	0.991 (0.029, 0.003)	0.997 (0.009, 0.001)	0.990 (0.026, 0.002)	0.997 (0.010, 0.001)
TP Rate	0.991 (0.029, 0.003)	0.997 (0.009, 0.001)	0.990 (0.026, 0.002)	0.997 (0.010, 0.001)
FP Rate	0.007 (0.025, 0.002)	0.001 (0.003, 0.000)	0.007 (0.020, 0.002)	0.002 (0.008, 0.001)
Precision	0.993 (0.023, 0.002)	0.998 (0.008, 0.001)	0.992 (0.021, 0.002)	0.997 (0.009, 0.001)
Recall	0.991 (0.029, 0.003)	0.997 (0.009, 0.001)	0.990 (0.026, 0.002)	0.997 (0.010, 0.001)
F-measure	0.991 (0.031, 0.003)	0.997 (0.009, 0.001)	0.990 (0.027, 0.002)	0.997 (0.011, 0.001)

(**a**)

Dataset D2	J48	Naive Bayes	Random Forests	SMO
Accuracy	0.727 (0.151, 0.014)	0.699 (0.173, 0.016)	0.807 (0.138, 0.013)	0.742 (0.171, 0.016)
TP Rate	0.727 (0.151, 0.014)	0.699 (0.173, 0.016)	0.807 (0.138, 0.013)	0.742 (0.171, 0.016)
FP Rate	0.129 (0.075, 0.007)	0.130 (0.072, 0.007)	0.089 (0.061, 0.006)	0.129 (0.081, 0.007)
Precision	0.774 (0.137, 0.013)	0.794 (0.133, 0.012)	0.848 (0.110, 0.010)	0.776 (0.179, 0.016)
Recall	0.727 (0.151, 0.014)	0.699 (0.173, 0.016)	0.807 (0.138, 0.013)	0.742 (0.171, 0.016)
F-measure	0.712 (0.158, 0.014)	0.688 (0.184, 0.017)	0.801 (0.143, 0.013)	0.723 (0.190, 0.017)

(**b**)

Dataset D3	J48	Naive Bayes	Random Forests	SMO
Accuracy	0.795 (0.193, 0.018)	0.690 (0.198, 0.018)	0.829 (0.153, 0.014)	0.903 (0.131, 0.012)
TP Rate	0.795 (0.193, 0.018)	0.690 (0.198, 0.018)	0.829 (0.153, 0.014)	0.903 (0.131, 0.012)
FP Rate	0.093 (0.087, 0.008)	0.141 (0.081, 0.007)	0.084 (0.077, 0.007)	0.054 (0.075, 0.007)
Precision	0.863 (0.152, 0.014)	0.714 (0.234, 0.021)	0.878 (0.114, 0.010)	0.922 (0.116, 0.011)
Recall	0.795 (0.193, 0.018)	0.690 (0.198, 0.018)	0.829 (0.153, 0.014)	0.903 (0.131, 0.012)
F-measure	0.785 (0.205, 0.019)	0.656 (0.230, 0.021)	0.823 (0.161, 0.015)	0.899 (0.142, 0.013)

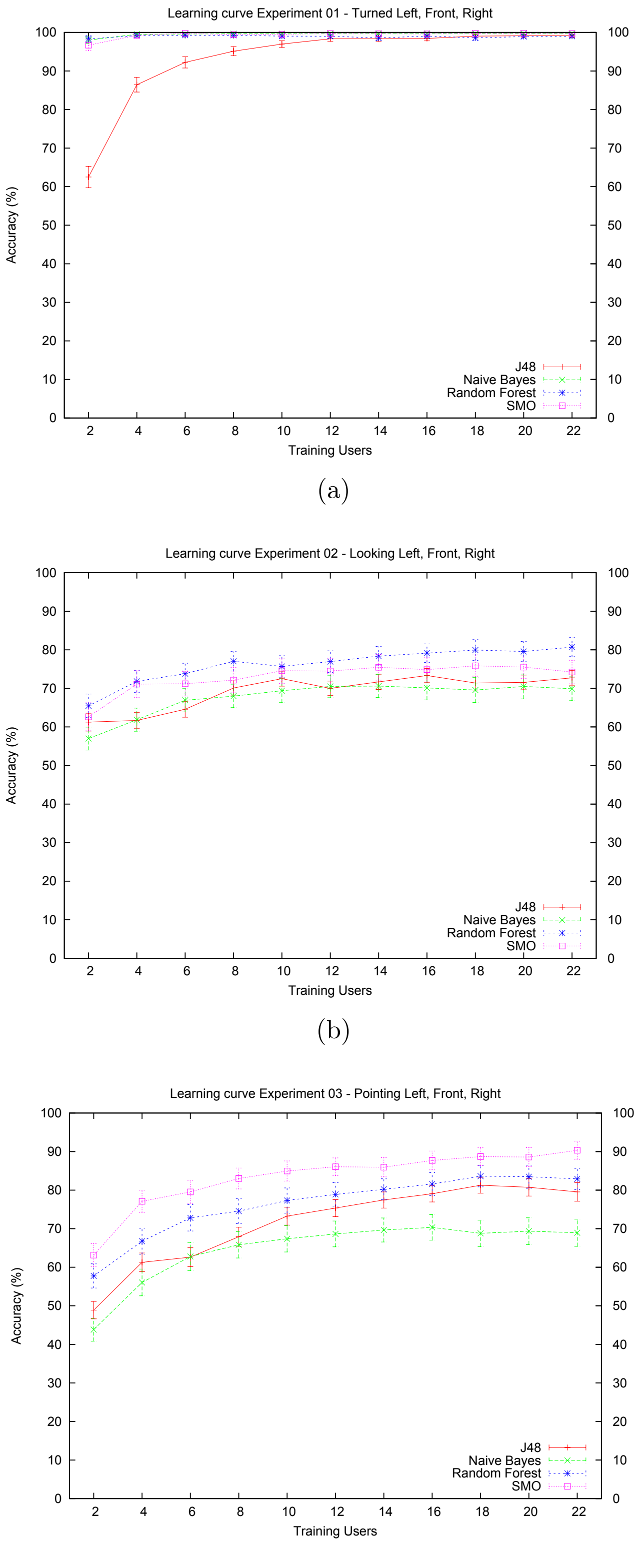
(**c**)

Table 3.2: Learning performance for the three datasets. Results are: *Mean* (*Std.Dev.*, CI_{95}) Note: TP = True positive; FP = False Positive. (**a**) Learning performance for dataset D_1 (*turned: left, forward, right*); (**b**) learning performance for dataset D_2 (*looking: left, forward, right*); (**c**) learning performance for dataset D_3 (*pointing: left, forward, right*). SMO, Sequential Minimum Optimization. CI, confidence interval.

important for discerning where the user is looking. Unfortunately, the OpenNI algorithms provide less precise data for the head than for other joints; therefore, since the learning system had to deal with inaccurate data, its performance decreased.

There is no clear difference between algorithms if they are trained with few examples. With only examples from one user, their performance ranges from nearly 60% by Naive Bayes to 65% by Random Forests. When the number of users is increased to 12, Random Forests performs better than J48 and Naive Bayes. Finally, when we have 22 users for training, Random Forests stands above the rest, achieving an 80% accuracy (see Table 3.2b).

In this dataset, all the algorithms slightly increase their performance when adding more users to training. Although this increment is not significant from 12 users



(a)

(b)

(c)

Figure 3.6: Learning curves for the three datasets. (**a**) Dataset D_1 (turned left, front, right); (**b**) dataset D_2 (looking left, front, right); (**c**) dataset D_3 (pointing left, front, right).

thereafter, we hypothesize that adding more users to D_2 might help to increase the overall performance of the system.

- Dataset D_3 (*Pointing*, Figure 3.6c) is where we found the biggest differences among algorithms. Here, Naive Bayes showed the lowest performance. With only one training user, it barely reached a 45% accuracy whereas SMO nearly scored 65%. With 22 users, Naive Bayes showed just a 69% accuracy. In this case, SMO was the algorithm that showed the best performance for this dataset, with nearly 86% accuracy with 12 training users and roughly 90% with 22 users.

  Similarly to the other datasets, none of the algorithms showed a significant improvement when increasing the number of training users from 12 to 22. However, like in dataset D_2, we believe that adding more users might increase the performance, especially in the cases of the J48, the Random Forests and the SMO algorithms.

3.5.4 Discussion

The presented results show that it is possible to learn poses from examples provided by the users in an interactive way. Nevertheless, Figure 3.6 indicates that the examples provided by one single user are not enough to generalize the learned concepts to other users. On the other hand, the system needed only 12 users to achieve good classifying results.

In general terms, during the training, we observed a great variability between the poses that each user taught the robot in datasets D_2 (*looking*) and D_3 (*pointing*). That is, when the robot was learning a pose, the examples shown by each user differed considerably. This effect was especially relevant in dataset D_3 (*pointing*), where some users used their right hand to point, while others, their left hand. Even more, in some cases, some users used their right hand to point to their right and their front, but changed to the left hand when pointing to their left (see Figure 3.7). In fact, we also observed some cases in which the users looked to the direction where they were pointing, while others looked to the robot instead.

In the case of dataset D_2 (*Looking*), we observed less differences, but still significant ones. Here, some users exemplified the looking poses by only turning their heads to the left or right, while others slightly turned their torso and waist, as well.

These differences between users is what may produce lower results when the robot is trained only with a few users. As it gathers examples from more users, it discovers new ways of how each pose is executed and, therefore, improves its classification accuracy.

We consider this variability one of the key justifications of our natural learning system, since it enables the robot to learn by directly asking the user, who does not need to be an expert in robotics to teach it.

A possible way to ameliorate to this issue is to obtain examples from many users. However, in some cases, it might be difficult or costly to acquire more data. For instance,

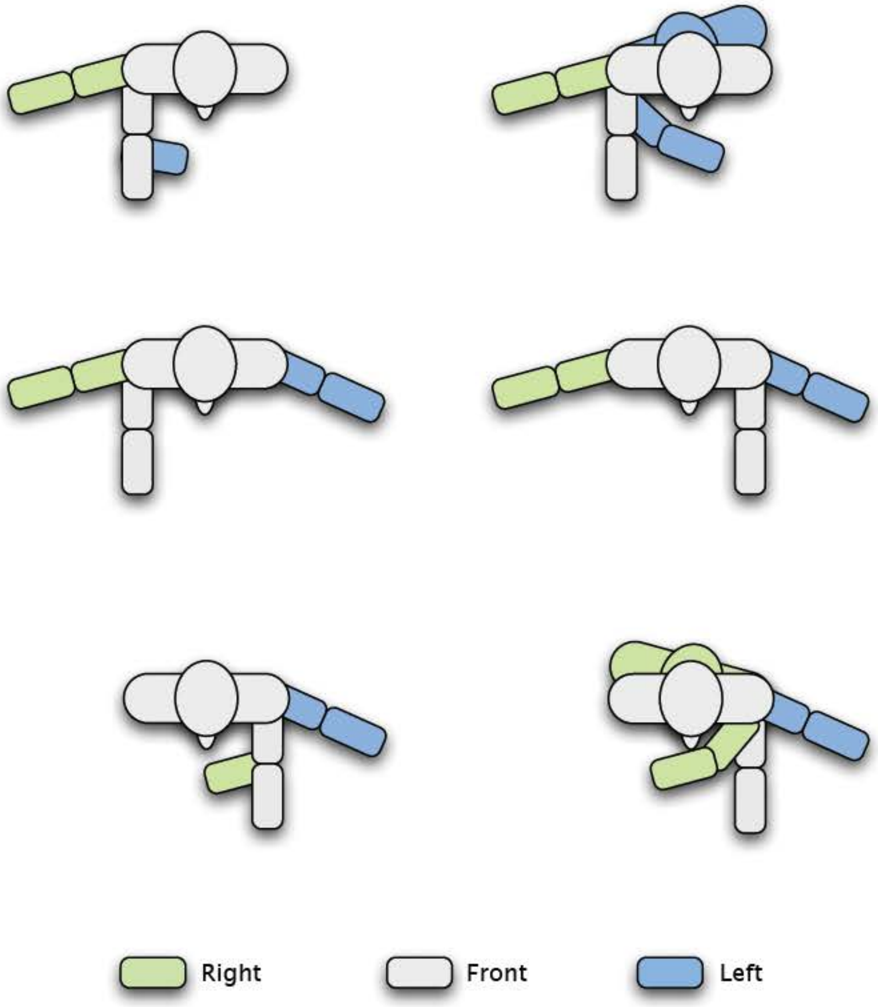


Figure 3.7: Examples of how different users pointed during the training for D_3. The figure shows six persons pointing to the left (blue), front (grey) and right (green) seen from above. Note that some people uses the same hand for the three pointing gestures, e.g people in top row, while others use different hands depending on the direction of the pointing pose, e.g. people in the middle row.

the user might not have the time to train the robot. For that reason, it might be interesting to improve the generalization capabilities of the learning system. In such a way, active learning techniques, where the robot actively asks the user for information regarding the poses, might accelerate the robot's learning speed.

To conclude the discussion, we must stress that forcing the users to stay inside a fixed area during the training phase might reduce the naturalness and realism of the scenario. The purpose of limiting the training area was ensuring that the user joints were inside the field of view of the Kinect during the whole training process. However, as a consequence, most users tended to place themselves near the geometric centre of the area and barely moved from there. We are currently considering the possibility of a more natural environment to allow scenes where the user might be located in some positions in which the robot might not see some of the joints of the user. On the other hand, we plan to allow the robot to track the user by moving itself, so it can adapt to the changing conditions of the scene, such as the user standing closer to or further from the robot, *etc.*

3.6 Conclusions

This chapter presented a system to endow a social robot with the capacity to learn interactively by maintaining a natural conversation with its human teacher. The natural interaction is achieved using a grammar-based ASR, whose aim is to recognize different sentences and to extract their semantic meaning. Using the semantics as labels of the concept being learned, the robot is able to understand users that are not robotic experts.

Our system has been tested in the application of pose recognition, in which the robot learns the poses adopted by the teacher, listening his/her explanations. Our experiment consisted of 24 non-robotics experts training the robot nine different poses in three training exercises. We evaluated our system by comparing four learning algorithms, achieving satisfactory results in all of them for the three exercises.

A robot with interactive learning capabilities can adapt rapidly to different situations, since the user can train it *ad hoc* for that situation. Moreover, since the robot is capable of establishing natural interactions, the teacher does not need any expertise in robotics.

Despite the promising results, our system still presents a major limitation. The maximum number of poses it can learn is limited by the number of semantics coded into the ASR's grammar. Moreover, these grammars are pre-written in a text file by the robot programmer. However, we already started working on an extension to our system, targeted at solving this limitation. This extension consists of combining a grammar-based ASR with statistical language models. Combined, the user will be able to add new semantics to the grammar that will be used to label the learned concept, as well.

Additionally, our work leaves other paths open for exploration. Firstly, from the HRI point of view, this chapter has focused on the HRI from the robot's point of view. It remains to study how users perceive what the robot has learned and how this fact changes their relation and their expectations towards it. Even more, understanding what the user thinks about the learning process might lead to better training scenarios that would end in robots that learn better from the users. Secondly, this work opens the door for building a continuous learning framework, where the robot actively seeks for new examples and asks questions of its teacher about the concepts being learned. Thirdly, with only a few minor modifications, this learning system can be extended to other applications, such as gesture learning, activity learning or the interactive learning of new objects.

Chapter 4

In Hand Object Detection and Tracking

This chapter explores another approach to robot learning. It focuses more in the vision rather than interaction aspects—although we will extend the interaction aspects of this system in Chapter 6. This chapter describes a vision system that learns to recognize household objects being held by a user. The system works using RGB (Red, Green, Blue) images as well as depth information in several stages of the vision pipeline: Region of Interest (ROI) finding, feature extraction, etc. We also use a skeleton model of the user to track its hand and define a ROI around it, thus reducing the computational requirements in later steps. RGB and Depth data are finally combined in the exploitation phase to produce a prediction of the being held by the user. The system operates in real time and, with few training examples, it achieves an F1 score of nearly 80% in a pool of 6 household objects. Our experiments also demonstrate that, when combining RGB and depth data the recognition accuracy improves compared to the predictions by separate.

4.1 Introduction

This chapter, together with chapter 6 presents a system that is able to learn objects using Active Learning. The focus of this chapter will be on the perception modules that enable the system to learn, while chapter 6 focuses on the interaction and the active learning components of the system.

Being able to recognize objects can help robots to understand better their context and adapt their behaviour accordingly. For instance, imagine that the robot needs to interact with a person who is reading a book. The robot can associate the book in the user's hand with the reading activity and, thus, infer that this person might not want to be disturbed unless is necessary.

For this purpose we have developed a system that is able to learn and recognize hand-held objects. This system is capable of recognizing the user's skeleton and, from it, extract the hands position. With this information the system studies the area around the hands and compares it with a previously acquired dataset.

There are various examples in the literature that have already studied the importance of the objects in the context of action recognition. For example, in [Delaitre et al., 2011], a study of the human-object interaction in still images was performed in order to relate those objects to actions. This relation between object and action may also be seen in [Fathi et al., 2012], in which wearable cameras are used as sensory input to recognize hand-held objects that later are associated to the action to be learned.

4.1.1 Proposed solution for in-hand object recognition

Our system has two differentiated modes, the learning or data acquisition mode and the recognition or system exploitation modes. In the first mode, the system learns a model that enables it to recognize different objects. In the second mode, the system exploits this model to estimate which object is being presented to it. The developed system is able to detect that the user wants to change the mode by analysing his/her pose. In order to trigger the learning mode, the user only needs to extend his/her arm towards the sensor, showing the item to it (Figure 4.1, left). This is a natural gesture usually performed between humans when introducing new objects to one another. Having the hand closer to the body the recognition mode is triggered and the program outputs a unique *id* for the learned object (Figure 4.1, right).

Apart from switching modes evaluating the user's pose, the system also detects the hand that is holding an object. Our system is able to work with one hand at a time by choosing the one that is closer to the robot. The system has been designed to be as modular and reusable as possible enabling the development of complementary software such as handbags recognition or hats recognition with slight modifications.

LEARNING
VIEW1
RECOGNIZING

Figure 4.1: OCULAR working modes: learning (left) and recognizing (right). The learning mode is triggered when the user stretches his/her arm towards the robot. The recognizing mode is triggered when the user pulls his/her arm towards his/her chest. This is the default mode.

4.1.2 Objectives

The objectives of this chapter are listed below.

- To develop a system capable of learning objects in real time. This includes the storage of the dataset for further usages. The learning process is performed acquiring a definable number of views per object.
- To develop a system capable of recognizing objects in real time. Recognition means the detection of new objects and the comparison with a previously obtained dataset outputting the ID number of the most similar template.
- The system must be able to detect the person that is in front of the robot.
- The software must detect the location of the hands of the user in order to extract from there the object to be recognized and learned.
- The system must be able to learn more than one view per object.

4.1.3 Structure of the Chapter

The chapter continues with a brief overview of the computer vision in section 4.2. Section 4.3, details similar work to ours. Section 4.4 describes the proposed solution, where the system is presented as a whole and each individual component of our approach is detailed. It is followed by section 4.5, which describes the experiments that have been carried out to validate the system. These results are shown in section 4.6, and discussed in section 4.7. Finally, the chapter concludes with some conclusions and remarks for further work in section 4.8.

4.2 Computer Vision fundamentals

This section introduces the main computer vision vocabulary and methods related with object learning and recognition. A typical computer vision system consists in a pipeline composed of several steps in which each one is in charge of one single data processing functionality. These steps are data acquisition, raw data input processing, segmentation, feature extraction (also called object description), and, image classification.

1. **Data acquisition: Hardware**
   In this chapter, we use a Kinect to acquire the RGB-D (Red, Green, Blue and Depth) images. This sensor was already described in section 3.3.2.

2. **Raw input data processing**
   Usually, the acquired information needs some preprocessing. This step transforms the Kinect's raw data to data structures that are manageable by the computer vision libraries that operate in the next steps. Section 4.2.1 describes this process.

3. **Segmentation**
   This step crops the input image to a Region of Interest (ROI) in which the object to be detected is expected to be located. Section 4.2.2 further explains this concept.

4. **Object description methods**
   In order to apply recognition and matching algorithms, it is necessary to extract certain relevant and unique information of each object that differentiates it from other objects. This process is done in this feature extraction step of the pipeline. Section 4.2.3 further describes this process.

5. **Image classification**
   The final step is to apply some classifier to learn and predict which is the object that corresponds to the analysed data. This part can be considered similar to other machine learning problems: a classifier is trained with some data (the descriptors extracted in the previous step) and builds some model that will be used in the exploitation phase.

4.2.1 Raw data preprocessing

The Kinect's raw data is passed to three libraries: OpenCV, PCL (Point Cloud Library) and OpenNI/NITE. The aim of OpenCV and PCL is to detect and track the object, while the objective of OpenNI/NITE is to detect the hand that holds it. The main difference between OpenCV and PCL is that PCL is able to process the depth information from the Kinect. In essence, OpenCV main data type is a 2D matrix (a flat image of pixels), while a PCL main's is a 3D matrix called *point cloud* (a cloud of points: x, y, z). For the rest of the document, we will use, indistinguishably 2D or RGB object recognition to the former and 3D or Point Cloud (PC) object recognition to the latter.

4.2.2 Segmentation

The segmentation consists in the extraction of the Region Of Interest (ROI), that is, separating the interesting parts of the image from the background of the input data. The purpose of segmentation is to simplify the image to make later steps in the analysis process easier [Shapiro and Stockman, 2001].

Different segmentation techniques exist depending on the application. For example, in the application of detecting objects on a table, the typical used method is to first locate the flat surface representing that table and crop the image around it. In our object recognition approach, the object itself is hand-held by the user, so our approach is to locate it and crop a square around his/her hand.

4.2.3 Object description methods

In order to be able to compare two objects, unique features or descriptors should be extracted from them. The definition of feature changes depending on the computer vision application. In the case of this chapter, we define a feature or descriptor as "*an interesting or important characteristic, point or region of an image*".

The best feature or descriptor is the one that possess a higher repeatability, i.e the ability of obtaining the same output given different inputs. Therefore, repeatability can be described as "*the ability of obtaining the same predicted object given different views of it*". Recognition algorithms depend directly on the repeatability of the features they use since in the recognition phase, they need to compare the extracted features against the learnt ones, If these features do not match for the same object, the recognition might produce misleading results.

4.2.3.1 2D Feature Extraction Algorithms

This section presents some of the most relevant algorithms for general object recognition. They are listed in chronological order.

Scale Invariant Feature Transform (SIFT)

SIFT (Scale Invariant Feature Transform) is a scale and rotation invariant feature descriptor [Lowe, 2004]. It has been widely used in computer vision due its capabilities to extract features independently from scale and rotation, which makes SIFT robust to noise and to changes in illumination [Mikolajczyk and Schmid, 2005].

The algorithm is based on four main phases: First, it finds the scale-space extrema, which is the location of potentially interesting points that are invariant to scale and rotation. Second, it selects the more relevant of these points by measuring their stability. After, SIFT assigns one or more orientations to each point using local image gradient directions. Finally, it creates the descriptors of these points by transforming the local image gradients into a representation that is enough descriptive and robust to allow various levels of shape distortion and changes in illumination.

However, the main drawback of SIFT is its high computing requirements during the creation of his descriptor vector, which is of considerably big size. The fact of using a highly distinctive descriptor, produces a slower detection, description and matching processes.

Speeded Up Robust Features (SURF)

SURF (Speeded Up Robust Features) is a scale and rotation invariant interest point or keypoint detector and descriptor [Bay et al., 2006]. SURF simplifies the detection, extraction of the descriptors and matching steps making it faster than SIFT, but keeping similar levels of repeatability, distinctiveness and robustness.

The algorithm operates by firstly identifying interesting points in the image such as corners, blobs or T-junctions[1], and then representing these keypoints together with their neighbourhood as a feature vector. In the recognition phase, these vectors are ususally compared using distance-based techniques. SURF aims to reduce that size without losing distinctiveness in the features.

Oriented FAST and Rotated BRIEF (ORB)

ORB (Oriented FAST and Rotated BRIEF) is a fast rotation invariant, noise resistant binary descriptor based on BRIEF [Rublee et al., 2011]. According to the authors, ORB it is up to two orders of magnitude faster than SIFT while matching its performance in many situations. However, ORB is not scale invariant. This makes ORB perform slightly worse than SIFT in situations where are noticeable scale differences in the images.

ORB is based in two other algorithms for detecting and describing the keypoints of an image. For finding the most interesting keypoints it uses the Accelerated Segment Test (FAST) [Rosten and Drummond, 2006] keypoint detector. Then, uses FAST's output to build a descriptor vector using the Binary Robust Independent Elementary Features (BRIEF) [Calonder et al., 2010] descriptor. Both FAST and BRIEF offer good performance in low computing times.

FAST is mainly used to find keypoints in real-time systems that match visual features. The orientation operator included in this algorithm is described in [Rosin, 1999]. This

[1] A T-junction is a junction were two lines meet forming a T

technique is not computationally demanding and also, unlike SIFT, it returns a single dominant result.

BRIEF uses simple binary tests with a performance that is similar to SIFT's in robustness to lighting, blur and perspective distortion. However, it is sensitive to in-plane rotation. In order to eliminate this drawback, the lowest computing costing solution is to steer BRIEF accordingly with the orientation of the keypoints.

According to ORB authors, ORB's inliers percentage is higher and do not variate as much as SIFT's or SURF's. This makes ORB a good alternative for the latter if the application does not need a scale invariant descriptor. Another advantage is that ORB has an Open Source implementation that, unlike SIFT and SURF, is patent-free. Despite both SIFT and SURF licenses allow their use for research, they force a payment to whom want to use them with commercial purposes. This last fact, has made us decide to use ORB in our system.

4.2.3.2 3D Feature Extraction Algorithms

Applied to 3D, a feature or descriptor is a characteristic that describes a point in the space. Features can be compared to determine whether the point described is the same in two different inputs. Depending on the application the developer must select the features depending on the specifications. Examples of geometric point features are the underlying surface's estimated curvature or the surface's normal at a specific query point. Both features are local and they describe the point by providing information of its surrounding neighbours.

Like in 2D images, 3D objects can be described using local or global features. Local features are usually less time-expensive but at the cost of lesser robustness. For instance, two objects may obtain very similar local features even when these objects are different. Global descriptors generalize the information obtained for keypoints in the mesh. In certain conditions they can be more robust than local descriptors, but at the cost of being more time expensive.

Many 3D feature extraction algorithms exist, each one with different approximations to extract the geometric characteristics of the input point clouds. As an example, the 3D SIFT descriptor [Scovanner et al., 2007] performs a 3D gradient and magnitude for each pixel, directly derived from its computation in 2D. 3D SIFT is rotation and scale invariant but is very time-consuming.

Our system's 3D descriptor is the Point Feature Histogram (PFH) due its good performance and relatively low computing requirements.

Point Feature Histogram (PFH)

Point Feature Histogram [Rusu et al., 2008] is a local 3D descriptor that computes the descriptors by approximating the geometry of a point's K-neighborhood with a few values. This fact results in the possibility of obtaining a similar set of points in a very different object.

PFH descriptors are invariant to rotation, position, and point cloud density. Besides, they also perform well with noisy data inputs. It creates the features by representing the mean curvature around the query point using a histogram of values.

Fast Point Feature Histogram (FPFH)

There is also a faster version than PFH, the Fast Point Feature Histogram (FPFH) which is based in the former [Rusu et al.]. FPFH is faster because it only takes into account the direct relations between the query point and its neighbours. However, FPFH is less robust than the previous descriptors. In our approach we preferred to use PFH over FPFH since our the former it already meets our real time requirements.

4.3 Related Work

This chapter is devoted to describe some of the most relevant algorithms and techniques involved in the process of learning and detecting objects. This brief literature review focuses in two of the main aspects of our system: first in object learning using 2D and 3D input data and, second, in-hand object tracking and recognition.

4.3.1 Object learning and recognition using 2D and 3D input data

One of the first works in the field was performed by Gavrila and Groen who developed a system that performs 3D object recognition using 2D input data [D.M.Gavrila and Groen, 1991]. They created a 3D model and which was matched with an input 2D projection using a hashing method. The system was tested under a controlled environment and using textured objects, however, its performance decreased when it was tested in a real environment with noise and loosely defined objects. Also, the system had a high computing cost due of the use of the hashing algorithm, which limited the size of the dataset.

Later systems [Sheta et al., 2012] explore different descriptors that are less time-consuming. In this case, Sheta and colleagues used fuzzy logic to match the learned features to the new ones being observed. Those descriptors did include Affine [Reiss, 1991], Zemike [Teague, 1980], and Hu [Hu, 1962] moments invariants among others. This approach lead to a high percentage of true positives, But, again, the system was tested in a very controlled set up: The input was collected from three cameras with known illumination and orientation. The learned objects are white with significantly different shapes and they rest in a black background during the learning and recognition phases. Hence, the applications of this system are related to the industrial field.

In another approach, Zia [Zia et al., 2013] demonstrated the effectiveness of two methods when combined: local descriptors and 3D wireframes. The system was able to distinguish between cars and bicycles and to estimate their pose. Nevertheless, the training was made off-line, using a high number of Computer-Aided Design (CAD) models views.

These systems differ from our approach in one major aspect: Both the learning and recognizing processes in our system are performed easily and on-line. There is no setting needed nor no previous processing. The combination of 2D and 3D information in our system is, to the best of our knowledge, novel. In our system we first define a ROI around the user's hand –which is located using a skeletonized model of the user provided by the OpenNI library–, and then we combine 2D and 3D descriptors to estimate the user being held in the user hand.

4.3.2 In-hand object learning and recognition

Most of the literature in in-hand object learning and recognition has been developed using wearable cameras as the input of the system. One of the most representative works is [Roth et al., 2006], which uses this approach to capture daily objects to construct a dataset more easily. Its main feature is that it eliminates the necessity of manually segmenting and labelling the learning datasets for object recognition libraries. However, the system only learns new templates that are later fed to an off-line learning algorithm.

Another example is [Philipose, 2009], where the authors perform a benchmarking of an egocentric object recognition system. The input of his approach is an image in which the target object is already centered. The background is not segmented in this system because the errors produced by it can be neglected. The target object occupies most of the image frame and hence the processing of the input data does not need to be as exhaustive as our approach.

The system presented in this chapter proposes a different approach: in our in-hand object recognition and learning, the user locates himself/herself in front of the camera and we find ROI around the user's hand using a skeleton model of the user. Besides, the switching between learning and recognizing modes is controlled by the user in a natural manner, taking use, again, of the skeleton model of the user.

4.4 System Description

This section presents the object recognition system that we have developed. Our approach implements a software capable of learning and recognizing hand-held objects. We have built our system in a modular manner where each step of the computer vision pipeline is carried out in a different process. Each of these processes are implemented as a Robot Operating System (ROS) [Quigley et al., 2009] node so the information between these processes is exchanged using the regular ROS mechanisms. All the computer vision algorithms used by our system are from the OpenCV Library [Bradski, 2000] for pixel-based image processing and from the Point Cloud Library (PCL) [Rusu and Cousins, 2011] for point-cloud based analysis.

The input of our system comes from the Kinect RGB-D sensor, which produces two data sources: a 2D raw image from its RGB camera; and a 3D information in form of a raw point cloud from the depth sensor. The former data source is passed to the OpenCV library while the latter is processed with the PCL Library. Our system also receives a third data source. This is a kinematic model of the skeleton as presented in chapter 3.4.1.1. This third input is used by our system to find the user's hand to crop a ROI section around it.

We also track the user's hand to decide whether the user is showing the object to the robot or not (see Figure 4.1). We use this predicate as a gestural interface that tells the robot when the user wants to teach the robot a new object (hand extended towards the robot indicates that the robot must learn the object) and when the learning should finish (when the hand is not extended, but it still holding the object, indicates that the learning should stop).

Figure 4.2 depicts a simplified view of our system's flowchart. The system has two main operating modes: the learning phase and the exploitation phase. In the next sections each of those modes are detailed.

4.4.1 Common Steps for both modes

There are two steps in the image pipeline that are common to both the learning and exploitation modes. These steps are selecting the Region of Interest (ROI) around the user's hand, both in 3D point cloud and in the 2D image; and the extraction of the features from the point cloud and the RGB image.

The ROI selection steps aim to reduce the image and point cloud sizes so latter steps would require less computation requirements and time. Note that in Figure 4.3 the 3D ROI information output is passed to the 2D ROI extractor. That is because the hand location is given in 3D coordinates. The 3D ROI is created by creating a prism around the hand which is later passed to the 2D ROI, which mission is to convert the 3D coordinates to pixel coordinates to create a square ROI.

Once the ROI is selected, the next step is to create the descriptors (features) that uniquely define the segmented image and point cloud. As it was described in section 4.2.3, the system uses PFH [Rusu et al., 2008] for extracting the 3D features from the

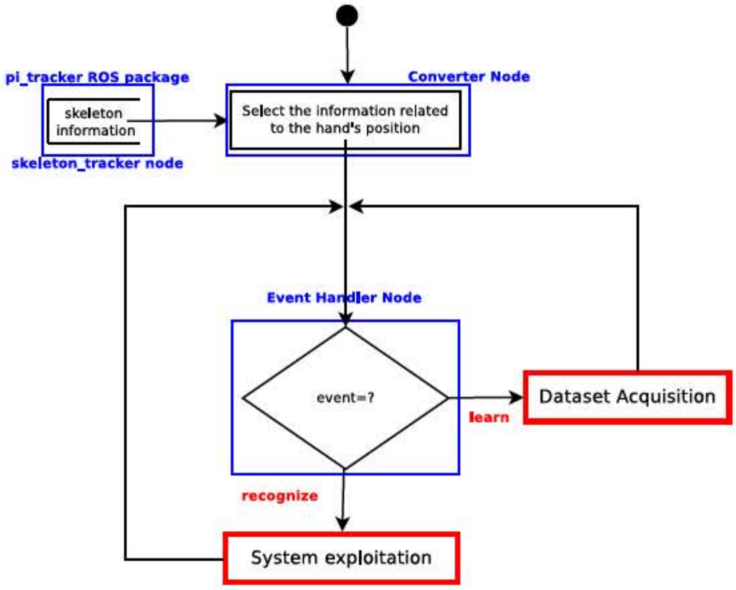


Figure 4.2: Simplified system's flowchart. Node names are in blue. The figure shows the two main operation modes: the learning phase (Dataset Acquisition) and the exploitation phase (System exploitation). The decision of switching between modes is carried out in the *Event Handler Node*, which tracks the user's hand position with respect to his/her own body.

point cloud and ORB [Rublee et al., 2011] for creating the descriptors of the RGB image (2D features). These features are stored in the form of matrices in separate files per object. Figure 4.4 depicts an example of the features extracted from an image using the ORB feature extractor.

4.4.2 Learning mode

The learning mode is the mode where the system creates a dataset of visual examples that will be used for matching in the exploitation phase. The dataset is created using the features from section 4.4.1. Figure 4.5 shows the process followed in the segmentation.

During the learning mode, the system captures a number of different views of an ob-

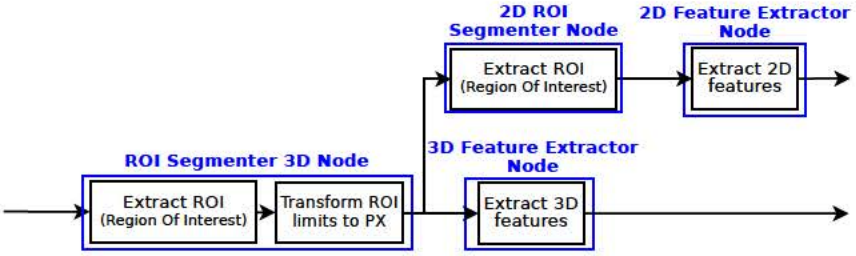

2D ROI Segmenter Node
Extract ROI (Region Of Interest)
2D Feature Extractor Node
Extract 2D features
ROI Segmenter 3D Node
Extract ROI (Region Of Interest)
Transform ROI limits to PX
3D Feature Extractor Node
Extract 3D features

Figure 4.3: Flowchart for ROI and Feature Extraction steps. The outputs in this stage are the 2D features from the RGB image (upper part of the output), and the 3D features from the Point Cloud (lower part of the output)

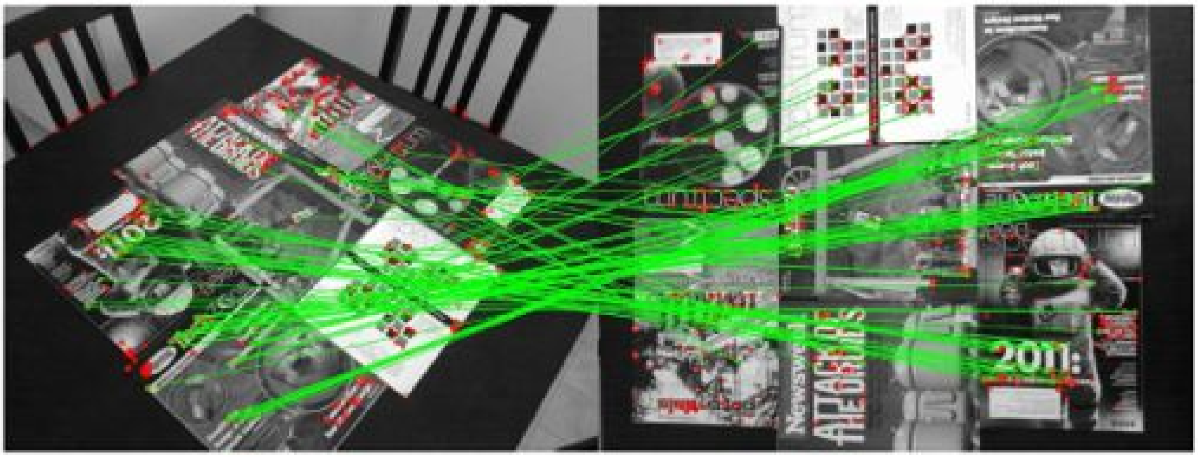


Figure 4.4: Example of the features produced by the ORB algorithm in grayscale image. The figure shows 2 different views of the same object (a table cluttered by objects) and the descriptors produced by ORB (red circles). The matching between descriptors is depicted in green lines between both views. Image retrieved from [Rublee et al., 2011]. ©2011 IEEE.

ject. In each view, the system extracts the features for the view and stores them so they can be matched during the exploitation phase. The system learns better when it is fed with lots of quality data. That means that adding more views during learning, may increase, potentially, its detection performance during the exploitation phase. Additionally, showing different views of the same object increases the robustness to rotation and pose misclassification.

To increase the quality of the views the learning system stops acquiring features during a second between views, giving the time to the user to rotate the object. This avoids having many similar views of the same object, which would not give any new information, but would increase the processing time.

At the end of the learning mode, the *learner_recognizer_node* (depicted in Figure 4.5) stores all the views in a file that can be latter used for matching during the exploitation phase. At this point, the system also has created the vector Ω which contains the *id* of all the learned objects in the session.

4.4.3 System exploitation mode

Once the system is trained by the user, it is able to start recognizing learned objects. This phase is the exploitation mode (Figure 4.6). This mode starts, like in the learning mode, by selecting the ROI and extracting the features from the Kinect's input. After that, the system tries to match the retrieved features to the previously stored ones. This matching is carried out using the FANN (Fast Approximate Nearest Neighbour Search) [Muja and Lowe, 2009] algorithm, which produces an approximate Nearest Neighbour for a given point in a high dimensional space. Although FANN's accuracy is outperformed by other

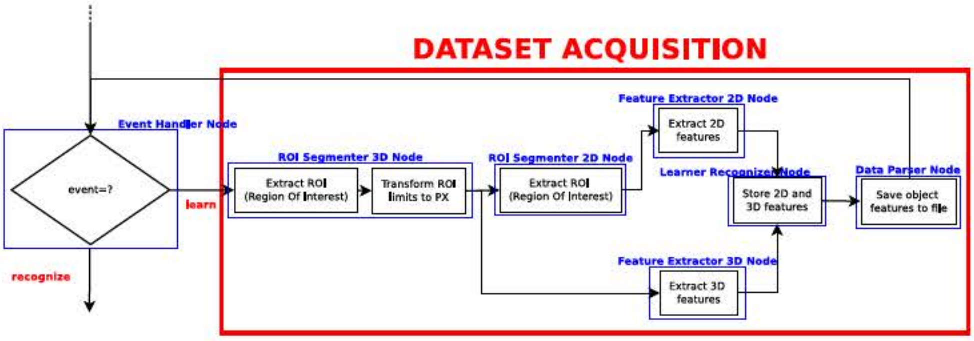


Figure 4.5: Flowchart of the Learning Mode.

algorithms in the state of the art it has been widely adopted because it combines good accuracy results with lower computational requirements that enable FANN to compute its results faster than other algorithms.

Note that Figure 4.6 shows separate matchers for the 3D Point Cloud (*PC Matcher*) and for the 2D image (*RGB Matcher*)[1]. Despite we use FANN for both cases, we perform the matching separately so we can later evaluate the results independently.

4.4.4 System Output

Both RGB and PC matchers give a prediction for each every frame they receive from the Kinect. That is, both matchers output 30 predictions per second.

[1] We will refer, indistinguishably, to the Point Cloud matcher as *PC Matcher* or *3D Matcher*. Also we will refer to the RGB matcher as *RGB Matcher* or *2D Matcher*

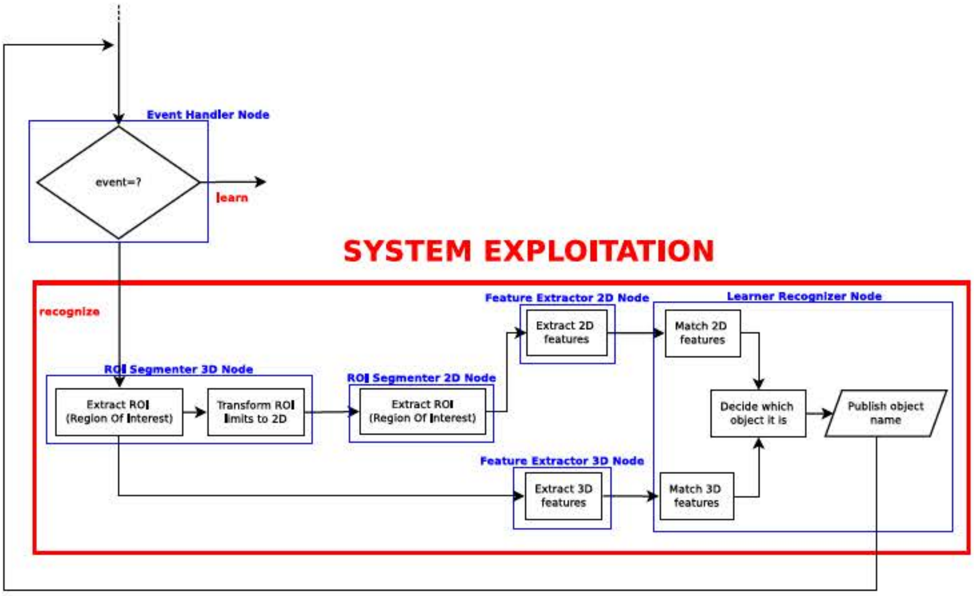


Figure 4.6: Flowchart detailing the system exploitation or recognition phase.

We observed that, due light conditions and the approximate nature of FANN matchers, some of these predictions might be incorrect. These misclassifications are isolated and rarely come in bursts. Therefore, they can be considered as random white noise that occurs at high frequencies. To mitigate these adverse effects, the last step of our estimator is a low-pass filter that produces an aggregated final prediction at a frequency of $1Hz$. This filter works as follows:

The filter receives, every second, two vector of predictions p'_{RGB} and p'_{PC} which contain the 30 last predictions from the *RGB Matcher* and the *PC Matcher*, respectively. With these predictions is possible to calculate the Probability Mass Function (PMF) $\mathrm{pmf}(x)$ of each prediction vector. These PMFs are calculated by counting the frequencies f'_{RGB} and f'_{PC} in which each learned object $o_{id} \in \Omega$ appear in p'_{RGB} and p'_{PC}, respectively, and then dividing them by the number of predictions per second. In the case of the *RGB Matcher*:

$$\begin{aligned} f'_{RGB} &= \mathrm{pmf}(p'_{RGB}) && (4.1) \\ &= \big[\frac{1}{|\Omega|}\,\mathrm{freq}(o_{id},\ p'_{RGB}) \mid o_{id}\ \in\ \Omega\big] && (4.2) \end{aligned}$$

where Ω is the vector containing all the object *ids* learned in the Learning Mode, and $\mathrm{freq}(o_{id},\ p'_{RGB})$ is the number of times that a particular learned object o_{id} appears in p'_{RGB}. We also obtain the frequencies for each prediction of the *PC Matcher*:

$$\begin{aligned} f'_{PC} &= \mathrm{pmf}(p'_{PC}) && (4.3) \\ &= \big[\frac{1}{|\Omega|}\,\mathrm{freq}(o_{id},\ p'_{PC}) \mid o_{id}\ \in\ \Omega\big] && (4.4) \end{aligned}$$

note that f'_{RGB} and f'_{PC} are vectors of length $|\Omega|$. The next step is to combine both vectors in a single one in the following way:

$$f' = w \cdot f'_{RGB} + (1 - w) \cdot f'_{PC} \qquad (4.5)$$

where $w \in \mathbb{R}, [0 \leq k \leq 1]$ is a constant that gives more weight to the matcher whose predictions are considered more reliable. We found empirically that $w = 0.6$ produced the best results with our data. That means that, in our case, the *RGB Matcher* produced more accurate results than the *PC Matcher* in most cases. Both matchers are compared in section 4.6.2.

The last step of our estimator is estimating which of the learned objects corresponds to the object the user is holding in his/her hand. To do so, the system returns the predicted object *id* y' which is the *id* of the most likely prediction in f':

$$y' = \underset{f'}{\text{index}}(\text{argmax}(f')) \tag{4.6}$$

where $\text{index}_v(x)$ returns the position of the value x in the vector v. Note that this is actually the object *id* that was predicted most times during the last second. This allows to filter isolated misclassifications and reduces the impact of short burst of errors that might occur when the user moves the object, etc.

Example of the output of our system:

As an example of how the predictions of our matchers are combined, imagine that our system has learned 3 different objects with *ids* 0, 1, and 2. The *ids* of the learned objects are stored in the vector $\Omega = [0, 1, 2]$, where each position of Ω is the *id* of one object. Now imagine that our matchers produce 8 predictions per second (our real system produces 30 predictions per second, but we reduce it here for brevity) and that the RGB and Point Cloud matchers produce the following predictions:

$$\begin{aligned} p'_{RGB} &= [1, 0, 2, 1, 1, 2, 0, 1] \\ p'_{PC} &= [1, 2, 2, 1, 2, 2, 2, 2] \end{aligned}$$

where each position in p'_{RGB} and p'_{PC} corresponds to the predicted *id* of one object $o_{id} \in \Omega$. These predictions produce the following probability mass functions.

$$\begin{aligned} f'_{RGB} &= (0.25, 0.5, 0.25) \\ f'_{PC} &= (0, 0.25, 0.75) \end{aligned}$$

and then, following Eq. 4.5 we obtain the final prediction:

$$\begin{aligned} y' &= \underset{f'}{\text{index}}(\text{argmax}(0.6f'_{RGB} + 0.4f'_{PC})) \\ y' &= 2 \end{aligned}$$

That is, our system predicted that the object being shown was the third one.

4.5 Experimental Method

To validate the system, we evaluated its performance in terms of accuracy of its predictions and of its computing requirements. The motivation of the latter is to ensure that the learning is interactive, which requires that the system must be able to perform the processing and analysis of its visual input in real time.

Our experiment consisted in a user showing the system a set of 6 objects (see Figure 4.7) which must be detected after the training. The light conditions were a mixture of natural and artificial light. The natural light was cast from a window situated on the left part of the room. The user presented the objects at an approximate distance of 1.7m of the robot, which used a Kinect as its visual input.

The computing was carried out in an Intel Core i7-3630QM CPU (4 cores) operating at 2.4Ghz[1], 8GB of RAM (1600MHz) and a 120GB Solid State Drive (SSD).

4.5.1 Experimental procedure

The experiment was carried out in the following way. In Learning Mode (described in section 4.4.2), the user showed each object to the system by extending his/her hand towards it. trying not to occlude it too much with the user's fingers. To compare how the system learns with different number of views, we repeated this process three times, to acquire 1, 5, and 10 views per object respectively. In the case of the 5 and 10 view trials, the user rotated slowly the object while the system was capturing 1 view per second (we capture one view per second to give the user time to rotate the object so we do not get many views that are almost identical).

[1] With turbo of 3,4GHz

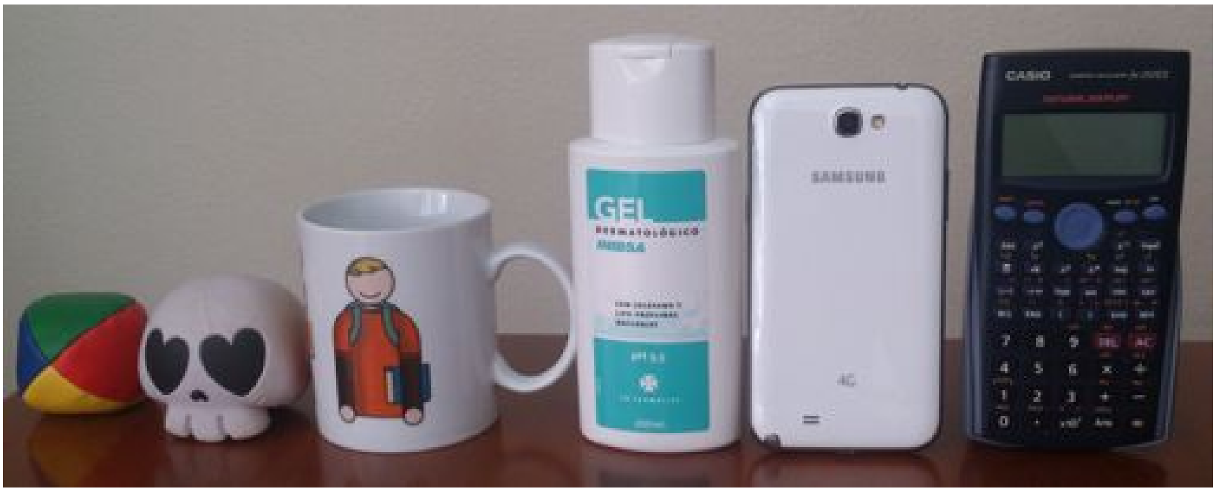


Figure 4.7: Dataset for the experiment. From left to right: ball, skull, cup, mobile phone, and calculator.

4.5.2 Evaluation Metrics

The accuracy of the system was measured by calculating the F1-Score metric:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.7}$$

where the *precision* and *recall* can be obtained from the confusion matrix in the following way:

$$precision_{ij} = \frac{M_{ij}}{\sum M_j} \tag{4.8}$$

$$recall_{ij} = \frac{M_{ij}}{\sum M_i} \tag{4.9}$$

where M_{ij} are the elements of the confusion matrix, and i, j the indexes of its rows and columns respectively.

To evaluate the system requirements, we used the metrics CPU consumption and RAM consumption to measure the computing load; and BandWidth (BW) and publishing rate to measure the network load of the system.

4.6 Experimental Results

This section presents the results of the experiments described in the previous section. The section starts with the results of the computing requirements evaluation and follows with the results of the learning performance of the system.

4.6.1 Computing Performance evaluation

This section presents the results of the computing performance evaluation. We evaluate the computing performance of the system by firstly monitoring its CPU and RAM consumption in section 4.6.1.1 and secondly monitoring its network usage requirements in section 4.6.1.2.

4.6.1.1 CPU and RAM usage

We monitored the CPU and RAM usage of every node involved in the experiment. The results are summarized in Table 4.1. Note that the table differentiates between the third party drivers and the nodes developed for the experiment.

Node	CPU Usage (%)	RAM Usage (%)
Converter	0.20	0.50
ROI segmenter 2D	0.84	0.50
ROI segmenter 3D	13.34	1.50
Feature extractor 2D	2.44	0.10
Feature extractor 3D	4.74	0.50
Event handler	0.16	0.50
Learner recognizer	1.04	0.80
System output	0.13	0.30
Total Nodes:	**22.88**	**4.70**
pi_tracker node: skeleton_tracker	4.86	3.2
openni_launch nodelet	16.19	1.00
Total (including drivers):	**43.84**	**8.90**

Table 4.1: CPU and RAM usage of the system

The aggregated total CPU consumption of the system is under 45%, including the computation requirements of third party drivers. The RAM consumption was nearly a 9% of the 8GB available.

4.6.1.2 Network usage

The results of the network load evaluation are summarized in Table 4.2. Our data shows that the system publishes information at nearly 30FPS, which means that the system can work in real time. Note that the last entry in the table the node "*Final Object ID*" outputs its results at a frequency slightly lower than 1Hz. This value is the expected one, since this node operates at that frequency as described in section 4.4.4.

Regarding the Bandwidth (BW) consumption, the system requires slightly less than 7MB/s, which is enough to operate in a WiFi connection in case it is needed. This network consumption, enables the system to be distributed across several computers in case it should be is installed in a robot with less computing power. For instance, it would be possible to run some nodes in the robot while sending the most intensive CPU-bound calculations to a central server with dedicated CPUs for such tasks.

4.6.2 Evaluation of the object recognition performance

Table 4.3 shows the F1 scores from our object recognition experiment. The results are for 1, 5, and 10 views. These results are further described in sections 4.6.2.1, 4.6.2.2, and 4.6.2.3). After, we compare the effect that *RGB Matcher* and the *PC Matcher* had in the final prediction in section 4.6.2.4.

Topic	Publishing Rate [Hz]	BW (MB/s)
Hand location	27.56	$1.38 \cdot 10^{-3}$
Segmented image	26.70	1.79
Segmented image with keypoints	25.91	$1.64 \cdot 10^{-3}$
Segmented coordinates (px)	11.56	0.212
Segmented point cloud	18.18	1.48
Descriptors 2D	25.98	$29.48 \cdot 10^{-3}$
Descriptors 3D	15.29	2.60
Event	27.72	$1.06 \cdot 10^{-3}$
ObjectID	26.40	$1.39 \cdot 10^{-3}$
Final object ID	0.75	1.15
Total:	**-**	**6.65**

Table 4.2: Network load results

	1 View			**5 Views**			**10 Views**		
Object	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
ball	0.55	0.55	0.55	0.53	0.83	0.65	0.59	0.93	0.72
skull	0.87	0.45	0.59	0.43	0.62	0.51	0.69	0.69	0.69
cup	0.48	0.52	0.50	1.00	0.66	0.79	0.79	0.76	0.77
bottle	0.44	0.59	0.50	0.77	0.69	0.73	0.96	0.83	0.89
mobile	0.37	0.55	0.44	0.95	0.67	0.78	0.95	0.72	0.82
calculator	0.88	0.52	0.65	0.82	0.62	0.71	0.92	0.76	0.83
Total	0.60	0.53	**0.54**	0.75	0.68	**0.70**	0.82	0.78	**0.79**

Table 4.3: F1-score, precision and recall metrics for 1, 5, and 10 views per object

4.6.2.1 Performance Results when using 1 view

Table 4.4 is the confusion matrix obtained during the system exploitation phase. The rows of the matrix correspond to the real class, while the columns are the predicted objects. Note that the diagonal of the matrix indicates the *success rate* of the experiment, that is, the number of *true positives* over the total number of estimations made by the system.

The F1-score is 0.54 (see Table 4.3), which is low but expected considering the robot acquired only 1 view per object. As observed in the confusion matrix (see Table 4.4), many object are confused with others, especially the skull, the cup and the bottle, which have a dominant white component.

Real / Predicted	ball	skull	cup	bottle	mobile	calculator
ball	**0.55**	0.00	0.03	0.14	0.28	0.00
skull	0.03	**0.45**	0.28	0.24	0.00	0.00
cup	0.00	0.03	**0.52**	0.14	0.28	0.03
bottle	0.21	0.00	0.03	**0.59**	0.17	0.00
mobile	0.14	0.03	0.14	0.10	**0.55**	0.03
calculator	0.07	0.00	0.07	0.14	0.21	**0.52**

Table 4.4: Confusion matrix when 1 view per object.

4.6.2.2 Performance Results using 5 views

Tables 4.3 (F-score) and 4.5 (Confusion Matrix) summarize the results for the training with 5 views per object. Here, the system achieves a greater precision (0.75) and recall (0.68), which reflects in an increase of the F1-score (0.70). Objects that were easily confounded, are now easier to detect since the robot has acquired more information of them during training.

Real / Predicted	**ball**	**skull**	**cup**	**bottle**	**mobile**	**calculator**
ball	**0.83**	0.14	0.00	0.00	0.03	0.00
skull	0.34	**0.62**	0.00	0.00	0.00	0.03
cup	0.03	0.21	**0.66**	0.07	0.00	0.03
bottle	0.14	0.14	0.00	**0.69**	0.00	0.03
mobile	0.10	0.13	0.00	0.07	**0.67**	0.03
calculator	0.10	0.21	0.00	0.07	0.00	**0.62**

Table 4.5: Confusion matrix using 5 views per object.

4.6.2.3 Performance Results using 10 views

Tables 4.3 (F-Score) and 4.6 (Confusion Matrix) summarize the results for a training of 10 views per object. Again, precision (0.82), recall (0.78) and F1-score (0.79) increase when compared with 5 views.

Real / Predicted	**ball**	**skull**	**cup**	**bottle**	**mobile**	**calculator**
ball	**0.93**	0.00	0.07	0.00	0.00	0.00
skull	0.31	**0.69**	0.00	0.00	0.00	0.00
cup	0.03	0.14	**0.76**	0.00	0.03	0.03
bottle	0.03	0.14	0.00	**0.83**	0.00	0.00
mobile	0.07	0.03	0.14	0.00	**0.72**	0.03
calculator	0.21	0.00	0.00	0.03	0.00	**0.76**

Table 4.6: Confusion matrix using 10 views per object.

4.6.2.4 Comparison of the 2D and 3D independent recognition results and the system's output

As described in section 4.4.3, the system includes two matchers: the *RGB Matcher* (2D Matcher) and the *Point Cloud Matcher* (3D Matcher). Each matcher produces an independent prediction which latter is combined as described in section 4.4.4.

Table 4.7 compares the prediction power of each matcher by separate while table 4.8 shows how the system benefits when both are combined.

	RGB Matcher			Point Cloud Matcher		
Object	**Precision**	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
ball	0.49	0.84	0.62	0.62	0.79	0.69
skull	0.65	0.69	0.67	0.53	0.42	0.47
cup	0.52	0.73	0.61	0.43	0.64	0.51
bottle	0.61	0.87	0.72	0.56	0.72	0.63
mobile	0.73	0.79	0.76	0.45	0.63	0.53
calculator	0.89	0.82	0.85	0.66	0.82	0.73
Total	0.65	0.79	0.71	0.54	0.67	0.59

Table 4.7: F1-score of the *RGB* and the *PC* Matchers (10 views per object).

	F1 Score		
Object	**RGB**	**Point Cloud**	**Combined**
ball	0.62	0.69	**0.72**
skull	0.67	0.47	**0.69**
cup	0.61	0.51	**0.77**
bottle	0.72	0.63	**0.89**
mobile	0.76	0.53	**0.82**
calculator	0.85	0.73	**0.83**
Total	0.71	0.59	**0.79**

Table 4.8: F1-score comparison for *RGB and Point Cloud Matchers* combined (10 views per object and $w = 0.6$).

Our data shows that the RGB matcher is a better predictor than the Point Cloud (0.71 vs. 0.59). A possible explanation for this difference might come from the depth resolution of the Kinect, which decreases exponentially with the distance [Khoshelham and Elberink, 2012]. At the operating distances of our experiment ($1.7m$), the typical depth error of the sensor is nearly $1cm$.

Despite the performance differences between matchers, the F1-score increased in 5 of the 6 tested objects when both were combined. Only the calculator had a worse F1-Score, but it was only a 2% decrease. We believe that this was caused because the calculator was the object with the richest textures, which facilitated its recognition for the RGB matcher.

4.7 Discussion

Our experiments showed that, as it was expected, increasing number of views also increased the F1-score. This suggests that the system can ask for more data to the user when it detects that the accuracy for detecting an object is not big enough.

We used objects with similarities either in texture or in shape since we hypothesized that these kind of objects are representative of the need of combining texture (RGB) and shape (Point Cloud) matchers. For instance the ball and the skull had a dominant spherical shape but differed in textures (see Figure 4.8). Also, when only using few views, the cup, bottle, mobile and calculator are an almost-rectangular shape if looked from the front. Even more, the bottle, mobile and calculator share a similar cubic shape (Figure 4.9).

In the case of the textures, some objects shared a strong white dominant (skull, cup, bottle, smartphone). However, they had other strong textured components that might have helped the *RGB matcher* to achieve better results than the *Point Cloud* Matcher.

As a general summary of the experiments, the combination of both matchers helped to discern between similarities in a single matcher. However, we found that the impact of the RGB matcher on the detection accuracy was greater than the PCL's. The reason for this difference might come from the fact that the depth resolution of the Kinect was around $1cm$. We believe that greater depth resolution might increase the accuracy of the PC Matcher.

4.7.1 Limitations and Future Work

The present section explains the limitations of our approach and describes which upgrades could be implemented to overcome them.

Hand location When a person shows an object to the system, the skeleton tracker considers this object as a part of the user's hand, introducing, therefore, an error in the

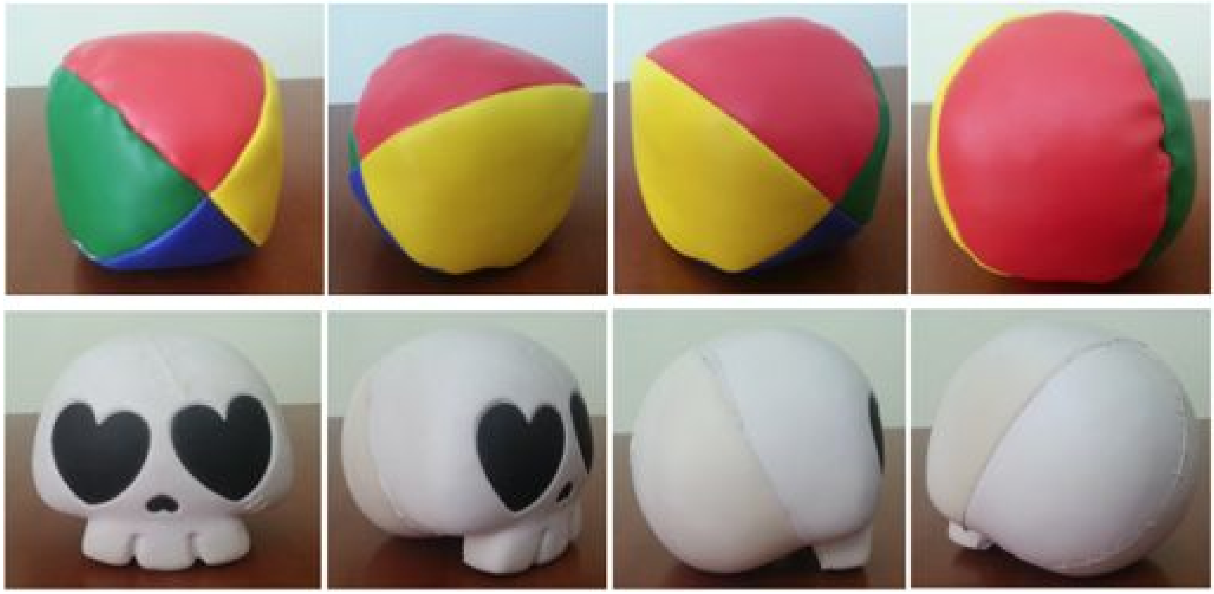


Figure 4.8: Detail of the ball and the skull using different views to illustrate their similar shape.

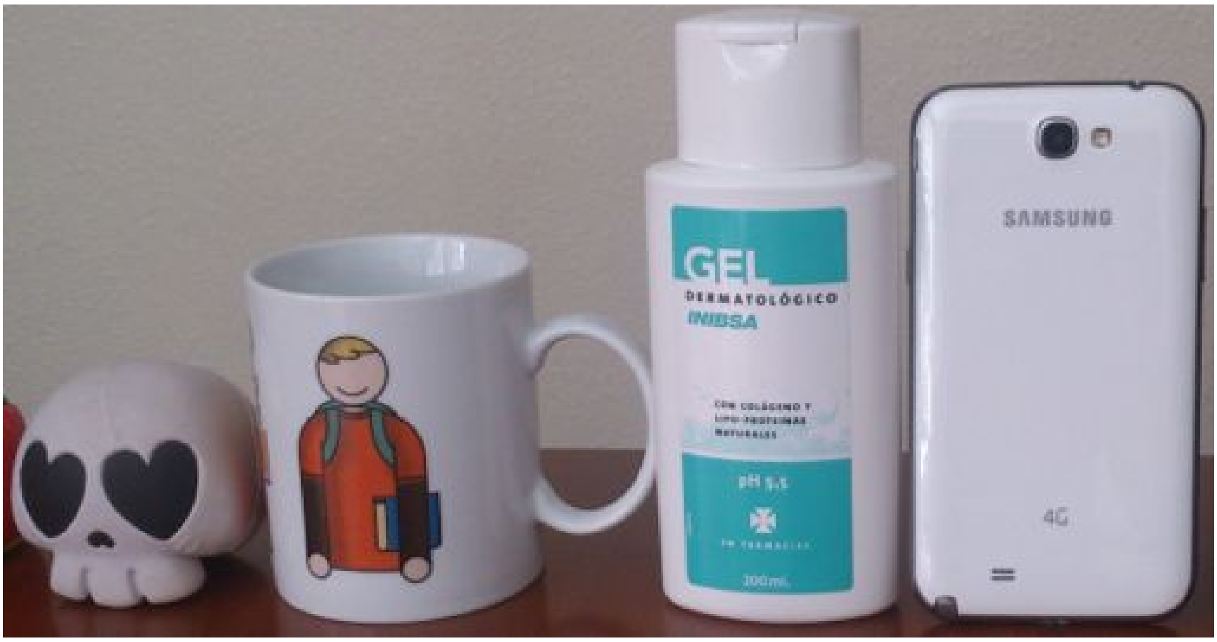


Figure 4.9: Detail of the skull, cup, bottle, and mobile objects to illustrate their similarities when using only 1 view.

user's hand location. This location error affects the ROI segmentation since big objects might not be segmented correctly (they may end cut by the ROI segmenter). A possible solution may be to implement a new hand location taking into account for example the skin color or to extend the ROI section automatically if the object is too big.

Feature extraction Feature extraction is performed in RGB and Point Cloud data. The RGB feature extractor is a state-of-the-art solution for this application. It is faster and less time-consuming than the alternatives with comparable experimental results [Miksik and Mikolajczyk, 2012]. The PC features are extracted using many approximations in order to reduce the huge computing time they require. Hence, they are less representative leading to more potential classification errors in the matching process.

Therefore, a potential upgrade of the system is to use other feature extractors for the Point Cloud. Some new algorithms such as LINEMOD [Hinterstoisser et al., 2012] might be suitable for our approach For instance, LINEMOD is able to extract features of texture-less objects. Such an extractor might help improve our matchers performance, specially in the cases where there are many texture-less objects such as house-hold objects, etc.

Learning and recognizing methods We followed a template matching approach as our first steps into the field of in-hand object recognition. Although we achieved good accuracy results in our dataset, we expect to expand the learning capabilities of the system by using machine learning algorithms such as Random Forests [Gall et al., 2012] or Support Vector Machines (SVMs) [Pontil and Verri, 1998]

4.8 Conclusions

This chapter presented a system that implements in-hand object recognition that fuses RGB and Point cloud data. The fact of using both types of information improve the robustness of the system to illumination changes or noise in the input data stream. The system works in real-time by matching the visual input of the system with templates of pre-learnt objects.

Our system is validated in an experiment aimed to evaluate two aspects: first, that the system is able to run in real time, and second, that combining RGB and Poing Cloud matchers increases the accuracy of the matching process.

The results demonstrated that the system runs in real time and that its accuracy is near an 80% when the dataset uses ten views per object. Depending on the evaluated object, the F1 score laid between a 70% and an 80%. We expect that increasing the number of views per object might increase these numbers a little bit more.

This system is, potentially, a good candidate to be a good benchmark for our Active Learning approach. AL can act in several points of the data pipeline. For instance, it is possible to extract the information gain of each matcher to analyse which one is more informative and tune the weights of Eq. 4.5 accordingly. Also with AL is possible to know when a prediction has high entropy (i.e. it is uncertain) and, therefore, ask for a label of the shown object to the user.

Part III

Active Learning for Social Robots

Chapter 5

How Much Should a Robot Trust the User Feedback? Analyzing the Impact of Answers in Active Learning

This chapter is the first one to incorporate Active Learning to robot learning. Here, using a similar system to the presented in Chapter 3, we analyse how the answers to different questions may affect the learning performance of the robot. This work has been submitted to the International Journal of Social Robotics and it is currently in the review process.

Active Learning (AL) allows robots to learn faster and better than passive learners by enabling them to ask questions to their human teachers during the learning session. However, if the teacher is not able to provide accurate answers to the robot's questions, the learning accuracy of the robot might decrease. As an additional problem, it might be difficult for the robot to know whether the user will provide an accurate answer or not before incorporating this answer to its knowledge. Hence, in some cases, the robot might be asking a question without knowing that its response may lead to a decrease the system accuracy instead of increasing it. This paper presents an experiment where, after teaching a robot, a group of users are asked several questions whose answers are used as feature filters in the robot's learning space. We study how the answers to different types of questions affect the learning accuracy of a social robot when it is trained to recognize poses and we compare the learning performance of a robot that learned the same poses actively and passively. Finally, we provide a method where the robot reduces the effects of inaccurate answers by lowering the trust in the user's responses. Our results show that, despite AL can improve the robot's learning accuracy, there are some cases where AL achieves significant worse results than Passive Learning (PL) if the

user provides inaccurate feedback when asked. In our experiment, our method has proven to maintain the benefits of AL even when the user answers are not accurate. With this method the robot can incorporate the domain knowledge given from the user's answers without worrying whether the user will produce a bad quality answer.

5.1 Introduction

If social robots are going to establish long-term relationships with humans, it is expected that there will be situations where robots would need to learn from people. Such learning process should be interactive, natural, and it would require robots to learn fast and cause the less disturbances as possible to humans. These topics are studied in the field of Socially Guided Machine Learning (SG-ML) [Thomaz et al., 2006; Thomaz and Breazeal, 2007, 2008] and receive inputs from research in Human-Robot Interaction (HRI) [Fong et al., 2003; Goodrich and Schultz, 2007].

Recently, new developments in SG-ML have started to include ideas from Active Learning (AL). This kind of learning enables social robots to ask questions[1] actively to the users instead of letting them the initiative of the interaction. AL comes from the Machine Learning field and it was introduced by Angluin [Angluin, 1988]. Whilst in Passive Learning (PL), it is the teacher who provides the examples to the learner and labels them, in Active Learning it is the learner who takes the initiative and asks queries. This queries can consist in asking for labels of the learning examples or demand information about certain parameters of the learning problem.

In HRI, AL is carried out interactively so the robot asks questions directly to the user during or after the learning session. This mimics how humans learn: first, by observing their teacher, and then, by asking questions when they have any doubts about the concept to be learnt or the examples they have seen.

The use of AL in robotics has three main motivations. First, active learners can potentially obtain better accuracy of the learned concepts [Cakmak et al., 2010]. Second, AL may reduce the number of training examples needed to acquire a concept [Settles, 2010]. This is specially relevant in robotics since in interactive learning the cost of acquiring a training example might time consuming. Finally, humans seems to prefer to train robots that learn actively over passive ones [Cakmak et al., 2010].

Nevertheless, people prefer robots that do not ask too many questions [Cakmak et al., 2010]. Therefore, knowing how different queries affect the learning performance can help the robot to ask the questions that maximize its learning. Hence, the robot can avoid being constantly querying the user.

There is literature that assessed different types of queries for AL [Cakmak and Thomaz, 2012] and that evaluated how to ask these questions to maximise the accuracy of the user answers [Rosenthal et al., 2009, 2012]. However, we did not found evidence on how different types of queries affect the robot's learning performance.

This chapter explores the impact of different types of Feature Queries -that is queries that seek information regarding the learning parameters-, on the accuracy of an interactive

[1] Questions and queries will be used indistinctly in this paper.

pose learning task. The chapter presents three different types of Feature Queries: Free Speech Queries (FSQ), Yes/No Queries (YNQ) y Rank Queries (RQ) and we use them to seek and filter the parameters which are less relevant for the learning. We studied how these data filters affect the robot's learning performance. Additionally, we found that people answer's can sometimes be too simplistic producing a negative impact in the learning. As a consequence, the robot might learn worse than a passive learner.

Despite it seems that in such cases it might be wiser not to ask the user, most times it is unknown whether the user responses are accurate or not. Accordingly, and knowing that AL can be highly beneficial from the interaction point of view, we believe that robots should ask queries even though their answers might be inaccurate.

Therefore, it is necessary to address this problem by minimizing the impact of inaccurate answers. We propose a method which consists in reducing the robot's confidence on the user's answers. Our method consists in extending the filters created from the robot's queries by including more parameters than the user answered.

We carried out an experiment where 24 users trained a social robot to detect different poses. The users were asked for Feature Queries to assess the relevance of their limbs for each pose. In our experiment we found that FSQ produced simpler filters than YNQ and RQ. When these filters were applied to the training data, we found that, in some cases, the user answers led the robot to learn worse. We hypothesize that this was originated because their answers were to simple and included fewer limbs than the ones actually involved in the pose. Hence, we decided to lower the robot's confidence in the user's answers by adding more limbs to the user filters. This new Extended Filter enabled the robot to learn better than with simply using user's filters.

The remainder of this chapter is divided as follows. First, section 5.2 introduces other works related to AL in robotics. Following, section 5.3 describes our learning approach were we apply AL for pose learning. After, we present our experiment in section 5.4, its results in section 5.5. The results are discussed in section 5.6. Finally, the conclusion and future remarks are in section 5.7.

5.2 Related Work

Although AL is a mature topic in Machine Learning, its potential for robotics has not been noticed until recent years. However, it has quickly attracted the attention of the robotics community [Cakmak et al., 2010; de Greeff et al., 2009; Gribovskaya et al., 2010; Lopes and Oudeyer, 2010; Martinez-Cantin et al., 2010; Merrick, 2010; Singh et al., 2010].

Two research trends can be distinguished in the field. In the first one, robots learn by self-exploring the environment while in the second one, robots leverage on HRI to learn from humans [Lopes and Oudeyer, 2010]. Here, the main vehicle of AL is asking queries to their teachers interactively.

This chapter focuses on the second approach inspired, mainly, by the works of Rosenthal [Rosenthal et al., 2009, 2012] and Cakmak [Cakmak and Thomaz, 2012; Cakmak et al., 2010]. Starting from their research, we attempt to go further by understanding how answers to different types of questions could affect the robot's learning.

Rosenthal [Rosenthal et al., 2009] explores how different questions affect the accuracy and correctness of the user's responses. The paper shows some guidelines on how to ask questions to the user so the error rate of their answers is reduced, even for those people who are not robotic experts. However, they do not studied the impact of the user's inaccuracies on the robot learning performance.

Cakmak et al. [Cakmak et al., 2010] explore the AL field from the human point of view. They propose that robots which ask questions interactively during the learning process are perceived by people as more enjoyable and are preferred over Passive Learners. Additionally, in their experiments, they found that AL might increase the learning speed and accuracy of the robot.

They studied how the robot was perceived when it showed three different degrees of interactivity when asking questions. The three modes of interactivity achieved better learning results than passive learning, but there were no differences among them. However, there were differences on how users perceived interactive robots. Despite users valued positively active learners, they preferred not to lose the control of the interaction, and not being bombarded constantly with questions.

In a further work, Cakmak [Cakmak and Thomaz, 2012] studies how humans ask questions when learning and applies her findings to enable a robot to learn tasks actively by combining active learning and *Learning from Demonstration*.

They introduce three main types of queries the robot can ask when learning: *Label Queries*, *Demonstration Queries*, and *Feature Queries*. The three of them are inspired from traditional AL techniques and adapted to the robot learning problem.

- *Label Queries*, consist in the robot executing an action and then asking the user for the label of that action. These queries are inspired by the original works of AL [Angluin, 1988].

- In *Demonstration Queries*, the robot asks the user for a demonstration of a certain label. They are inspired by Lomasky's *Active Class Selection* [Lomasky et al., 2007], in which an active learner actively asks for new examples.

- *Feature Queries*, consist in asking the user if a certain feature of the learning space is relevant or not. These are inspired by [Druck et al., 2009; Raghavan et al., 2006].

Cakmak presents an experiment where they found that people tend to ask feature queries when they are learning. Even more, when teaching a robot, people perceive it as smarter if it asks for Feature Queries rather than Label or Demonstration Queries.

In general terms, Feature Queries tend to be very verbal, but grounded to the physical world. This fact, together with the idea that users prefer them over other queries, led us to focus our research on the study of Feature Queries. Cakmak studied how different queries are perceived by users, but did not study how they affect the robot's learning capabilities. This chapter combines Cakmak's Feature Queries [Cakmak and Thomaz, 2012], with Rosenthal's ideas on how different questions can lead to inaccuracies in the user's responses [Rosenthal et al., 2009]. We study how these user inaccuracies can lead to a decrease in the robot learning and propose a method to reduce the impact of this problem.

5.3 Learning Scheme

We will apply our learning scheme to pose learning. When the user starts training the robot, he has to carry out two tasks. The first one is to put herself in the pose she wants to show the robot. The second task is to tell the robot at which pose is she standing. From the robot's point of view, firstly, it has to detect the human pose and, secondly, it has to understand what the user is saying it. Using this data, the robot builds a dataset which feeds a learning algorithm. The questions that the robot ask to the user are used to manipulate which information of this dataset is used to learn.

In this section we describe the components that participate in the learning process as well as the learning itself. First, we describe the visual system that enables the robot to see its surroundings. We continue describing the elements that enable our robot to hear and talk with the user, so the user can tell the robot the labels of the poses interactively. After, we present the learning mechanisms used by the robot and we end this section by introducing how our robot is able to learn actively.

5.3.1 Visual Input

We use a Kinect RGB-D camera as the main visual sensor. Concretely, we rely only on its depth data, which it is processed using the OpenNI[1] API. This API builds a skeleton model of the user returning the positions and orientations of 15 joints of the user (see figure 5.1). The positions and orientations of these 15 joints are what will be used as the main input for our learning algorithm.

[1] http://structure.io/openni

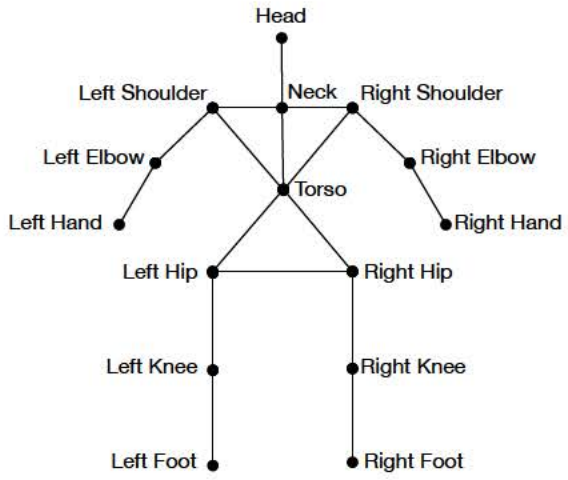


Figure 5.1: OpenNI's kinematic model of the human body. OpenNI algorithms are able to create and track a kinematic model of the human body. The model has 15 joints and their positions and orientations are updated at a rate of 30 Frames per Second.

5.3.2 Interactive Labelling of what the Robot Sees

First, the user needs to tell the robot at which pose is she standing so the robot can put a name to what is observing. In other words, the user has to tag the pose she is showing to the robot. The process of tagging (labelling) the poses is carried out interactively by voice.

Our robot is equipped with an ASR (Automatic Speech Recognition) system described in [Alonso-Martín and Salichs, 2011]. This ASR is grammar-based, which means that the robot extracts semantic meaning from the user's speech providing she follows certain pre-written grammar rules when speaking. A grammar is a set of rules defined by the tuple $G : (U, S)$ which relates a set of utterances U with a set of semantic meanings S. For instance, it is possible to write a grammar that relates the sentences (utterances u_i) "*Please, could you help me?*" and "*Could you give me a hand with this, please?*" to the semantic meaning of soliciting assistance to the robot (Eg. $s_i = user\_needs\_help$)

Note that many utterances $U = \{u_1, u_2, \ldots, u_n\}$ can produce the same semantic meaning, s_i. Each utterance, u_i, might be composed of single words, phrases or any combination of them. For instance, the semantic meaning $s_1 = raining$ might be composed of the utterances:

```
u_1 = [*] raining [*]
u_2 = [*] [take the] umbrella [*]
```

Listing 5.1: Example of utterances that provide the same semantic meaning: *raining*

where the asterisk, "$*$", acts as a wildcard to indicate any word or set of words and the text between square brackets "[ ]" indicate that the elements enclosed by them are optional. Therefore, utterances like "*It seems it is raining.*" and "*Take the umbrella before going* *out.*" trigger s_1.

We have built one grammar $G_p =: (U_p, S_p)$ to detect if a user is telling a pose to the robot. This grammar allows us to detect up to 9 pose definitions separated in three different categories as will be described later. For the sake of brevity, and since the robot was taught in Spanish, we omit all the possible U_p for each $s \in S_p$ (although we provide a couple of examples later in this sub-section). However, we fully describe S_p since they contain the concepts taught to the robot:

$$S_p = \{s_a, s_d\} \tag{5.1}$$

where s_a, s_d are the semantic meanings of the user's speech:

1. **Action Semantics, $\mathbf{s_a}$**: This can take one of the following values:

   $$s_a = \{turned\ , \ looking\ , \ pointing\}$$

   where:

   (a) *turned* defines that the user is turned (oriented her body) to the direction specified in s_d.

   (b) *looking* defines that the user has oriented her head to the direction specified in s_d.

   (c) *pointing* defines that the user is pointing in the direction specified in s_d.

2. **Direction Semantics, $\mathbf{s_d}$**: This can take one of the following values:

   $$s_d = \{left\ , \ forward\ , \ right\}$$

   where:

   (a) *left* defines that the action defined in s_a is carried out to the user's left side. For example, if the trainer is pointing (s_a = *pointing*), she is doing it to her own left.

   (b) *forward* defines that the action defined in s_a is carried out toward the user's front. For instance, if the trainer is pointing (s_a = *pointing*), she is doing it toward her own front.

   (c) *right* defines that the action defined in s_a is carried out to the user's right. For instance, if the trainer is pointing (s_a = *pointing*), she is doing it to her own right.

When the robot detects a speech, the ASR evaluates it against G_p. If the evaluation produces a valid result, the ASR tags the speech with a label $l_p^{(i)} \in G_p$. To be considered a valid label $l_p^{(i)}$, our grammar needs both s_a and s_d semantic meanings.

Some label examples might be $l_p = (looking, left)$, which indicates the user is looking to her left; or $l_p = (turned, forward)$, which indicates the user is turned towards to the robot. Note that, as we defined before, the user can arrive to the same semantic meaning in different ways. For instance, phrases such as "*I'm looking to the right*" or "*I'm in a chair, looking towards my right*" would produce the same semantic meaning: $l_p = (looking, right)$

5.3.3 Learning

With the visual and verbal inputs, the robot is able to create a dataset of training instances that it will use to learn the poses of the user. The visual input is the training data itself while the verbal labels are used to establish the class label of each training instance.

Each training instance $I_{tr} = (J, l)$ is composed of the joint set (J) retrieved from the visual system and the class label $l_p \in G_p$ which defines the pose of J and is retrieved from the grammar G_p. The joint set $J = (j_1, j_2, \ldots, j_{15})$ contains the 15 joints of the user as depicted in figure 5.1. Each joint j has the following parameters:

$$j_i = (x_i, y_i, z_i, q_{x_i}, q_{y_i}, q_{z_i}, q_{w_i} : 1 \leq i \leq 15) \tag{5.2}$$

where x, y, z represent the position of the joint in $\mathbb{R}^3$ and q_x, q_y, q_z, q_w is a quaternion representing its orientation.

While the user keeps training the robot, each instance is added to dataset D_{tr}:

$$D_{tr} = \{I_{tr}^{(1)}, \ldots, I_{tr}^{(m)}\} \tag{5.3}$$

where m is the number of training instances of the dataset. Since the user is who decides when to stop the training, m might vary between users.

Once the user stops the training by emitting a voice command specified in a control grammar, the robot can apply a learning algorithm to learn from D_{tr}. In this chapter we use three different learning algorithms: J48 [Quinlan, 1993], Random Forests [Breiman, 2001] and SMO [Shevade et al., 2000].

The algorithms that we use in this chapter are freely available through the Weka Framework [Hall et al., 2009]. Since our purpose is not to develop or compare learning algorithms, but to observe the impact of AL on them, we did not run any model fitting nor parameter optimization. Instead, we use the default parameters as provided by the framework. Despite not being optimal, these parameter choice provides good performance on a wide range of datasets.

Once the robot has applied one of these learning algorithms to D_{tr}, it obtains a model of the poses the user has taught to it. In other words, the robot is able to establish the relationship $J \rightarrow l_p$. This model can be used, later, in the exploitation phase, to detect at which pose is the user standing. At this point, this learning process is passive since the robot does not ask actively any question to the user. Following, we introduce the additions to our learning scheme so our robot can behave as an active learner.

5.3.4 Proposed approach for active learning

We use AL to ask questions to the teacher related to these poses. We focus on feature queries to find which features of the learning space are more relevant since users perceive them as the smartest and they use the most preferred [Cakmak and Thomaz, 2012].

In our approach, we ask the questions once the training session is over. At this moment, the robot asks the user which parts of her body have been the most important for each pose. For instance, if the user has taught the robot a pointing pose, it is expected that when the robot asks for the features that are more relevant for this pose, the user's answer should indicate some part of her arm.

With the user's answers, the robot filters all the features which has been told to be less relevant. In this chapter, we focus on the learning effects of filtering parameters due AL. In other words, we do not study how users perceive the questions or which kind of queries are most helpful to explore the unknown learning space. Contrarily, we study how different user responses affect the learning performance for the case of pose learning.

One aspect that must be taken into account is that feature queries for parameter selection might be too technical from the point of view of the user. Even though we are not focusing on the human side of the HRI, our aim is to have interactions as much natural as possible. Hence, we simplified the parameter selection process by asking non technical questions to the user. For instance, instead of asking for the orientation in certain axes of the user's hand, the robot can ask whether the user's hand is considered important or not for a certain pose. If she answers affirmatively, then we use all the parameters related to the hand. Conversely, if she answers negatively, then the robot filters these parameters not using them in the learning phase.

Regarding the questions themselves, we have taken into account the effect in which the user's responses can be affected by the way the questions are asked [Rosenthal et al., 2009]. For that reason, we considered three types of questions to the user.

The first one, which we called "*Free Speech Queries*" (FSQ) consists in open questions which allow the user to answer freely. The second type, "*Yes/No Queries*" (YNQ), force the user to answer with a "yes" or "no" statement. Finally, in the third type, which we call "*Rank Queries*" (RQ), the user must answer quantifying the importance of a limb from *not important* to *very important*. Examples of these type of questions are:

- **Free Speech Queries (FSQ)**:
  - "*Which is the most important limb in this pose?*"
  - "*Which limbs are important in this case?*"
- **Yes/No Queries (YNQ)**:
  - "*Is the hand important?*"
  - "*Should I pay attention to your head when you are pointing?*"
- **Rank Queries (RQ)**:
  - "*How important is your hand?*"

The answers to these questions produce different information. Typically, answers to FSQ provide a single limb or a short list limbs which the user considered important. For instance "*I suppose my arm*" or "*My hand and my head*". Moreover, in these questions, is the user who decides from which limbs she will provide feedback. Therefore, their use might be interesting when the robot does not know which limbs might be important. Conversely, YNQ and RQ force the user to answer only about the limb the robot asked for. Hence, these questions are better to retrieve information from a specific limb.

The major drawback of FSQs is that their answers need to be parsed in order to map what the user has told to the limbs shown in Fig 5.1. This is not needed in YNQ and RQ since they are directly related to a limb and their answers are typically prefixed (yes/no, very important, quite important, etc.).

Although YNQ and RQ are similar, there are some differences among them. The main one is that YNQ provide less information than RQ. I.e. YNQ only tell whether a limb is important or not whilst RQ quantify this importance.

Once the robot has all the answers from the user, it processes them to decide what limbs are the most relevant to learn a certain pose. Our system makes this decision establishing a threshold in which, if a limb has a value below it, it will be filtered out and therefore, not used for learning.

This threshold is calculated differently depending on the type of the questions. In FSQ, each user that have trained the robot, provides a list of limbs which she has considered important, so we can calculate the number of times a limb has been mentioned by the users to get a score of its relevance R_l (Note that we use the terms *relevance* and *importance* indistinctly). This score can oscillate between 0, if no one considered that limb important, and N_u (number of users that have trained the robot) if every user mentioned it when they were asked. Therefore, with the user answers we can build a list of relevances:

$$R_{FSQ} = \{R_{head}, R_{neck}, ...\} \tag{5.4}$$

in which are stored the relevances for all the 15 limbs. We then calculate the mean relevance $\overline{R_{FSQ}}$ of this vector. Our threshold Th_{FSQ} is established as this mean plus one standard deviation:

$$Th_{FSQ} = \overline{R_{FSQ}} + \sigma_{R_{FSQ}} \tag{5.5}$$

We decided to add a standard deviation $\sigma_{R_{FSQ}}$ to the threshold in order to ensure that the limbs which are chosen stand out from the rest.

The threshold in YNQ is calculated similarly, but instead of summing the number of times the user mentioned a limb, we sum all the positive answers (the number of times "*Yes*" was answered). In RQ, the process is slightly different since the answers are not binary (yes/no) but a direct measure of what is the perceived relevance of each limb. What we did is, with the answers from several users, calculate the average relevance for each limb. So R_l in RQ is actually $\overline{R_l}$. The rest of the process is exactly the same as in FSQ and YQ except the criteria in which a limb passes the threshold. In this case, since the relevances are random variables, we decided that a limb passes the threshold if its 95% Confidence Interval (CI) is above it.

The thresholds define which limbs are used to learn and which are filtered out. Hence, AL allows us to build, directly from user's answers, parameter filters that enable the robot to filter the data which is not relevant for learning.

Note that our approach allows mixing the answers from FSQ, YNQ and RQ to build a filter. However, we decided not to do so since our aim for this chapter is to analyse how these type of questions affect the robot learning isolatedly.

5.3.4.1 Parameter Filters from User's Answers

We have built 3 different filters that can be used to pre-process the data before its fed to the classifier. We called these filters:

1. **FSQF**, built from *Free Speech Queries*

2. **YNQF**, built from *Yes/No Queries*

3. **RQF**, built from *Rank Queries*

In the three cases, the final "F" comes from *Filter*. Additionally we created an additional fourth filter:

1. **Extended Filter (EF)**, built as an extension of RQF.

The EF is an RQF which includes all the adjacent limbs included in a normal RQ Filter. For instance, if we have an RQF formed with $\{head,\ neck\}$, its associated EF will include: $\{head,\ neck,\ left\ shoulder,\ right\ shoulder,\ torso\}$. A map of adjacencies is shown in Figure 5.1.

As it will be shown later, this filter improves the RQF in the situations where, due a low quality answer from the user, the RQF (and other AL filters) can be outperformed by Passive Learning. In such cases, the use of our EF ameliorates this effect.

5.4 Experiment

The aim of our experiment is to understand how the answers to FSQ, YNQ, and RQ affects the robot learning. For that reason we prepared an experiment in which several users teach the robot different poses and they are asked afterwards several questions in the form of FSQ, YNQ, and RQ. Therefore, the robot can use this information to improve its learning performance.

Despite the learning session involved natural interaction between the user and the robot, we evaluated the effectiveness of the user's answers in offline questionnaires that were handed to the users just after the training session. This was motivated because the aim of our experiment is to explore the uses of different queries, which required to ask many questions to the users so we can explore which ones are better to ask. Because of that, if the robot asks too many questions, we believe that we are in danger of causing boredom and fatigue to the user and, therefore, he/she might not answer correctly.

5.4.1 Platform. The Social Robot Maggie

The user trained the robot Maggie [Salichs et al., 2006], see Fig. 5.2, a social robot developed in our own lab aimed for research in HRI and Social Robotics. The robot is equipped with a Kinect, an ASR (Automatic Speech Recognition) [Alonso-Martín and Salichs, 2011] and TTS (Text to Speech) systems, coupled in a Natural Dialogue Management System [Alonso et al., 2013], which enable the robot to carry out natural interactions. These components are tightly coupled by using the ROS [Quigley et al., 2009] framework. Although most of these components are self-built, any robot equipped with a Kinect, a TTS and an ASR, can replicate our experiments easily.

5.4.2 Experimental setup

We tested our active learning approach in three experiments where 24 users trained the robot while interacting with it. The 24 users participated in the three experiments. Each

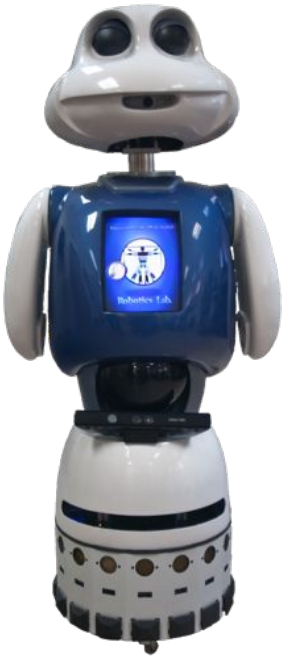


Figure 5.2: Maggie: the robot used in our experiments

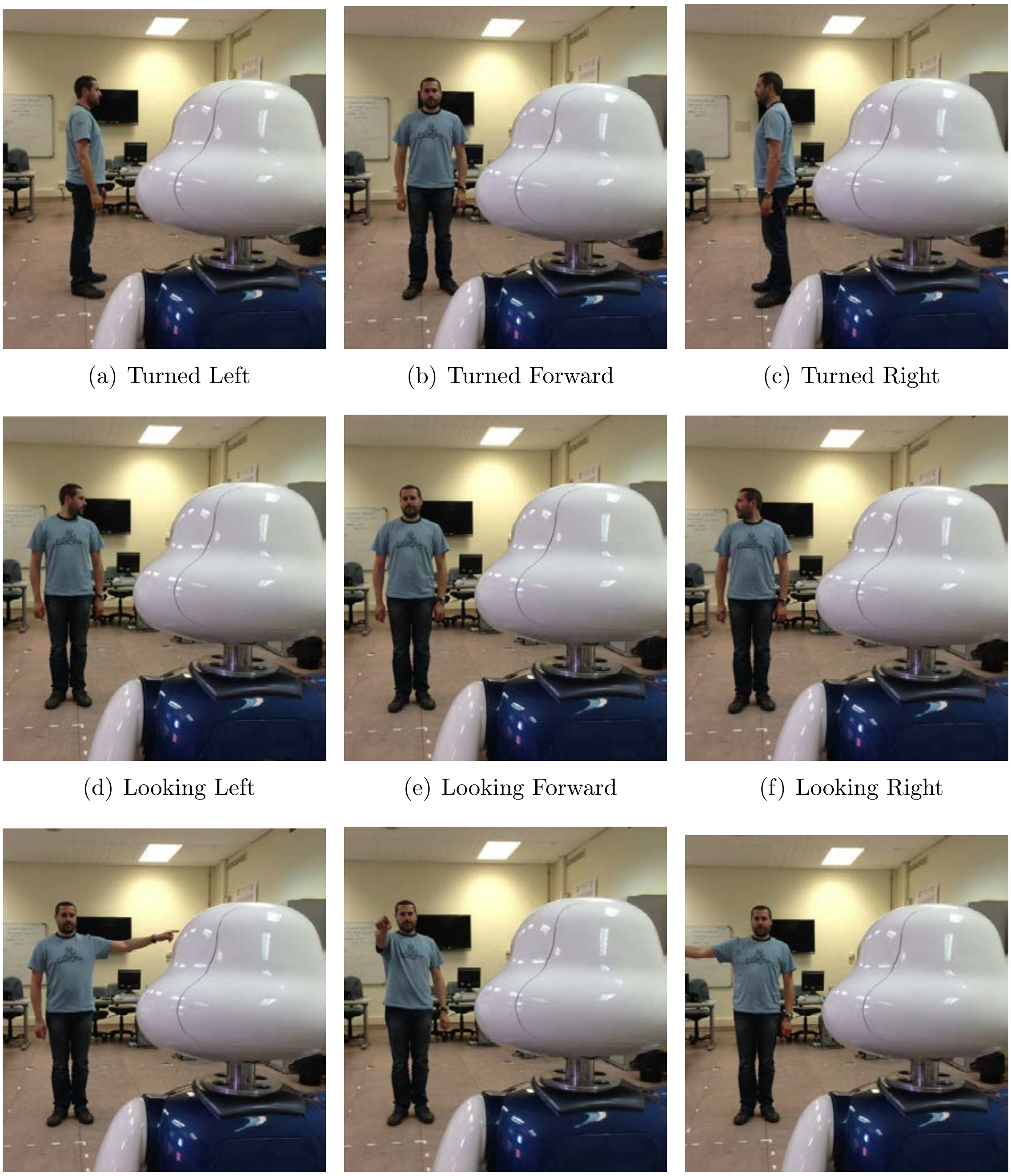
experiment consisted in the user teaching three poses to the robot:

- **Experiment 01**: The users showed the poses *Turned left, turned forward, and turned right.* (Figs. 5.3(a), 5.3(b), 5.3(c)).
- **Experiment 02**: The users showed the poses *Looking left, looking forward, and looking gright.* (Figs. 5.3(d), 5.3(e), 5.3(f)).
- **Experiment 03**: The users showed the poses *Pointing left, pointing forward, and pointing right.* (Figs. 5.3(g), 5.3(h), 5.3(i)).

The training was carried out in the scenario depicted in Fig. 5.4. The users were told to remain inside the rectangle depicted in the figure. This rectangle was drawn in the ground so the users could see whether they were inside it or not. We limited the experimental area to ensure that the user was always inside the Kinect's field of view. However, the users were allowed to move wherever they wanted as long as they stayed inside the limits. Finally, they were told to warn Maggie before changing their pose. This was done so we can separate the transitions between poses.

5.4.3 Method

The users were told to stand in front of Maggie. Then, the experimenter told them all the experimental procedure. After, the users were told that they could ask the experimenter whenever they had any doubts on the labels of the poses or other grammar commands.



(a) Turned Left (b) Turned Forward (c) Turned Right

(d) Looking Left (e) Looking Forward (f) Looking Right

(g) Pointing Left (h) Pointing Forward (i) Pointing Right

Figure 5.3: Examples of poses that the users taught to the robot.

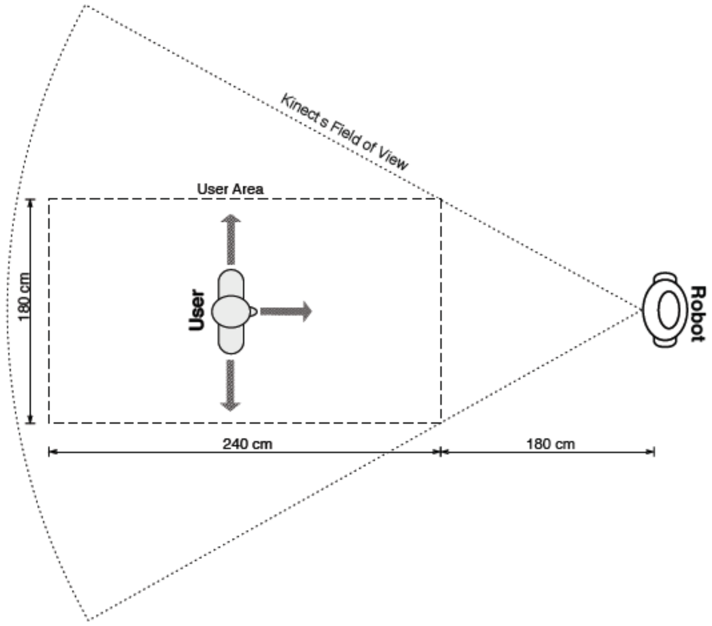


Figure 5.4: Scenario of the experiment. The cone represents the field of view of the Kinect sensor. The user was allowed to move inside the rectangle, as long as she always kept in the same pose while moving.

During training, the users had the freedom to start, pause, and finish the training session whenever they wanted. The procedure was: First, they stand in front of the robot. When the robot tells the user that it is ready to start, the user can start the training whenever she wants. To record each pose, the user was told to: first stand at this particular pose and then tell the robot the label for that pose. Before changing to other pose, the user had to tell the robot to stop recording this pose. This is done with issuing commands provided in a control grammar. Once the robot tells the user that it has finished recording the pose, she is free to move to the next pose and start the process again. The user finishes the teaching session by issuing the "*stop*" command of the control grammar. The session dynamics are similar to a previous work [Gonzalez-Pacheco et al., 2013].

We encouraged the users to start with the left poses, then the front poses, and finish each round with right poses. We did this because, although the order in which the robot learns is not relevant, it helped us to analyse the data after the training. For instance, with this procedure, we found that three users tagged their pose to the left when they were looking (one case) and pointing (two cases) to their right.

We stored three datasets per user, each one corresponding to a single experiment. Each dataset stored the examples of the user in the form of instances as described in eq. 5.3. The robot will feed these datasets to different classifiers to learn the poses.

After having trained the robot, they filled a questionnaire where they were asked questions relative to the poses they taught to the robot. These questions were, first FSQ, then YNQ, and finally RQ. All users had to respond to the three types of questions, although their answers were treated separatedly. The following examples of the asked questions are provided[1]:

- **FSQ**: *What parts of your body do you think the robot has used to learn whether you were TURNED left, forward or to the right?.*

  This example is related to the experiment 1. The same question was repeated for experiments 2 and 3 but changing *TURNED* with *LOOKING* and *POINTING*, respectively. To answer an FSQ, the users were asked to write in an open-text box what limbs were considered most important in each experiment.

- **YNQ**: *What parts of your body do you think the robot has used to learn in Experiment 1?.*

  This question was the same for each experiment, except we change the number of experiment indicated. Together with the number of the experiment, we provided an explanation of what was the experiment. To answer the YNQ, the user had to fill a multiple-choice list of the limbs depicted in Fig. 5.1. Every limb marked by the user, indicated that was relevant for the experiment.

- **RQ**: *Mark the importance of each of the parts of your body so that the robot can learn. [*limb*]*[2].

  RQs consisted in fifteen questions per experiment, each one asking for the importance of a single limb. The limb that was asked for was marked in brackets. For instance, in this example, the asked limb is the *Head*. The answers consisted in a 4-point scale rating the importance of a limb ranging from *Not important at all* to *Very Important*. Notice that in a real learning session is unlikely that the robot would ask for all limbs, but our motivation here was to understand which is the maximum performance gain from each type of question. We believe that this maximum gain is achieved when the robot gathers information from all the parameters.

5.5 Results

5.5.1 Filters for active learning

First, we evaluate the responses of the users to the questionnaires, from which we build the AL filters that will be used for learning. Figure 5.5 shows the results from the user's

[1] The original questions were asked in Spanish. Here we provide the most accurate translations we have found

[2] In the real questionnaire, *limb* was substituted by the name of a limb. For instance *Head*, *Torso*, etc.

Experiment 01 (Turned)	**Free Speech Queries**	LS, RS, Torso
	Yes/No Queries	Head, LS, RS, Torso
	Rank Queries	LS, RS, Torso
Experiment 02 (Looking)	**Free Speech Queries**	Head
	Yes/No Queries	Head, Neck
	Rank Queries	Head, Neck
Experiment 03 (Pointing)	**Free Speech Queries**	LS, LE, LH, RS, RE, RH
	Yes/No Queries	LE, LH, RE, RH
	Rank Queries	LE, LH, RE, RH

Table 5.1: Summary of the selected limbs for each experiment and question types. Note: LS (Left Shoulder), RS (Right Shoulder), LE (Left Elbow), RE (Right Elbow), LH (Left Hand), RH (Right Hand)

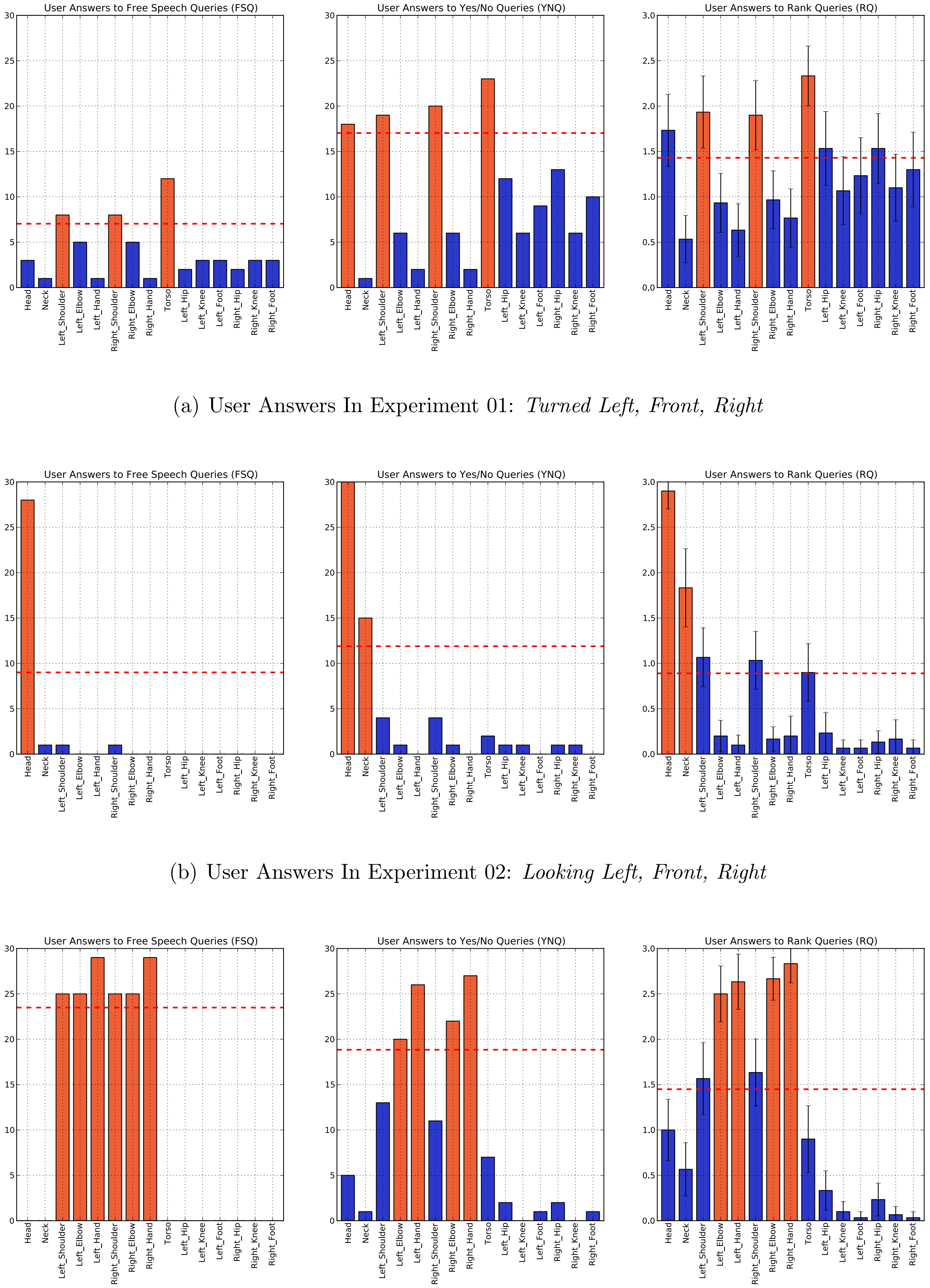
answers in each experiment. The horizontal dashed line indicates the filter thresholds. The limbs that passed the threshold are painted in orange, indicating that they will to be used in the learning phase. The results are also summarized in Table 5.1 and briefly discussed following:

- **Experiment 01, *Turned*** (Fig. 5.5(a)) - FSQ and RQ produced the same filters, selecting both shoulders and the torso. In YNQ the users also selected the head as an important limb.

- **Experiment 02, *Looking*** (Fig. 5.5(b)) Here FSQ were stricter, selecting only the head. On the other hand, users selected the same limbs in YNQ and RQ: head and neck.

- **Experiment 03, *Pointing*** (Fig. 5.5(c)) In FSQ most of the users answered "their arm/s" as response. These included user's both shoulders, elbows and hands. In the case of YNQ and RQ, however, the users selected only their both hands and elbows.

When analysing the user's answers to FSQ, we realized that sometimes their answers were too broad. This was the case of the answers to experiment 3, in which most users answered with "*my arm*" or "*my arms*". There is not a direct mapping between "*arm*" and any of the OpenNI joints, hence, we had to make a decision on how to interpret these answers. Moreover, no user indicated which arm was referring when they answered "*my arm*". In this case we did not know if she was referring to her left or right arm. Since different users pointed with different arms and, since we wanted to be conservative in the mapping, we decided that all the uncertain answers that covered different possible limbs will be mapped including all of these limbs. For instance, in the case of the "*my arm*" answer, we included both left and right arms. We applied the same principle to decide



(a) User Answers In Experiment 01: *Turned Left, Front, Right*

(b) User Answers In Experiment 02: *Looking Left, Front, Right*

(c) User Answers In Experiment 03: *Pointing Left, Front, Right*

Figure 5.5: User answers for experiments 01, 02 and 03. The limbs that are selected are painted in orange.

which OpenNI joint was mapped to the "*my arm*" answer. Hence we mapped "arm" to the joints hand, elbow and shoulder.

YNQ and RQ produced nearly the same filters, which might occurred because, contrarily to the FSQs, the users had a list of limbs in front of them when choosing their answers. This might have allowed them to think more carefully about their answers than in the FSQ. But, if observed in detail, although the graphs of YNQ and RQ had almost the same shape, there is a slight difference in the limbs that were not selected. In RQ, these limbs tended to be closer to the threshold than YNQ. For instance, this effect is clearly shown in Fig. 5.5(b). In the figure both YNQ and RQ selected the Head and the Neck, and when looking to the unselected limbs, the shoulders and torso stood over the rest. However, while in YNQ their values were lower than in RQ, where they almost passed the threshold.

Therefore, there were differences between them. These differences might have been produced because, although the user had the list of limbs in both cases, in RQ they were forced to give an answer to each limb.

Two conclusions might be drawn from these effects. The first is: the more you force the users to think of their limbs, the most limbs are included in their answers. In other words, if you let them to decide which limbs are important, they provide more strict filters than when you ask them for particular limbs.

The second conclusion we can extract from this fact is that RQ and YNQ obtained the same filters because we were too strict selecting our threshold. Had been the threshold lower, RQ would have had more limbs included than YNQ. In that case RQ could have been considered as an extended version of YNQ.

This idea is what led us to create the Extended Filter (EF) which we presented in section 5.3.4.1. Since the user answers were, too simplistic, we decided that the robot should not blindly trust in her answers. However, since she were not completely wrong, we opted to extend her answers including more information than she actually gave.

5.5.2 Learning Results

We compared the AL performance when using the filters built from the FSQs, YNQs and RQs against a Passive Learner[1] (PL). Aditionally, we also compare the Extended Filter (EF) presented in section 5.3.4.1 against PL. Therefore, we consider PL as a reference metric to benchmark our AL approach.

These filters pre-process the data, removing all the parameters associated to the limbs that were filtered out. The remaining data is fed to three classifiers: a J48 [Quinlan, 1993], which is a modification of the C4.5 decision tree, a Random Forest [Breiman, 2001], and

[1] This is a classifier built without filtering any parameters

an SMO [Shevade et al., 2000]. We compared the results of each classifier separately, so we can reduce the likelihood of the results being caused by the algorithm itself.

The metric we used for our evaluation is the F-score:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{5.6}$$

This metric is a single value metric that indicates the accuracy of a learning system having into account both its precision and recall.

We evaluated the learning performance using a modification of the Leave-One-Out (LOO) evaluation method. A traditional LOO evaluation is similar to a Cross-Validation but with the difference that, in each fold it trains with all all the dataset except one instance. Once the system is trained, the left out instance is used for evaluation which produces a result of one or zero for success or failure respectively. This process is repeated with every instance of the dataset and then the evaluation results are averaged. Our modified LOO consists in, instead of leaving out one single instance, we remove all the instances that correspond to a single user. The rest of the process is kept the same as the traditional LOO. We call this method *user-based LOO* and is motivated to avoid over-optimistic estimations produced by including instances of the same user both in the training and test sets.

The results are shown in Fig. 5.6 and summarized following. Notice that, as described in section 5.5.1, YNQ and RQ produced the same filters[1]. Therefore, from now on, we only describe RQ results:

- **Experiment 01, *Turned*** (Fig 5.6(a)): This experiment produced the best learning results, achieving F-Scores of nearly 100% with just only 2 or 3 training users.

- **Experiment 02, *Looking*** (Fig 5.6(b)): In the looking experiment AL lead to worse results compared to PL. In general terms, FSQ and RQ performed worse than PL with the exception of the cases in which only one user trained the robot. Here, the EF performed better. Although its performace was lower than PL, it was closer to it than to FSQ and RQ. Even more, EF was significantly better than PL in the SMO.

  The reason of the lower performance of AL (FSQ and RQ) might be caused because the verbal cues may be useful at the beginning of the training process, when the robot has few examples to build a good model of the problem. In this case, introducing domain knowledge from the users compensate the lack of training examples. However, once the classifier gets more training examples, the training data becomes

[1] The only difference occurred in Experiment 01 (*Turned*), where YNQ added the Head to the RQ limbs. However, since in such experiment the results achieved scores of nearly 100% so quickly, we decided that the differences in such case were irrelevant.

more informative than the user's responses, which we believe that in this case were being too simplistic and leaded to worse results. The extended filter did include these joints and, therefore, was able to achieve better results.

- **Experiment 03, *Pointing*** (Fig 5.6(c)): In this experiment RQ and EF performed slightly better than FSQ and PL, specially when used Random Forests. In the case of SMO, these differences are less significant and they disappear with the J48.

  Despite the differences are not as clear as in experiment 2, in this experiment the algorithms also performed differently when they had few training examples than when they were trained with all the users. When only one user trained the system, two of the three algorithms, the J48 and the SMO, produced the best results in the AL approaches obtaining similar results. However, when they were trained with the 23 users, we did not observe differences between the AL and non-AL approaches.

  On the other hand, the Random Forests Algorithm behaved just in the opposite way. When trained with only one user, the AL approaches were significantly worse than the non-AL version. But with 23 training users, AL supposed a significant improvement over the non-AL.

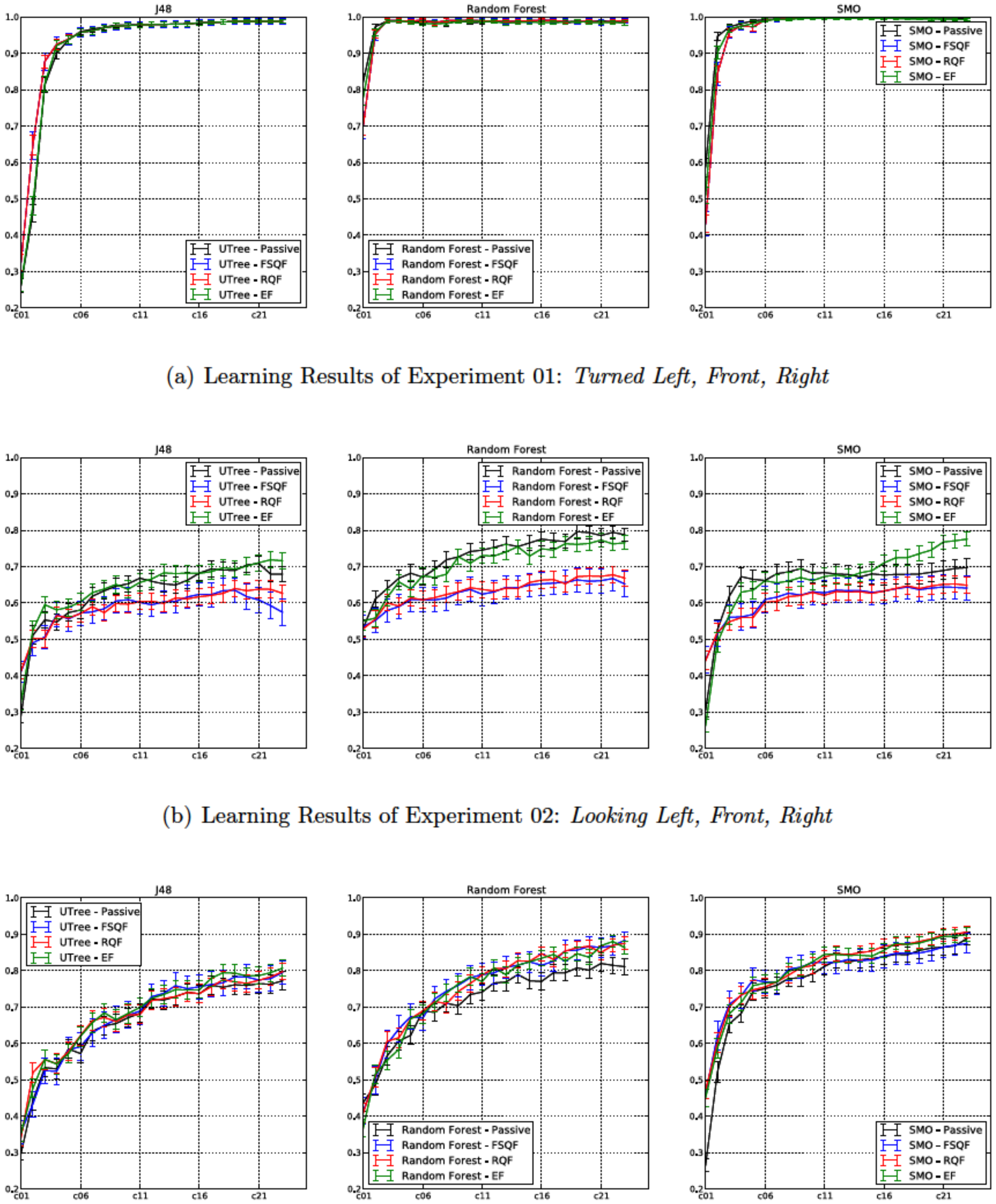
  When comparing the results between algorithms, SMO showed the best performance followed by Random Forest and J48.

5.6 Discussion

It seems that FSQ answers produced stricter filters (included less limbs) than YNQ and RQ. Although YNQ and RQ produced the same filters, it also seems that YNQ was slightly stricter than RQ. Therefore, if the robot needs to create more inclusive filters, it should prefer to ask Rank Queries over the others. However, each RQ is tailed to a single limb, which might produce a more verbose robot. Since users prefer not to be bombarded with many questions [Cakmak et al., 2010].

FSQs have the advantage of being more natural and they can be much more efficient than the other types of queries. In an interactive setting, YNQ and RQ would require the robot to go through all possible dimensions to have an unbiased estimate of relevance for all dimensions. Whereas with FSQ the robot just asks "*What is relevant?*" and the person can just directly mention the relevant ones and ignore the irrelevant ones, which would take much less time.

One way to mitigate the number of questions issued by YNQ and RQ might consist in combining feature selection algorithms with AL. For instance, the robot might ask the user YNQ or RQ for the features being chosen by the feature selection algorithm just to



(a) Learning Results of Experiment 01: *Turned Left, Front, Right*

(b) Learning Results of Experiment 02: *Looking Left, Front, Right*

(c) Learning Results of Experiment 03: *Pointing Left, Front, Right*

Figure 5.6: Learning Curves in experiments 01, 02 and 03. X-axis shows the number of users used for learning, while in the Y-axis is shown the F-score.

confirm or discard them. Also, to reduce the number of questions, these might be asked in batches. An example of such queries would be asking questions of the type "*Which of the following are relevant: X, Y, or Z?*". or, perhaps "*Do you thing that X, Y, and Z are relevant?*".

All this, leads to one aspect that has not been considered in our work, and that should be addressed in future work. That is, studying which is the cost of each question. For such purpose different metrics could be used depending of what are the requirements of the system: Time required to build a filter, number of questions required to build a filter, etc.

As we seen before, user's answers to YNQ and RQ ended up with the same filters. This has been caused because our method of choosing the selection threshold. To calculate it we used the mean relevance of the limbs plus one standard deviation. Our motivation was to differentiate the limbs that stood out of the rest. Perhaps choosing another threshold, for instance using only the mean, could have lead to producing different filters. Another aspect to consider is the method which we have followed to build the user-based filters. We followed an aggressive approach in which we removed all the parameters associated to a limb when the user considered that this limb was irrelevant. A more conservative approach might differentiate, for instance, between positions and orientations. However, as a side effect, asking too many questions to the user might lead to worsening the interaction [Cakmak et al., 2010].

As we have seen, in the second experiment (*looking*), AL got worse performance than PL. Our intuition leads us to think that this might be caused because the user's answers were too generic. We believe that, when they indicated that their heads and necks were important, they should have included their shoulders as well. It seems that when they turned their heads, they also slightly moved, imperceptibly, their shoulders. Therefore, when filtering the shoulders, the classifiers lost information that, in the end, resulted to be important.

The main conclusion that might be drawn from the experiment is that user verbal responses in AL can lead to filtering data of the parameter space which might be actually relevant for learning. When this happens, AL can produce worse results than PL. Even more, it might not be easy for the system to know in which cases the user's answers would lead to a decrease in the algorithm performance. To solve this problem we developed the notion of the Extended Filter (EF). This filter achieved a performance similar or better than other AL filters in the situations where AL did not worked as well as expected since EF included those limbs that might have been relevant. What is more interesting, when AL did performed better than PL, the EF behaved as a regular AL approach. Therefore, our EF gets the best of the two worlds, being a good choice to avoid the pitfalls we commented before.

Perhaps the reason EF behaves in this way is because it can be considered as a filter in which the robot trusts less in the user's answers. The EF feeds the learning algorithm with more data than the user would do, but at the same time, it filters the data which is likely to be irrelevant for the learning problem. In other words, when using an EF, the robot does not take the user response as literal and accurate as other AL approaches, by generalizing the user's answer to other parameters.

We believe that the EF concept can be exported to other fields than pose learning. This is possible because the main notion of the EF is to include in the filter not only the features from the user's feedback, but also the ones which are closely related to them. In the case of pose learning, this relation is spatial, i.e. the joints which are closer are more likely to move together. To implement the EF in other fields, the designer of the learning scheme must be aware of the relationships between parameters just to build the extension to the user-selected parameters

Another way of extending the user filter is lowering the selection threshold of the user answers. As we have seen, when many users answer questions, a long tail of unselected answers appears. The higher is the standard deviation of the user answers, the more likely will be that lowering the threshold will produce an extended filter. This is, if the user answers are variate, it is more likely that lowering the threshold will include other parameters.

5.7 Conclusions

The main contribution of this chapter is two-fold. First, we evaluated how different types of Feature Queries affect the learning performance of a robot that learns actively. We found that, in some cases, user's answers might be too simplistic leading to a reduction in the robot's learning accuracy. In such cases, if the robot trusts too much in the user's responses, Active Learning approaches might be outperformed by Passive Learning. The second contribution of this chapter is a method in which the robot reduces its confidence on the user's answers by extending them to other related parameters of the learning space. This method has proven to keep the learning performance high even in the cases where users did not provide accurate answers.

We tested our approach in an experiment where 24 users trained a social robot to recognise poses. In the experiment, they were asked for three types of Feature Queries: Free Speech Queries (FSQ), Yes/No Queries (YNQ), and Rank Queries (RQ). The answers to these queries were used to build feature filters that pre-processed the training data before it was fed to the learning algorithm of the robot. We found that FSQ produced less-inclusive filters than YNQ and RQ leading to simpler answers. We tested our method by reducing the robot's confidence on the answers to RQ. In essence, it consisted in building a filter which was an extension to the RQ-based filter. Our Extended Filter outperformed other AL-based filters when the user's provided inaccurate answers, and kept a similar performance compared to pure AL-based filters when the users provided good answers

Even though our method is applied for pose learning, we believe that it can be applied to other AL-based learning approaches. This is because our approach is based on feature selection, Hence, other learning approaches can apply it as long as the robot knows the relationship between the features it is asking for. Knowing these relationships, the robot might extend the user answers to other related features, thus reducing the effects of her inaccuracies. Our method might help robots to learn better from people who are not expert in robotics. If the robot expects that the user answer might not be accurate, it can choose to apply our Extended Filter instead of discarding it.

As a future work, it remains how to apply this method when different types of feature queries are combined. Because FSQ produce simpler answers than YNQ and RQ, it might be interesting to apply different filters to each type of questions. In the cases in which the robot expects inaccurate answers, it can increase the filter extension just to reduce the side-effects of these inaccuracies. Besides, we want to explore the use of automatic feature selection algorithms together with our extended filter. For instance, if the user answer and the automatic feature selection differ too much, it might mean whether the user gave a bad answer or the robot does not have enough training data. In such cases, it might be wise to extend the user's answer and/or ask more queries.

Chapter 6

Active Learning for in-hand Object Recognition

This chapter extends the object detection and tracking system presented in Chapter 4 by improving the robot's interaction and active learning capabilities. In this chapter, the robot uses active learning techniques that allows it to take the learning initiative after the training has finished. We also improved the robot's interaction capabilities so it to establish more complex interactions with the user, giving him/her the freedom to deliver the same message in many different ways. This new interaction system consists of a Dialog Management System that controls the interaction flow, an open-grammar Automatic Speech Recognition (ASR) system, and some Natural Language Processing (NLP) modules that enable the user to tag virtually any object without the need of precoding its label in a grammar. We demonstrate the viability of our system in an experiment where the robot is trained with 4 objects. In our experiment we evaluated the number of questions the robot asks to the user when it sees the objects of a test set. Our results show that the robot is able to learn continuously, but that the degree of initiative has to be controlled if we do not want to saturate the user with many questions.

6.1 Introduction

The aim of this chapter is to extend the work in Chapter 4 by adding to the OCULAR system rich interaction capabilities and to improve the robot's learning using active learning. The purpose of using active learning is not only to increase the robot's learning speed, but also to enable it to learn proactively once the learning session has finished. The idea of this chapter is to enable the robot to understand that, during the exploitation phase, when it observes an object whose label is uncertain to its learned model, the robot can ask a human for such label.

The novelty of our approach is not in the active learning aspects, but in how is combined with the interaction that it is carried out during learning. Our objective in this chapter is to integrate a Dialogue Management System (DMS) with a rich Automatic Speech Recognition (ASR) so a user can user who is teaching a robot can use a wider range of expressions when addressing to it.

6.1.1 Objectives

The objectives of this chapter can be summarized here:

1. To enable rich dialogues between the user and the robot during learning.

2. To enable the user to interact with the robot in a natural way, relaxing the limitations of a small set of vocabulary.

3. To enable the user to give the robot any object name. That is, without having to pre-write a finite number of object names in a grammar.

4. To let the robot to take the initiative of its own learning after the first traning session has ended.

6.1.2 Organization of the chapter

The remainder of this chapter is as follows. Section 6.2 gives an overview of the system by, first, detailing the differences with the system presented in Chapter 4 and, second, stressing the interaction and the active learning modules that are presented in this chapter. Section 6.3 describes an experiment where we evaluate our approach. The results of our experiment are discussed in section 6.4. Finally, in section 6.5, are presented some concluding remarks together with some future lines of work.

6.2 System Description

This section describes the description of the OCULAR system with the addition of the interaction and the active learning components that enable the robot to actively learn objects that are held in the user's hand. Figure 6.1 shows a "black-box" overview of the system with these additions. In short, our active in-hand object recognition system is composed of three modules: OCULAR, which it has been already described in Chapter 4, an Interaction Manager (IM) module, and an Active Learning (AL) module. Describing the former two is the aim of this chapter.

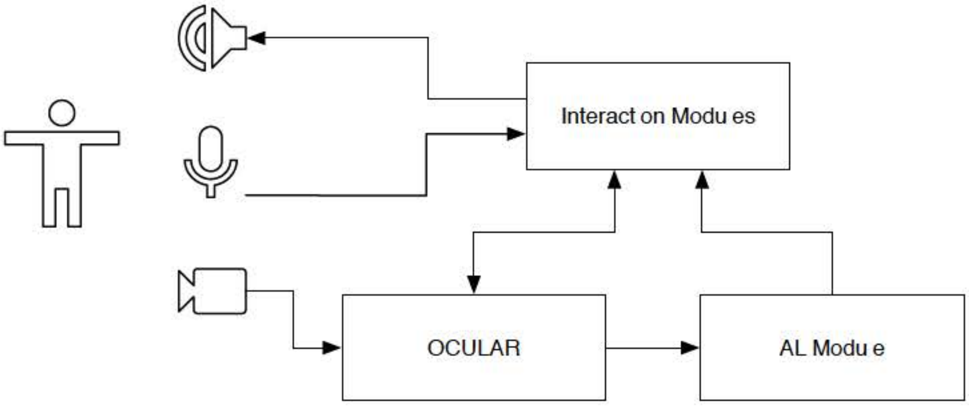


Figure 6.1: Black box view of the active in-hand object recognition system. The figure shows the three main modules of the system and which are the inputs and outputs with the user.

The Interaction Manager is a Dialog Manager that controls the interaction flow and enables the communication with the user. It can be considered as the user interface of the system that aims to provide a natural way of communicating with the robot. The Active Learning module is the component of the system that receives the predictions from OCULAR and decides until when it is necessary to keep learning and which queries might provide more help to accelerate such process.

An example of the system working might be the following situation. First, a user that wants the robot to learn a new object. To do so, he/she tells the robot by voice that he/she wants to teach a new object. The robot accepts and asks the name of the new object, which is then told by the user, again by voice. After the robot gets the object name, it tells the user to stretch his or her arm closer to it so it can see the object properly. Once the user does it, the learning process starts and the robot starts recording views of the object as described in chapter 4. When the robot has acquired some views of the object, it compares these views against its previous knowledge. It does so by comparing what it is currently seeing, the new learnt object, against other objects that has previously learned. If the robot has some troubles discerning between the new object and the previous ones[1],

[1] As we will see later in this chapter, These "troubles" can be modelled by some of the metrics described in chapter 2.

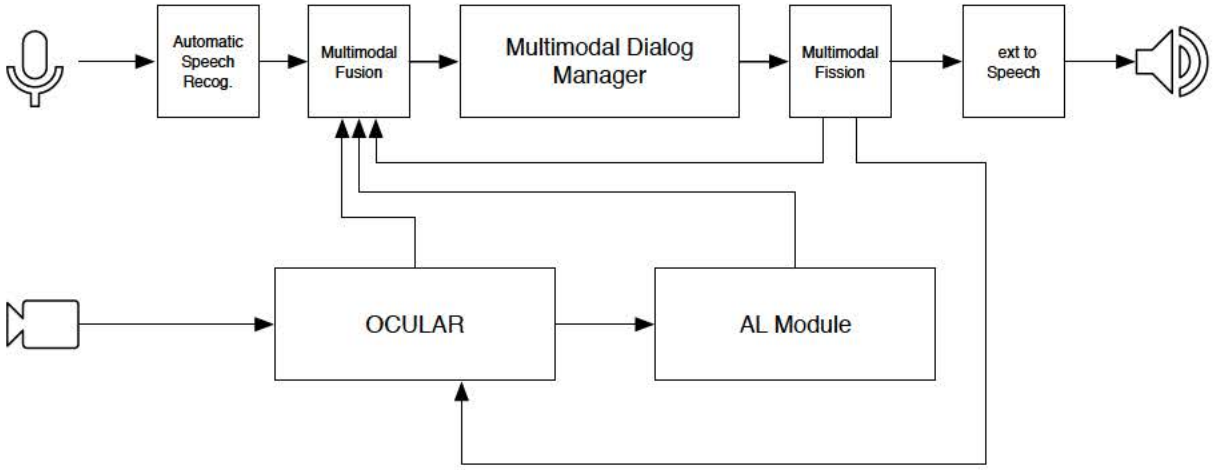


Figure 6.2: White box view of the active in-hand object recognition system The main interaction components are, from left to right, the Automatic Speech Recognition (ASR) System for understanding what the user says; the Multimodal Fusion Module, which aggregates information from different sources; the Dialogue Manager (DM), which controls the interaction process; the multimodal fission module, which controls the outputs of the DM; and the Text To Speech (TTS) module that synthesises the robot's voice.

it then decides to ask for more views of the object or, if the robot is confusing this object with another in particular, it can ask for more examples of the other object. The learning of this object is concluded when the robot reaches a certain threshold in some metrics such as accuracy or F-score.

Figure 6.2 shows the white box view of the object recognition architecture. There, the different modules that compose the Interaction Modules box are detailed. From left to right, these modules are the Automatic Speech Recognition (ASR) Module, The Multimodal Fusion, the Multimodal Dialog Manager, The Multimodal Fission Module and the Text To Speech Module (TTS).

6.2.1 Automatic Speech Recognition Module

The ASR is similar to the one described in section 3.3.4.2 but with the only difference that the ASR used here is based on an open vocabulary model rather than being grammar-based. The reason for using an open-language model instead of a grammar-based as it is done in Chapters 3 and 5 is the following: in a grammar-based model, the grammars need to be pre-coded, which may constrain the robot's vocabulary and limit the number of objects it can learn to what the grammar programmer initially considered. Conversely, an open-language model does not constrain the vocabulary of the robot, so the user can virtually speak any word present in the language that he/she is speaking. This is beneficial for our learning set-up, in which the user might want to teach objects that are unknown to the robot. Hence, when the robot asks for the name of the object, it needs to use an ASR that supports recognizing words that might have not been pre-coded in a grammar. However, an open-language model requires a post-processing step to extract useful information from raw text. In our system, these post-processing steps are carried out in the Multimodal Fusion Module.

6.2.2 Multimodal Fusion Module

The Multimodal Fusion module is in charge of aggregating data from different sources, and of preparing these aggregated data to a format that is manageable by the Multimodal Dialog Manager. Its functionalities are achieved by two layers of ROS nodes composed of information aggregators and data translators, respectively. The aggregators produce high level information from lower-level data such as raw data coming from sensors. An example of an aggregator can be a node that gets RAW data from the Kinect and monitors whether the user's arms are in a certain position or range of positions (e.g pointing to some direction).

The object recognition system presented in this chapter uses several aggregators. Two of them are worth being mentioned here since they process the ASR raw data to produce higher level information that is required by the Dialogue Manager. The first aggregator, which we call *user_ command_ aggregator*, seeks for user commands such as *start*, *stop learning*, etc. The second, *object_ name_ extractor*, extracts the object name when the user tells it. Both produce higher level information from the ASR output by using Part-Of-Speech (POS) tagging together with other Natural Language Processing techniques.

The *user_ command_ aggregator* uses POS to extract the verbs from the ASR output and conjugate them to their infinitive form. Then, it checks if any of the extracted verbs is present in a command database. For instance, a user can tell the robot "*Would you like to learn a new object?*"[1] from which the aggregator would extract "*learn*" as the command. If the *learn* command is present in the database, then the aggregator sends it to the Dialogue Manager. The *user_ command_ aggregator* uses the same POS tagger, but configured to extract the nouns from the ASR output instead the verbs. The first noun found in an ASR transcription is then relayed to the Dialogue Manager, which will use it to name the object being learned. For instance, when the robot is learning an object and it asks for its name, the Dialogue Manager will receive the noun naming the object from the user's answer. For example, from the sentence "*This is a ball*", the aggregator produces *ball* as its output.

Both aggregators have been implemented using the *Pattern* Toolkit [Smedt and Daelemans, 2012], since it supports the required NLP techniques described before not only in English, but also in Spanish. *Pattern*'s Spanish POS parser has been trained on the Spanish portion of Wikicorpus [Reese et al., 2010] using 1.5M words and it has an accuracy of

[1] In the real set-up the interaction is in Spanish

around 92%. For Spanish verb conjugation, it uses a lexicon of around 600 common Spanish verbs composed by Fred Jehle[1]. For unknown verbs it will fall back to a rule-based approach with an accuracy of about 84%.

The translators receive the data from the aggregators or directly from raw sensors and translate them to a format which is compatible with the Dialog Manager's data format. The need of translators comes, mainly, from Software Engineering reasons. To avoid entering in too much detail, and to simplify the architecture description, the translators will be omitted from here on[2].

6.2.3 Multimodal Dialogue Manager

The *Multimodal Dialogue Manager* (MDM[3]) is the component in charge of coordinating the execution of the learning and training phases by interacting with the user. Our dialogue manager is a fork of the *iwaki* dialogue manager[4], a rule-based production system, which is inspired in COLLAGEN plan trees [Rich et al., 2001]. The rules of the production system, which are called recipes, are written in XML files and enable interactions with aspects of finite-state, frame, and information-state-based dialogue management.

The MDM processes the data received from Multimodal Fusion layer and decides what action should be executed. For instance, imagine that a user asks the robot to start the learning process. This is captured by the ASR, which translates the user's sentence to text that it is interpreted by an aggregator as a command to start learning. This input is received by the MDM and decides to start the learning process. This activate a dialogue (also called recipe) which has been pre-coded as a plan in an XML file. For instance, the plan can start by the MDM sending a command to the TTS to tell the user that it is accepting his/her command. At the same time, the MDM can send a command to the learning node to start the learning process. The plan (the dialogue) can include the MDM asking several questions to the user (e.g. the object name) and handling several responses or requests from the user.

6.2.4 Multimodal Fission Module

The output of the MDM pass through the Multimodal Fission Module, whose functionality is symmetric to the Multimodal Fusion. That is, the Fission module enables the connection between the MDM and external modules but, in this case, these external modules

[1] Jehle, F. (2012). Spanish Verb Forms: http://users.ipfw.edu/jehle/verblist.htm

[2] In short, the translators mission is to abstract the Dialog Manager from the implementation details and the diverse data formats that it might receive from its many different data sources. In this way, the translators act as an abstraction layer that makes the Dialog Manager independent on the information types of the different data sources that can feed the Dialog Manager.

[3] We will also refer to the MDM as Dialogue Manager (DM)

[4] https://github.com/UC3MSocialRobots/iwaki-ros-pkg

receive the information and commands from the MDM. An example can be sending a text string to the TTS, which is in charge of converting it to the spoken audio wave. As the Fusion, the Fission Modules are composed of information translators, which adapt the data format from the MDM to the output modules that receive these data.

6.2.5 Text To Speech Module

The TTS Module is similar to the TTS already described in section 3.3.4.1. It consists in a ROS wrapper of the Loquendo TTS that receives strings containing the text to be synthesized. Although the engine supports different emotion configurations, in this case we use the neutral set-up.

6.2.6 OCULAR Module

The OCULAR module has already been presented in Chapter 4. As a brief reminder, its main function is to learn and recognize the objects the user is holding in his/her hand by analysing both RGB images and Point Clouds.

In this chapter OCULAR has been slightly modified to communicate with the interaction modules presented above. The main difference here is that its `event_handler` and the `learner_recognizer` nodes are not connected directly anymore. Instead, this connection is redirected through the Interaction Manager modules. Concretely, the `event_handler` node output is redirected to the Multimodal Fusion, and the `learner_recognizer` node receives the *learning* and *recognizing* inputs from the Multimodal Fission Module. This enables a better control of the interaction flow since it is now controlled by the Interaction Manager. In short, now is the Interaction Manager who decides when to start the learning and exploitation phases.

6.2.7 The Active Learning Module

The Active Learning Module of the system calculates the amount of uncertainty of a given instance. In case the uncertainty of an instance is above certain parameters, the robot can ask for the label of such instance.

The Active Learning (AL) Module enables the system to use the active learning algorithms described in Chapter 2. This module receives 30 predictions per second from the OCULAR Module and analyses the uncertanty of such predictions using two metrics: first, the Margin metric (as described in section 2.2), which it is the difference between the two most likely predictions on an instance; and, second, the Jensen-Shannon Divergence (JSD), which is similar to the Kullback-Liebler (KL) Divergence shown in equation 2.7 from section 2.3.3. The main differences are that the JSD is symmetric and their values ar bounded between 0 and 1. This is an advantage since we can use this value directly as the probability of asking a question.

In our case, the committee that computes the JSD consists in only two members: the RGB Matcher and the Point Cloud Matcher described in Chapter 4. When the predictions of these two matchers differ, the robot will ask the user for the true label of the instance being observed. However, the decision is not made by the AL module, but by the Dialogue Manager, who uses the uncertainty calculated in the AL module to evaluate whether the robot should ask the user or not.

6.2.8 System Working Modes

The main working modes of the system are the *training* and the *recognition* modes. Those are equivalent to the *training* and *exploitation* phases in a regular machine learning setup. Following we present a working example of the system that explains the interaction steps that occur to change between modes.

1. First, in the *training phase*, the user trains the system using natural interaction. The training starts when the user tells the robot that he/she wants it to teach it a new series of objects. The learning session consists in the user showing different objects to the robot while telling it their names. In short, the robot asks the user, for the label of the next object that is going to learn. After the user tells the object's name, the robot asks him/her to extend his/her arm towards it so it can "see" the object. Once it detects that the user has extended his/her arm, it stores a set of views of the object together with its label in the training dataset. Then it tells the user that it has already recorded the object and that it is prepared for more objects. At this point the user can tell the robot whether that he/she wants to keep teaching more objects or that the learning session has finished. In that case, the system enters in the exploitation phase.

2. In the *exploitation phase*, the user can start showing several objects to the robot. The robot, using its learned model, estimates the object *id* and tells it to the user.

3. If, during the *exploitation phase*, the robot is uncertain of the *id* of an object, it issues a question to the user asking for the label of that object. If the user answers, the robot updates the training dataset with the new view of the object and its associated label. After this training update, the system enters again the regular *exploitation phase.*

This approach proposes a mixed initiative that enables the robot to continuously learn from its observations. First, in the initial training the robot yields the initiative to the user. But, once its training has bootstrapped, it takes the initiative of its own learning by asking the questions it considers appropriate every moment. If the robot is uncertain of what it sees, it means that this is a possible opportunity to expand its knowledge base (that is, its training data set).

6.3 Experiment

We designed an experiment to demonstrate that our system is able to learn continuously after the system has been trained. In the experiment the robot has to take the learning initiative after the training session has finished. To do so, we train the robot with a set of objects and later we evaluate the number of questions it asks on a test set. This will allow us to understand two things: first, if the system is capable of having the initiative to learn and, second, how verbose the system will be.

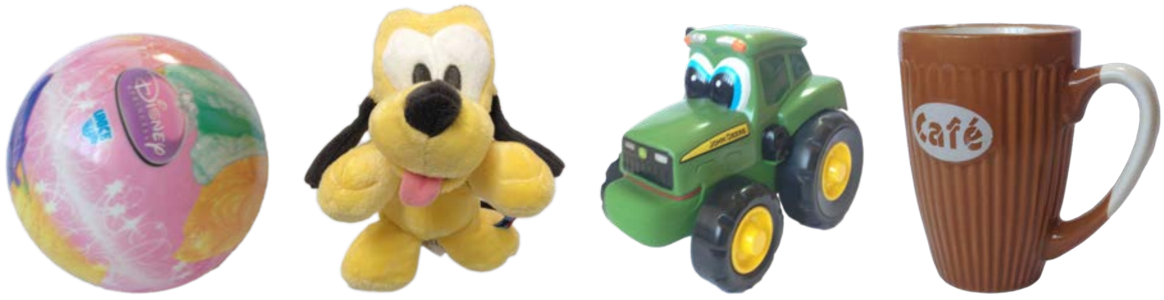
6.3.1 Scenario Description

The user stood in front of the robot at a distance of $1.5m$ approximately in a room with a combination of artificial light and natural light. The natural light came from a side of the room while the artificial light was uniform and came from the ceiling. The robot was equipped with a Kinect for visual processing (both RGB and depth data), and with a microphone to capture the user's speech. It was also equipped with a pair of speakers to talk with the user. All the systems are the same as described in section 6.2.

6.3.2 Method

The experiment starts when the user tells the robot that he/she wants to teach it some objects. Then, the robot agrees, and asks the user for the name of the first object. The user can use a relatively short sentence to describe the object such as "*This is a car*" or "*I'm going to show you a car*". In short, any sentence that contains a noun is processed by the robot. If this occurs, the robot responds to the user telling him/her the name of the object: "*Ok, you're going to show me a car*". Following, it continues by asking him/her to show the object by extending his/her arm. When the robot sees the object, it tells the user "*Ok, I see it. Wait a second while I'm capturing it*". Once the robot has recorded the object, it tells it to the user and asks him/her if he/she has more objects to teach it. If the answer is positive, the process is repeated again. Otherwise, the learning phase finishes.

During the whole process, all the communication between the human and the robot is carried out verbally. To minimize communication errors, we implemented several fallback



(a) Ball (b) Dog (c) Tractor (d) Cup

Figure 6.3: Objects shown to the robot.

dialogues to the Dialogue Manager to handle the situations where the ASR did not recognize the user speech. For instance, some of these fallback dialogues consisted in asking the user to repeat a sentence, or asking for confirmation of what the robot understood from the user.

The trainer showed 4 objects depicted in Figure 6.3. While showing them, the trainer tried to oclude them with his/her hands the less as possible. We obtained 3 training datasets each one consisting of 1, 10, and 20 views per object respectively. The test dataset was recorded in the exploitation phase, showing the robot the same objects. Our test data consisted in recording the uncertainty levels of each prediction as well as the number of questions the robot asked while observing the test data.

6.3.3 Results

This section summarizes the results of the experiment. The results are divided in 3 different aspects: learning performance (Figure 6.4), uncertainty scores (Figure 6.5), and number of questions asked to the user in the test set (Figure 6.6). All of them are evaluated against the tree datasets of 1, 10, and 20 training instances.

Learning Results. The learning results are depicted in Figure 6.4. Here, both matchers obtained similar results for 1, and 10 training examples. But they showed a significant different behavior with 20 training examples. Here, while the RGB Matcher obtained a 0.75 F1-Score, the Point Cloud scored a 0.26 F1-Score. We discuss the reasons for the lower score of the Point Cloud matcher in section 6.4. Despite the low results of the Point Cloud Matcher, we observe that the combination of both matchers lead to an increase in the F1-Score up to a 0.82.

Uncertainty levels of the robot. In our experiment we measured two uncertainty metrics: the Jensen-Shannon Divergence (JSD) between matchers' predictions and the margin between the two most likely predictions.

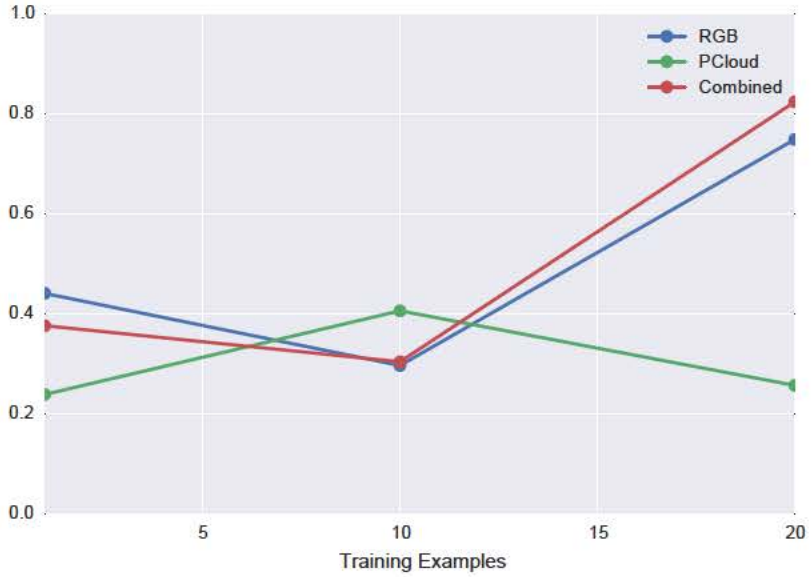


Figure 6.4: F1 Score when the system is trained with 1, 10, and 20 views per object.

The JSD results are depicted in Figure 6.5(a). Its values lay between 0.41 with 1 single training instance, to 0.47 with 20 training instances. The reason for this low variance might be produced because of the different performance results of each matcher.

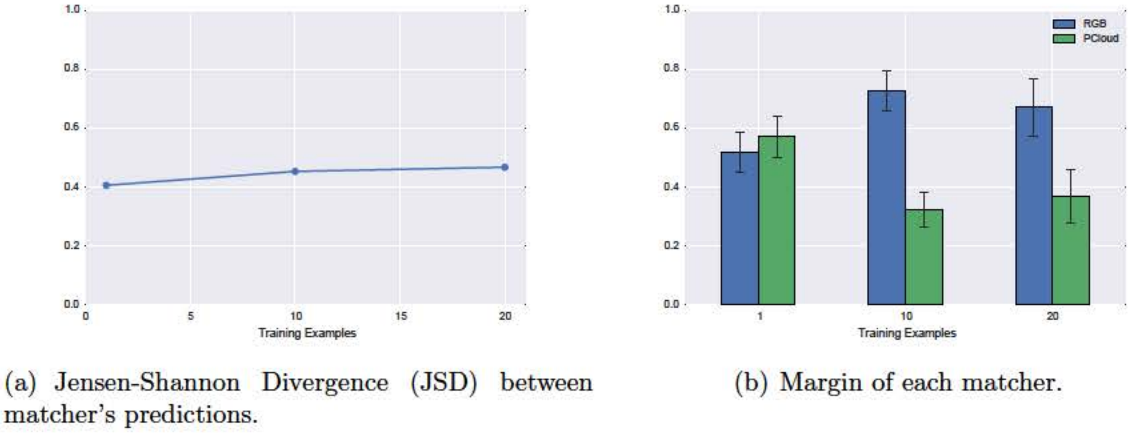
The Margin results (Figure 6.5(b)) are considered for each matcher separately and are detailed in Table 6.1. In the case of the RGB Matcher, the values move between 0.52 for 1 training instance to a 0.67 in the case of 20 training instances. It seems that adding more data increased the margin, producing more certain predictions.

This did not happen in the Point Cloud Matcher. In this case, our results seem to indicate that adding more training data increased the levels of uncertainty of the model (thus lowering the margin). One reason for this fact might be the similarity of the point clouds of some objects. For instance, in some views, the point clouds of the *tractor* and the *dog* were vaguely familiar: the legs of the dog resembled the wheels of the tractor, and the dog's snout might have been confused with the tractors' hood.

Verbosity of the robot. The last metric of our experiment is related to the amount of questions asked by the robot while observing the test dataset. Figure 6.6 shows the

Training Examples	**1**	**10**	**20**
RGB	0.52	0.73	0.67
PCloud	0.57	0.32	0.37

Table 6.1: Margin Results



(a) Jensen-Shannon Divergence (JSD) between matcher's predictions.

(b) Margin of each matcher.

Figure 6.5: Uncertainty levels of the robot in the test set

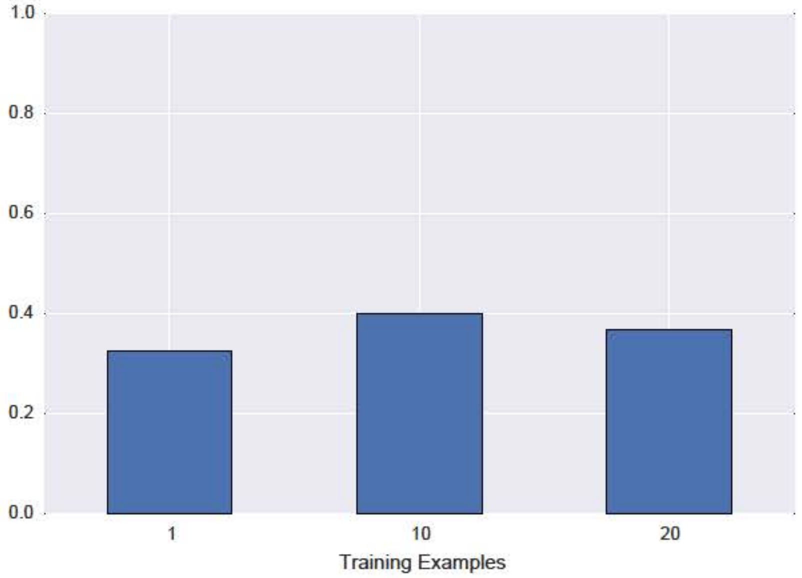


Figure 6.6: Ratio of questions asked to the user in the training set.

ratio of questions asked by the robot with respect to the number of instance in each test dataset. The numbers vary from a 33% when the robot was trained with a single instance to a 37% when the training consisted in 20 examples.

The decision of asking was carried out evaluating the JSD of a prediction against a threshold. If the instance produced a JSD >= 0.5 then the robot asked to the user for the label of the object. Since the matchers diverged in their predictions (see Figure 6.5(a)), the robot behaved in a very verbose manner, asking a great number of questions to the user. In the discussion section we offer some alternatives to have a less verbose robot.

6.4 Discussion

The results of our experiment demonstrate that our system is able to detect uncertain instances and incorporate them after asking for their label. However, if the training of the robot has not been thorough, it might be very verbose asking for almost each new instance that it observes in the exploitation phase.

Besides, if one of the matchers does not work properly against some data, then the divergence between matcher's predictions grows and the robot becomes more verbose, even if the combined prediction is good. This is what it might have happened with our dataset. When the robot was trained with 20 examples, the RGB matcher managed to get fairly better results compared to the Point Cloud Matcher. This lead to big divergence between matchers and ended up with the robot asking too many questions.

The amount of questions asked by the robot caused a saturation effect in the user who, in some cases, it was asked several times for the same object. This produced a negative effect since, if while the user is holding the same object, the robot should remember its label (and thus avoid asking the user for more labels of the same object). Some ideas to mitigate this effects are:

1. Limit the frequency at which the robot asks questions. That is, after having asked a question, discard the other questions it might have until a certain period has passed.

2. Wait until the user takes another object to ask again. That is, if the robot has asked for a certain object label, wait until the robot is sure that the user is holding a different object before ask again.

From the engineering point of view, the first method is easier, although it might present some problems like the one described above. The second method seems more elegant, but presents the problem of knowing that the user has left the object that was asked. It is possible to do some hand tracking, but understanding that the user has left the object, for instance in a table, supposes a much harder problem to solve.

From the communication perspective, our system has proven to be robust and flexible. In essence, the user only needs to know a certain number of commands to control the robot, but the system lets the user the freedom to tell the commands in many different ways. The same happens when the robot asks for a new object name. Our open grammar model allows to process an infinite number of names. The only condition the user needs to know is that, when the robot asks for the object name, the user has to answer the robot with any sentence he/she wants but providing that a noun is in the sentence (which will be used as the label of the object).

6.5 Conclusions

This chapter has extended the object detection and tracking system presented in Chapter 4 with improved interaction and active learning capabilities. The active learning has been used to enable the robot to take the learning initiative once the training session has finished. That is, our approach allows the robot to ask the user for the labels of those instances who are uncertain to the trained model even after the training has ended.

In this chapter, the interaction capabilities of the system have also been improved. Mainly, the system has been extended by including a Dialog Management System that allows rich and complex interactions between humans and robots. Besides, the interaction system incorporated an Open-Grammar Automatic Speech Recognition (ASR). This, together with the Natural Language Processing (NLP) modules that we have developed, enables the robot to engage the user in rich interactions, giving him/her the freedom to communicate in many different ways. Additionally, our system allows the user to virtually label any object he/she wants, without having the need of pre-coding the object names in a grammar.

Our system has been tested in an experiment where, after the robot was trained with 4 different objects, the robot had to ask for the labels of the uncertain objects. The results of our experiment demonstrated that the robot can have the initiative to ask for labels of uncertain objects and can incorporate these new information into its training dataset. However, our system has proven to be too verbose, asking too many questions to the users. Therefore, we provided some indications of how mitigate this effect.

Future lines of work involve adding more matchers to the system and evaluate how the system behaves when these matchers behave as a committee that evaluate the uncertainty of new objects. Also, it would be interesting to evaluate how the user perceives a robot that it has the initiative to learn continuously. For that purpose, it might be interesting to study which methods could be used to control the number of questions the robot asks to the user.

Chapter 7

Novelty Detection for Interactive Pose Recognition

This chapter addresses the problem of Active Learning for pose learning from another perspective. The approach here is to enable the robot to learn continuously from what it observes. That is, while in previous chapters the robot learns during the learning session, here we study the mechanisms that enable it to ask questions to the user during the exploitation mode. The objective is that the robot should know if it needs more training data—either in form of new examples or labels—and, hence, ask for them to the user. The contents of this chapter have been published in [Gonzalez-Pacheco et al., 2014] which is an extension of [Gonzalez-Pacheco et al., 2015].

Active robot learners take an active role in their own learning by asking queries to their human teachers when they receive new data. However, not every input is useful for the robot, and asking for non-informative inputs or asking too many questions might worsen the user's perception of the robot. We present a novelty detection system that enables a robot to ask for a label of new stimulus only when it seem both novel and interesting. Our system separates the decision process in two steps: first, it discriminates novel from known stimuli and second, it estimates if these stimuli are likely to happen again. Additionally, our approach uses the notion of curiosity*, which controls the eagerness in which the robot asks questions to the user. We evaluate our approach in the domain of pose learning by training our robot with a set of pointing poses. Our results show that the system is able to detect up to an 84%, 79%, and 78% of the observed novelties in three different experiments. Our approach enables robots to keep learning continuously, even after the training session has finished. This is controlled by the curiosity parameter, which tunes the conditions in which the robot asks more training data to the user.*

7.1 Introduction

In *active learning* (AL), the robot is an active actor during the learning process. Since the robot can understand and decide what is relevant and what is unknown, it can be intrinsically motivated to explore the areas of the learning space that it does not know and ask the user about them so it can add these new data to its knowledge base. The main benefits of active learning are that active robot learners are capable of learning faster and better, and are also perceived as more intelligent by humans [Cakmak et al., 2010].

However, once the training has finished, the robot does not receive more training unless the human teacher decides so. This might limit the robot performance since after the training is done, there are many situations that the robot might face that were not predicted by the trainer. One solution is to allow a robot to actively participate in its own learning process by, for instance, letting it to ask questions to the user when it perceives any new stimulus.

Notwithstanding, the stimuli perceived by the robot can be either noise or be worth learning. If the robot asks the user for the label of each new stimulus, the user might end up considering the robot as not intelligent or annoying. For this reason, it is important that the robot should ask the minimum questions as possible and make them as most interesting as possible. In other words, the robot should ask for the most interesting novelties.

We present a system that allows a robot to detect when a stimulus is new and to decide whether it is worth learning it or not by assuming that interesting novelties tend to accumulate forming clusters. When the robot tags new data as strange and interesting, the robot can actively ask the user for a label that describes these data. The decision process is carried out in two steps. First, our system is able to detect if a new visual stimulus is different to anything seen before; and second, it is able to understand whether this stimulus is worth learning it. If both conditions apply, the robot asks the user for the label of the stimulus. Otherwise it discards it.

We validate our approach in the field of pose detection and learning, where a set of users trains a robot to detect certain pointing poses. In our evaluation we train the system with a set of poses and test it by showing new poses that it has never seen before. Our system is able to detect new poses and to differentiate them from noisy instances by incorporating a curiosity, which modifies the behaviour of the novelty detection algorithms to make them more or less sensitive depending on the robot needs.

The remainder of this chapter is as follows: section 7.2 gives an overview of the concepts involved in novelty detection that are used in the rest of the chapter. Section 7.3 summarizes in an example the problem we are addressing in this chapter and introduces some definitions that establish a common language used in our approach. Following, section 7.3.1 presents the related work and shows why our approach is novel and different from others. After that, section 7.5 describes our approach, which is evaluated in section 7.6. Finally, we conclude this chapter in section 7.7 summarizing our contribution and giving some insights of future work.

7.2 Novelty Detection

Our approach uses Novelty Detection techniques [A.F. Pimentel et al., 2014; Chandola et al., 2009]. Novelty detection is defined as "detecting previously unobserved (emergent, novel) patterns in the data", and incorporating these novel patterns into the normal model afterwards [Chandola et al., 2009]. In essence, novelty detection is applied in classification problems where new data differs in some aspects from the data available during training [A.F. Pimentel et al., 2014].

Novelty detection tries to detect (or identify) *abnormal data*. That is, data that differs from the data in the training dataset. It has become very popular in applications with the need of identifying abnormal behavior such as failure detection in industrial systems, or mass-like structures in mammograms [A.F. Pimentel et al., 2014]. All these applications have in common that their complexity leads to a limited understanding of a direct identification between a cause and a consequence of what is *normal* and *abnormal*.

One of the main characteristic of novelty detection is that the set of *abnormal* examples is under-sampled if compared with the *normal* set. Also, there is a large number of possible *abnormal* and *normal* modes. In some of the applications there is a high cost of obtaining examples of *abnormal behavior*, for example in industrial damage, to obtain a new abnormal instance one machine has to be broken on purpose, with its associated cost. This results in typical Machine Learning classification methods not being suited for detecting novelties, mainly because there are not enough examples to create a *normal* class and an *abnormal* class, and because there could be multiple unidentified *abnormal* classes.

To deal with these issues, Novelty Detection techniques treats the classification problem as a one-class binary classification. That is, the knowledge base of the robot is considered as only one class: the known or *normal* class. All the other data that the robot observes, might belong to the normal class (if it is known) or to the abnormal (unknown) class.

Therefore, the formal approach in Novelty Detection is to create a large one-class model of "normality", formed by as many examples of normal instances as possible. The new entries are tested against this model of normality, resulting in some sort of *novelty score*. Anomalies present high novelty scores while normal instances present low novelty scores. In essence, higher novelty scores indicate a more *abnormal* instance.

Thus, when the robot detects unknown data, it can ask the user for the label of this novel entry if it considers that these new data is worth learning. However, differentiating between noisy entries and useful data is not trivial [Chandola et al., 2009].

7.3 Problem Definition

To give an intuition of the problem, we are going to use the example depicted in Fig. 7.1. The figure shows two clusters, N_1 and N_2, which are the learnt model. These clusters are instances belonging to two different classes, for instance, the user pointing to two different directions. Fig. 7.1a depicts the initial conditions after the model is built and with some outliers marked as, o_1, o_2 and o_3. Asking the user for the labels of each of these outliers, might not have sense since they might be noise gathered by the robot's sensors or situations that occurred randomly and which are not worth learning. In Fig. 7.1b, the system has received more outliers, but some of them have started to form a cluster O_3. This cluster might be indicating that some specific event has been repeated over time. Therefore, the robot may decide to ask the user for a label describing this event. Finally, in Fig. 7.1c, the robot has received a label for the new cluster and thus, it has incorporated these data to the learnt model as the class N_3. Following, we formalize some of these terms in the following definitions: *new* data, *strange* data, and *interesting* data.

7.3.1 Definitions

We refer to **new data** to the data entry that has not been presented to the system any time before. This aspect is a characteristic of the data entry itself and not of how the model interprets these data.

Strange data entries are those which do not conform with the learnt model. I. e. those entries in which the model would yield a prediction with low confidence. For instance, o_1, o_2, o_3, and all the points of the cluster O_3 in Figs. 7.1a and 7.1b would be considered *strange*. The contrary to *strange* data is *known* data, which is data that is predicted with high confidence and, usually, belongs to the known model. For instance, any new datum that falls inside the clusters N_1 or N_2 would be considered as known data instead of strange data. Note that *strange* data refers only to how well the data fit on the current model. In other words, *strange* data may be both, an anomaly or a novelty. We need, therefore, another term that helps distinguishing between these two.

For that purpose we define the term **Interesting data**. Interesting data describes some phenomena that occurs in the world. It is opposite to *noise*, whose definition is "a phenomenon in data which is not of interest to the analyst" [Chandola et al., 2009]. From

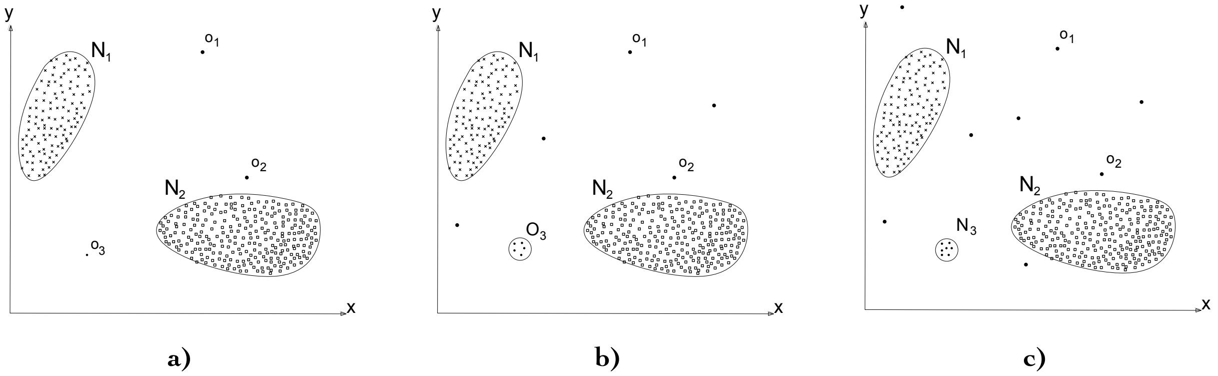


Figure 7.1: A sequence of anomalies and novelties appearing over time. We assume that noise (anomalies) is uniformly distributed (o_1, o_2) while interesting novelties tend to form clusters (o_3 becoming into O_3) that later can be incorporated to the learnt model. Modified from [Chandola et al., 2009].

the robot's point of view, interesting data have the potential to change or update the current learnt model if these data is incorporated to it. We assume that this kind of data tend to form clusters in the learning space. In Fig. 7.1b, O_3 is considered *interesting*.

Since 0_3 is considered both *strange* and *interesting*, the robot may decide that it is worth to ask the user for the label of this cluster. Therefore, after identifying a new stimulus as strange and interesting, the robot might want to incorporate this stimulus to the learnt model. This step means adding the entry to the set of data considered as "normal" and recalculating the model for the one-class classification, thus recalculating the threshold values for the novelty score. This means that when new stimuli, similar to the previous novel stimulus, are shown to the system, they will now be identified as interesting and normal, that is, as already known.

7.3.2 Curiosity Factor of the Robot

We assume that interesting stimuli tend to form clusters like O_3 in Fig. 7.1c. We need, therefore, a parameter to define which is the minimum cluster size or density that indicates that an interesting event is occurring.

We called this parameter, the curiosity factor K of the robot, since it will be key to decide the tendency of the robot to ask questions to the user. A very curious robot would ask the user as soon as it thinks a strange and interesting cluster is being formed, while less curious robot would need a much bigger cluster before it asks to the user.

7.4 Related Work

The applications of Novelty Detection techniques vary widely in the area of application and performance. Ding et al. [Ding et al., 2014] provide a comparative evaluation of Novelty

Detection methods for 10 different experiments in different areas. The datasets in these experiments varied from breast cancer detection to phonemes analysis. The methods used were a One class SVM based algorithm; a Nearest Neighbor based technique; a clustering technique, such as K means; and a parametric probability density estimation, a Gaussian Mixture. The results showed a better and more stable performance of the K neighbors algorithm. They also showed that the One Class SVM algorithm was more sensitive to the size of the training data, requiring more data than the other three methods to increase its performance.

Literature presents many problems related with Novelty Detection. For instance, in fields similar to the problem addressed in this paper, we can find the work by Drews et al. [Drews et al., 2013]. The article proposes a framework to detect and segment changes in robotics datasets, using 3D robotic mapping as a case study. The main applications are video surveillance or exploration of dangerous environments. In this case, noise avoidance is very important, the data is pre-processed by two consecutive methods (i) a simplification algorithm and (ii) a sparse outliers and ground plane removal methods. The novelty algorithm used is based on Gausian Mixture Models (GMMs).

One of the studied applications more related to our work is [Pinto et al., 2011]. In their article, the authors present an approach to learn the semantics of a room from the human user. For this purpose the agent must be able to identify gaps in its own knowledge. They propose a method based on graphical model to identify novel input which does not match any of the previously learned semantic descriptions. Their method employs a novelty threshold defined in terms of conditional and unconditional probabilities. Our approach also attempts to identify novel inputs by applying novelty thresholds, and is able to make the agent to identify gaps in its knowledge. However we decided to build this novelty filters with algorithms from different fields, to be able to compare their performance, and to use pose recognition as the dataset in the experiments. They do not enter into the problem of abstraction and tolerance to noisy data, a problem that we address in this paper.

In the field of Cumulative Learning Robots, [Nehmzow et al., 2013] presents an article on Novelty Detection as an intrinsic motivation for cumulative learning robots. The article describes the theoretical basis of habituation, the task of ignoring perceptions that are similar to those seen during training, but being able to highlight anything different. They explain different novelty detection methods for habituation, including "grow-when-required" (GWR) networks, a similar approach of expanding the base of knowledge when necessary, as the one used in this paper.

Nehmzow and colleagues conclude their work in [Nehmzow et al., 2013] explaining that existing Novelty Detection approaches show a number of strengths and weaknesses, and that there is no single universal method for novelty detection, rather than a suitable choice depending on the task.

To the extent of our knowledge, there is no reference on Novelty Detection for pose recognition in a Human-Robot interactive application.

7.5 Description of the Proposed Solution

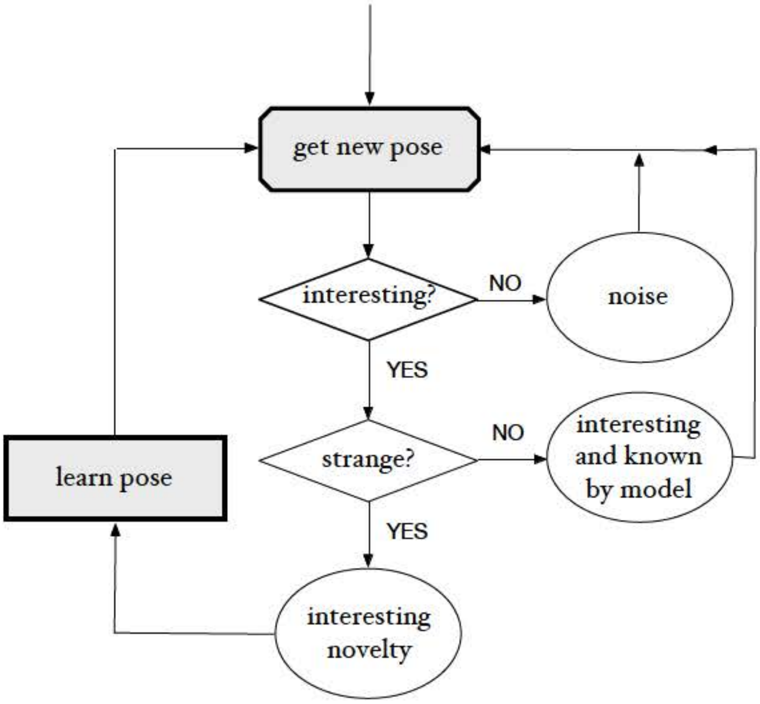


Figure 7.2: System general scheme. The new entry is first tested to be classified as *interesting* or noise. If it is interesting, it is then tested to check whether it is *strange*, or already known by the model. Only interesting but unknown novelties are learned. Note that the learning that novelty will entail asking for the label of that new entry and incorporating to the dataset

This section describes our novelty detection system and how it is applied to detect poses. Our pose-detection system is described in detail in a previous work [Gonzalez-Pacheco et al., 2013][1]. The vision inputs come from a Kinect camera and its software libraries extract a kinematic model of the human body. The shape of the data of our novelty detection system is further described in section 7.6. In this section we describe the Novelty Detection system rather than how the data for the poses is gathered and pre-processed. More information on the data acquisition and processing can be found in [Gonzalez-Pacheco et al., 2013].

The process of detecting new poses starts with the arrival of a new pose data from the Kinect (Fig 7.2). This entry is tested against all other data entries received up to that point. This test consists in evaluating whether the received pose forms a cluster with

[1] Note: this paper corresponds to Chapter 3.

other poses. If the result is positive, then the entry is considered *interesting*. After that, the entry is tested against the knowledge base (the model) to determine if this new pose is strange. I. e. if the new pose cannot be predicted by our model. If the result is positive for both tests, the entry is classified as novelty, and the robot will ask a query to the user asking for the label of this new data. Once the user answers, the entry is learned by the system by adding it to the robot's knowledge base.

If the new pose data is regarded as not interesting it is classified as noise. However, it is not discarded, since we will need it later to check if newer data forms a cluster with the currently received instance. If it is regarded as not strange, it is classified as known and is also kept in the dataset of all entries received.

Note that the system checks first if the new data is interesting instead of strange. The reason behind this is to avoid wasting computational resources in checking if a data entry is strange, prior to detecting whether this instance is interesting or not. I.e. we first apply a noise filter to the received data.

The step of quantifying both the interestingness and the strangeness of a stimulus is very similar. Mainly, it consists in obtaining the model of normality $M(\theta)$ of a Novelty Detection algorithm, where θ represents the parameters of the model. This model produces a novelty score $z(x)$, when it is evaluated against new data instance x. Then, we evaluate the novelty score $z(x)$ against a novelty threshold k

$$z(x) \geq k \tag{7.1}$$

and, if this condition holds, then x is considered *abnormal* [A.F. Pimentel et al., 2014].

7.5.1 Enabling the system to filter noise by evaluating the interest of a stimulus

If we want to ask the user the fewer queries as possible, we need to keep these questions as much informative as possible. This implies that the sensitivity of the system should be as lower as possible while being capable of detecting novel and interesting events. Summarizing, we need a step that it is able quantify the interestingness of an event and make questions to the user only when interesting events occur. Note that we call uninteresting events as noise.

We assume that interesting events tend to form clusters in the learning space while noise is distributed uniformly. To enable the system to detect clusters, it has to store and track all the past events or, at least, the most recent ones. Therefore, the system needs to account for all the entries it has seen recently, to see patterns of repetition.

Another approach to detect interesting patterns is to use a novelty detection algorithm and fit it with all the data the system has ever observed. With such algorithms, new events that occur randomly are seen as outliers. However, events that are not noise produce instances that, eventually, will start forming a clusters. Once a cluster appears, the novelty algorithm would stop detecting the elements of this cluster as outliers. In other words, if an event occurs so often that its data stops being an outlier, the robot can consider that it is likely that the event would occur again in the future, and it can decide to ask the user for a label that describes the event.

7.5.1.1 Obtaining the noise score

To enable the system to decide whether a new instance is either interesting or noise, we build a model $M_{\mathcal{I}}$ from a dataset $\mathcal{I} = [i_1, i_2, \ldots, i_m]$ that contains all the previously observed instances[1]. These include both the instances added to the dataset and the ones that are considered noise.[2]

This dataset $\mathcal{I}$ is expanded every time a new pose is presented to the system. Fitting a novelty algorithm with $\mathcal{I}$, produces a fitting score:

$$Z_{M_{\mathcal{I}}} = [z(i_1), z(i_2), z(i_3), ..., z(i_m)]$$

which represents a distribution of the scores of all entries. This distribution can be normalized, obtaining a mean μ_{all} and a standard deviation σ_{all} of the interestingness. We can obtain a standardized noise score $z_{\varepsilon}(o_1)$[3]:

$$z_{\varepsilon}(o_1) = abs(\frac{z(o_1) - \mu_{all}}{\sigma_{all}}) \tag{7.2}$$

7.5.1.2 Obtaining the interestingness threshold

The *interestingness threshold* is obtained from the scores of the normal dataset. We use the Extreme Value Theorem (EVT) to obtain its value. EVT is a “branch of statistics which deals with extreme deviations of a probability distribution” [A.F. Pimentel et al., 2014]. It considers extremely large or small values in the tails of the distribution that is assumed to generate the data to obtain the threshold [A.F. Pimentel et al., 2014]. For instance, setting the novelty threshold at one σ from the mean, implies that any instance whose score is $z_1 > \sigma$ or $z_1 < -\sigma$ will be considered strange.

Formalizing the equation that represents entries outside the threshold region:

$$1 \geq abs(\frac{z(o_1) - \mu_{all}}{\sigma_{all}}) \tag{7.3}$$

[1] If this data is too big, it is possible to use a memory of recent events and use only the data that has been stored in that memory

[2] Notice that the instances considered as noise are needed so we can evaluate later if they form clusters

[3] The standard score formula can also be denominated the normal score or z-score in the literature

In this example, the score of the new data entry is one σ units away from the average value, μ of the normal scores. A value of one σ means that, a new entry is labeled as *normal* if lays approximately within the 68% of the closest scores to the mean of the dataset. If the standard score is higher than 1, then the score $z(o_1)$ is higher than the 68% of the normal entries, and it can be classified as strange.

This is for the case in which we want to consider a 68% (one σ) of the closest instances as normal. But it might be the case in which our model should be more or less restrictive when considering new entries as *normal* or *abnormal*. To do so, we introduce the concept of Curiosity Factor K. In essence, we use a threshold $+K$ and $-K$, instead of $+1$ and -1 to allow our model to be more or less restrictive. Therefore, we have:

$$K \geq abs(\frac{z(o_1) - \mu_{all}}{\sigma_{all}}) \tag{7.4}$$

and if we develop this expression:

$$1 \geq abs(\frac{z(o_1) - \mu_{all}}{K \times \sigma_{all}}) \tag{7.5}$$

where $abs(\frac{z(o_1) - \mu_{all}}{K \times \sigma_{all}})$ is the *noise score* z_ε of our system.

Therefore, with Eq. 7.5, any value of o_1 which produces a noise score below or equal to 1, will be considered *interesting*.

7.5.2 Enabling the system to evaluate the strangeness of a stimulus

In the previous step, the system was able to detect if a new stimulus is likely to happen again. After this step the robot needs to know if this interesting pattern can be predicted by the robot's model. If the stimulus is unknown by the model, it means that it is an interesting and unknown event that might be worth to learn.

To evaluate if a stimulus is known, we evaluate new instances against the normal dataset. The normal dataset consists of the poses that have been considered normal in the past and that were learned at some point. The model of normality is generated by the novelty detection algorithms using the normal set as the training set. The normal set is expanded as the system learns new poses. The system learns whatever the user teaches it. We consider that the learned poses are a consistent base of knowledge.

7.5.2.1 Obtaining the strangeness score

The model is obtained by using one-class classification methods. These classifications methods allow to compute a score of how well each instance fits the model. The score is calculated differently for the distinct methods used in this paper (see section 7.6.3.1).

If we consider the normal dataset $\mathcal{N} = [n_1, n_2, n_3, ..., n_m]$ which contains all the instances that have been considered normal, and we fit a model $M_{\mathcal{N}}$ with it, we can obtain a fitting score for each its instances:

$$Z_{M_{\mathcal{N}}} = [z(n_1), z(n_2), z(n_3), ..., z(n_m)]$$

which represents a distribution of the scores of the normal entries. This distribution can be normalized, obtaining a mean μ and a standard deviation σ.

The strangeness standard score z_{str} of a new entry o_1[1], can be calculated as follows:

$$z_{str}(o_1) = abs(\frac{z(o_1) - \mu_{\mathcal{N}}}{\sigma_{\mathcal{N}}}) \tag{7.6}$$

Note that this is similar to Eq. 7.2, but using with μ and σ calculated from the dataset $\mathcal{N}$ rather than $\mathcal{I}$.

7.5.2.2 Obtaining the interestingness threshold

Following the same procedure as in section 7.5.1.2, we can obtain the noise threshold:

$$z_{str}(o_1) = abs(\frac{z(o_1) - \mu_{\mathcal{N}}}{K \times \sigma_{\mathcal{N}}}) \tag{7.7}$$

Here, again we use the notion of the *curiosity factor* to control how much close to the average should an instance be to consider this instance interesting.

7.5.3 Curiosity level of the robot

The value of K represents where we place the threshold that decides which are the extreme values of a fitting score. By increasing K, we decrease the range for the extreme values and thus, we are considering more instances as normal.

This K factor, as we have seen, can be applied to both strangeness and interestingness evaluation. This is key to the sensitivity of the system. We can increase the sensitivity by decreasing the curiosity factor, because, to be normal, the new score will need to be closer to the mean with respect to all other instances in the base of knowledge.

An important remark is to note that, as the value of the *curiosity factor* depends on the sensitivity degree we want to achieve, and also on the nature of the data in the application, it needs to be computed empirically.

[1] The convention is using the letter o to label outliers

7.5.4 Enabling the system to learn from novel stimuli

If a new instance passes the *interestingness* and *strangeness* filters, it is classified as a novelty that has to be learnt. This means that the robot asks the user for a label for this instance. Once the robot knows the label of the instance, it can learn it.

The learning process consists in adding the novel data to the robot's base of knowledge, that is, the normal data set. With the new data, the Model of normality is recalculated and expanded. Therefore, a new score is added to the set of normal scores so its mean μ and standard deviation σ are recalculated.

7.5.5 Description of the used novelty detection methods

In sections 7.5.1 and 7.5.2 we refer to the models $M_{\mathbb{J}}(\theta)$ and $M_{\mathbb{N}}(\theta)$. The models highly depend on the Novelty Detection algorithm being used. In this section we describe the different ND algorithms that can generate models such the ones described in these sections.

Novelty detection techniques can be classified in the following categories [A.F. Pimentel et al., 2014]: (i) probabilistic, (ii) distance-based, (iii) reconstruction-based, (iv) domain- based, and (v) information-theoretic techniques. Each category has a series of advantages and disadvantages, and different computational costs. Since there is no single universal method for novelty detection [Nehmzow et al., 2013], the choice of the appropriate algorithm depends on the task.

7.5.5.1 Probabilistic-based novelty detection methods

These techniques use probabilistic methods that often involve a density estimation of the *normal* class. An entry in a low density area indicate that there is probability of it being a *normal* object [A.F. Pimentel et al., 2014].

The method used in this category is **Gaussian Mixture Model**, a GMM. A GMM is a probabilistic model that assumes that all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters [Pedregosa et al., 2011].

This paper used the implementation of GMM provided by Scikit-Learn Library [Pedregosa et al., 2011]. It does not provide directly a *normal* or *abnormal* method, but it does provide a built-in function called *score*. The score represents the log probability of a sample under the model.

7.5.5.2 Distance-based novelty detection methods

This category includes the concepts of nearest-neighbour and clustering analysis that have also been used in classification problems. It assumes that *normal* data are tightly clustered, while novel data occur far from their nearest neighbours [A.F. Pimentel et al., 2014].

This paper tested one method from this category, **K-Means**. The K-means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the "inertia" of the groups. This algorithm requires the number of clusters to be specified, in this case, as we are interested in a one-class classification, the number of clusters will be 1. The *K-means* algorithm aims to choose centroids C that minimize the within cluster sum of squares objective function with a dataset X with n samples [Pedregosa et al., 2011].

The K-means method was also implemented in the the Scikit Learn Library [Pedregosa et al., 2011]. It does not provide a *normal* or *abnormal* label either. The score function in this case represents the opposite of the value of X on the K-means objective [Pedregosa et al., 2011].

7.5.5.3 Domain-based novelty detection methods

Algorithms in this category use domain-based methods to characterize the data for the model of normality. These methods typically try to describe a domain containing *normal* data by defining a boundary around the *normal* class such that it follows the distribution of the data [A.F. Pimentel et al., 2014].

Methods used from this category are specifically categorized as "oultier detection methods", and provide directly a label categorizing test data as *normal* or *abnormal*.

One of the algorithms chosen was **One Class SVM**, from the Scikit Learn Library [Pedregosa et al., 2011]. The One-Class SVM has been introduced to decide whether a new observation belongs to the same distribution as exiting observations (it is an inlier), or should be considered as different (it is an outlier). It requires the choice of a kernel and a scalar parameter to define a frontier. The RBF kernel is usually chosen although there exist no exact formula or algorithm to set its bandwidth parameter. This is the default in the scikit-learn implementation. The ν parameter, also known as the margin of the One-Class SVM, corresponds to the probability of finding a new, but regular, observation outside the frontier [Pedregosa et al., 2011].

The other algorithm used is **Least Squares Anomaly Detection**. It is a flexible, fast, probabilistic method for calculating outlier scores on test data, given training examples of inliers. The model is controlled by two parameters: σ (a kernel length scale, controlling how "*smooth*" the result should be) and ρ (a regularisation parameter, which controls the sensitivity to outliers) [J.A. Quinn, 2014].

7.6 Experimental Evaluation

7.6.1 Data Acquisition

To carry out our experiment we used the Poses Dataset recorded in a previous work [Gonzalez-Pacheco et al., 2013]. The dataset consists in 30 users teaching a robot a set of poses captured using a Kinect camera. Each pose is recorded as a data instance I:

$$I = (t, J, l) \tag{7.8}$$

where t is the timestamp of the pose, J the set of joints returned by the Kinect and, l the label for that pose. The set of joints $J = (j_1, j_2, ..., j_{15})$ is composed by 15 joints where each joint j_i is composed of:

$$j_i = (x, y, z, q_x, q_y, q_z, q_w) \tag{7.9}$$

whose fields correspond to its 3D position, and a quaternion defining the joint orientation.

7.6.2 Interactive Labelling of what the Robot Sees

To fill the label l from Eq.7.8, the user has to tag the pose he/she is showing to the robot. The process of tagging (labelling) the poses is carried out interactively by voice during the training stage.

Our robot is equipped with an ASR (Automatic Speech Recognition) system described in a previous work [Alonso-Martín and Salichs, 2011]. This ASR is grammar-based, which means that the robot extracts semantic meaning from the user's speech providing she follows certain pre-written grammar rules when speaking. A grammar is defined by the tuple $G : (U, S)$ which relates a set of utterances U with a set of semantic meanings S. For instance, it is possible to write a grammar that relates the sentences "*Please, could you help me?*" and "*Could you give me a hand with this, please?*" to the semantic meaning of soliciting assistance to the robot.

We have built one grammar $G_p =: (U_p, S_p)$ to detect if a user is telling a pose to the robot. This grammar allows us to detect up to 9 pose definitions separated in three different categories: *turned*, *looking*, and *pointing*. The content of set U_p is out of the scope of this paper—some examples can be found in [Gonzalez-Pacheco et al., 2013]—, however, we define S_p since it contains the concepts we will teach the robot:

$$S_p = \{s_a, s_d\} \tag{7.10}$$

where s_a, s_d are the semantic meanings of the user's speech:

1. Action Semantics (s_a). Which can take one of the following values: $s_a = \{turned,\ looking,\ pointing\}$, where:
   (a) Turned ($s_a = turned$). Defines that the user has oriented her whole body towards some direction defined in s_d.
   (b) Looking ($s_a = looking$). Defines that the user has oriented her head towards some direction defined in s_d.
   (c) Pointing ($s_a = pointing$). Defines that the user is pointing with her arm towards some direction defined in s_d.
2. Direction Semantics (s_d). It can take one of the following values: $s_d = \{left, forward, right\}$, where
   (a) Left ($s_d = left$). Defines that the action defined in s_a is executed to the user's own left side. For example, if it is pointing, it is doing it to her left.
   (b) Forward ($s_d = forward$). Defines that the action defined in s_a is executed to the user's front.
   (c) Right ($s_d = right$). Defines that the action defined in s_a is executed to the user's right

When the robot detects a speech, the ASR evaluates it against G_p. If the evaluation produces a valid result, the ASR tags the speech with a label $l_p^{(i)} \in G_p$. To be considered a valid grammar, our grammar needs both s_a and s_d semantic meanings.

Some label examples might be $l_p = (looking, left)$, which indicates the user is looking to her left; or $l_p = (turned, forward)$, which indicates the user is turned to the robot.

Note that the user can express the same semantic meaning in different ways. For instance, phrases such as "*I'm pointing to the right*" or "*Look carefully, I am pointing towards my right*" would produce the same semantic meaning: $l_p = (pointing, right)$

7.6.3 Method

From all the poses the users recorded for the dataset, we only use 3 of them: pointing left, right and forward (Fig. 7.3). The main reason for this decision is because the pointing poses where the ones that presented more differences between users, so it was easier to simulate novelties by just adding users that pointed differently (e.g. robot learning labels from right-handed users and then presenting left-handed as novelties). Therefore, although the users trained the system with three labels, they did it differently from ones to others. In fact, we counted an average of three different ways of pointing for each class. That makes, potentially, up to 8 different classes of novelties to be detected in our dataset (1 training, the rest in the test set).

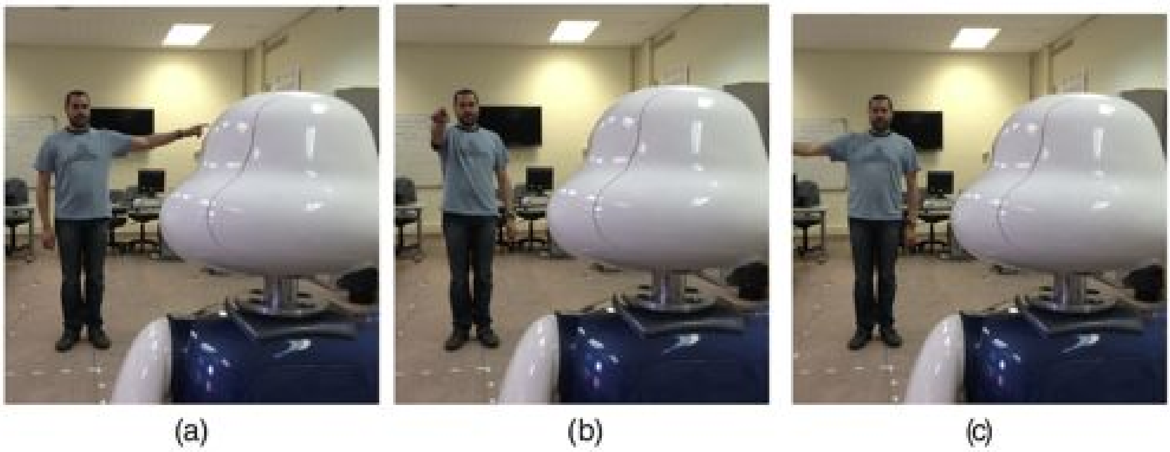


Figure 7.3: Example of a user pointing. With permission from [Gonzalez-Pacheco et al., 2013].

We cleaned the dataset to adapt it to the experiment requirements. Mainly, we removed the parts of the body that were less significant for pose learning: that is, the lower-part of the users' body. In that way, the used joints were Head, Neck, Torso, and Right and Left Shoulder, Elbow, and Hand. Additionally, we normalized all the data to have the frame of reference in each user's torso instead of the kinect's.

We evaluated the detection performance of four novelty detection algorithms: Gaussian Mixture Models (GMM), K-Means, One-Class Support Vector Machines (OCSVM) and Least Squares Anomaly Detection (LSA). For the first three algorithms we used the Scikit-Learn implementation [Pedregosa et al., 2011], while for the latter we used the LSA author's implementation [J.A. Quinn, 2014].

7.6.3.1 Mapping the output of the algorithms

OCSVM and LSA provide directly a numeric label classifying the entries as anomalous (1) or normal (0). Table 7.1 describes the meaning of these labels in each step of our system. Note that the objective of our system is detect novelties, that is, instances that are interesting but strange for our model. In terms of these algorithms this would reflect that a novel entry would score 1 and 0 in the *Noise Filter* and the *Strangeness Test* respectively.

Score	Noise Filter	Strangeness
1	Noise	Strange
0	Interesting	Known

Table 7.1: Interpretation for the OCSVM and LSA algorithms predictions applied to our system. A new entry is considered a novelty when it is both *Interesting* and *Strange*.

Novelty Score for x_i	Noise Filter	Strangeness
≥ 1	Noise	Strange
< 1	Interesting	Known

Table 7.2: Interpretation for the K-Means and GMM algorithms predictions applied to our system. A new entry is considered a novelty when it is both *Interesting* and *Strange*.

The output from K-Means and GMM are slightly different. Instead of providing a binary value, their output is a score (see sections 7.5.2 and 7.5.1). Table 7.2 describes the mapping between the values of these scores and their corresponding interpretation in the *Noise Filter* and *Strangeness* tests. Note that a novel entry (interesting but strange to our model) needs a score below 1 in the *Noise Filter* and equal or greater than 1 in the *Strangeness* evaluation test.

7.6.3.2 Graphic Interface

A graphic interface was created to show the results of the tests and the shape of the skeletons we were considering, it can be seen in Figure 7.4. The figure depicts an example of how our system operates. The left part represents the *Noise Filter* tests while the right part shows the *Strangeness* evaluations. The 3D plots represent all the observed poses. In red are depicted the poses that have been incorporated to the system which, in the example of the figure, all are *pointing right* poses. In black are represented the last observed pose. In blue are the observed poses that have not been incorporated to the dataset. The bar plots indicate the score of each algorithm in each test. If one algorithm score is below the threshold (which is 1), then the test is passed (indicated by the bar painted in green)[1]. Table 7.3 summarizes the colors of the figure and their representation.

In the upper row the system observes, for the first time, a *pointing left* pose, which is evaluated against the Noise Filter. Since it is the first time that the system is exposed to that pose, the filter considers the pose as noise. However, the instance is not discarded but stored in memory for further evaluation in case similar poses appear in the future. In the second row, another *pointing left* pose is observed. This time the filter scores lower results since there is another similar pose in the observed dataset (the previous one). In the lower row, a third *pointing left* pose is observed. This time it passes the Noise filter and, therefore, is evaluated against the *Strangeness* step. Since the pose is not known by the model (scores low in the test), the test is passed, indicating that the pose is a novelty that should be incorporated to the dataset of known poses (red). In this case, the robot would ask the user for a label describing the observed pose.

[1] Note that, since the important value is the threshold, the scales of the bar plots are not in scale to facilitate a better visualization of the plots.

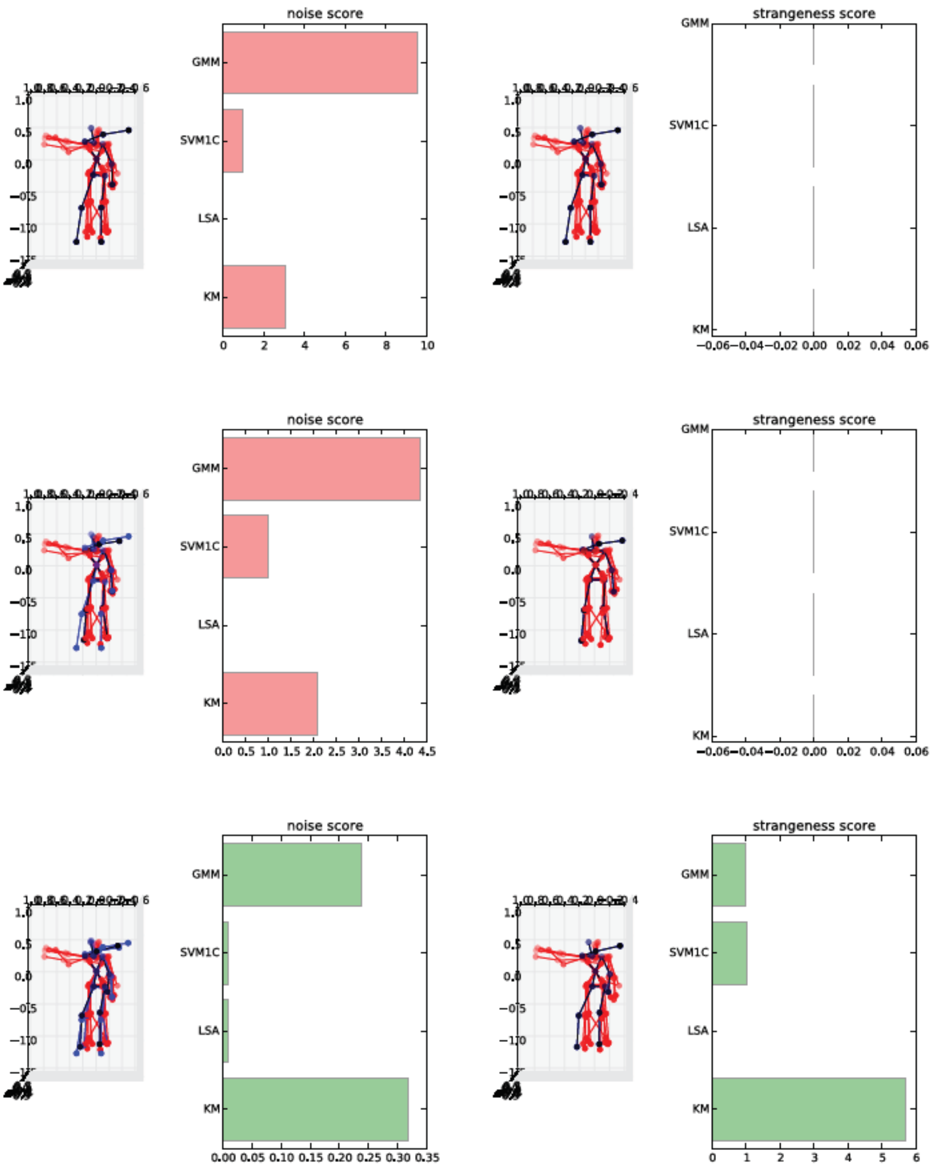


Figure 7.4: Graphical representation of the algorithm when exposed for the first time to new poses. Poses that were used to learn are in red, current observed pose is in black, and previously observed but not yet incorporated are in blue. The figure shows a system that has been trained with *pointing right* poses (in red) that, after training, is exposed to *pointing left* poses Each row represents a new observation of a *pointing left*, which is, finally, marked to be learnt in the third row. Note that, to provide better detail, the scales or the figures are not the same. See Table 7.3 for more info regarding the threshold and the color codes of this figure.

7.6.4 Results

This section presents the results of the experiments that asses the performance of our approach. We carried out two different experiments, one for evaluating how well the system interprets the interestingness of new data, and another one to evaluate how it calculates the strangeness of these data. Also we evaluated how well the system is able

Noise score	Color (noise)	Strangeness score	Color (strange.)	Prediction
≥ 1	red	-	-	*Noise*
< 1	green	< 1	red	*Known*
< 1	green	≥ 1	green	*Novelty*

Table 7.3: Color codes for Figure 7.4.

to detect novelties that belong to already known classes (e.g. new ways of pointing to the same direction).

7.6.4.1 Evaluating the Interestingness of New Data

As we defined in previous sections, to prevent asking too many questions to the user, new data is not considered interesting until it does not start forming clusters. In the case of poses, a new pose is considered noise until the robot has seen it several times.

Therefore, we were interested in knowing how many times should a pose be presented to the robot. That is, how many times do the user needs to stand in front of the robot in the same pose, to until the robot gets interested for that pose. Because in the dataset each user showed the same pose only once, we show the same pose with different users.

Fig. 7.5, shows the interestingness evaluation results. The plot shows the evolution of the noise score for the different algorithms. The X axis represents the cumulative number of users that showed the same pose to the robot, while the Y axis represents the averaged noise score from the algorithms, extracted from 63 try-outs. A score below the threshold, indicates that the entry is detected as interesting. Also, one of our design choices is to avoid the robot asking the user the first time it sees a new pose. Therefore, we consider that the algorithm should mark as noise each pose when it is observed for the first time.

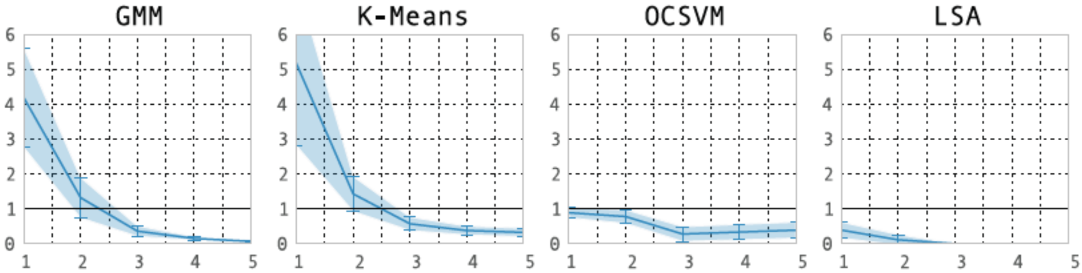


Figure 7.5: Noise score (y-axis) evolution when adding from 1 to 5 instances (x-axis) of an unknown pose. Points below the threshold indicate that the pose is considered interesting.

We found that setting the curiosity factors to $K_{GMM} = 3$ and $K_{Kmeans} = 1$ produced good results for GMM and K-Means respectively. Both GMM and K-means do not start considering a pose as interesting until it is observed, at least, three times. On the other side, OCSVM and LSA detected new poses as interesting the first time they observed them. In that way, they are over sensitive to new stimuli. In other words, in our set up, GMM and K-Means acted as better noise filters than the other algorithms. As we will discuss in section 7.6.5, the value of K has been of paramount importance with respect to the algorithm's performance.

7.6.4.2 Strangeness evaluation

The next step after new data is detected as interesting, is finding out whether these data are already known by the model or if, on the contrary, the current model cannot predict them confidently.

To evaluate how well the system predicts the strangeness, we train with poses of one class and test it against poses of other classes. In such set-up, poses with other classes should be classified as novelties (strange) while poses of same class should be classified as not novel (known). For instance, if we train our system showing it only poses of the user pointing to the right, the model should consider strange any other pose that is not pointing to that direction.

To evaluate the performance of this task, different experiments were carried out with the existing dataset. The dataset was separated by poses, *pointing right*, *pointing left* and *pointing forward*. Each user showed the robot each pose once.

In the experiment we built three different models, one for each pose and then tested how these models were able to predict the other two poses. For instance, we built a model from instances of users pointing to their left, and then tested with users pointing forward and right.

After building the models and evaluating them independently, we merged and averaged the results to give a general overview of how well the system works. The expected result is that all the entries from different poses should be detected as strange, while the entries from the same pose should be detected as known.

This process was repeated three times, where the model was built with training datasets of 5, 10 and 20 instances for each pose. The test datasets were always of size 20, containing 5 entries of each of the other 2 poses, and 10 entries of that same pose.

We considered the problem as a one-class binary classification where a known class was marked as negative (0) and new classes marked as positive (1). Table 7.4 summarizes the results for the experiment using the F-Score [Van Rijsbergen, 1979]. Note that a higher F-score means poses classified correctly while lower F-scores indicates the model failed predicting such poses.

Size	GMM	OCSVM	LSA	K-Means
5	**0.73** (0.05, 0.04)	**0.73** (0.10, 0.07)	0.28 (0.21, 0.15)	0.62 (0.18, 0.13)
10	**0.81** (0.07, 0.05)	0.74 (0.11, 0.08)	0.53 (0.26, 0.18)	0.55 (0.19, 0.13)
20	**0.84** (0.06, 0.05)	0.78 (0.10, 0.07)	0.58 (0.25, 0.17)	0.42 (0.27, 0.19)

Table 7.4: Strangeness F-score results. Note: Results are in Mean (Std. Dev, Std. Err). Size (s) = number of users in the base of knowledge, Curiosity factor for K-Means: $K_{Kmeans} = 1$ Curiosity factor for GMM: $(s = 5) : K_{GMM} = 30$, $(s = 10) : K_{GMM} = 3$, $(s = 20) : K_{GMM} = 0.1$

7.6.4.3 In-class novelties

If compared separately, we found a worse performance of the algorithms when trained with the classes *pointing left* and *pointing forward*, so we decided to test these classes more specifically. In-class novelties are defined as novelties within a class. We divided the poses of these classes in sub-classes. The aim is to test if the system would detect if the user is pointing left with her right arm or with her left arm.

To test novelties between classes we did the following procedure. The training dataset had 5 instances (each one from one user). The test dataset consisted on 6 known instances plus 6 unknown instances. Note that the ground truth of known instances is *non-novelty*. That is, they are instances created by users that pointed in the same manner as the training users did. For example, if the pointing poses in the training dataset were recorded from a right handed person, these 6 known instances were recorded from other right handed people. The motivation of using known and unknown instances in the test dataset is to evaluate how well our system detects both the known and the unknown.

The 6 unknown instances (whose ground truth is novelties) were distributed in a 2 + 4 sub-classes[1]. Two of them were *in-class novelties*. That is, novelties that belong to the same class as the known model. For instance, if the training data consisted in examples of right handed people, these two instances were retrieved from left-handed people pointing to the same directions as the trainig data. The remaining 4 were novelties belonging to other classes (e.g. *pointing-right* when the training data was with *pointing-left* poses). Note that we only used 5 users to train the system because the division in sub-classes did not allow us to have a bigger training datasets.

Our hypothesis is that instances belonging to the same sub-class should be detected as normal while instances from different sub-classes or from different classes should be detected as novel. The sub-classes that we used are described in Table **??**, and table 7.6

[1] We consider a sub-class the instances of one class that differ with the rest in some aspect. For instance, a pointing pose would be different if executed by a right-handed than if done by a left-handed

summarizes the F-Score of the algorithms for these different poses.

PL-LH	*Pointing left* with the *left* hand
PL-RH	*Pointing left* with the *right* hand
PFW-RH	*Pointing forward* with the *right* hand

Table 7.5: Description of the sets for in-class novelty detection

	GMM	OCSVM	LSA	K-Means
PL-LH	0.92 (0.07, 0.02)	0.77 (0.08, 0.03)	0.95 (0.07, 0.02)	0.84 (0.09, 0.03)
PL-RH	0.73 (0.08, 0.03)	0.72 (0.07, 0.02)	0.32 (0.20, 0.07)	0.62 (0.16, 0.05)
PFW-RH	0.71 (0.13, 0.04)	0.69 (0.10, 0.03)	0.58 (0.23, 0.08)	0.54 (0.22, 0.07)
Average	**0.79** (0.09, 0.07)	**0.73** (0.03, 0.02)	**0.58** (0.31, 0.22)	**0.67** (0.13, 0.09)

Table 7.6: Novelty Detection F-Score [Mean (Std. Dev, Std. Err.)] for in-class novelties. $K_{GMM} = 30$, $K_{Kmeans} = 1$.

When considered the averaged results, GMM performed better than the other three algorithms. In general, the average F-score is similar to the results from table 7.4, which implies that the system is capable of detecting both novelties that belong to new classes as well as in-class novelties.

7.6.4.4 Novelties in multi-class systems

In previous tests, we evaluated the capacity of the system to detect novelties when it was trained with a single class. In this section, we evaluate the system to detect novelties, but when it already has learnt two classes. The results of the experiment, shown in Table 7.7, indicate that the system keeps a similar performance compared to the previous experiments. In this experiment, OCSVM performed better than GMM.

7.6.5 Discussion

The results show that our approach enables a robot to detect when a user is standing in a pose that has never seen before and to decide whether this new pose is worth learning or not.

In general, GMM performed the best to detect when a new pattern started being interesting and to detect when new data was not known by the system (i.e. when it was *strange*). The key for its good performance was the possibility of modifying the curiosity

New Class	GMM	OCSVM	LSA	K-Means
PR	0.67 (0.19, 0.10)	0.79 (0.07, 0.03)	0.70 (0.18, 0.09)	0.84 (0.06, 0.03)
PL	0.68 (0.02, 0.01)	0.82 (0.03, 0.02)	0.60 (0.05, 0.03)	0.69 (0.13, 0.06)
PFW	0.78 (0.07, 0.03)	0.74 (0.10, 0.05)	0.53 (0.17, 0.09)	0.67 (0.05, 0.03)
Average	**0.71** (0.05, 0.04)	**0.78** (0.03, 0.02)	**0.71** (0.10, 0.07)	**0.73** (0.08, 0.05)

Table 7.7: F-score obtained when the system was trained with two classes and presented a new one. $K_{Kmeans} = 1$ $K_{GMM} = 0.8$ Note: New class = new class presented to the system. Results are in Mean (Std. Dev, Std. Err).

factor K, which allowed us to tune the algorithm to be insensitive to new stimuli until this stimuli is not observed a certain number of times.

The main drawback of our approach is that as the knowledge base (that is, the size of the training set) grows, K has to be adapted accordingly. Despite K has to be set empirically, we believe it is possible to find the relation between K and the dataset size. However, this requires further work since, K will depend highly on the algorithm used (K-Means, GMM, etc.) and on the nature of the data themselves.

The curiosity Factor K_{GMM} had to be adapted depending on the size of the base of knowledge. As an example, if we kept $K_{GMM} = 3$ for a training dataset of 20 users, all the entries were detected as normal. Thus, K_{GMM} needs to be decreased, to make the system more sensitive. In the trial with 5 users, K_{GMM} needs to be increased to 30, since none of the entries were detected as normal with $K_{GMM} = 3$, thus the system is less sensitive. We found that the best values for K_{GMM} were 30, 3 and 0.1 for 5, 10 and 20 users respectively. The curiosity factor for K-Means, K_{Kmeans}, on the other hand, was suitable for all the trials.

In our preliminary findings we found good values for K_{GMM} when followed:

$$K_{GMM} = 300e^{-0.45x}$$

where x is the size of the dataset containing the known instances. However, the finding a general rule for K depending on data still needs further work. In future work, we will assess further how K affects the interpretation of novel stimuli.

Our work also needs to explore more how well our system will scale to a greater number of classes. Despite our system had to face to up to 8 different novelties, we want to known how its performance varies as new novelties are being discovered and incorporated to the model. If the system is designed to be continuously acquiring knowledge, it is expected that, over time, it will be discovering new poses that should be learnt. Eventually, it will have to handle a big number of classes. Despite the algorithms that we used already work well with big datasets, we need to understand how the system as a whole behaves in such situations.

Another limitation of our system is that we learn with a predefined dataset. Despite our approach is valid for the input data, we did not tested our system in real time. We still have to see how people reacts to the robot's questions for new data. In this way, there are two metrics that might be evaluated. First, from the robot's point of view, replicating our work, but in real scenario, would produce a better estimate of how well is our system to work in real set ups. Second, from the user point of view, it is interesting to know how robot's questions affect the people's perception on the robot's intelligence and capabilities, especially in the long term where people can see how robot's knowledge evolves over time. Our further work would focus in these two points of view, together with a further exploration of how the K parameter affects learning.

7.7 Conclusions

We presented a system that endows a robot with the capacity of actively deciding whether to learn or ignore novel stimuli when exposed to it for the first time. We separate this decision in two steps. First by deciding if a visual stimuli is interesting, i.e. it is worth to be learnt; and second by deciding whether the current model is able to predict it confidently, i.e. the stimuli is strange to the model. In that way, if a stimuli is both interesting and strange, the robot can actively ask the human for a label tagging the new data. Both steps are based in Novelty Detection algorithms with an extension to parametrize how much curious the robot should be.

Our system has been tested in the application of pose recognition, in which the system learns the poses adopted by a human teacher. To validate our approach, we evaluated our system with 28 non-robotics experts training the robot three different poses. We compared four novelty detection algorithms, for both the noise filter (interestingness filter) and the strangeness detection filter and we found that GMM and K-means are more suitable for the noise filter, while GMM stands out as a strangeness filter

The main advantage of our approach is the use of the curiosity factor K. This factor enables the robot to adapt the algorithm to specific application domains. That is, K enables to robot to be very sensitive or insensitive to novelties depending on the application requirements. For instance, we tuned K to make the robot insensitive to new stimuli until these stimuli are presented a determined number of times.

However, using K as a parameter to model the robot's curiosity presents two main issues that have yet to be solved. First, finding a good value for K has to be done empirically. Therefore, the designers of the system must to tune manually this parameter before the system can be applied in a certain domain. Second, as the knowledge base of the robot grows, the robot starts becoming less sensitive to new stimuli. One way to mitigate this is by automatically adapting K to the training dataset size but, since the value of K has to be defined manually, the designers have to calculate beforehand how different values of K will affect the robot's curiosity as it is gaining more knowledge.

Part IV

Conclusions

Chapter 8

Conclusions

This thesis has contributed to the field of robot interactive and natural learning by combining techniques from the fields of Machine Learning, Natural Language Processing, and Human Robot Interaction, among other fields. This chapter concludes this thesis by summarizing its main contributions and by presenting the future works that remain open or that have been opened by these contributions.

8.1 Conclusions

The contributions of this thesis enabled a social robot to learn while interacting with a human teacher in a similar way as people learn from other people. We tested our interactive learning approaches in two supervised learning settings. In the first one, the robot had to learn to classify human poses while in the second one the task consisted in classifying different objects the user was holding in his/her hand.

A major contribution of this thesis has been improving the natural, interactive learning capabilities of the robot with the inclusion of Active Learning techniques. The aim of this was to enable the robot to learn better and faster from the examples it gathers during the training session. Finally, we adapted these techniques to enable the robot to keep learning beyond the training session. That is, the robot is able to decide whether it needs more training data or labels and ask for them, even after the training phase has finished. This enables the robot to detect when there are situations that might require more learning and ask the user for more training data (or labels) before the user notifies that the robot has this learning necessity.

8.2 Summary of the Key Contributions

The main contributions of this thesis are summarized as follows:

Regarding interactive learning: We developed a system that endows a robot to learn while interacting with people. For such system we developed two interaction approaches:

- The first one focused on understanding the user using a grammar-based ASR where the semantics of what the robot's trainer tells to the robot were predefined in grammar text-files. This was the limiting factor of the interactive system since the trainer could not teach the robot anything that was not pre-written in the grammar files.

- The second approach consisted in using a dialogue management system combined with some Natural Language Processing techniques. This approach, despite not using a grammar-based ASR, increased the flexibility of the system in two different ways. First, by overcoming the limitation of the number of objects, and second, by increasing the naturalness of the interaction in both directions of the communication: understanding the user and talking to him/her.

Regarding interactive object recognition: We developed a system that is able to recognize objects that the user is holding in his/her hand. The system uses both RGB (Red-Green-Blue) images and depth dapta in form of point clouds. Combining both data improves the robustness of the system to illumination changes or noise in the input data stream. Also, in our pre-processing we used Point Cloud data to define the Region Of Interest (ROI) where to carry out the object detection. This enabled us to have a system that is able to operate in real time.

Regarding Active Learning for pose learning: Here, we contributed in two different manners:

- We studied how Active Learning can be applied in interactive learning by studying the impact of the user answers to the robot's queries when using these queries to introduce domain knowledge from the trainer as feature filters. In our experiments we observed that there are some cases where the trainer answers can be vague and, hence, produce feature filters that over generalize and produce simplistic models.

- We proposed a method that overcomes this problem by reducing the trust in the user's answers. This method compensates the over-generalization of the user by extending the user-defined filter to other features that have a correlation with the user's (in our case, body-joints that were directly connected). Our method has proven to work better than active-learners in such situations at the same time that produced results that were at the same level as active learners when the user produced good and informative answers.

Regarding Active Learning for object detection and tracking: This chapter has presented two main contributions. The first one is the use of active learning to enable the robot to keep learning once the training session has finished. The second contribution is the interaction system of the robot, which combines a Dialogue Manager, an open-grammar ASR, and some self-developed Natural Language Processing (NLP). The combination of these three modules enabled the robot to establish natural and rich interactions with the user, enabling him/her to use a rich vocabulary of sentences.

Regarding Novelty detection for pose recognition: We presented a system that enables the robot to keep learning after the training session ended. The system endows the robot to actively decide whether new stimuli is interesting (is not noise), and strange according to a pre-learnt model. For that we used novelty detection algorithms that evaluate incoming data following these two principles. If a new stimulus results to be interesting and the learned model decides that is strange to it, the robot can ask the user a query to tag this data.

8.3 Future Works

Interactive pose learning Our work leaves other paths open for exploration:

- First, from the HRI point of view, this thesis has focused on the HRI from the robot's point of view. It remains to study how users perceive what the robot has learned and how this fact changes their relation and their expectations towards it. Even more, understanding what the user thinks about the learning process might lead to better training scenarios that would end in robots that learn better from the users.

- Second, this work opens the door for building a continuous learning framework, where the robot actively seeks for new examples and asks questions of its teacher about the concepts being learned.

- Third, with only a few minor modifications, this learning system can be extended to other applications, such as gesture learning, activity learning or the interactive learning of new objects.

In-hand object tracking and recognition Most of the future works for the in-hand object tracking and recognition are related to the computer vision field. The most interesting future work in this are can be:

- Using better feature extractors. This might be interesting specially in the case of the Point Cloud (PC). The PC features are extracted using many approximations in order to reduce the huge computing time they require. Hence, they are less representative leading to more potential classification errors in the matching process. Another potential upgrade of the system is to use other feature extractors for the Point Cloud. Some new algorithms such as LINEMOD [Hinterstoisser et al., 2012] might be suitable for our approach For instance, LINEMOD can extract features of texture-less objects. This might help improve our matchers' performance in the cases where there are many texture-less objects such as house-hold objects, etc.

- A second improvement to the in-hand object detection system is to use other computer vision methods for learning the objects. We followed a template matching approach as our first steps into the field of in-hand object recognition. Although we achieved good accuracy results in our dataset, we expect to expand the learning capabilities of the system by using machine learning algorithms such as Random Forests [Gall et al., 2012] or Support Vector Machines (SVMs) [Pontil and Verri, 1998]. Specially interesting might be the case of Random Forests, since they are an ensemble which can provide easily some committee-based heuristics that can be used in active learning scenarios.

Active Learning in Pose Recognition Regarding our active learning contributions, several future work are still open.

- Even though our method for reducing the robot's trust on the user's answers is applied for pose learning, we believe that this method can be applied to other active learning-based approaches providing that the robot knows the relation (or correlation) between the features which is asking for. In our case those relation was spatial—i.e. body joints that are directly interconnected—, but other type of correlation between features might serve. Knowing these relationships, the robot might extend the user answers to other related features, thus reducing the effects of his/her inaccuracies. Our method might help robots to learn better from people who are not expert in robotics. If the robot expects that the user answer might not be accurate, it can choose to apply our Extended Filter instead of discarding it.
- Also, it is unclear how to apply this method when different types of feature queries are combined. Because Free Speech Queries (FSQ) produce simpler answers than Yes/No Queries (YNQ) and Rank Queris (RQ), it might be interesting to apply different filters to each type of questions. In the cases in which the robot expects inaccurate answers, it can increase the filter extension just to reduce the side-effects of these inaccuracies. Besides, we want to explore the use of automatic feature selection algorithms together with our extended filter. For instance, if the user answer and the automatic feature selection differ too much, it might mean whether the user gave a bad answer or the robot does not have enough training data. In such cases, it might be wise to extend the user's answer and/or ask more queries.

Active learning for in-hand object tracking and recognition Regarding the active learning for object recognition and tracking, there are some works that might be worth pursuing. Among them, the two most interesting would be

- First, to evaluate how the user perceives a robot that it has the initiative to learn continuously. And how different degrees of verbosity might be perceived from the user perspective.
- Additionally, it might be interesting to study which methods could be used to control the number of questions the robot asks to the user.

Novelty Detection for continuous pose learning Finally, in the case of our novelty detection approach some possible future works are:

- The most interesting future work is to apply it to other fields, for instance, applying it in our object recognition system.

Also, using K as a parameter to model the robot's curiosity presents two main issues that have yet to be solved.

- First, finding a good value for K has yet to be done empirically. As a consequence, the system designer must tune manually K before the system can be applied in a certain domain.
- Second, an implication of how we defined K is that, as the knowledge base of the robot grows, the robot becomes less sensitive to new stimuli. One way to mitigate this is by automatically adapting K to the training dataset size but, since the value of K has to be defined manually, the designers have to calculate beforehand how different values of K will affect the robot's curiosity as it gains more knowledge.

8.4 List of Publications

8.4.1 Journals

This section presents a list of journals that have been published during this thesis. The three journals are indexed in the Journal Citation Reports (JCR), being one in the first quartile (Q1) and two in the third quartile (Q3).

- **V. Gonzalez-Pacheco**, A. Sanz, M. Malfaz, and M. a. Salichs, "*Novelty Detection for Interactive Pose Recognition by a Social Robot*," Int. J. Adv. Robot. Syst., vol. 12, no. 43, p. 1, 2015.
- **V. Gonzalez-Pacheco**, M. Malfaz, F. Fernandez, and M. A. Salichs, "*Teaching human poses interactively to a social robot*," Sensors, vol. 13, no. 9, pp. 12406–12430, 2013.
- **V. Gonzalez-Pacheco**, A. Ramey, F. Alonso-Martin, A. Castro-Gonzalez, and M. A. Salichs, "*Maggie: A Social Robot as a Gaming Platform*," Int. J. Soc. Robot., vol. 3, no. 4, pp. 371–381, Sep. 2011.

8.4.2 Conferences and Talks

- **V. Gonzalez-Pacheco**, A. Sanz, M. Malfaz, and M. A. Salichs, "*Using novelty detection in HRI: Enabling robots to detect new poses and actively ask for their labels*," in 2014 IEEE-RAS International Conference on Humanoid Robots, 2014, pp. 1110–1115.

- **V. Gonzalez-Pacheco**, M. Malfaz, and M. A. Salichs, "*Asking rank queries in pose learning*," in Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14, 2014, pp. 164–165.

- **V. Gonzalez-Pacheco** and M. A. Salichs, "*Active Learning for Pose Recognition. Studying what and when to ask for Feature Queries*," in Proc of the 8th HRI Pioneers Workshop, 2013, pp. 3–4.

- J. Sequeira, P. Lima, A. Saffiotti, **V. Gonzalez-Pacheco**, and M. A. Salichs, "*MOnarCH: Multi-Robot Cognitive Systems Operating in Hospitals*," in Proc of the ICRA 2013 Workshop on Crossing the Reality Gap – From Single to Multi to Many Robot Systems, 2013, p. 1.

- A. Valero-Gomez, J. Gonzalez-Gomez, **V. Gonzalez-Pacheco**, and M. A. Salichs, "*Printable creativity in plastic valley UC3M*," in Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON), 2012, pp. 1–9.

- A. Ramey, **V. González-Pacheco**, and M. A. Salichs, "*Integration of a low-cost RGB-D sensor in a social robot for gesture recognition*," in Proceedings of the 6th international conference on Human-Robot Interaction - HRI '11, 2011, pp. 229–230.

- F. Alonso-Martin, **V. Gonzalez-Pacheco**, A. Castro-Gonzalez, A. A. Ramey, M. Yébenes, and M. A. Salichs,"*Using a social robot as a gaming platform*," in 2nd International Conference on Social Robotics, 2010, pp. 30–39.

References

N. Abe and H. Mamitsuka. Query learning strategies using boosting and bagging. In *Proceedings of the 25th International Conference on Machine learning*, volume 388, pages 1–9. Morgan Kaufmann, 1998. ISBN 1558605568. 20

M. A.F. Pimentel, D. a. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, June 2014. ISSN 01651684. doi: 10.1016/j.sigpro.2013.12.026. 125, 130, 131, 134, 135

F. Alonso, J. Gorostiza, and M. Salichs. Preliminary Experiments on HRI for improvement the Robotic Dialog System (RDS). In *Robocity2030 11th Workshop: Robots Sociales*, Leganes, Spain, 2013. 95

F. Alonso-Martín and M. A. Salichs. INTEGRATION OF A VOICE RECOGNITION SYSTEM IN A SOCIAL ROBOT. *Cybernetics and Systems*, 42(4):215–245, May 2011. ISSN 0196-9722. doi: 10.1080/01969722.2011.583593. 38, 89, 95, 136

F. Alonso-Martin, A. A. Ramey, and M. A. Salichs. Maggie : el robot traductor. In UPM, editor, *9th Workshop RoboCity2030-II*, number Breazeal 2003, pages 57–73, Madrid, May 2011. Robocity 2030. 38

D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, Apr. 1988. ISSN 0885-6125. doi: 10.1007/BF00116828. 85, 87

B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009. ISSN 09218890. doi: 10.1016/j.robot.2008.10.024. 4, 34

R. Barber. *Desarrollo de una arquitectura para robots móviles autónomos. Aplicación a un sistema de navegación topológica.* Phd thesis, Universidad Carlos III de Madrid, 2000. 37

H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, 2006. 62

A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100, 1998. ISBN 1581130570. doi: 10.1145/279943.279962. 26

R. Boulic, J. Varona, L. Unzueta, M. Peinado, A. Suescun, and F. Perales. Evaluation of on-line analytic and numeric inverse kinematics approaches driven by partial vision input. *Virtual Reality*, 10(1):48–61, 2006. ISSN 1359-4338. doi: 10.1007/s10055-006-0024-8. 33

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 66

L. Breiman. Bagging predictors, 1996. ISSN 0885-6125. 20

L. Breiman. Random Forests. *Mach. Learn.*, 45(1):5–32, 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. 50, 91, 102

P. Breuer, C. Eckes, and S. Müller. Hand Gesture Recognition with a Novel IR Time-of-Flight Range Camera - A Pilot Study. In A. Gagalowicz and W. Philips, editors, *Computer Vision/Computer Graphics Collaboration Techniques*, volume 4418 of *Lecture Notes in Computer Science*, pages 247–260. Springer Berlin / Heidelberg, 2007. doi: http://dx.doi.org/10.1007/978-3-540-71457-6_23. 33

M. Cakmak and A. L. Thomaz. Designing robot learners that ask good questions. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction - HRI '12*, page 17, New York, New York, USA, 2012. ACM Press. ISBN 9781450310635. doi: 10.1145/2157689.2157693. 85, 87, 88, 92

M. Cakmak, C. Chao, and A. L. Thomaz. Designing Interactions for Robot Active Learners. *IEEE Transactions on Autonomous Mental Development*, 2(2):108–118, June 2010. ISSN 1943-0604. doi: 10.1109/TAMD.2010.2051030. 85, 86, 87, 104, 106, 124

M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, 2010. 62

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):1–58, 2009. 125, 126, 127

L. Chen, H. Wei, and J. Ferryman. A survey of human motion analysis using depth imagery. *Pattern Recognition Letters*, (0):–, 2013. ISSN 0167-8655. doi: 10.1016/j.patrec.2013.02.006. 33

D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15:201–221, 1994. ISSN 08856125. doi: 10.1007/BF00993277. 19

S. Dasgupta and D. Hsu. Hierarchical sampling for active learning. In *Proceedings of the 25th international conference on Machine Learning - ICML '08*, pages 208–215. ACM, 2008. ISBN 9781605582054. doi: 10.1145/1390156.1390183. 24, 25

J. de Greeff, F. Delaunay, and T. Belpaeme. Human-Robot Interaction in Concept Acquisition: a computational model. In *Development and Learning, 2009. ICDL 2009. IEEE 8th International Conference on*, pages 1–6, 2009. doi: 10.1109/DEVLRN.2009.5175532. 35, 86

V. Delaitre, J. Sivic, and I. Laptev. Learning person-object interactions for action recognition in still images. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 1503–1511. Curran Associates, Inc., 2011. 58

X. Ding, Y. Li, A. Belatreche, and L. P. Maguire. An experimental evaluation of novelty detection methods. *Neurocomputing*, 135:313–327, July 2014. ISSN 09252312. doi: 10.1016/j.neucom.2013.12.002. 127

D.M.Gavrila and F. Groen. 3D object recognition from 2D images using geometric hashing. 1991. 64

P. Drews, P. Núñez, R. P. Rocha, M. Campos, and J. Dias. Novelty detection and segmentation based on Gaussian mixture models: A case study in 3D robotic laser mapping. *Robotics and Autonomous Systems*, 61(12):1696–1709, Dec. 2013. ISSN 09218890. doi: 10.1016/j.robot.2013.06.004. 128

D. Droeschel, J. Stückler, and S. Behnke. Learning to interpret pointing gestures with a time-of-flight camera. In *Proceedings of the 6th international conference on Human-robot interaction*, HRI '11, pages 481–488, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0561-7. doi: http://doi.acm.org/10.1145/1957656.1957822. 33

G. Druck, B. Settles, and A. McCallum. Active learning by labeling features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 81–90. Association for Computational Linguistics, 2009. 87

A. Fathi, Y. Li, and J. Rehg. Learning to recognize daily actions using gaze. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *Computer Vision – ECCV 2012*, volume 7572 of *Lecture Notes in Computer Science*, pages 314–327. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33717-8. doi: 10.1007/978-3-642-33718-5_23. 58

S. Foix, G. Alenya, and C. Torras. Lock-in Time-of-Flight (ToF) Cameras: A Survey. *IEEE Sensors Journal*, 11(99):1, 2011. ISSN 1530-437X. doi: 10.1109/JSEN.2010.2101060. 32

T. Fong, I. Nourbakhsh, and K. Dautenhahn. A survey of socially interactive robots. *Robotics and Autonomous Systems*, 42(3-4):143–166, Mar. 2003. ISSN 09218890. doi: 10.1016/S0921-8890(02)00372-X. 34, 85

B. Freedman, A. Shpunt, M. Machline, and Y. Arieli. Depth mapping using projected patterns, 2008. 32, 36

Y. Freund and R. E. Schapire. A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. *Journal of Computing Systems and Science*, 55:119–139, 1997. ISSN 00220000. doi: 10.1007/3-540-59119-2_166. 20

A. Fujii, T. Tokunaga, K. Inui, and H. Tanaka. Selective Sampling for Example-based Word Sense Disambiguation. *Comput. Linguist.*, 24(4):573–597, 1998. ISSN 0891-2017. 23

K. Fujimura. Hand gesture recognition using depth data. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.*, pages 529–534. IEEE, May 2004. ISBN 0-7695-2122-3. doi: 10.1109/AFGR.2004.1301587. 33

J. Gall, N. Razavi, and L. V. Gool. An introduction to random forests for multi-class object detection. *Outdoor and Large-Scale Real-World*, pages 1–21, 2012. 80, 154

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995. 37

C. Goerick, J. Schmudderich, B. Bolder, H. Janssen, M. Gienger, A. Bendig, M. Heckmann, T. Rodemann, H. Brandl, X. Domont, and I. Mikhailova. Interactive online multimodal association for internal concept building in humanoids. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 411–418. IEEE, 2009. ISBN 978-1-4244-4597-4. doi: 10.1109/ICHR.2009.5379549. 35

S. B. Gokturk and C. Tomasi. 3D head tracking based on recognition and interpolation using a time-of-flight depth sensor. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages 211–217, 2004. doi: 10.1109/CVPR.2004.1315166. 33

V. Gonzalez-Pacheco, M. Malfaz, F. Fernandez, and M. a. Salichs. Teaching human poses interactively to a social robot. *Sensors (Basel, Switzerland)*, 13(9):12406–30, Jan. 2013. ISSN 1424-8220. doi: 10.3390/s130912406. 31, 98, 129, 136, 138

V. Gonzalez-Pacheco, A. Sanz, M. Malfaz, and M. A. Salichs. Using novelty detection in HRI: Enabling robots to detect new poses and actively ask for their labels. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 1110–1115. IEEE, 2014. ISBN 978-1-4799-7174-9. doi: 10.1109/HUMANOIDS.2014.7041507. 123

V. Gonzalez-Pacheco, A. Sanz, M. Malfaz, and M. a. Salichs. Novelty Detection for Interactive Pose Recognition by a Social Robot. *International Journal of Advanced Robotic Systems*, 12(43):1, 2015. ISSN 1729-8806. doi: 10.5772/60057. 123

M. a. Goodrich and A. C. Schultz. Human-Robot Interaction: A Survey. *Foundations and Trends® in Human-Computer Interaction*, 1(3):203–275, 2007. ISSN 1551-3955. doi: 10.1561/1100000005. 34, 85

Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pages 529–536. MIT Press, 2005. 26

E. Gribovskaya, F. D'Halluin, and A. Billard. An active learning interface for bootstrapping robot's generalization abilities in learning from demonstration. In *RSS Workshop Towards Closing the Loop: Active Learning for Robotics*, 2010. 86

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11 (1):10–18, 2009. 40, 45, 91

N. Haubner, U. Schwanecke, R. Dörner, S. Lehmann, and J. Luderschmidt. Recognition of Dynamic Hand Gestures with Time-of-Flight Cameras. In *ITG / GI Workshop on Self-Integrating Systems for Better Living Environments 2010: SENSYBLE 2010*, pages 1–7, 2010. 33

M. Heckmann, H. Brandl, J. Schmuedderich, X. Domont, B. Bolder, I. Mikhailova, H. Janssen, M. Gienger, A. Bendig, T. Rodemann, M. Dunn, F. Joublin, and C. Goerick. Teaching a humanoid robot: Headset-free speech interaction for audio-visual association learning. In *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 422–427. IEEE, 2009. ISBN 978-1-4244-5081-7. doi: 10.1109/ROMAN.2009.5326338. 35

S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE transactions on pattern analysis and machine intelligence*, 34(5):876–88, May 2012. ISSN 1939-3539. doi: 10.1109/TPAMI.2011.206. 79, 154

M.-K. Hu. Visual Pattern Recognition by Moment Invariants. pages 66–70, 1962. 64

M. S. J.A. Quinn. A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters*, 40:36–40, 2014. 135, 138

A. Jaume-i Capó and J. Varona. Representation of human postures for vision-based gesture recognition in real-time. *Gesture-Based Human-Computer*, pages 102–107, 2009. 32

G. H. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, UAI'95, pages 338–345, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-385-9. 50

J. Kang, K. Ryu, and H.-C. Kwon. Using cluster-based sampling to select initial training set for active learning in text classification. In H. Dai, R. Srikant, and C. Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 384–388. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22064-0. doi: 10.1007/978-3-540-24775-3_46. 24

K. Khoshelham and S. O. Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437, 2012. ISSN 1424-8220. doi: 10.3390/s120201437. URL `http://www.mdpi.com/1424-8220/12/2/1437`. 77

A. Kolb, E. Barth, R. Koch, and R. Larsen. Time-of-flight sensors in computer graphics. In *Eurographics State of the Art Reports*, pages 119–134, 2009. 33

S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. ISSN 0003-4851. doi: 10.1214/aoms/1177729694. 21

H. Lahamy and D. Litchi. Real-time hand gesture recognition using range cameras. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences [on CD-ROM]*, page 38, 2010. 33

R. Lomasky, C. E. Brodley, M. Aernecke, D. Walt, and M. Friedl. Active Class Selection. In *Proceedings of the 18th European conference on Machine Learning*, ECML '07, pages 640–647, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74957-8. doi: 10.1007/978-3-540-74958-5_63. 87

M. Lopes and P.-Y. Oudeyer. Guest Editorial Active Learning and Intrinsically Motivated Exploration in Robots: Advances and Challenges. *IEEE Transactions on Autonomous Mental Development*, 2(2):65–69, 2010. ISSN 1943-0604. doi: 10.1109/TAMD.2010.2052419. 86

D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. 61

S. Mahadevan, G. Theocharous, and N. Khaleeli. Rapid Concept Learning for Mobile Robots. *Autonomous Robots*, 5(3):239–251, 1998. ISSN 0929-5593. 34

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-13360-1. 22

R. Martinez-Cantin, J. Peters, and A. Krause, editors. *Robotics Science and Systems (RSS'10) Workshop on Towards Closing the Loop: Active Learning for Robotics*. Zaragoza, Spain, 2010. 86

A. K. McCallum and K. Nigam. Employing EM and pool-based active learning for text classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 359–367. Morgan Kaufmann, 1998. 23, 26

P. Melville and R. J. Mooney. Diverse ensembles for active learning. In *International Conference on Machine Learning*, volume 69, pages 584–591, 2004. ISBN 1-58113-828-5. doi: 10.1145/1015330.1015385. 20

P. Melville, S. M. Yang, M. Saar-Tsechansky, and R. Mooney. Active learning for probability estimation using jensen-shannon divergence. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3720 LNAI, pages 268–279, 2005. ISBN 3540292438. doi: 10.1007/11564096_28. 22

K. E. Merrick. A Comparative Study of Value Systems for Self-Motivated Exploration and Learning by Robots. *IEEE Transactions on Autonomous Mental Development*, 2 (2):119–131, June 2010. ISSN 1943-0604. doi: 10.1109/TAMD.2010.2051435. 86

K. Mikolajczyk and C. Schmid. Performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–30, Oct. 2005. ISSN 0162-8828. doi: 10.1109/TPAMI.2005.188. 61

O. Miksik and K. Mikolajczyk. Evaluation of local detectors and descriptors for fast feature matching. *Pattern Recognition (ICPR), 2012 21st*, 2012. 79

T. M. Mitchell. Generalization as search, 1982. ISSN 00043702. 18

S. Mitra and T. Acharya. Gesture Recognition: A Survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 37(3):311–324, May 2007. ISSN 1094-6977. doi: 10.1109/TSMCC.2007.893280. 32

M. Muja and D. G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009. 68

I. Muslea, I. Muslea, S. Minton, S. Minton, C. A. Knoblock, and C. Knoblock. Selective sampling with redundant views. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 621–626. AAAI Press, 2000. ISBN 0262511126. doi: citeulike-article-id:1727957. 20

I. Muslea, S. Minton, and C. A. Knoblock. Active + Semi-Supervised Learning = Robust Multi-View Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, number 1998, pages 435–442, 2002. ISBN 1558608737. 26

U. Nehmzow, Y. Gatsoulis, E. Kerr, J. Condell, N. Siddique, and T. M. Mcginnity. Intrinsically Motivated Learning in Natural and Artificial Systems. pages 185–207, 2013. doi: 10.1007/978-3-642-32375-1. 128, 134

G. Ngai and D. Yarowsky. Rule Writing or Annotation: Cost-efficient Resource Usage for Base Noun Phrase Chunking. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 117–125, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics. doi: 10.3115/1075218.1075234. 22

H. T. Nguyen and A. Smeulders. Active learning using pre-clustering. In *International Conference on Machine Learning Conference on Machine Learning (ICML)*, pages 79–86. ACM, 2004. ISBN 1581138285. doi: 10.1145/1015330.1015349. 24

K. Nickel and R. Stiefelhagen. Visual recognition of pointing gestures for human-robot interaction. *Image and Vision Computing*, 25(12):1875–1884, 2007. ISSN 02628856. doi: 10.1016/j.imavis.2005.12.020. 33

M. Nicolescu and M. Mataric. Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 31 (5):419–430, 2001. ISSN 10834427. doi: 10.1109/3468.952716. 35

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 134, 135, 138

M. Philipose. Egocentric recognition of handled objects: Benchmark and analysis. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8, June 2009. doi: 10.1109/CVPRW.2009.5204360. 65

A. Pinto, A. Pronobis, and L. Reis. Novelty detection using graphical models for semantic room classification. In L. Antunes and H. Pinto, editors, *Progress in Artificial Intelligence*, volume 7026 of *Lecture Notes in Computer Science*, pages 326–339. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24768-2. doi: 10.1007/978-3-642-24769-9_24. 128

M. Pontil and a. Verri. Support vector machines for 3D object recognition, June 1998. ISSN 01628828. 80, 154

M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, number Figure 1, Kobe, May 2009. 37, 66, 95

J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers, 1993. ISBN 9781558602380. 50, 91, 102

H. Raghavan, O. Madani, and R. Jones. Active Learning with Feedback on Features and Instances. *J. Mach. Learn. Res.*, 7:1655–1686, 2006. ISSN 1532-4435. 87

A. Ramey, V. González-Pacheco, and M. A. Salichs. Integration of a low-cost RGB-D sensor in a social robot for gesture recognition. In *Proceedings of the 6th international conference on Human-Robot Interaction - HRI '11*, HRI '11, pages 229–230, New York, New York, USA, 2011. ACM Press. ISBN 9781450305617. doi: 10.1145/1957656.1957745. 33

S. Reese, G. Boleda, M. Cuadros, L. Padró, and G. Rigau. Wikicorpus: A word-sense disambiguated multilingual Wikipedia corpus. In *In Proceedings of 7th Language Resources and Evaluation Conference (LREC'10)*, pages 1418–1421, 2010. ISBN 2-9517408-6-7. 113

T. H. Reiss. The revised fundamental theorem of moment invariants. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(8):830–834, Aug. 1991. ISSN 0162-8828. doi: 10.1109/34.85675. 64

C. Rich, C. Sidner, and N. Lesh. Collagen: Applying Collaborative Discourse Theory to Human-Computer Interaction. *AI magazine*, 2001. 114

R. Rivas, A. Corrales, R. Barber, MA, and M. A. S. R. Rivas, A. Corrales, R. Barber. Robot skill abstraction for ad architecture. *6th IFAC Symposium on Intelligent Autonomous Vehicles*, 47(4):12–13, 2007. 37

S. Rosenthal, A. Dey, and M. Veloso. How robots' questions affect the accuracy of the human responses. *Robot and Human Interactive ...*, 2009. 85, 87, 88, 92

S. Rosenthal, M. Veloso, and A. K. Dey. Acquiring Accurate Human Responses to Robots' Questions. *International Journal of Social Robotics*, 4(2):117–129, 2012. ISSN 1875-4791. doi: 10.1007/s12369-012-0138-y. 85, 87

P. Rosin. Measuring corner properties. *Computer Vision and Image Understanding*, 1999. 62

E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, pages 1–14, 2006. 62

P. Roth, M. Donoser, and H. Bischof. On-line learning of unknown hand held objects via tracking. *Int. Conf. on Computer Vision*, 2006. 65

N. Roy and A. McCallum. Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *Proc 18th International Conf on Machine Learning*, pages 441–448, 2001. ISBN 1558607781. 26

E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571. IEEE, Nov. 2011. ISBN 978-1-4577-1102-2. doi: 10.1109/ICCV.2011.6126544. 62, 67, 68

R. Rusu, N. Blodow, Z. Marton, and M. Beetz. Aligning point cloud views using persistent feature histograms. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3384–3391, Sept. 2008. doi: 10.1109/IROS.2008.4650967. 63, 66

R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. 66

R. B. Rusu, N. Blodow, and M. Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. 64

P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso. Interactive robot task training through dialog and demonstration. In *Proceeding of the ACM/IEEE international conference on Human-robot interaction - HRI '07*, pages 49–56, New York, New York, USA, 2007. ACM Press. ISBN 9781595936172. doi: 10.1145/1228716.1228724. 35

M. A. Salichs, R. Barber, A. M. Khamis, M. Malfaz, J. F. Gorostiza, R. Pacheco, R. Rivas, A. Corrales, E. Delgado, D. G. Maggie, A, and D. Garcia. Maggie: A Robotic Platform for Human-Robot Social Interaction. *In Submitted to IEEE International Conference* *on Robotics, Automation and Mechatronics (RAM*, pages 1–7, 2006. doi: 10.1109/RAMECH.2006.252754. 4, 32, 36, 95

D. Scharstein and R. Szeliski. High-Accuracy Stereo Depth Maps Using Structured Light. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1: 195, 2003. ISSN 1063-6919. doi: http://doi.ieeecomputersociety.org/10.1109/CVPR.2003.1211354. 32

L. A. Schwarz, A. Mkhitaryan, D. Mateus, and N. Navab. Human skeleton tracking from depth data using geodesic distances and optical flow. *Image and Vision Computing*, (0):—-, 2011. ISSN 02628856. doi: 10.1016/j.imavis.2011.12.001. 33

P. Scovanner, S. Ali, and M. Shah. A 3-dimensional sift descriptor and its application to action recognition. *Proceedings of the 15th international conference on Multimedia - MULTIMEDIA '07*, (c):357, 2007. doi: 10.1145/1291233.1291311. 63

B. Settles. Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2010. 85

B. Settles. Active Learning, 2012. ISSN 1939-4608. 7, 11, 12, 14, 16, 17, 18, 19, 22, 23, 24, 25, 26

B. Settles and M. Craven. An analysis of active learning strategies for sequence labeling tasks. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1069–1078, 2008. doi: 10.3115/1613715.1613855. 23, 24

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory - COLT '92*, pages 287–294, 1992. ISBN 089791497X. doi: 10.1145/130385.130417. 20

C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423, 1948. ISSN 00058580. doi: 10.1002/j.1538-7305.1948.tb01338.x. 15

L. G. Shapiro and G. C. Stockman. *Computer Vision*. Prentice-Hall, New Jersey, USA, 2001. ISBN 0-13-030796-3. 61

A. F. Sheta, A. Baareh, and M. Al-Batah. 3D object recognition using fuzzy mathematical modeling of 2D images. *2012 International Conference on Multimedia Computing and Systems*, pages 278–283, May 2012. doi: 10.1109/ICMCS.2012.6320118. 64

S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. Improvements to the SMO algorithm for SVM regression. *Neural Networks, IEEE Transactions on*, 11 (5):1188–1193, Sept. 2000. ISSN 1045-9227. doi: 10.1109/72.870050. 50, 91, 103

J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, Colorado Springs, CO, June 2011. IEEE. ISBN 978-1-4577-0394-2. doi: 10.1109/CVPR.2011.5995316. 33

S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically Motivated Reinforcement Learning: An Evolutionary Perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, June 2010. ISSN 1943-0604. doi: 10.1109/TAMD.2010.2051031. 86

T. D. Smedt and W. Daelemans. Pattern for Python. *Journal of Machine Learning Research*, 13:2063–2067, 2012. ISSN 1532-4435. 113

M. Teague. Image analysis via the general theory of moments. *J. Optical Soc. Am.*, (vol. 70, no. 8):920–930, 1980. 64

A. Thomaz, G. Hoffman, and C. Breazeal. Reinforcement Learning with Human Teachers: Understanding How People Want to Teach Robots. In *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*, pages 352–357. IEEE, 2006. ISBN 1-4244-0564-5. doi: 10.1109/ROMAN.2006.314459. 85

A. L. Thomaz and C. Breazeal. Robot learning via socially guided exploration. *2007 IEEE 6th International Conference on Development and Learning*, pages 82–87, July 2007. doi: 10.1109/DEVLRN.2007.4354078. 85

A. L. Thomaz and C. Breazeal. Experiments in socially guided exploration: lessons learned in building robots that learn with and without human teachers. *Connection Science*, 20(2-3):91–110, Sept. 2008. ISSN 0954-0091. doi: 10.1080/09540090802091917. 85

K. Tomanek and U. Hahn. Semi-supervised active learning for sequence labeling. *Proceedings of the Joint Conference of the 47th . . .*, (August):1039–1047, 2009. doi: 10.3115/1690219.1690291. 26

G. Tur, D. Hakkani-Tür, and R. E. Schapire. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2):171–186, 2005. ISSN 01676393. doi: 10.1016/j.specom.2004.08.002. 27

C. J. Van Rijsbergen. *Information Retrieval.* Butterworth, 2nd ed. edition, 1979. 142

I. H. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann series in data management systems. Elsevier Science & Technology, 3rd edition, 2011. ISBN 9780123748560. 49

Z. Xu, R. Akella, and Y. Zhang. Incorporating Diversity and Density in Active Learning for Relevance Feedback. In *ECIR European Conference on Information Retrieval*, volume 4425, pages 246–257, 2007. ISBN 978-3-540-71494-1. doi: 10.1007/978-3-540-71496-5_24. 24

D. Yarowsky. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196, 1995. ISSN 0736587X. doi: 10.3115/981658.981684. 26

K. Yu, J. Bi, and V. Tresp. Active learning via transductive experimental design. *Proceedings of the 23rd international conference on Machine learning ICML 06*, 148(6): 1081–1088, 2006. doi: 10.1145/1143844.1143980. 27

Z. Zalevsky, A. Shpunt, A. Maizels, and J. Garcia. Method and System for Object Reconstruction, 2007. 36

Z.-H. Zhou, K.-J. Chen, and Y. Jiang. Exploiting unlabeled data in content-based image retrieval. In *Proceedings of ECML-04, 15th European Conference on Machine Learning*, pages 525–536, 2004. 27

X. Zhu and A. B. Goldberg. Introduction to Semi-Supervised Learning, 2009. ISSN 1939-4608. 25

X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, pages 58–65, 2003. 27

Y. Zhu, B. Dariush, and K. Fujimura. Kinematic self retargeting: A framework for human pose estimation. *Computer Vision and Image Understanding*, 114(12):1362–1375, 2010. ISSN 10773142. doi: 10.1016/j.cviu.2009.11.005. 33

M. Z. Zia, M. Stark, B. Schiele, and K. Schindler. Detailed 3D representations for object recognition and modeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2608–23, Nov. 2013. ISSN 1939-3539. doi: 10.1109/TPAMI.2013.87. 65