UNIVERSITY OF BREMEN
INSTITUTE FOR
ARTIFICIAL INTELLIGENCE

Task-adaptable, Pervasive Perception for Robots Performing Everyday Manipulation

Ferenc Bálint-Benczédi

Vollständiger Abdruck der vom Fachbereich 3 (Mathematik und Informatik) der Universität Bremen zur Erlangung des akademischen Grades eines

Doktor-Ingenieur (Dr. Ing.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Udo Frese *Universität Bremen*
1. Prüfer:	Prof. Michael Beetz, PhD *Universität Bremen*
2. Prüfer:	Prof. Dr. Markus Vincze *Technische Universität Wien*
Beisitzer:	Christian A. Müller, PhD *Universität Bremen*

Die Dissertation wurde am 03.12.2019 bei der Universität Bremen eingereicht und durch den Prüfungsausschuss am 11.02.2020 angenommen.

Abstract

Intelligent robotic agents that help us in our day-to-day chores have been an aspiration of robotics researchers for decades. More than fifty years since the creation of the first intelligent mobile robotic agent, robots are still struggling to perform seemingly simple tasks, such as setting or cleaning a table. One of the reasons for this is that the unstructured environments these robots are expected to work in impose demanding requirements on a robot's perception system. Depending on the manipulation task the robot is required to execute, different parts of the environment need to be examined, the objects in it found and functional parts of these identified. This is a challenging task, since the visual appearance of the objects and the variety of scenes they are found in are large.

This thesis proposes to treat robotic visual perception for everyday manipulation tasks as an open question-answering problem. To this end RoboSherlock, a framework for creating task-adaptable, pervasive perception systems is presented. Using the framework, robot perception is addressed from a system's perspective and contributions to the state-of-the-art are proposed that introduce several enhancements which scale robot perception toward the needs of human-level manipulation. The contributions of the thesis center around task-adaptability and pervasiveness of perception systems. A perception task-language and a language interpreter that generates task-relevant perception plans is proposed. The task-language and task-interpreter leverage the power of knowledge representation and knowledge-based reasoning in order to enhance the question-answering capabilities of the system. Pervasiveness, a seamless integration of past, present and future percepts, is achieved through three main contributions: a novel way for recording, replaying and inspecting perceptual episodic memories, a new perception compo-

nent that enables pervasive operation and maintains an object belief state and a novel prospection component that enables robots to relive their past experiences and anticipate possible future scenarios. The contributions are validated through several real world robotic experiments that demonstrate how the proposed system enhances robot perception.

Zusammenfassung

Seit Jahrzenten streben Robotikforscher die Entwicklung intelligenter Roboter an, die uns bei unserer täglichen Arbeit helfen können. Mehr als fünfzig Jahre nach der Entwicklung des ersten intelligenten mobilen Roboters haben diese immer noch Schwierigkeiten scheinbar einfache Aufgaben, wie zum Beispiel das Decken oder Reinigen eines Tisches, auszuführen. Einer der Gründe dafür ist, dass die unstrukturierten Umgebungen in denen diese Roboter arbeiten sollen, hohe Anforderungen an das Wahrnehmungssystem eines Roboters darstellen. Abhängig von der von dem Roboter auszuführenden Manipulationsaufgabe, müssen verschiedene Teile der Umgebung untersucht, die darin enthaltenen Objekte gefunden und deren Funktionsteile identifiziert werden. Dies ist eine herausfordernde Aufgabe, da die visuelle Erscheinung der Objekte und die Szenen, in denen sie vorkommen, vielfältig sind.

In dieser Dissertation wird vorgeschlagen, die robotisch-visuelle Wahrnehmung für alltägliche Manipulationsaufgaben als offenes Frage-Antwort-System zu behandeln. Zu diesem Zweck wird RoboSherlock, ein Framework für die Erstellung von anpassungsfähigen und durchdringenden Wahrnehmungssystemen, vorgestellt. In diesem Framework wird die Wahrnehmung von Robotern aus der Perspektive eines Systems angegaben. Es werden Beiträge zum Stand der Technik vorgeschlagen, die verschiedene Verbesserungen einführen, die die Wahrnehmung von Robotern an die Bedürfnisse menschlicher Manipulation anpassen. Die Beiträge der Dissertation beschäftigen sich mit der Anpassungsfähigkeit und Durchdringung von Wahrnehmungssystemen. Eine Aufgabensprache und ein Sprachinterpreter, der aufgabenrelevante Wahrnehmungspläne erstellt, werden vorgeschlagen. Die Aufgabensprache und der Sprachinterpreter nutzen Wis-

sensrepräsentation und wissensbasiertes Schlussfolgern, um die Fähigkeit des Systems zur Beantwortung von Fragen zu verbessern. Die Durchdringung, eine nahtlose Integration vergangener, gegenwärtiger und zukünftiger Wahrnehmungen, wird durch drei Hauptbeiträge erreicht. Eine neuartige Methode zur Aufzeichnung, Wiedergabe und Untersuchung der episodischen Wahrnehmungserinnerungen, eine neue Wahrnehmungskomponente, die einen Objektglaubenszustand beibehält, und eine neuartige Prospektionskomponente, welche es den Robotern ermöglicht, ihre vergangenen Erfahrungen erneut zu erleben. Die Beiträge werden durch mehrere reale Roboterexperimente validiert, die zeigen, wie die vorgeschlagenen Systeme die visuelle Wahrnehmung von Robotern verbessern.

Acknowledgments

The work conducted for this thesis would not have been possible without the support of several people, who I would like to acknowledge in the following.

First and foremost, I am grateful to Prof. Michael Beetz for giving me the opportunity of researching in an excellent academic environment and trusting me with the responsibility of further developing the RoboSherlock framework. Our long discussions about robotic perception and his excitement and unique vision about the topic have always inspired and motivated me in continuing down the road I started.

Throughout the years of my doctoral studies I had the pleasure of working together with excellent colleagues, who have greatly influenced my research. I would like to thank the team contributing to the RoboSherlock framework: Nico Blodow for his early guidance in continuing his work, Jan-Hendrik Worch and Thiemo Wiedemeyer for the endless discussions and support in further developing the framework. Special thanks to Zoltán-Csaba Márton for his continuous support and advice related to computer vision problems and Daniel Nyga for the lengthy discussions about robotics and AI.

The contents of the thesis benefited enormously from the robot demonstrations that were developed together with colleagues. Gayane Kazhoyan, Gheorghe Lisca, Alexis Maldonado, Georg Bartels, Simon Stelter, Jan Winkler, thank you all for your feedback, support and patience while running the perception system on the robotic platforms. Many thanks to Daniel Beßler and Andrei Haidu for their assistance in integrating with the software systems they are developing.

I also had the opportunity of supervising the work of several excellent students who have contributed to the proposed framework. I would like to thank Patrick Mania, Le Thai An, Constantin Wellhausen and Shixin Li. Their contributions to

the code-base of the framework proved to be of great value.

Last but not least I want to thank my family. I owe a great deal to my father and sister, who have encouraged me in all my decisions. I am forever grateful to my wife, Réka, for her patience and support throughout the preparation of this thesis.

This work has received funding from the European Union Seventh Framework Programme FP7 project RoboHow (grant number 288533) and Horizon 2020 project Refills (grant number 731590) as well as the German Research Foundation DFG, as part of the Collaborative Research Center (Sonderforschungsbereich) 1320 “EASE - Everyday Activity Science and Engineering”, University of Bremen (http://www.ease-crc.org/) and the RoboSherlock project (grant number 260410154).

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

In the last decades robots have gradually migrated from operating in well-controlled, structured environments, such as factory floors, to more unstructured, human environments, and are slowly becoming everyday companions in the day-to-day lives of people. Whether it is an autonomous vehicle passing you on the highway (Waymo, 2017), a semi-humanoid greeting you at a restaurant (Ulanoff, 2017) or a custom-built robot flipping burgers in a restaurant (Kolondy, 2017), the diversity of tasks robots can undertake is increasing every day. Commercially available mobile robots are serving you in your hotel room when you order room service, are inspecting supermarket shelves while you are shopping for your everyday groceries, are vacuuming your living room or mowing your lawn while you are away from home. As the title of a recent article in Forbes magazine put it, *"the autonomous mobile robot market is taking off like a rocket ship"* (Banker, 2019). While this is true for a lot of robots whose main tasks are navigation and simple interaction with their users, only very few perform manipulation tasks. The ones that do, tackle very specific tasks on very specific objects in very specific conditions (Coldewey, 2018). Most mobile robots that interact with their environment through manipulating the objects around them are still confined to operating in research laboratories. In these environments, specifically arranged for robots, a wide variety of manipulation tasks can be performed. These range from mundane tasks, such as folding clothes (Maitin-Shepard et al., 2010), preparing meals (Beetz et al., 2011) or cleaning a window (Leidner et al., 2014) to specialized ones, such as performing chemical experiments (Lisca et al., 2015) or assembling

furniture (Dogar et al., 2015). Even though each of these tasks is an impressive achievement, mobile manipulation robots are not yet ready to perform these in real world settings. The complexity of the unstructured, constantly changing environment and the huge diversity of objects that need to be perceived and interacted with are limiting the manipulation tasks that robots can perform.

In order to cope with these limiting factors, robots need to be able to perceive the world around them, disambiguate and translate it into machine-understandable terms. In the case of autonomous robots performing everyday manipulation tasks, the role of perception is to extract information from the observations it has of the world, needed for accomplishing the respective task. The observations of the world are gathered using various sensors of a robotic agent, a very popular category of which are ones that enable visually perceiving the environment. While other sensing modalities are without a doubt an important part of an integrated robotic system and offer valuable and often indispensable information, when it comes to building artificial cognitive systems the potential inherent in visual perception is by far the largest (Kragic and Vincze, 2009). The strong need for sophisticated visual perception systems is emphasized by many in the community. In a recent essay, Brooks (2018) identified object recognition capabilities of two-year-old children as one of four important future goals of robotics. It is easy to see why. Small children already have a sophisticated understanding of object classes that enables them to transform functional knowledge of one object to another and their ability to learn about new objects from single or very few examples are truly amazing. When asked about the perception systems of robots, Brooks replied[1]:

> “ If we were only able to provide the visual capabilities of a 2-year old child, robots would quickly get a lot better ”
>
> *Rodney Brooks*

Indeed, the potential of visual perception is by far the largest, which stems from the fact that a great proportion of the necessary perception tasks can be accomplished through the use of vision alone. To exemplify this, let us consider a robot that has been operating in a kitchen for a while now, tasked with setting the

[1] `therobotreport.com/amazon-challenges-robotics-hot-topic-perception/`, Accessed: 24th of September, 2019

Figure 1.1: Typical scenes in a kitchen that a robotic agent needs to be able to perceive in order to set a table for breakfast. The goal of the robot would be to end up with an arrangement of objects similar to that on the right side.

table, preparing breakfast and cleaning up afterwards and exemplify the kinds of vision tasks this robot has to perform. The first part of this process is exemplified in Figure 1.1. As a first step, the objects need to be fetched from their current location and brought to a table. For example, the container with the milk needs to be found. The robot might recall that there was a milk on the counter, but it should also know that it was soy milk and that we prefer whole milk. So it has to find a milk container in the cluttered fridge. Once in front of the fridge, the door is closed, so it has to be opened. To do so, the handle of the fridge needs to be identified and grasp points calculated. If there are multiple milk containers inside the fridge, the intended one has to be picked. To fetch the cereal, spoon, a plate or a bowl, containers, such as drawers and cupboards, need to be searched. A cereal box with a specific flavor or the one that is already open needs to be identified. When fetching the bowl, the one that is big enough to hold just the right amount of milk and cereal needs to be chosen. Once all objects are on the table, it is time to prepare the meal, hence, the cereal needs to be mixed with the milk. For this it is necessary to identify functional parts of the container, such as the handle or the mouth of the container. The robot has to check whether the box of cereal and milk carton are open or closed, and in order to pour the milk into the bowl, the opening of the milk carton should be positioned right above the center of the bowl. While pouring, the robot should track the amount of liquid in the container such that it does not spill it. When putting the spoon in the bowl, it has to make sure that the handle of the spoon is sticking out and not the

other way around. When it is time to clean up the table, the objects need to be re-identified. In order for the robot to safely remove the plate, the bowl on top of it would have to be detected and removed first. To optimize the sequence of objects to be picked, those that are similar and need to be placed in the same cupboard or drawer should be detected and grouped (i.e. the cutlery goes into the drawer, dirty items go in the dishwasher, etc.). The robot should also remember where objects were taken from and put them back where they belong: the milk in the refrigerator, the cereal in the cupboard, and so on. It could be that there is still some cereal left over in the bowl, so that needs to be detected in order to not spill it and to put it in the sink or a dishwasher.

Most of these perception tasks are driven by visual sensing, highlighting the need for a robust vision system. Even for a seemingly simple task, such as setting a table, there is a great amount of variations in the types of perception tasks that need to be considered. In order to further examine these I propose to formulate them as questions that a robotic perception system needs to answer. Based on the described scenario I identify a collection of general questions that I consider robot vision systems should be able to answer:

Past	Present	Future
Where have I seen this object before?	*Is there an object X in my environment?*	*How would the scene look like if I [...]?*
Has the object always been like this?	*Is there an object of type Y or property Z?*	*How would the object look if it were to[...]?*
What was that object [...]?	*Does this object have a part that [...]?*	*...*
...	*Where should I grasp that object?*	
	...	

Table 1.1: Set of questions that should be answerable by a robotic perception system

The questions are split into three categories based on the availability of information needed in order to answer them. Questions in the first column can be answered based on what the robot has seen before, the ones in the second column are related to what the robot perceives right now while the third category of questions address

possible future scenarios. The list of questions is not meant to be exhaustive, rather it covers the most common categories of requests that I considered to be important for a cognitive robotic system to answer when looking for objects in its environment.

Answering all of these queries is a challenging task for a robot. First, the variation of visual characteristics of objects is high: they can be either textured or textureless, shiny or opaque, can be occluded or can have functional parts that might be important in order to manipulate them. When they need to be manipulated, recognizing the type of the object is not sufficient. It is important to know the geometry of an object, its function, how and where to grasp it, etc. The objects often have to be perceived in scenarios where it is difficult to recognize them, such as in extremely cluttered drawers or fridge interiors that make it difficult to separate individual objects from each other. It can also happen that objects simply look different depending on where they are located, i.e. stacked bowls look different when found on a table then in a cupboard. When performing a task, objects are moved around and can temporarily be out of sight. When manipulation actions are performed, it is possible that an object's characteristics drastically change: plates, bowls become dirty, or cups seen from above change color after some liquid is poured into them. Geometry of an object can also easily change: just consider removing the lid of a pot. Besides the challenges posed by the large variety of object properties, another source of difficulty arises from the imperfect sensors that robots are equipped with. The images of objects can often be noisy, blurred, over- or under-exposed, induced by the movement of the robot or the changes in lightning conditions. I collectively refer to these challenges as the **problem of robotic visual perception for everyday manipulation tasks**.

Addressing this problem typically happens on the level of individual perceptual challenges, investigated by computer vision approaches. Specialized solutions exist for most of them, but there is no single algorithm that can cope with all the complexities at the same time. Thus, in order to enable a robot to competently *perceive everything*, it is necessary to combine the results of multiple algorithms. As we could only reasonably expect robots to deal with the complexities of these perception tasks through the combined strengths of multiple algorithms, we need

to address the ways in which these algorithms are chosen and combined. In order to do this competently, a belief state containing all relevant information about state of the current world, the task that a robot should perform, the perception tools it has at hand, as well as past and possible future states of the world needs to be accessible. Most of the time, though, *perceiving everything* is not necessary and a subset of algorithms suffices. In the majority of situations all perceivable properties of all objects are not needed. As an example, consider the last scene from Figure 1.1. When a robot is to remove the plate from under the bowl, the brand of the cereal box next to it is of no interest. It follows then that obtaining each of the needed information pieces can be considered a separate perception task that specifies what needs to be perceived. As Kragic and Vincze (2009) also observe, computer vision creates the necessary foundation for visually perceiving the environment, by investigating specialized techniques for understanding individual scenes, but robot vision needs to go further and take the task a robot is performing into consideration.

Based on these observations, I consider the problem of robotic visual perception for everyday manipulation tasks to be dependent on the following three components:

- the **observations** that a robot has of the world, which, in the case of robot vision, is the collection of images it has to process,
- a **perception task or query** that needs to be performed or answered by the robotic agent,
- a **belief state** of the world the robot is operating in that contains all relevant information related to robot and world state.

1.1 Motivation and Proposition

Several recent surveys about robot vision (Loncomilla et al., 2016; Ruiz-del-Solar et al., 2018; Cadena et al., 2016) demonstrate that today's robot perception systems typically only provide a subset of the functionalities required by a robot performing everyday manipulation tasks. At the same time most robot perception systems provide their functionality in an overly general manner. They try to

accomplish the perception task by running predefined detection, categorization, identification and interpretation algorithms mostly on images as their sole input. This means that the perception routines cannot exploit prior knowledge, the structure of manipulation tasks and the environment, and that the robot cannot prepare for future perception tasks. There is a mismatch between the general methods applied to interpret the raw data and the semantic structure of the underlying information. For example, an overly generalized system cannot exploit that in a cupboard plates are stacked and therefore appear as horizontal lines in the images at their known location. If plates have a texture, most systems do not know that the pattern cannot be used for re-detection after a meal, because the plates might have become dirty. The environment and its structure are also not exploited to their fullest, i.e. in slowly changing environments many perception tasks or sub-tasks can be performed earlier in order to be prepared when the actual task is issued. These observations are reminiscent of what Drew McDermott formulated about the problems faced by automatic generation of an action sequence in robot planning:

> “ As stated, the problem turned out to be too hard and too easy. It was too hard because it was intractable. [...] It was too easy because action sequences are not adequate as a representation of a real robot's program. As often happens in AI, we are now trying to redefine the problem or do away with it. ”
>
> *Robot Planning by Drew McDermott*

Similarly to this statement, in light of the existing solutions, we can say that perception tasks currently used are both too easy and too hard. They are too easy because they only cover a subset of the needed functionalities under context conditions. These conditions though, often cannot be met by applications. Therefore, the current perception tasks cannot enable open and competent manipulation in realistic environments. They are, thus, too hard because they are intractable once context conditions are not met and regularities that prior knowledge has to offer, in order to simplify perception tasks, are not exploited. As a simple example of this seemingly contradictory nature of

perception tasks consider the previous example of detecting plates in a cupboard: it is easy to create a specific detector that finds lines in the images, but the lines will only correspond to edges of plates under the specific condition that the robot is looking into a cupboard where plates are stacked. In this case, detecting plates can be considered easy. Without considering and exploiting the knowledge about the environment, creating a general detector for plates that will work in all possible conditions for all possible plates is a much more difficult task, hence in this situation detecting plates is hard. As a consequence of these observations, in order to address this duality of perception tasks, I believe that it is necessary to treat robotic vision on a system's level and a framework is needed that enables the creation of perception systems that fulfill the following requirements:

R1: are task adaptable. The robot control program can request the perception system to detect objects, ask it to examine aspects of the detected objects such as their 3D form, pose, state, etc. Perception can, thus, be viewed as a question-answering system that answers the queries of the robot's control system based on perceived scenes. The query language needs to be expressive enough to cover all possible tasks that a robot might have to execute and flexible enough to be extended when new tasks need to be added.

R2: enhance perception with knowledge and reasoning. The framework should support reasoning about the objects to be detected and the task and environment context that objects are found in, in order to make the perceptual processes faster, more efficient and robust. Through the use of knowledge processing the systems created using the framework can interpret the results returned by the perception algorithms and thereby increases the set of perceptual tasks that they can accomplish. Knowledge-based reasoning also enables the specialization of perception tasks in order to resolve ambiguities in the data.

R3: are equipped with ensembles of expert perception algorithms. Instead of relying on one particular perception algorithm, integrating algorithms with complementary, similar or overlapping functionality needs to be supported. Control mechanisms and data structures should be provided

to direct the algorithms to synergistically cooperate, to communicate relevant information, fuse their results and hence to combine the strengths of individual methods.

R4: can incorporate beliefs maintained about the world to improve future perception tasks. Maintaining an internal belief state and memories of percepts that led to forming these beliefs are the basis for adapting and improving perception components through various learning techniques and as such it needs to be an integral part of any perception system that is built for robots performing everyday manipulation tasks.

R5: is modular and extensible. Robot vision is in fact embodied vision. This means that several strategies for interpreting the environment are possible, especially for long term operations, where a robot moves around in its environment. Since there is no single algorithm to solve every perception problem, it is essential that a perception framework is easily extensible with new modules and modalities.

With these requirements of a perception system in mind, in this thesis I propose a promising way for addressing the problem of robotic visual perception for everyday manipulation tasks, by creating **a task-adaptable, pervasive perception system** for autonomous robots performing human-level manipulation actions. The proposed system is capable of accomplishing a large variety of perception tasks through **incorporating background knowledge about the environment and the objects in it**. Pervasive operation is achieved by spreading perception throughout the lifetime of a robot, **seamlessly integrating past, present and possible future percepts.**

1.2 Contributions

To address all of the outlined challenges, in this thesis I present a perception framework, called RoboSherlock. RoboSherlock was introduced by Blodow (2014) as a framework for maintaining dynamic object belief states for robot perception through treating perception as an Unstructured Information Management (UIM) problem. I extend the framework in order to facilitate the realization of

task-adaptable, pervasive perception systems that combine results from multiple expert algorithms with the use of knowledge-based reasoning techniques, fulfilling the requirements specified in Section 1.1. Specifically the contributions of the thesis with respect to the state-of-the-art and the prior work of Blodow, grouped around task-adaptability and pervasiveness can be summed up as follows:

In order to achieve a perception system that is task-adaptable:

- **I propose a perception query language that can be interpreted and reasoned about in order to generate and optimize task-specific perception plans.** The query language allows the formulation of robot vision tasks for everyday manipulation as a query-answering problem. It acts as an interface between the high-level control system of a robotic agent and the perception system. The task interpreter is able to reason about the elements of the query in order to find the correct perception algorithms that are best suited to solve it. The reasoning capabilities are used in a planning algorithm that automatically builds a complete perception processing pipeline. I demonstrate the expressive power and applicability of the language using several application scenarios showing that the proposed syntax is general enough to cover a large variety of perception problems;
- **I realize and investigate a perception system that leverages knowledge representation and reasoning in order to boost query-answering, task adaptability, robustness and efficiency**. I propose to represent and store perception experts using a common symbolic knowledge base. Given a perception task, the system reasons about how to solve it, leveraging available background knowledge about objects, the environment and the perception experts, in order to boost query-answering capabilities of the robotic agent. I will demonstrate how the proposed knowledge-based approach improves the query-answering capabilities of robots performing everyday tasks to extents that go far beyond what current state-of-the-art systems can handle.

Building on task-adaptability, in order to realize a pervasively operating perception system:

- **I propose to equip robots with the capability of recording perceptual episodic memories and demonstrate how these are used to improve the recognition of objects.** To do this, I present a novel logging mechanism that is able to offer detailed records about *everything* that RoboSherlock has processed, and propose a Domain Specific Language (DSL) for describing past percepts that allows accessing the episodic memories using the perception query language. Using the query language and the episodic memories I demonstrates how object recognition modules of a robotic agent can be improved.
- **I investigate and realize a preparatory, amortized perception system that spreads the act of perceiving through the lifetime of the robotic agent.** This is achieved by maintaining an object belief state through the combination of a preparatory perception component that *continuously* analyzes the environment and an amortized perception component that *opportunistically* refines the belief state. I demonstrate through robotic experiments how these two components contribute to a pervasively operating perception system and improve the query-answering capabilities of a robotic agent;
- **I propose equipping robots with the ability to relive variations of past experiences in order to adapt their object detection and recognition capabilities.** The goal of the proposed ability is to enable prospection and is realized through the implementation of an internal simulation component in RoboSherlock that enables the generation of mental images, depicting possible future scenes. The prospection component consists of: (1) a knowledge-based approach for generating scene variations based on episodic memories; (2) the simulation of these variations in a virtual environment and the recording of off screen rendered images of these; and (3) algorithmic approaches for adapting object detection and recognition modules using the resulting images. I demonstrate the reliability of the adapted modules through

a series of experiments comparing the results of several state-of-the-art object recognition and detection approaches tested on both real and rendered images.

The contributions of this thesis have been published at international conferences and in journal articles in the field of robotics and artificial intelligence. The most relevant of these are listed below in chronological order of their appearance. A complete list of publications can be found in Appendix A.

Journals and Book Chapters:

Michael Beetz, Ferenc Bálint-Benczédi, Nico Blodow, Christian Kerl, Zoltán-Csaba Márton, Daniel Nyga, Florian Seidel, Thiemo Wiedemeyer, Jan-Hendrik Worch, "RoboSherlock: Unstructured Information Processing Framework for Robotic Perception", *In Handling Uncertainty and Networked Structure in Robot Control, Springer International Publishing, Cham, pp. 181-208, 2015.*

Ferenc Bálint-Benczédi, Jan-Hendrik Worch, Daniel Nyga, Nico Blodow, Patrick Mania, Zoltán-Csaba Márton and Michael Beetz, "RoboSherlock: Cognition-enabled Robot Perception for Everyday Manipulation Tasks", *Preprint In Arxiv.org, revision under review for the International Journal of Robotics Research, 2019*

Conference Papers:

Ferenc Bálint-Benczédi, Michael Beetz, "Variations on a theme:'it's a poor sort of memory that only works backwards'", *In International Conference on Intelligent Robots and Systems, IEEE, Madird, Spain, 2018*

Ferenc Bálint-Benczédi, Michael Beetz, "Amortized Object and Scene Perception for Long-term Robot Manipulation", *Preprint In Arxiv.org, 2019.*

Ferenc Bálint-Benczédi, Patrick Mania and Michael Beetz, "Scaling perception towards autonomous object manipulation — in knowledge lies the power", *In International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016*

Ferenc Bálint-Benczédi, Zoltán-Csaba Márton, Maximilian Durner and Michael Beetz, "Storing and retrieving perceptual episodic memories for long-term manipulation tasks", *In Proceedings of the 2017 IEEE International Conference on Advanced Robotics (ICAR), Hong-Kong, China, 2017 finalist for Best Paper Award.*

Michael Beetz, Ferenc Bálint-Benczédi, Nico Blodow, Daniel Nyga, Thiemo Wiedemeyer and Zoltán-Csaba Márton, "RoboSherlock: Unstructured Information Processing for Robot Perception", *In IEEE International Conference on Robotics and Automation (ICRA), Seattle,Washington, USA, 2015 best Service Robotics Paper Award.*

Daniel Nyga, Ferenc Bálint-Benczédi and Michael Beetz, "PR2 Looking at Things: Ensemble Learning for Unstructured Information Processing with Markov Logic Networks", *In IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014*

Open-source Contributions In addition to the scientific contributions, implementations of the presented systems have been made publicly available as a collection of open-source packages under BSD or the Apache, Version 2.0 licenses. The implementations are part of the RoboSherlock perception framework and they are located in the main repository of the project[2]. A project website[3] provides interested readers with installation instructions and tutorials on how to use the framework. Parts of the datasets used in this thesis (described in detail in Appendix B) are made publicly available through the project website.

1.3 Validation Scenarios

I validate the proposed task-adaptable and pervasive system through robotic experiments performed in several application scenarios. I consider the case of mobile manipulation robots that are expected to perform everyday long-term manipulation tasks. The tasks I examine take a considerable amount of time to execute, are repetitive in their nature and require interaction with the environment.

[2] `http://www.github.com/robosherlock`
[3] `http://www.robosherlock.org`

(a) Scanning shelves **(b)** Preparing a meal **(c)** Performing a chemistry experiment

Figure 1.2: Example scenes from the demonstrator scenarios that will be used in the thesis.

Specifically, throughout the thesis, three examples of such scenarios will be used: an assisted living scenario, a robot performing chemical experiments and a robot monitoring the stock levels of products in a supermarket (Figure 1.2). All of these scenarios have the following characteristics in common:

- the high-level tasks performed by the robot take a considerable amount of time
- if no other agents intervene, the scenes tend to change at a slower pace
- most manipulation tasks involve some form of pick-and-place task
- and the respective perception task can be simplified by using common sense reasoning and background knowledge.

Furthermore, in all three scenarios it is considered that the robot is performing a high level task that entails the need for object perception. I will demonstrate that, with the help of the concepts and systems presented in the thesis, a wide variety of perception problems can be successfully tackled.

Let us briefly look at the *assisted living* scenario as it is the one where most of the experiments take place. I believe it to be one of the most exciting areas of mobile service robotics research, where robots are expected to perform everyday household chores, such as setting or cleaning a table or preparing various dishes. Such robots are still only developed in research laboratories, but it is anticipated that they will be essential in helping out in areas such as elderly care or hospitals (EU-MAR, 2016). In this thesis, the example scenario consists of

a Personal Robot 2 (PR2) and a custom-built mobile robot, Boxy, performing fetch-and-place and meal preparation tasks, such as preparing a pizza or a pancake in a kitchen environment (Figure 1.2b). The kitchen is an excellent demonstrator domain, since most people have a very good understanding of the tasks that need to be performed and the objects that are needed for these tasks. Even if a weak-closed-world assumption is considered the challenges and diversity of perceptual tasks makes such an environment ideal for finding new and improved ways for object perception. A weak-closed-world assumption means to assume that all objects and object classes that can be encountered, are known beforehand, but there is a possibility of still encountering unseen objects, object categories. The assisted living scenario serves as a test bed for new functionality of the perception framework. The objects that need to be recognized are commonly referred to as objects of daily use, ranging from small shiny objects, such as knives, to large opaque ones, such as an electric pancake maker. The challenge in finding these objects is the diversity of constellations they can be found in (in drawers, refrigerators, stacked and occluded) and the fact that they can easily change visual characteristics depending on the executed task, i.e. the pizza dough changes color after tomato sauce is spread all over it.

1.4 Outline

The thesis is structured based on the contributions outlined in Section 1.2. It is recommended to read the chapters in the order they are presented in. Some of the chapters are self-contained, investigating the goals set out in this thesis, while others depend on concepts introduced in preceding chapters. The short summary of each clarifies these dependencies, highlights the contributions described in each and specifies the peer-reviewed publications that the chapters are based on and where the contributions were originally presented. Furthermore, each chapter presents a survey of the state-of-the art that is related to the topic covered in it. The rest of the thesis is structured as follows.

Chapters 2 and *3* present the contributions related to the realization of a task-adaptable perception system. In *Chapters 2* I present the approach for treating robot vision for everyday manipulation actions as a query-answering

problem. A formal definition of the framework is given and main components are introduced. I present the concepts that the framework makes use of. I propose a language for defining perception tasks, and show how it is used to guide the process of perception. The chapter explains the basics of RoboSherlock, what a processing pipeline is, how data is read in, stored, passed around, how query-answering is implemented and as such it constitutes the basis for all other chapters. The chapter is based on previous work presented in Beetz et al. (2015a) and Bálint-Benczédi et al. (2019).

In *Chapter 3* I investigate ways for combining knowledge-based reasoning and perception techniques in order to improve robustness and extend the domain of answerable queries by a robotic agent. I detail how symbolic knowledge bases and knowledge-based reasoning are used during the query-answering process to enhance perception with background knowledge and how knowledge about the environment simplifies perception problems. Through the introduction of a symbolic representation for perception algorithms, a means for generating perception plans is proposed. I present how knowledge is stored, what the current reasoning capabilities are, how these can be extended and the advantages of such an approach. The benefits of the proposed fusion of knowledge processing and perception are highlighted through investigating the types of queries that need to be answered in the three scenarios briefly mentioned in Section 1.3. The chapter is based on previous work presented in Bálint-Benczédi et al. (2016) and Bálint-Benczédi et al. (2019).

The remainder of the chapters presents individual components of the system that contribute to the realization of a pervasively operating perception system. In *Chapter 4* I propose to store perceptual episodic memories in the form of semantic image logs. The chapter gives an overview of how these memories are gathered and used to improve in a lifelong learning manner the performance of a perception system. I extend the perception query language presented in Chapter 2 to allow querying for past precepts and accessing low-level numerical perceptual data. The logging mechanisms presented in this chapter are used in Chapters 5 and 6. The chapter is an extended version of the concepts presented in Bálint-Benczédi et al. (2017).

In *Chapter 5* I propose an amortized preparatory subsystem for the framework.

I present how the gathered episodic memories are used during runtime to prepare the system for new perception tasks. I will show that using such an integrated approach the query-answering accuracy of the system can be greatly improved. The chapter is partly based on joint work with Thiemo Wiedemeyer (Wiedemeyer et al., 2015) and on previous work presented in Bálint-Benczédi and Beetz (2019).

In *Chapter 6* I discuss the topic of prospection and how it can be beneficial for robotic systems. A subsystem is proposed that allows a robotic agent to anticipate and adapt to new scenarios and environments. The key role of knowledge integration is highlighted in generating plausible alternative realities that semantically resemble the original scenarios. Recent advances in virtual reality allow a robot to perform perception tasks in this environment and off-screen render its results. The resulting images then serve as input data for supervised learning problems. The contents of this chapter have previously been published in Bálint-Benczédi and Beetz (2018).

In *Chapter 7* I conclude the thesis with a summary of the proposed contributions, discuss how they were achieved and how they relate to different question-answering subproblems. The chapter offers a short overview of the robotic demonstrations the framework was used in and discusses future opportunities and challenges.

CHAPTER 2

RoboSherlock: Task-adaptable Perception for Open Question-answering

The primary role of perception in autonomous robots performing everyday manipulation is to extract the information needed to accomplish a specific task. Robotic perception systems must provide relevant information about the environment and the objects in it to various modules of a complete robotic system, such as the high-level planning or the lower-level motion controllers. Robot vision plays a key role in supplying these information. As I have argued in Chapter 1, if we are to scale towards human-like manipulation capabilities, a promising way is to address the problem on a system's level and create a general perception framework that, among other criteria, forms a tight collaboration with other cognitive processes of a robotic system. To this end I present RoboSherlock, a software framework designed for robots performing manipulation tasks. The framework facilitates the realization of task-adaptable and pervasively operating perception systems with the needed functionality to specialize and leverage existing structure and knowledge. It enables the combination of perception, representation, and reasoning methods in order to scale robot perception towards the needs implied by general manipulation tasks.

In RoboSherlock, interpretation of realistic scenes is formulated as an Unstructured Information Management (UIM) problem. The application of the UIM principle enables the design and implementation of perception systems that satisfy the requirements outlined in Section 1.1:

- ... *adaptability to perception tasks through a query language and as such the capability to answer task-relevant queries about objects and scenes,*
- ... *tight integration of knowledge-based reasoning techniques, support for knowledge-enabled reasoning about objects and automatic, knowledge-driven generation of processing pipelines,*
- ... *incorporation of beliefs maintained about the world to improve future perception tasks,*
- ... *support for the implementation of multiple complementary algorithms and boosting of object recognition performance through the combination of these* and
- ... *extensibility, allowing the implementation of various indispensable perception subsystems (belief state management, semantic logging etc.).*

In this chapter I focus on formally introducing the general problem of perception for everyday manipulation, phrased as a question-answering problem. I present the RoboSherlock framework in detail and propose the perception query language for formulating perception tasks. Using the framework I propose a perception system, highlighting individual contributions of the thesis, how these relate to the RoboSherlock framework and how they interact with each other. I conclude with a survey of the related work. Experimental analysis of perception as a question-answering problem will be given at the end of Chapter 3, after presenting how exploiting regularities, prior knowledge and knowledge-based reasoning can simplify perception tasks. The chapter is based on work from previous publication (Beetz et al., 2015a; Bálint-Benczédi et al., 2019), offering additional detail on how the system is implemented. The formal description of a processing cycle in RoboSherlock is the result of joint efforts with Daniel Nyga.

2.1 Perception for Everyday Manipulation Tasks

Throughout the thesis I use the term *everyday robotic manipulation task* to refer to a sequence of actions that are performed by an autonomous manipulation robot

and are aimed at accomplishing a well defined goal. The sequence of actions takes a considerable amount of time to execute, is repetitive and requires interaction with the environment. Examples of such tasks are setting and cleaning a table, performing chemical experiments or monitoring the stock levels of products in a supermarket. Depending on the current context and the action performed, different perceptional cues and functional parts of objects become important. For instance, in the case of a robot that is to put a cooking pot on the stove, the handles of the pot need to be detected, in order for the robot to grasp it. If, however, it is to pour water into it, the handles are less relevant, but the center of its top opening matters more. As it is infeasible to exhaustively process a robot's sensor readings for all possible aspects of the world at all times, its perception system must be able to answer targeted questions about the environment on demand. In order to proficiently accomplish these tasks in human environments, perception needs to be adaptable to the current action being executed. In addition, for the world being only partially observable in nearly all real-world scenarios, it is necessary for a robot to maintain an internal representation of which state it believes the environment to be in. This representation is referred to as the *belief state* of the robot.

(formal description of the perception-action loop by courtesy of Daniel Nyga)
In Chapter 1 I introduced the perception problem for everyday manipulation tasks as being dependent on the observations, the task queries and a belief state about the current world. To formally define this problem, we consider a robot model that is inspired by the concept of rational agents originally introduced by Russell and Norvig (2010). The main concepts are depicted in Figure 2.1. A *robot* is a physically embodied agent located in an *environment* (which we also call a *world*). It perceives the world through its sensors and it is able to manipulate the world by its actuators. Let us denote the set of possible states of the world by $\mathcal{X}$. Typically, the state of the world is not directly accessible to the robot, but its sensors yield a filtered, noisy representation of a certain part of it. The so-called *perceptual filter* function determines how a world state is transformed into signals of the robot's sensors, $f_p : \mathcal{X} \mapsto \mathcal{O}$, where $\mathcal{O}$ denotes the set of possible sensor readings, which we also refer to as *observations*. Given an observation, the robot's control program has to decide on the next action to conduct, which is executed

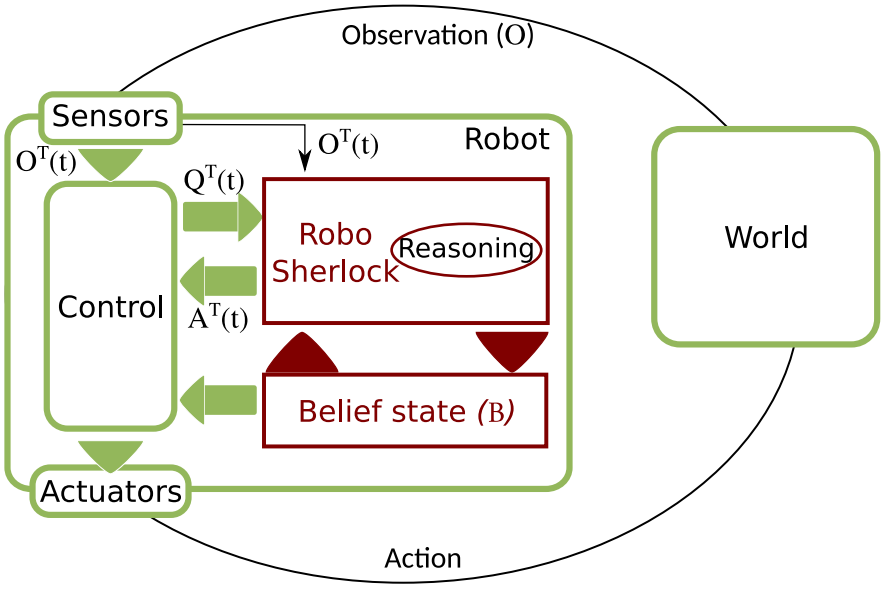


Figure 2.1: The classical perception-action loop, with RoboSherlock's role highlighted in red

by the robot's actuators. Let $\mathcal{AC}$ denote the set of possible actions, then the transition function $f_e : \mathcal{AC} \times \mathcal{X} \mapsto \mathcal{X}$ specifies the effects that an action has on the environment, when being executed in a particular world state. The repeated execution of

1. perceiving the state of the world,
2. selecting an action to conduct,
3. manipulating the world according to the action

is called the perception-action loop. Let $T = \{1, \ldots\}$ denote a set of iterations (time steps) denoting executions of the perception-action loop. Then, the robot's actions produce a sequence of world states $X^T : T \mapsto \mathcal{X}$, which are perceived by the robot as a sequence of observations $O^T : T \mapsto \mathcal{O}$, where $O^T(t) = f_p(X^T(t))$. The robot maintains an internal representation of the world, which is called the robot's *belief state*. Let $\mathcal{B}$ denote the space of all possible belief states[1]. In

[1]In literature (Lee et al., 2014a; Platt et al., 2010) the belief state is considered to be a probability distribution over all possible beliefs about the world. I use the term to refer to the most probable state of the world. Section 2.3.1 details how the belief state is represented in this thesis.

every iteration of the perception-action loop, the belief state gets updated by the perception system, such that a trajectory $B^T : T \mapsto \mathcal{B}$ of belief states is generated over time. The belief state is used by the robot's control program to select the next action to conduct, and in turn serves as a basis for formulating a query Q that is issued to the perception system. The query Q and the observation O are used to to update the belief state with task-relevant knowledge about the environment. Let $\mathcal{Q}$ denote the set of possible queries, and $Q^T : T \mapsto \mathcal{Q}$ the sequence of queries that the control program asks over time. Similarly let $\mathcal{A}$ denote the set of all possible answers that is a subset of the belief state ($\mathcal{A} \subset \mathcal{B}$), and $A^T : T \mapsto \mathcal{A}$ the sequence of answers generated by the perception system. Using these components, *perception for everyday robot manipulation*, phrased as a question-answering problem, can now be formulated as the pair of functions:

$$perceive : \mathcal{B} \times \mathcal{Q} \times \mathcal{O}^T \mapsto \mathcal{B}, \tag{2.1}$$

$$answer : \mathcal{B} \times \mathcal{Q} \mapsto \mathcal{B} \tag{2.2}$$

such that the following conditions hold:

1. $B^T(0) = \emptyset$
2. $perceive(B^T(t-1), Q^T(t), O^t) = B^T(t), \forall t \in T,$
3. $answer(Q^T(t), B^T(t)) = A^T(t), \forall t \in T$

where O^t denotes the sequence of past observations up to time t. This allows reasoning about world states taking into account sensor readings from multiple time steps, which may lead to more accurate estimates than considering only one-shot observations. Examples of such approaches are Bayesian filtering models, such as the Kalman filter, SLAM (Thrun et al., 2005), or Kinect Fusion (Izadi et al., 2011). Note, that $B^T(0)$ is initialized as the empty set only in the most general case, where there is no previous information available and the robot is expected to explore the world. In this thesis $B^T(0) = \mathcal{P}$, where $\mathcal{P} \subset \mathcal{B}$ represents pre-existing knowledge about the environment, the known objects in it and the robot state, used to initialized the belief state with.

2.2 RoboSherlock Overview

RoboSherlock is a perception framework that leverages implementations of the *perceive* ans *answer* functions from Equations (2.1) and (2.2). RoboSherlock considers perception as *Content Analytics (CA) in unstructured data*. CA denotes the discipline of applying methods from the field of statistical data analysis to large amounts of data in order to extract semantically meaningful knowledge from those. The data are considered *unstructured* since the structure does not reflect its semantics as it for instance does in relational databases or spreadsheets. The paradigm of Unstructured Information Management (UIM) offers an implementational framework for realizing high-performance CA systems. The most prominent example of a UIM system is *Watson* (Ferrucci et al., 2010), a question-answering system that won the US quiz show *Jeopardy!*, competing against the champions of the show and demonstrating an unprecedented breadth of knowledge. The paradigm of UIM can be summed up by three computational processes: hypothesizing about data, annotating the hypotheses and testing-ranking of results. In UIM, pieces of unstructured data, such as web pages, text documents, audio files or images are processed by a collection of specialized information extraction algorithms, and each algorithm contributes pieces of knowledge with respect to its expertise. Results of different algorithms are allowed to be complementary, overlapping or even contradictory. Subsequently the collected annotations are rated and consolidated to come to a consistent final decision.

Images are perfect examples of unstructured data. They depict hierarchical structures of object constellations, objects, object parts and relationships between these, but they are simply stored as an array of pixels. Thus, the UIM paradigm of hypothesize, annotate and test-rank can be, with minor modifications, applied to streaming sensor data. RoboSherlock operates based on this principle. It creates object hypotheses for pieces of sensor data that it believes to represent objects or object groups. Subsequent perception algorithms analyze these hypotheses and annotate them with semantic metadata. Further algorithms then test and rank these based on the combination of sensor and metadata. To this end, the framework supports the application of multiple algorithms, also referred to as experts, and the combination of their results. This is achieved by:

- providing means for the communication among expert methods;
- providing infrastructure that supports reasoning about how results of different experts should be combined;
- enabling the system to reason about its perceptual capabilities and decide which methods are the best for a certain perceptual sub-task.

In addition to the UIM paradigm of hypothesize, annotate and test-rank, RoboSherlock takes as input a perception query that describes the information needed by the robot in order to accomplish its manipulation tasks. RoboSherlock is designed to accomplish these perception tasks in complex scenarios that include objects with different perceptual characteristics. To do so, objects are perceived taking the scene and task context into account and knowledge-based algorithms are employed in order to decide which methods to apply to which objects. Using background knowledge simplifies perception tasks, e.g. through the use of salient distinctive characteristic of objects that simplify perceiving them. Knowledge-based reasoning techniques are also used when combining the results of different perception methods. Because of the ubiquitous presence of knowledge-based reasoning, RoboSherlock is also referred to as a knowledge-based perception framework.

2.2.1 Origins of RoboSherlock

RoboSherlock was first introduced by Blodow (2014) as a perception system for service robots in order to manage dynamic object belief states. In his work, Blodow pioneered the use of Unstructured Information Management (UIM) for robot perception, adapting the framework of Unstructured Information Management Architecture (UIMA) to handle streams of image data, implementing all the necessary data structures and processing mechanisms to achieve this. The focus of his work was mainly on how UIM can be used to induce a paradigm shift in approaching perception systems, advocating for the use of multiple expert algorithms. The UIM based framework of Blodow addresses the building of a dynamic object belief state for service robots, drawing parallels to the human memory system in the way it does this. Contributions of his work were the demonstration of the

applicability of UIM in the field of robotic perception and the introduction of a general *type system* that allows the semantic description of perception results. His work strongly emphasizes individual algorithmic contributions that are part of the overall framework, such as semantic real-time segmentation methods, interfaces to web services or point cloud processing algorithms.

The topics of this thesis build on Blodow's system, extending and broadening the framework's general goals, such that the implementation of task-adaptable, pervasively operating perception systems is possible. The most relevant changes have to do with the way adaptability to different tasks performed by a robot is addressed by proposing to treat perception as a query-answering system and by extending the framework with a knowledge representation and reasoning component. While Blodow highlighted in his experiments the richness of object descriptions that can be achieved using the RoboSherlock framework, in this thesis I take this idea further and look at how these descriptions can be generated on demand as part of a complete robotic system, formally describing the components of the framework and the roles that they can have. In addition, several perception subsystems are proposed, which investigate different modalities of robotic perception: preparatory perception, amortization effects or prospection. The changes introduced to the RoboSherlock framework allow for investigating these novel implementations of robotic perception systems. In conclusion, whereas RoboSherlock is considered to be the main outcome of Blodow's work, in this thesis the continued development of the framework serves as a means to achieving the implementation of the pervasive task-adaptable perception system.

2.3 Conceptual Overview

RoboSherlock has been designed with two major implementational aspects in mind:

1. it does not replace any existing perception system or algorithm but rather enables easy integration of previous work in a unifying framework that allows these systems to synergistically work together

2. new methods can be easily wrapped into RoboSherlock processing modules to extend and improve existing functionality and performance.

Conceptually there are two main parts of the framework. One for query interpretation and perception plan generation and one for processing the observations, a collection-processing component. The former is a contribution to the problem of Content Analytics (CA) while the latter is based on key concepts of UIM that were adapted to fit the needs of robot perception. These key concepts that RoboSherlock builds on are:

- the *Common Analysis Structure (CAS)* containing the data to be analyzed and temporary results; it is the common data structure and acts as a short-term memory of the RoboSherlock system;
- a *type system*, a hierarchical collection of data structures, defined by the user, that acts as a common language, allowing communication between components;
- the *Analysis Engines (AEs)* which are the core processing components of the framework, implementing single perception algorithms (Primitive Analysis Engine (PAE)) or complete perception pipelines (Aggregate Analysis Engine (AAE)). AEs share and operate on the CAS by generating, interpreting and refining hypotheses;
- *Collection Readers* that interface with the sensors of a robotic system and initialize the CAS;
- *CAS Consumers* that solve inconsistencies, merge results and update the world model;
- a *query language and query interpreter* that enable the description of perception tasks and generation of task-specific perception plans
- a *belief state representation* that represents our current understanding of what the world looks like and allows for processing mechanisms that the AEs can use as resources for reasoning about the objects and scenes they interpret.

With the exception of the query language and the belief state representation these concepts all originate from UIM. Each of these concepts has its own imple-

mentation and through combining them novel perception systems can be created. A detailed description of each concepts will now follow.

2.3.1 Belief State Representation

The robot's belief state is a formal representation of the state that the robot believes the world to be in. It is therefore crucial that the representational formalism is expressive enough to capture the essence of the world at a sufficiently high level of detail. At the same time, one should be able to reason about aspects of the model in order to derive new knowledge about the world that is only implicitly entailed.

In RoboSherlock, *Description Logic (DL)* was chosen as the representation language, for two reasons: (1) DL provides an intuitive, standardized and clean way for constructing models of the real world and (2) DL has well-established implementations used in both the knowledge representation and the AI-based robotics communities. In this thesis KnowRob (Tenorth and Beetz, 2013), a framework for representing worlds and reasoning about them using DL is used.

Description logics is a subset of first-order logic that facilitates ontological engineering of world representations. An ontology in DL consists of two main components: a *TBox* and an *ABox*. The $TBox$ defines the terminological building blocks a belief state may be composed of. It contains a set of *type symbols* $\top$ and universal rules on how they relate to each other. Examples of type symbols are *Container*, *Cup*, *Milk* or *Number*. One of the most prominent relations is the *subsumption*, which hierarchically arranges the types in a taxonomy, i.e. $\sqsubseteq \subseteq \top \times \top$. Propositions like

$$\begin{aligned} \mathit{Knife} &\sqsubseteq \mathit{Cutlery} \\ \mathit{Fork} &\sqsubseteq \mathit{Cutlery} \\ \mathit{Cutlery} &\sqsubseteq \mathit{PhysicalThing} \\ \mathit{Milk} &\sqsubseteq \mathit{Liquid}, \end{aligned}$$

for instance, state that the types *Knife* and *Fork* are both specializations of the type *Cutlery*, which in turn is a specialization of the type *PhysicalThing*, and that *Milk* is a specialization of the *Liquid* type. Additional relations can be used to

introduce more detailed type definitions, such as

$$\mathit{Knife} \doteq \mathit{Cutlery} \sqcap \exists\, \mathit{has.Blade} \sqcap \exists\, \mathit{has.Handle}.$$

This defines the concept of a knife as the intersection of the concept of cutlery that has a handle and has a blade. A specific instance of a world modeled in DL is stored in the $ABox$, which contains symbols referring to individuals in the world. Concept assertions and relations can be used to assign certain individuals to one or more types from the $TBox$, like

$$\mathit{Cup}(c) \text{ and } \mathit{Milk}(t),$$

if an entity c is an instance of the concept *Cup* and a different entity t is an instance of the concept *Milk*. If the relation $\mathit{relation}(\cdot,\cdot)$ holds for the pair $\langle c, t\rangle$, then the assertion

$$\mathit{relation}(c, t)$$

must hold. Further detailing of description logic would be out of scope. For an excellent introduction refer to Rudolph (2011).

It is important to note that all data structures and algorithms integrated in RoboSherlock are required to be modeled in the belief state. The $TBox$ of the belief state thus, includes the RoboSherlock *type system* and the Primitive Analysis Engines (PAEs) that wrap perception algorithms. This guarantees the interchangeability of data in a common language between components with unique, well-defined semantics, and in turn allows explicit reasoning about the perceptual capabilities of the robot. For example,

$$\begin{aligned}\mathit{ClassificationAnnotation} \sqsubseteq\ & \mathit{SemanticAnnotation} \\ & \sqcap \exists\, \mathit{classLabel}.\top \\ & \sqcap \exists\, \mathit{classConfidence}.\mathbb{R} \\ & \sqcap \exists\, \mathit{classifierName}.\mathit{String}\end{aligned}$$

defines a type that holds results of classification algorithms in the *type system* as *SemanticAnnotation* with attributes *classLabel*, *classConfidence*, *classifierName*. The *type system* further specifies that the *classConfidence* must be filled with exactly one floating point value and *classLabel* itself is a type. By storing this in a centralized knowledge base, a list of analysis engines that generate a specific type of annotation as output can easily be retrieved. The pre- and post-conditions in the $TBox$ of perception algorithms implemented in the framework specify under which circumstances algorithms may be applied and the results they produce. This enables automatic generation of perception plans which will be detailed in Section 3.2.

The RoboSherlock *type system* defines types for each annotation and object hypothesis. Besides the types that serve as blueprints for objects that hold interpretations of the raw data, types for low-level vision data structures (point clouds, images, regions of interest, rigid transforms) are also defined. A description of the current type system is given in Section 2.4.2.

To summarize, the belief state in this thesis is represented using DL and is considered to have two parts. One that holds the static knowledge about the environment, the objects in it and the robot with all of its capabilities (including the perceptual capabilities of RoboSherlock). We referred to this as the $TBox$ in this section but in the remainder of the thesis I will often refer to it as a knowledge base, to denote that KnowRob is used as the framework for storing this information. The other part, referred to as the $ABox$, is dynamic and is a specific instance of the world model from the $TBox$, populated and maintained during the operation of the robot.

2.3.2 A Query Language for Formulating Perception Tasks

Having the belief state represented in a formal, logic-based language allows answering queries about different aspects of the world. A *query* is a statement in a formal declarative language that describes conditions that entities in the belief state need to satisfy in order to be among the query results. The result of a query is thus a set of tuples that match that description. For a robot, such reasoning capabilities are extremely important, as it needs to be able to retrieve information on demand, such as “a free location on the table to put down an object”, “an

empty container object that is able to hold at least two liters of water" or "the turning knob on the oven". For the belief state in DL, tuple calculus known from relational algebra Codd (1970) can be used. A generic query in relational calculus has the form $\{v \mid \Psi(v)\}$, where v denotes a variable representing a candidate tuple and $\Psi(v)$ is a statement that formulates the requirements that v needs to satisfy in order to qualify as a result. A query for "an empty container that is able to hold at least 2 liters of water" can be formulated as follows:

$$\{c \mid c \in \mathit{Container} \wedge c.\mathit{capacity} \geq 2 \\ \wedge \neg\exists s \in \mathit{holds}(c.\mathit{id}, s.\mathit{id})\}, \tag{2.3}$$

and a query for "the pose of the turning knob on the oven" can be formulated as

$$\{[n.\mathit{pose}] \mid n \in \mathit{Knob} \wedge \exists o \in \mathit{Oven} \wedge o.\mathit{id} = \text{'oven-01'} \\ \wedge o \in \mathit{has}(o.\mathit{id}, n.\mathit{id})\}. \tag{2.4}$$

Queries to the belief state of the robot can also be thought of as queries to the perception system. The queries encode the perception task expected to be carried out by the robot, thus, they **enable the task adaptability of the perception system.** For example, query (2.3) requests all tuples of the type *Container* that have a capacity of at least two liters and are not in a *holds* relation with anything else (so are empty). This naturally corresponds to a **detect** perception task. Query (2.4), on the other hand, asks for an object of type *Knob*, which is in a *has* relation with a specific object of type *Oven*, and the *pose* attribute of that object is returned. Query (2.4), thus, corresponds to an **inspect** query.

Using relational algebra the generation of an answer, based on the updated belief state, can be formulated as a combination of *selection* and *project* operations. With the help of the formal description introduced in Section 2.1, a generalized selection performed on the belief state can be defined as $\sigma_\varphi(B^T(t)) = A'^T(t)$, where φ is a propositional formula describing the query and $A'^T(t)$ is an intermediary answer. The results of the selection are then filtered using the projection operator $\Pi_{k_1,k_2,\ldots}(A'^T(t)) = A^T(t)$, where $k_1, k_2, \ldots$ are attribute names that we queried for and $A^T(t)$ is the final answer.

These queries are difficult to parse by a machine, so in the following I propose a machine-interpretable query language in which perception queries for robots performing manipulation tasks can be stated. The semantics of these queries is defined in terms of queries in relational tuple calculus, as described above. The perception task language enables a better understanding of the capabilities of perception algorithms and systems. The language is used to state the tasks that perception algorithms are expected to accomplish. I propose to state perception tasks in a symbolic language that consists of *terms* and *perception tasks*.

The terms of the language are *object descriptions*, *object hypotheses* and *task descriptions*. Using such descriptions, a red spoon can be described as *(an object (category spoon) (color red))*.

Perception tasks are then formulated as one of the following operations on object descriptions and hypotheses:

1. **(detect *obj-descr*)** asks RoboSherlock to detect objects in the sensor data that satisfy the description *obj-descr* and return the detected matching *object hypotheses*,

2. **(inspect *obj-hyp attributes*)** asks the perception system to examine a given hypothesis *obj-hyp* in order to extract additional information as requested by *attributes* and add the information to *obj-hyp*,

3. **(track/scan *task-descr*)** runs continuous perception for a description defined in *task-descr*; these are perception tasks that typically require a stream of images, i.e. track an object while performing a manipulation task.

In more detail an object detection task has the form:

$$
\begin{aligned}
&(\textit{detect}\ (\langle \textit{det} \rangle\ \langle \textit{type} \rangle \\
&\qquad (\langle \textit{attr}_1 \rangle\ \langle \textit{val}_1 \rangle) \\
&\qquad \ldots \\
&\qquad (\langle \textit{attr}_n \rangle\ \langle \textit{val}_n \rangle)))
\end{aligned}
$$

where $\langle \textit{det} \rangle$ is the determiner of the description, which can be one of the key words *a(n)* or *the*. If the key word is *a(n)* than any hypothesis is accepted. If the determiner is *the*, there is assumed that exactly one hypothesis is generated.

The determiner *the* is needed to express a perception task such as find *my* cup. ⟨*type*⟩ can be *object*, *object-part* and *scene*. *object* asks for objects and object groups that cannot be further segmented by the perception system. For example, simple segmentation algorithms typically cannot segment a set into the cup and the saucer underneath. *object-part* specifies parts such as the handle of a spatula, and *scene* is used for constellations of objects. The determiner of the description is followed by optional attribute value pairs. Some attributes can be abstract characteristics such as affordances (*graspable*, *can-hold*), while others can refer to visual characteristics (*color*, *shape*). In addition, the locations where the objects can be found, such as in a container, on a surface, or relative to a reference object, such as in front of the cup, can be expressed.

The second category of perception tasks are inspection tasks that allow the robot to perceive additional information about detected objects. Formally, an inspection query has the form:

(*inspect* ⟨ *#uid* ⟩
⟨*:attr*$_1$⟩ ⟨*:attr*$_2$⟩ . . . ⟨*:attr*$_n$⟩)

where *#uid* is a unique identifier of an object hypothesis and ⟨*:attr*$_1$⟩ ⟨*:attr*$_2$⟩ . . . ⟨*:attr*$_n$⟩ is the list of attributes we want to examine. Object attributes that can be inferred from perception algorithms are for example *pose*, *2D/3D model*, *object state*, or *grasp points*.

A special category of perception tasks are compound queries. These queries use the detection task as a base query in order to achieve a more complex behavior. Such tasks are, for example, ones that require the perception system to behave in a continuous manner. The description can contain information such as a command for starting and stopping a task, description of an object, or a given hypothesis:

(*track/scan/...* ⟨*det*⟩ ⟨*type*⟩ (
detect (⟨*det*⟩ ⟨*type*⟩
(⟨*attr*$_1$⟩ ⟨*val*$_1$⟩)
. . .
(⟨*attr*$_n$⟩ ⟨*val*$_n$⟩)))))

These tasks include, for example, the tracking of an object while performing a manipulation task, or the scanning of a region for fusing results from multiple images.

Attribute	**Description**
shape	semantic shape label
color	semantic color label
type	type of an object e.g. super-class in taxonomy, or object affordance
location	semantic location of an object (e.g. on table top, in a drawer)
class	result of a classification, be it classification of object instances or class
pose	specify a pose
cad-model	specify a CAD model to fit
obj-part	specify the part of an object to detect (e.g. handle, opening etc.)
...	...

Table 2.1: List of query-language attributes currently implemented in RoboSherlock

Table 2.1 contains a subset of the valid keywords that are considered most relevant for object detection and perception in autonomous robot manipulation. Depending on the environment in which a robot is deployed, the query language can be extended to handle attributes that are specific to the tasks that need solving.

To better understand how queries are formulated let us look at a few examples. Using the introduced query language the task of finding a flat object that has the color black and is located in a specific drawer can be stated in the following way:

```
(detect (an object
            (shape flat) (color black)
            (location in (an object
                ((type container) (category drawer#3))))))
```

Once detected, examining it for its pose and grasp points can be done using an

inspection query:

```
(inspect (an object
             (obj-id #id))
         :pose :grasp-points)
```

As another example, we can state the perception task for tracking an object of a certain type with:

```
(track (an object (type ‘Spatula‘)))
```

Another example task that uses task-specific attributes is to look for a container that holds 200 milliliters of hydrochloric acid:

```
(detect (an object
            (category container)
            (capacity (≥ 0.2)) (contains 'HCl')))
```

Especially in the case of using task-specific attributes we need mechanisms to decide whether or not certain computational components should be run or not. To this end terms of the query language are interpreted and reasoned upon by a task interpreter component (Figure 2.2) in order to generate context-specific perception plans that invoke specialized perception algorithms (experts) as plan steps. Since reasoning about results of a query and task interpretation are closely related to the use of knowledge-based reasoning they are further detailed in Chapter 3, Section 3.2.

2.3.3 The Collection Processing Engine

The intermediary result of a query is a context-specific perception plan that will produce the results asked for in the upcoming processing cycle. The conceptual overview of a single processing cycle in RoboSherlock is shown in Figure 2.2. The figure is a detailed view of the role of RoboSherlock in the perception-action loop from Figure 2.1 and depicts two of the main components: the task

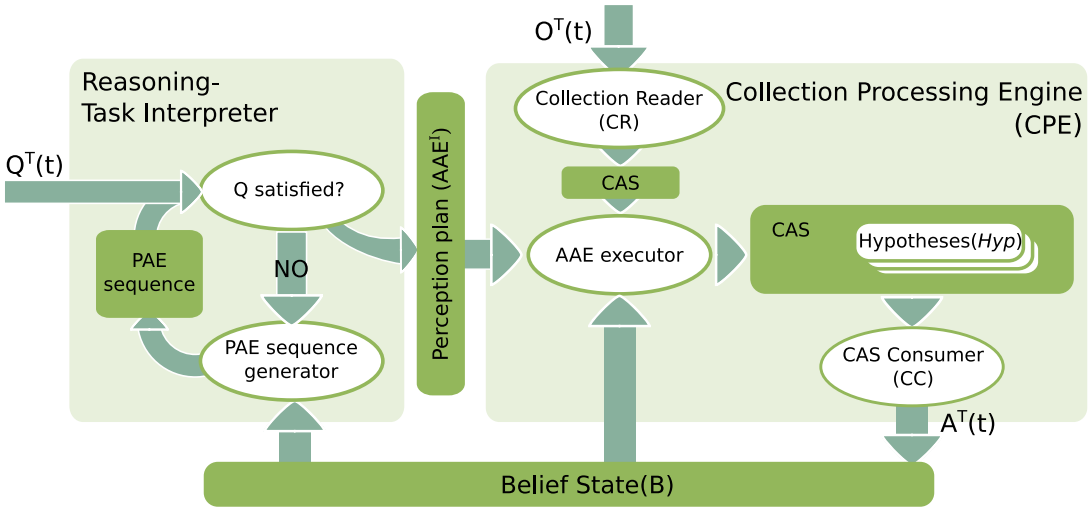


Figure 2.2: High-level conceptual overview of an execution cycle in RoboSherlock

interpreter and the Collection Processing Engine (CPE). The latter is addressed in this section, while parts of the former were already introduced and the rest will be the topic of Section 3.2. The Collection Processing Engine (CPE) consists of multiple processing components: *Collection Readers* that read in the observations, an *Aggregate Analysis Engine (AAE) executor* that takes as input the generated perception plan and executes its steps and a *CAS Consumer* that takes the result of the AAE executor and generates an answer. Results of these computational processes are returned in a data structure called the CAS. In order to understand the full cycle, the individual components shown in the figure and their behavior need to be introduced.

The central data structure that is common to all components of RoboSherlock-based perception systems is the Common Analysis Structure (CAS). A CAS consists of:

- **Subjects of Analysis (SofAs)**, the raw sensor data that is to be interpreted (e.g. RGB or depth images), which in our model is equivalent to the observations O^T (SofAs are also commonly referred to as *views*);

- and the **hypotheses** that are generated during a processing cycle, representing pieces of the raw data that are considered by the algorithms run in the cycle to be relevant to the task.

A hypothesis represents a certain region in our observations and a set of annotations attached to this region. A region in an observation is a set of indices of the vector of observations, e.g. pixels or 3D points. An annotation is an *ABox* representing a fragment of the belief state. Let $\mathcal{P}(\mathbb{N}^+)$ be the power set of pixels of an image or point indices of a point cloud. The space of all hypotheses $\mathcal{H}$ is defined as

$$\mathcal{H} := \mathcal{P}(\mathbb{N}^+) \times \text{ABox}.$$

An example for a hypothesis of a box-like red object which has a pose and a 3D feature descriptor, expressed through the use of description logic introduced earlier, has the form:

$$\begin{aligned} &Thing(h_1) \wedge shape(h_1, box) \wedge color(h_1, red) \\ &\qquad \wedge pose(h_1, [x, y, z, r, p, y]) \wedge vfh(h_1, [...]) \end{aligned}$$

With this we can not formally define what a CAS is. Let $\mathcal{P}(\mathcal{H})$ denote the power set of all hypotheses and $\mathcal{O}$ the set of all possible observations (SofAs). The space of all CASs is then defined as:

$$\mathcal{CAS} := \mathcal{O} \times \mathcal{P}(\mathcal{H})$$

CASs are both inputs and outputs of several computational components. An initial CAS is generated by processing modules called *Collection Readers* (*CR*s), special purpose modules that are designed to read in the observation from the sensors. A *Collection Reader* is defined as:

$$CR : \mathcal{O}^{\mathcal{T}} \longmapsto \mathcal{CAS} \;\; s.t. \;\; CR(O^t) = \langle O^t(t), \emptyset \rangle, \forall t \in T$$

creating a CAS with the most recent observation and an empty set of hypotheses.

The initial CAS servers as an input to processing components that generate enriched copies of it. The core processing elements of RoboSherlock are *Analysis Engines* (AEs). Their input is a CAS and they generate enriched copies of this by finding, annotating and refining object hypotheses. *Analysis Engines* can be divided into two categories: Primitive AEs (PAE) and Aggregate AEs

(AAE). Primitive AEs can be split based on their functionality into two categories: the first kind, called *hypothesis generators* analyze the observations and generate regions of interests. The second one, called *annotators* annotates the generated regions of interest. Some primitive AEs do not fit into one or the other category but combine traits of both. An example for such a primitive AE is a CAD model fitter which generates a hypothesis and, if successful, annotates the hypothesis with a label. Formally, both hypothesis generators and annotators are the same, both taking the CAS as an input and outputting a new CAS that is an enriched copy of the input:

$$PAE : \mathcal{CAS} \longmapsto \mathcal{CAS} \tag{2.5}$$

Aggregate AEs solve more complex tasks. An AAE consists of an ordered list of primitive AEs. The primitive AEs can be run sequentially, in parallel, or flexibly, e.g. on-demand or event-driven. The planning of a perception pipeline for a query produces an ordered sequence of primitive AEs, denoted by AAE^I:

$$AAE^I : I \longmapsto \mathcal{PAE} \tag{2.6}$$

where $I = \{1, \dots\}$ is the set of indices corresponding to the ordering of the primitive AEs and $\mathcal{PAE}$ is the power set of all primitive AEs. An AAE executor, also called a *flow controller*, takes the initial CAS with the observations and runs the sequence defined by AAE^I on it, resulting in a trajectory of CASs:

$$CAS^I : I \longmapsto \mathcal{CAS} \tag{2.7}$$

where $\mathcal{CAS}$ is the space of all possible CASs. The final CAS holds a partial semantic description of the world at time t. With the use of Equations (2.5)(2.6) and (2.7) the following formula holds:

$$CAS^I(i+1) = AAE^I(i)(CAS^I(i))$$

meaning that in order to obtain the next CAS in a sequence, the current primitive

AE has to be executed on the current CAS. This means that for every new observation $O^T(t)$ a unique trajectory of CASs, as defined in Equation (2.7), is generated. While formally this is equal to a new CAS being generated by each processing component, in practice the CAS is implemented as a whiteboard data structure (Boitet and Seligman, 1994) that gets enriched by each processing component and is cleaned at the end of the processing cycle.

In order to update the world, the resulting description need to be incorporated into the belief state B. Since annotators might employ heuristic interpretation methods or are more or less reliable and accurate, the set of annotations is allowed to be inconsistent or contradictory. Inconsistencies and erroneous annotations are handled by subsequent reasoning and hypothesis testing and ranking, using components called *CAS Consumers* (CC). CAS Consumers do not have a unique signature since they implement different kinds of post processing methods on the annotated data. For example, one important role of a CAS Consumer is the generation of the answer $A^T(t)$ to a query $Q^T(t)$ based on the updated belief state. A CAS Consumer that implements Equation (2.2) and generates an answer is defined as:

$$CC^A : \mathcal{CAS} \times \mathcal{Q} \times \mathcal{B} \longmapsto \mathcal{A}$$

Another example of a CAS Consumer that updates the belief state is defined as:

$$CC^B : \mathcal{CAS} \times \mathcal{B} \longmapsto \mathcal{B}$$

With these definition an iteration of a CPE, that implements both Equation (2.1) and (2.2) in RoboSherlock is defined as:

$$(CC^A \circ CC^B \circ B \circ AAE^I(i) \circ \cdots \circ AAE^I(0) \circ CR)(O^t, Q) = \langle A^T(t), B^T(t) \rangle$$

that is, the composition of a collection reader, the sequence of PAEs, the belief state and the CAS Consumers for a sequence of observations and a query Q. Figure 2.3 exemplifies the above explained process on a kitchen table scene, showing the trajectory of CASs as it is produced and updated by one iteration of the CPE. The cycle starts by executing a *Collection Reader* and populating the initial CAS with the raw data (depth and color images). The CAS is then passed on to the AAE executor that runs the planned sequence of PAEs (AAE^I).

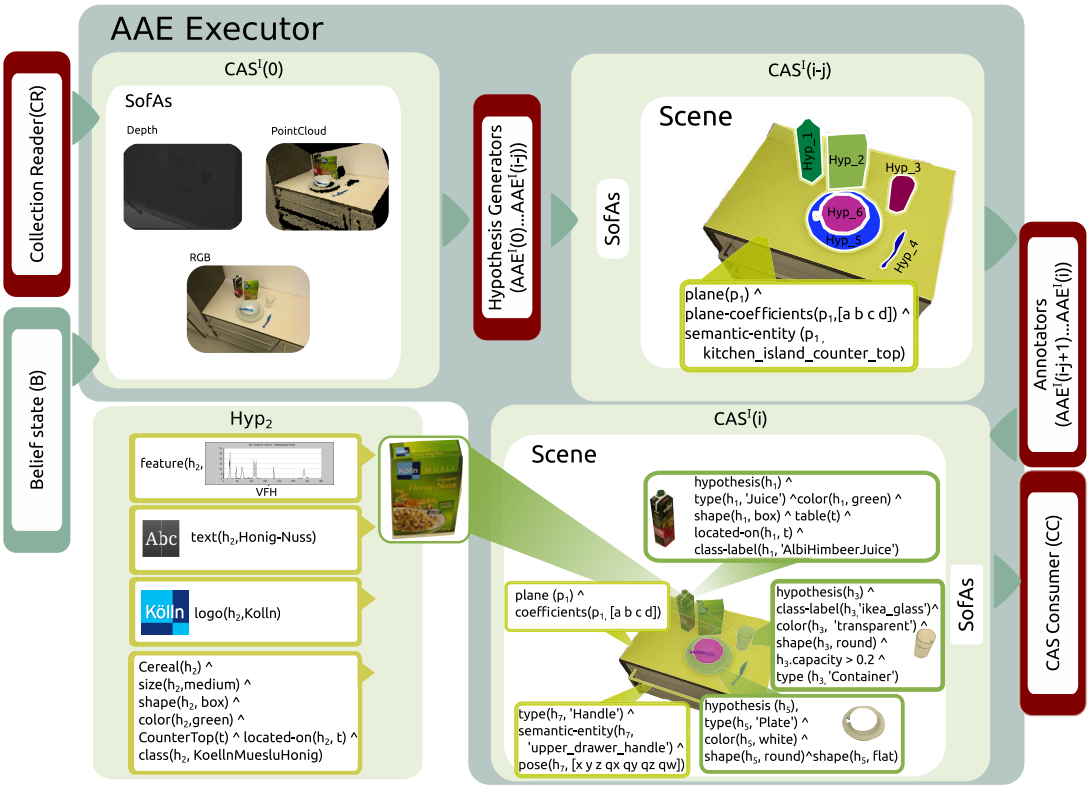


Figure 2.3: The Common Analysis Structure as it gets enriched by computational processes (highlighted in red) during a single iteration of a CPE, containing the interpretation of a kitchen breakfast scene

First hypothesis generators are run that find regions on the table that belong to objects. A second set of PAEs then annotate these hypotheses. Bottom left side of the figure highlights an object hypothesis that depicts a cereal box and examples of its annotations. Finally the CAS is passed to the CAS Consumers, that can incorporate results into the belief state, log them and generate answers.

2.4 Implementing a Perception System

Using the conceptual description of the RoboSherlock framework, in this thesis I propose a task-adaptable, pervasively operating perception system consisting of several computational components that enable a robot to perform everyday manipulation tasks. Figure 2.4 shows the high-level organization of the system, emphasizing the contributions of the thesis. At the center of the system is the

Collection Processing Engine (CPE) that is responsible for orchestrating the flow of information, reading in observations, executing a planned pipeline (AAE^I) and running several CAS Consumers for interpreting, further processing or storing the data.

Task adaptability is realized through the use of perception tasks formulated using the query language described in Section 2.3.2 and passed to the system's task interpreter. Generating all possible sequences of primitive AEs, as shown in Figure 2.2 is not feasible, instead, based on the query, the system plans a new sequence of PAEs specific to the query, using the proposed knowledge-based reasoning mechanisms.

In a typical execution cycle, after the initial CAS containing the raw data is passed to the AAE executor, the planned sequence of primitive AEs is run. PAEs mainly wrap perception algorithms performing various operations on the raw data. In the scenarios considered in this thesis, this sequence typically starts with the execution of object hypothesis generators. These analyze the images and detect regions that correspond to objects and object groups in the environment. Subsequent perception algorithms (referred to as *annotators*) analyze these hypotheses and annotate them with symbolic or numeric information. The results are inserted into the CAS as annotations of hypotheses. This process was already exemplified in Figure 2.3, where the object hypothesis generators have detected the objects on the table (indicated by the overlaid color boxes) and object annotators analyzed these generating descriptions of objects. The annotations of a hypothesis that depicts a breakfast cereal box are highlighted in the figure.

PAEs can also be categorized based on their applicability: general-purpose ones that run for any object hypothesis and task-specific ones that need a predefined condition fulfilled in order to be applicable. This distinction is easiest illustrated considering the scene depicted in Figure 2.3. A general purpose PAE would be a visual keypoint extractor, or a color histogram estimator for the objects in the scene. These algorithms can be run irrespective of the hypothesis and return annotations that can be useful for further PAEs that are executed later in the processing cycle. A task-specific annotator would be one that can only run and return positive results when task conditions are met. For example, an algorithm for drawer handle detection that identifies handles as 3D lines in the depth image.

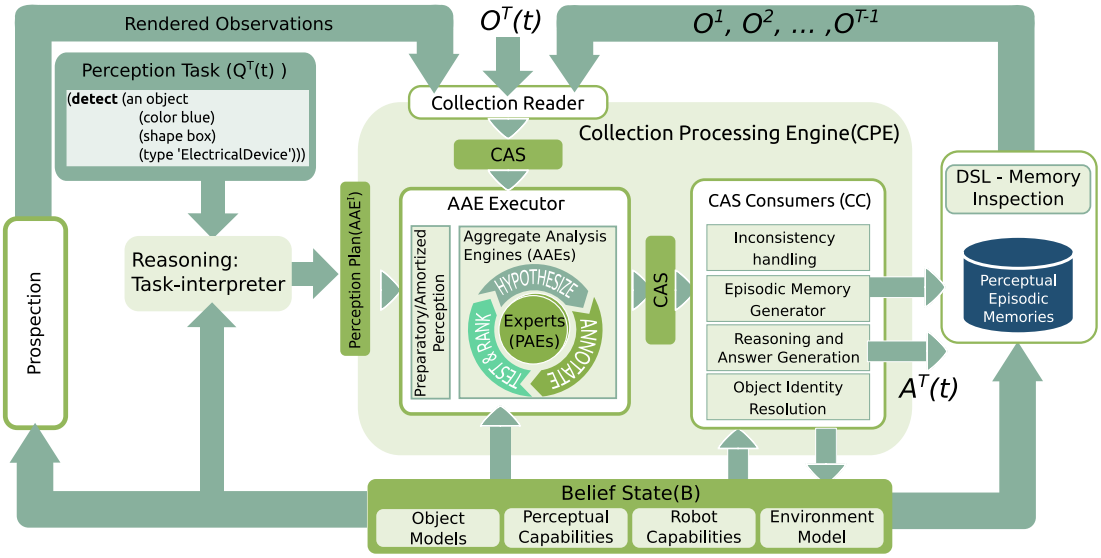


Figure 2.4: High level overview of the RoboSherlock framework showing the system components and the most important interactions

This annotator can only work and can only be used if the drawer handles in our world model meet the requirements. As another example, typically, the volume of an object is of no interest, unless a query specifically asks for it, so an annotator that estimates volumes of objects would only be run if needed. Planning the sequence of PAEs with the task-specific annotators is done through the application of knowledge processing and reasoning techniques, detailed in Chapter 3.

At the end of every processing cycle the CAS is transmitted to the consumers where further algorithms can analyze the generated hypotheses, filter them, merge them, etc. One such consumer is responsible for the generation of a final answer $A^T(t)$, by filtering results and applying knowledge-based reasoning. The belief state takes a central role when it comes to reasoning about tasks, storing static as well as dynamic knowledge about the objects, the environment, the robot and the available perceptual capabilities.

The components presented so far enable task-adaptability of the system. Pervasive operation is achieved through another set of computational modules that build on top of the task adaptable system. An extended version of the AAE executor proposed in Chapter 5 enables the preparatory, amortized perception to pervasively gather information about the environment and manage an object

belief state in an asynchronous manner. Several approaches for CAS consumers, for storing and retrieving perceptual episodic memories (Chapter 4), solving inconsistencies in hypothesis annotations through probabilistic first-order reasoning (Section 3.4) or enabling prospection (Chapter 6), complete the system. Figure 2.4 shows how and where these components are integrated in the proposed system.

All of these components will be presented in detail in the chapters to come. The remainder of this chapter gives an overview of the current state of RoboSherlock, offering implementation details.

2.4.1 Collection Readers

RoboSherlock is implemented using the Apache UIMACPP framework[2] and the Robot Operating System (ROS) middleware[3]. Interfacing with other components of the robotic system is done through standard ROS interfaces (topics, services and actions). In RoboSherlock an observation $O^T(t)$ corresponds to a set of images taken by the robot at timestamp t. *Collection Readers* are modules that read the images together with their metadata and store them in the CAS. In UIMA raw data to be processed are referred to as SofAs or *views*.

RoboSherlock interfaces to sensors of a robotic system through the means of a standard ROS server-client communication schema. While an observation $O^T(t)$ can come from any sensor, since the focus is on robot vision in particular, RoboSherlock is mainly designed to read sensory input from camera systems. This is realized through a general *CameraInterface* implementation that can be extended to handle various types of cameras, commonly used by autonomous manipulation robots. The main sensors used in this thesis are low-cost RGB-D sensors, such as the Microsoft Kinect v1 and v2[4], as well as Intel RealSense[5] or Asus Xtion[6] cameras. All of these sensors are well supported by the ROS middleware. Besides the sensors, *Collection Readers* can read data from other sources such as files or a database.

[2] https://uima.apache.org/doc-uimacpp-huh.html
[3] http://www.ros.org
[4] https://en.wikipedia.org/wiki/Kinect
[5] https://www.intelrealsense.com/
[6] https://www.asus.com/3D-Sensor/Xtion_PRO/

There are three main types of *Collection Readers* currently implemented:

1. **camera interfaces**:

   - *ROSCameraBridge*: interface for monocular cameras,
   - *ROSKinectBridge*: used for reading data from active RGB-D cameras,
   - *ROSRealSenseBridge*: similar to the KinectBridge, but developed for reading data from active stereo cameras such as the Intel RealSense camera series

   All of these interfaces include storing the camera intrinsic and extrinsic calibration information. In addition, filters such as blur or motion filtering are implemented in each.

2. **offline storage**: *DataLoaderBridge*, *MongoDBBridge*, for reading image data from a database or the file system.

3. **virtual cameras**: *UnrealVisionBridge* that acts like a normal camera but the data streaming are off-screen rendered depth and rgb images, with ground truth associated with them.

The images read by the camera interfaces are stored in the CAS with specific *view* names. Let us take an RGB-D Kinect sensor as an example. Typically, for such a sensor there are at least two image streams that are important (depth and color) and the camera parameters. The color image from the camera is stored with the name *VIEW_COLOR* and the depth image with the name *VIEW_DEPTH_IMAGE*. Additionally, if a point cloud is created, it is stored in a *view* with the name *VIEW_CLOUD*. This way any PAE that requires a color or a depth image or a point cloud simply needs to look in the CAS for a SofA with this name. Since most robotic systems can have several sensors of the same type, the *Collection Reader* can handle multiple cameras. This is achieved by generating a unique ID for each camera and pre-pending the name of the *view* with it. During the execution of the AAE a default camera ID can be set such that individual PAEs process the data from that specific sensor by default.

2.4.2 The Type System

The RoboSherlock *type system* is a taxonomy of data structures, where the name of a type also carries the semantic description of the information stored in it. For this reason the *type system* is considered to be a part of the *TBox* and the structures defined can be expressed using description logic, as I have shown previously. The UIMA framework supports the storage of basic data types (int, float, string, Boolean etc.) and arrays of these. The built-in type TOP acts as a parent type for all other types that will be defined. The *type system* is programming language agnostic, and is defined in a set ot XML files.

The types defined in the RoboSherlock *type system* can be categorized based on characteristics of the information they are supposed to store into three categories:

1. *low-level data types*: these types are data structures that are necessary in order to store results of computational processes performed on the input data. These are structures derived from the libraries that are often used in robot vision tasks. Converting from the native library data structure to the UIMA type is handled automatically. Specifically, the most prominent type categories are:

    - *cv_types*: types that store OpenCV (Bradski, 2000) specific data structures. Most notably the *rs.cv.Mat* type allows the storing of image data as OpenCV matrices. Other types are, for example, *rs.cv.Rect* for regions of interest or *rs.cv.Point* for storing a pixel location
    - *pcl_types*: types to store PCL (Rusu and Cousins, 2011) specific information, such as point clouds (*rs.pcl.PointCloud*) or point indices (*rs.pcl.PointIndices*)
    - *ros_types*: ROS and robot-specific types. These types mostly cover commonly used ROS messages, such as camera parameters, poses, or 3D transformations.

2. *annotation types*: types for storing results of PAEs that analyze object hypotheses. These types contain either numeric information (keypoints, feature descriptors) or symbolic information about the hypotheses (color, shape, class, etc.)

3. types *for storing hypotheses, objects and scenes*; since only data structures defined in the type system can be stored in the CAS, the type system defines higher level structures:

   - *object hypothesis*: type for representing regions in the image data that are believed to be objects. It is composed of an array of reference points in 3D, regions of interest in the image coordinates and an array of annotation types,
   - *objects*: a sub-type of *object hypothesis* with an additional array field for hypotheses ids. The type enables the storing of objects from the belief state inside RoboSherlock. Each object stores the series of hypotheses that were associated to it.
   - *scene*: the scene type allows the storage of what the robot believes to be true about the current observation, based on the readings from a specific sensor. It consists of an array of object hypotheses, the current location of the robot in the world, an array of scene-specific annotations (annotations that hold for all hypotheses in the current observation) and the time stamp.

The *type system* implemented in RoboSherlock serves only as a guideline for organizing the numeric and symbolic values extractable from image data. Depending on the application scenario, new types can be defined and integrated. The *type system* is a central element of the framework, since it acts like a common language between PAEs. For an analysis on how it compares to communication schema offered by other middleware, specifically how would a similar implementation be realized using standard ROS protocols, I kindly refer readers to the previous work of Blodow (2014).

2.4.3 Hypothesis Generation

In RoboSherlock, perception algorithms and existing perception systems are wrapped as primitive analysis engines (PAEs). The majority of these PAEs wrap perception algorithms implemented in PCL (Rusu and Cousins, 2011), OpenCV (Bradski, 2000) or ROS packages that are commonly used in robotics. Most of these algorithms can be assigned to one of the two categories of PAEs: hypothesis generators or hypothesis annotators.

Hypothesis generators are specialized segmentation algorithms that find regions in the data that depict objects and deal with objects that exhibit different perceptual characteristics, such as ordinary objects of daily use, flat objects, small shiny objects. These methods generate different kinds of representations:

- pixel coordinates generated by attention mechanisms that detect points of interest in order to create regions of interest (points and extents) in the camera frame,
- masks or region maps, referencing the respective part of the image generated by image segmentation algorithms (e.g. color-based),
- index vectors generated by point cloud segmentation relying on supporting planar structures.

These representations can be converted from one to the other, e.g. by projecting a point cluster from a point cloud into a camera image, or transforming an image region to a grasping pose in robot-local coordinates. This allows the retrieval of the camera image region of interest corresponding to a 3D point cluster, enabling the combination of image processing techniques and point cloud processing. I will now present the most commonly used PAEs in RoboSherlock for hypothesis generation.

RegionFilter Based on an existing semantic map of the environment (Pangercic et al., 2012) 3D regions relative to a fixed coordinate system in the environment are defined that are considered to be important in finding objects. The regions are typically 3D boxes, consisting of dimensions (height, width, depth) a rigid transformation and a semantic label. Based on these values, the RegionFilter

prunes the image and 3D data such that only measurements inside a given interest area are considered, given that the robot is localized in this map.

PlaneDetector Implements a RANSAC (Fischler and Bolles, 1981) based 3D plane estimation algorithm. This hypothesis generator detects supporting surfaces, which can subsequently be used to find objects on them. The output of the PAE is the plane equation and the array of inlier points.

PointCloudClusterExtractor This PAE implements a region growing algorithm, based on Euclidean distance in 3D space, to find objects on supporting surfaces. It requires a point cloud SofA and a detected supporting plane. The results of this hypothesis generator are clusters of 3D points representing clearly separable regions in the point cloud.

BinaryImageSegmenter Wraps a color-based segmentation useful especially in cases where objects are missed by the 3D clustering (e.g. flat cutlery in a table setting). It uses the color distributions of the supporting plane as background information and performs binary thresholding and blob detection.

SymmetrySegmentation Implementation of the 3D, model-free segmentation algorithm, presented by Ecins et al. (2016). Based on symmetry alone the algorithm can separate objects from cluttered, occluded scenes. The implemented PAE requires a point cloud SofA and optionally the supporting surface. This latter can simplify computations, and decrease processing time.

TransparentSegmentation Hypothesis generator that segments transparent objects. Transparency is one of the most challenging object properties for current computer vision algorithms. This PAE implements the approach introduced by Lysenkov et al. (2012), that takes a limitation of active depth sensors, such as the Microsoft Kinect (absence of measurement when objects are translucent), and turns it into a feature for detecting objects.

2.4.4 Object Annotators

Object annotators are the subclass of PAEs that enrich the CAS through the generation of annotations that are then attached to hypotheses. In general, annotators implement existing perception algorithms and result in numeric or symbolic values. From the large variety of PAEs that are implemented, I present only a subset of these that I consider to be important for everyday manipulation tasks and are used in the chapters to follow.

Some annotators make use of a 3D semantic map of an environment (Pangercic et al., 2012) in which the robot is localized to annotate an object cluster with a semantically meaningful location (e.g. "on top of counter_top#1"). Others, such as *PCLDescriptorExtractor*, process 3D point clusters and extract 3D feature descriptors implemented in the point cloud library PCL (VFH (Rusu et al., 2010), RIFT (Skelly and Sclaroff, 2007), SHOT (Salti et al., 2014), etc). Similarly to this, the *KeypointExtractor* finds image keypoints and calculates keypoint descriptors (ORB (Rublee et al., 2011), SIFT (Lowe, 1999), etc.) using their implementation from OpenCV.

Another set of annotators extract general properties of objects, such as color, size or shape, computing symbolic values for the hypotheses they are processing or numeric descriptions, such as an estimated 3D bounding box, with volume and estimated pose. There are annotators that wrap web services, such as Google Goggles, allowing the online analysis of an image. Google Goggles generates a highly structured list of matches, including product descriptions, barcodes, logo/brand recognition, OCR text recognition or a list of similar images (Blodow, 2014).

Some annotators wrap existing perception frameworks, like BLORT (Mörwald et al., 2010), Moped (Collet et al., 2011), Line-mod (Hinterstoisser et al., 2013) or SimTrack (Pauwels and Kragic, 2015). These are object recognition or categorization engines and are worth executing when the task involves an object that has a model. They also constitute an interesting category, since due to their bottom-down approach they produce and annotate hypotheses at the same time. Table 2.2 presents a more detailed view of PAEs that are implemented in RoboSherlock.

AE	Type	Description
ClusterColorHistogramAnnotator	Annotator	Returns semantic color annotation based on color distribution in HSV color space. Also annotates the hypotheses with a numeric color histogram
PCLDescriptorExtractor	Annotator	Extracts 3D features implemented in PCL (VFH, SHOT etc.)
KeyPointExtractor	Annotator	Extracts keypoints and calculates keypoint-descriptors using OpenCV (FREAK, FAST, SIFT, SURF, etc.)
RSSVM, RSRF, RSKNN, ...	Annotator	Support Vector Machine , Random Forrest, k-NN classifier wrappers for classification from OpenCV
CaffeAnnotator	Annotator	Extracts deep learning features using pre-trained models using the Caffe (Jia et al., 2014) framework
3DGeometry	Annotator	Estimates a pose based on a 3D oriented bounding box. Also classifies objects into *small*, *medium* or *big* depending on 3D volume
Goggles	Annotator	Sends the image of a region of interest of a hypothesis to the Google Goggles servers and parses the answer to extract text, logo, and texture information (Blodow, 2014)
PrimitiveShape	Annotator	Fits lines and circles to 3D point clusters projected onto the supporting plane using RANSAC (Goron et al., 2012). Values returned: *box, round, flat*
LineMod	Annotator & Hypothesis generator	Matches each object hypothesis to a set of object models that the robot should actively look for using the Line-Mod algorithm (Hinterstoisser et al., 2011).
SimTrack	Annotator & Hypothesis generator & Tracking.	Fits a CAD-model from a query to the current scene and tracks it (only works for textured objects) Pauwels and Kragic (2015).
TemplateAlignment	Annotator	Fits a CAD-model from a query to a hypothesis that was classified as the CAD label (Tombari and Stefano, 2010)
SACmodel	Annotator	Fits parametric models to objects if the query demands

Table 2.2: List of hypothesis annotators implemented in RoboSherlock

2.4.5 Implementing New Analysis Engines

Defining the Primitive and Aggregate Analysis Engines in RoboSherlock is done through a set of configurations files. Originally, in the UIM framework, this was done through the use of XML files. In RoboSherlock the less verbose YAML format was chosen. These configuration files are commonly referred to as descriptors, as their role is to describe what the analysis engine's role is.

Primitive Analysis Engines

Primitive Analysis Engines in RoboSherlock are the components that implement perception algorithms. There are several different PAEs already implemented and the framework supports easy addition of new ones. When creating a new PAE, developers are required to wrap the algorithm as a dynamic library using a predefined C++ interface and most importantly, each PAE needs to have a meta definition in the form of a YAML configuration file. The descriptor of a PAE that wraps a k-NN classifier is given below as an example:

```
annotator:
  implementation: rs_knnAnnotator
  name: KnnAnnotator
  description: wrapper for FLANN bassed Knn from OpenCV
parameters:
  class_label_mapping: extracted_feats/BVLC_REF_ClassLabel_pnp_5_obj.txt
  default_k: 5
  model_data: extracted_feats/BVLC_REF_5_obj.yaml
capabilities:
  inputs:
    - 'rs.scene.ObjectHypothesis',
    - 'rs.annotation.FeatureDescriptor': [BVLC_FC7]
  outputs:
    - rs.annotation.Classification: [CupEcoOrange, EdekaRedBowl,
     KoellnMuesliKnusperHonigNuss, BluePlasticSpoon, WeideMilchSmall]
```

The descriptor has three main parts: implementation data, capability definitions and parameter definitions. The first part (*annotator*) describes the PAE, defining the name with which it can be referred to and the implementation file name (library name). In the *parameters* section input parameters of the PAE can be defined. The parameters supported by the configuration file can be one of the standard data types or arrays thereof. The last part is perhaps the most important of them all. The capabilities of the PAE in terms of the RoboSherlock *type*

system and the *TBox* are defined. In the example above, the k-NN annotator takes as input data structures of type *rs.scene.ObjectHypothesis* and annotations of type *rs.scene.FeatureDescriptor*. Furthermore the *FeatureDescriptor* annotation needs to be a descriptor extracted from a deep learning module (BVLC refers to the reference implementation of AlexNet (Krizhevsky et al., 2012) in the Caffe deep learning framework (Jia et al., 2014)). The *KnnAnnotator* produces *rs.annotation.Classification* results, with a specified domain of objects. Based on these capability definitions the order of PAEs in an Aggregate Analysis Engine is deduced, as I will show in Chapter 3.

Aggregate Analysis Engines

Aggregate Analysis Engines are computational modules that are composed of a list of PAEs. We have seen that the sequence of primitive AEs in the aggregate are planned automatically based on the query that is asked. In the implementation of the system the list of PAEs that are used in the planning process need to be predefined. This way the planner has to only consider PAEs that are deemed necessary for that task a robot is solving. In practice this is done through the use of YAML configuration files. Such a configuration file is exemplified below:

```
---
ae:
  name: demo_aae
---
delegates:
  - ImagePreprocessor
  - RegionFilter
  - NormalEstimator
  - PlaneAnnotator
  - PointCloudClusterExtractor
  - Cluster3DGeometryAnnotator
  - CaffeAnnotator
  - ClusterColorHistogramCalculator
  - PrimitiveShapeAnnotator
  - KnnAnnotator
---
fixedflow:
  - ImagePreProcessor
  - RegionFilter
  - RegionFilter
  - NormalEstimator
  - PointCloudClusterExtractor
  - Cluster3DGeometryAnnotator
---
```

```
RegionFilter:
  semantic_map_definition: "semantic_map_iai_kitchen_ease.yaml"
  defaultRegions:['kitchen_sink_block_counter_top',
  'kitchen_island_counter_top','kitchen_meal_table_counter_top']
  border: 0.01
PointCloudClusterExtractor:
  polygon_min_height: 0.02
  cluster_min_size: 500
---
```

The configuration file is split into three sections: general information, list of delegate PAEs and definition of a *fixed flow*. Additionally to these three, parameters of individual PAEs can be overwritten. The *delegates* is used to define the list of PAEs that are used during the application. The *fixedflow* section defines an ordered subset of PAEs from the list of delegates, that constitutes a default ordering for continuous processing of input data. The end of the configuration file can be used to overwrite default parameters of PAEs from the delegate list, for example: in the configuration file above, the minimum size (number of 3D points) of a hypothesis, originally defined in the *PointCloudClusterExtractor* through the *cluster_min_size* parameter, is overwritten. Multiple AAE configuration files can be defined and loaded during runtime.

These components, namely *Collection Readers*, *Primitive and Aggregate Analysis Engines* are at the center of the framework, implementing various perceptual functionalities. References to these components will be made throughout the thesis when using them to realize task adaptability and pervasiveness.

2.5 Related Work

The scope of RoboSherlock goes beyond what currently constitutes the state-of-the-art perception systems. The goal is not to replace or reproduce any existing system, but to offer a framework where we can take advantage of the combination of these approaches. As such, the focus of the literature review offered in this section is to survey existing and previous perception frameworks or libraries and to highlight the current trends in algorithmic research in robot vision.

2.5.1 Frameworks and Libraries

There are a lot of works oriented at creating perception libraries which are a collections of task-specific algorithms, e.g. PCL (Rusu and Cousins, 2011), OpenCV (Bradski, 2000) or the V4R library [7]. The perception tasks demanded by the scenarios described in Section 1.3 go substantially beyond what is supported by current perception libraries and frameworks. These frameworks, while excellent from the perspective of algorithmic performance, do not encode the knowledge to automatically address higher level perception tasks, that is to offer solutions for the combination of the implemented algorithms that ease the development of a complete solution.

Frameworks, mostly based on middle-ware like ROS, such as SMACH (Bohren et al., 2011)) or REIN (Muja et al., 2011) have targeted the ease of program development but the problems of boosting perception performance through more powerful method combination has received little attention. An early example of a robotic perception system was described by Okada et al. (2007), where a particle filter based integration of multiple detectors and views was achieved. The probabilistic fusion of different results corresponds to a simple rule ensemble, i.e. one that is not trainable. Similar methods have been employed for semantic mapping approaches (Pronobis et al., 2010; Mozos et al., 2011). More recent works on general frameworks for robotic agents (Wyatt et al., 2010; Scheutz, 2006) approach the problem in a much broader sense, thus, do not address the specific needs of robot perception.

A recent highly integrated robotic learning system is presented by Skočaj et al. (2016), focused on interactive learning to associate linguistic and visual percepts by acquiring abstract knowledge of generic categories like object color, shape and type. The authors start with a comprehensive presentation of the field's development, and present a statistical approach for identifying missing information. To fill this knowledge gap, a human operator can be queried and perceptual tasks be performed automatically, or at the direction of the human. The presented system is highly complementary to RoboSherlock, as it focuses on building up a model database, learning what different shape, color and object

[7] https://www.acin.tuwien.ac.at/vision-for-robotics/software-tools/v4r-library/

type categories mean, and how to continuously update these beliefs using sensory data and interactions with humans. For these tasks RoboSherlock currently relies on an existing representation of the world and in this thesis the focus is on querying such common sense knowledge, instead of continuously learning it.

A related approach for actively generating pipelines to solve perceptual tasks was described by Sridharan et al. (2010), based on an efficient hierarchical partially observable Markov decision process formulation. The authors also focus on color, shape and type categories similar to Skočaj et al. (2016), and answer queries on object location, existence and identity, i.e. they also consider clutter segmentation by including a region of interest (ROI) splitting operator (which would be a hierarchical hypothesis generator in RS). The ROI splitting is a specific challenge for the POMDP, but overall the operators are simple, static and few. In contrast, RoboSherlock generates pipelines that are considerably more complex, and utilize background knowledge, but currently lack the probabilistic sequential decision-making during executions.

2.5.2 Algorithmic Developments

A lot of existing perception systems usually consider the case where a database of trained objects is used to match it with sensor data. Even more, many systems focus on individual algorithms that work on objects with specific characteristics, e.g. point features for 3D opaque objects (Aldoma et al., 2012a), visual keypoint descriptor based systems like MOPED (Collet et al., 2011) for textured or the work of Lysenkov et al. (2012) for translucent objects. RoboSherlock is capable of incorporating all of these different frameworks and combine their results meaningfully to build on the strengths and mitigate the weaknesses of the individual methods.

In recent years the research in the vision community has been dominated by deep learning. Initially only focusing on recognizing specific object instances and classes, recently the focus has shifted to recognizing and detecting objects (Huang et al., 2016) and estimating their pose (Li et al., 2018). One of the first well-performing algorithms that tackled object detection as a neural network regression problem is called YOLO (You Only Look Once) presented by Redmon and Farhadi (2016). Earlier algorithms usually used two separate steps for detection

and classification, e.g. (Girshick et al., 2014). The main advantages over those methods are the much faster processing speeds and the simple end-to-end training of the algorithm.

Liu et al. (2016) proposed a method called SSD to extend any existing image classification neural network to support the prediction of bounding boxes around objects. To that end, the existing network is truncated before the classification layers and an auxiliary structure, consisting of several convolutional layers, is added. Lin et al. (2017) recently introduced RetinaNet, with a proposed improvement to the loss function of object detectors, and obtained an upper envelope on existing methods when tuned for different inference time versus accuracy. While results are impressive and in some cases recognition rates are comparable with that of human performance, these approaches are rather complementary to the RoboSherlock framework, and can act as experts that hypothesize or annotate data.

Using commercial systems like the one offered by Cognex [8] or VIDI[9], all you need to do is provide a sufficiently large training set and you have a perception expert that can solve the specific task. The idea behind RoboSherlock is to build perception pipelines from the individual experts, not having to retrain networks for every single perception task that needs solving.

One of the exciting areas of research in deep learning, that relates well to the work presented in this thesis is that of visual query answering and description generation based on images. According to a survey on the *image to caption field* conducted by Bernardi et al. (2017), our work could be categorized as a *direct generation model*, whereby detections are made and then used by a following description generation step. However, the detection is guided by the task and by the a priori cues it receives from the high-level planning system. Caption generation can be performed using RNNs, for example by mapping sentences and images to a latent representation where they are related (Socher et al., 2014), and by applying LSTM or other architectures for further improvement (Donahue et al., 2017). However, most of them need image-sentence pairs for training, and the substantial time for retraining a network every time makes them so far not applicable.

[8] https://www.cognex.com/products/leading-technology/deep-learning-based-image-analysis

[9] http://imaging.market/vidi-green-object-scene-classification-10000337

Summarizing the most commonly used algorithms and toolboxes for object perception in robotics, Table 2.3 details a selection of these, highlighting their applicability and offering a short description for each. It is immediately visible from the table that, although there are a lot of overlapping and complementary approaches available, there are, as to date, no solutions that can cover all of the requirements imposed by the real world where we expect robots to work in. Hence, the investigation of how to reasonably combine these approaches is of the utmost importance.

	segmentation	recognition	tracking	pose estimation	textured	non textured	opaque	transparent	description
BLORT (Mörwald et al., 2010)		✓	✓	✓	✓		✓		interest point based object recognition, particle filter tracking, CAD model is needed for learning of models
RTM-Toolbox (Prankl et al., 2015)		✓	✓	✓	✓		✓		recognize and track objects based on interest points, RGBD-data is needed to create object models
SIMTRACK (Pauwels and Kragic, 2015)		✓	✓	✓	✓		✓		uses interest points for recognition and a CAD model is needed for learning of models
Line-Mod[10] (Hinterstoisser et al., 2013)		✓				✓	✓		detect and recognize non textured objects based on models trained from multiple views

[10]as implemented in OpenCV

MOPED Collet et al. (2011)		✓	✓	✓		✓		detect and recognize textured objects and their poses, trained on multiple views
Tenorth et al. (2013)	✓			✓	✓	✓		detect parts of objects based on their CAD-model
Marton et al. (2014)	✓	✓		✓	✓	✓		segments and recognizes objects in clutter, trained on partial 3D views of objects
Lai et al. (2011)	✓	✓		✓	✓			3D bag of words descriptors and kernel SVM for recognition; segmentation of bounding boxes around objects, CAD model needed for learning object models.
Lysenkov et al. (2012)		✓	✓				✓	CAD based transparent object detection and pose estimation
Ecins et al. (2016)	✓			✓	✓	✓		3D symmetry based segmentation of cluttered tabletop scenes
Tombari and Stefano (2010)		✓	✓	✓	✓	✓		Hough-voting based 3D CAD matching
Aldoma et al. (2012a)	✓	✓	✓	✓	✓	✓		table top segmentation using 3D point clouds and object recognition using a global or local matching

Redmon and Farhadi (2016) and Liu et al. (2016)	✓ ✓	✓ ✓ ✓	deep learning based semantic segmentation in image space returning object labels and 2D bounding boxes
He et al. (2017)	✓ ✓	✓ ✓ ✓	deep learning based pixel-wise semantic segmentation returning object labels and object-masks (in image space)

Table 2.3: State of the art open-source software commonly used in robotic perception, the perception-tasks

2.6 Summary

This chapter presented RoboSherlock, a cognitive perception framework that enables the creation of pervasively operating task-adaptable perception systems. The contributions to the framework are meant to deliver the necessary capabilities for implementing perception systems that provide the functionalities needed for everyday manipulation tasks, outlined in the introduction (Section 1.1):

- ... *can be tasked*: a query language was proposed that allows the description of a perception task. The framework adapts processing cycles to the descriptions of the task.
- ... *can be equipped with ensembles of expert perception algorithms*: RoboSherlock is based on UIM, as such, using ensembles of expert approaches is one of the core features of the framework.
- ... *is modular and extensible*: since RoboSherlock follows the processing logic of unstructured information management, it is highly modular. New Collection Readers, Collection Processing Engines and CAS Consumers can easily be added, extending current functionality through supporting new sensors, processing libraries.

... *can incorporate beliefs that it maintains about the world to improve future perception tasks*: we have already seen that the *type system* of RoboSherlock and the conceptual description all take into consideration past percepts. This capability of the framework will be the topic of chapters that follow.

... *can enhance perception with knowledge and reasoning*: a cornerstone of the framework, this will be the topic of the next chapter.

We have seen how perception for everyday manipulation tasks can be formulated as a query-answering problem that is dependent on the estimation of the current state of the world. Using the introduced formal description most perception processes can be defined. The concepts introduced in this section form the basis for the remainder of thesis, where I will detail parts of the system that contribute towards achieving a pervasive open question-answering perception for robots.

CHAPTER 3

Knowledge-based Perception

Mobile robots, operating in a human environment, face the challenge of recognizing objects found in visually challenging scenes and that posses a multitude of different characteristics. To address this challenge, perceptual capabilities of such robots need to go beyond detection or categorization of objects and be able to answer queries not only about where certain objects are located based on their class label, but also about properties of these. These properties are not always visual in nature. A lot of times achieving a task means identifying objects with functional properties (i.e. the cereal box that is open), or other characteristics (i.e. the one with no sugar). To achieve an optimal performance, perception systems need to be adaptable to the task that the robot is performing and the robots need to be aware not just of their environment and the task that they are to execute, but also of their perceptual capabilities that can help accomplish the task.

In the previous chapter we have seen that, in order to achieve task adaptability, perception for everyday manipulation tasks is treated by RoboSherlock as a question-answering problem and I have briefly discussed how knowledge-based reasoning plays a key role in this. RoboSherlock operates on captured images, in order to provide the robotic agent with the information and data necessary to successfully accomplish the intended actions. Consider, for example, a robotic agent setting the table. In this task context the robot needs to fetch plates and place them on the table. To enable this action the perception system has to provide information, including where to reach, where to grasp, how to avoid collisions when reaching for a plate, and so on. A precondition for answering these

queries is that the robot has recognized the plate to be grasped in all possible settings. This is not an easy task. Lying on the table, the plates can be detected by looking for ovals of the right size, or if the plates have a known texture, by detecting the respective texture. But, if the plates are stacked in a cupboard with poor lighting conditions then all the robot can see are the horizontal lines that individual plates form in the stack. The situation gets even more complicated when there might be a cup or a soup bowl placed on top of the plate or the plate is covered with food. The methods for detecting and categorizing the same objects in different contexts require very different perception strategies. Automatically choosing the right strategy depends on the knowledge that is available about the task and the objects.

A number of perception methods partly deal with some of these issues of determining *which objects stand where and how to act on them*. The set of algorithms developed and investigated for this problem category is huge. Some algorithms assume objects to be known beforehand and textured so that they can be detected by their visual appearance (Pauwels and Kragic, 2015). Others assume the shape of objects to be characteristic and they can detect objects such as plates, bowls, boxes (Mörwald et al., 2010). Many algorithms phrase the categorization of detected objects by learned classification systems that can be trained with huge numbers of images (Karpathy and Fei-Fei, 2014). Deep learning approaches for object categorization enable the capability of learning different representations for plates for different situations. Other approaches do not try to interpret the image

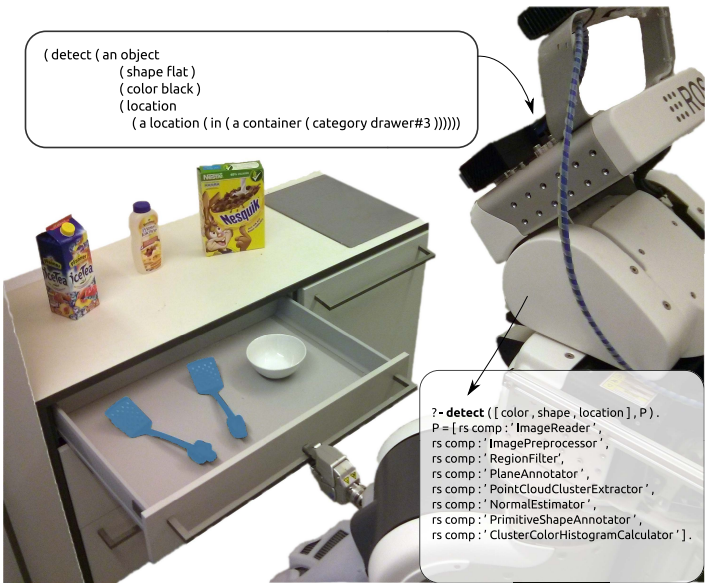


Figure 3.1: PR2 handling a query(top left) and detecting the corresponding objects using a planned pipeline (bottom right)

data to find out which objects are depicted, but rather approximate the shapes of objects sufficiently for grasping the object. Sporadic approaches investigated how to perceive more delicate characteristics of objects, (e.g. the actions they afford (Aldoma et al., 2012b)) and richer languages for object recognition based on perceptual attributes (Sun et al., 2013) and object parts (Felzenszwalb et al., 2010).

These methods have substantially advanced the state-of-the-art in robotic object and scene perception, but a central question still needs to be addressed: *how should a robotic agent decide which approach to use in which situation.* To formulate an answer to this question, **I propose to consider robot perception as a hybrid system that combines knowledge processing and image interpretation in order to answer the queries that the robot control system asks**. In this context knowledge processing can take on two important roles: first, it can be used to select problem-specific methods that can achieve higher recognition rates on restricted sets of objects and scenes and second, it can be used to interpret, disambiguate and correct the results of perception processes. This is similar to the manner in which expert systems work. In other research fields it has been demonstrated that the amount and quality of knowledge that a system has about the process is equally important as the reasoning mechanisms it uses. Feigenbaum, considered to be the father of expert systems, in his seminal work (Feigenbaum, 1992), formulated the following :

> “ .. knowledge of the specific task domain in which the program is to do its problem solving was more important as a source of power for competent problem solving than the reasoning method employed ”
>
> *Edward A. Feigenbaum*

This same idea can directly be applied to robotic perception when pursuing the creation of a system that scales towards perception tasks. By symbolically representing the task a robot should solve and the robot's perceptual capabilities, we can enable reasoning about the choice of methods for a given task and about the interpretation of the results of these methods. Perceptual capabilities in this sense refer to knowledge about a robot's sensing capabilities as well as its

computational ones, i.e. the algorithms that are able to process the data from the sensors.

In this chapter I further detail how the task adaptability of the proposed system is achieved and as such I will present how RoboSherlock uses knowledge-based reasoning techniques to answer the perception queries it receives. Specifically I am proposing:

- a representation of perception experts that specify their preconditions and the output information,
- a representation of complex specialized perception processes as perception plans,
- logical rules that automatically generate perception plans, a sequence of Primitive Analysis Engines (PAEs) tailored to given perception tasks,
- logical rules that extract the information requested by a given perception task from the perceptual evidence combined with the background knowledge of the robotic agent,
- a probabilistic approach for handling inconsistencies and uncertainty of results generated by PAEs.

The chapter concludes with an analysis of how query answering, together with knowledge-based reasoning operates in the application scenarios presented in Section 1.3. Contents of the chapter are mostly based on previous work described in Bálint-Benczédi et al. (2016) and Bálint-Benczédi et al. (2019) offering additional information about the reasoning mechanisms used for generating a task-dependent perception plan.

3.1 Knowledge Representation

In RoboSherlock I chose description logic (*DL*) as the representation language for two reasons: (1) DL provides an intuitive, standardized and clean way of constructing models of the real world and (2) DL has well-established implementations used in both the knowledge representation and the AI-based robotics communities. For implementing these representations KnowRob (Tenorth and Beetz, 2013) was chosen as the knowledge processing and inference framework.

KnowRob is designed to be used on robots, can load knowledge, stored in the Web Ontology Language (OWL) (Bechhofer et al., 2004) and can reason about the facts stored using first-order logic, implemented in the logic-based programming language Prolog.

3.1.1 Description Logic in KnowRob

KnowRob stores information in OWL ontologies. Ontologies in general are a form of representing pieces of knowledge and the interdependent properties and relations between these, while the Web Ontology Language (OWL) is a family of languages for representing, storing and exchanging the stored knowledge. The OWL family of languages support a wide variety of syntaxes. In KnowRob the Relational Data Format (RDF) XML syntax is used to store the OWL files. KnowRob has its own upper ontology, based on OpenCyc (2009), extending this with concepts that are relevant to the application domain of robots performing everyday manipulation tasks.

When representing knowledge in KnowRob we have the choice of defining three types of knowledge:

... **classes**: in description logics also referred to as the concepts, represent static knowledge about the world;

... **properties**: object or data properties that can describe relations between classes and individuals; in description logic terms are referred to as roles;

... **individuals**: instances of classes that can take additional properties, and are specific to the current execution.

The most common tasks performed by description logic reasoners deal with class subsumption and classification (e.g. *is this individual of type A?*). Tenorth and Beetz (2009) argue that classical reasoners in description logic are not suited for robotic tasks due to several causes:

1. a classified world is stored in memory: this is not ideal, since the world changes during robot operation. It is computationally inefficient to reclassify the whole world every time new information is available;

2. knowledge is not static: when the robot is started, not all knowledge is immediately available, hence inference methods need to be modified to take into account both preexisting and runtime-inserted knowledge.

To address these issues, KnowRob introduces Prolog-based reasoning. In Prolog, the knowledge is represented in terms of Prolog predicates to which the common Prolog inference methods can be applied. The knowledge stored in the OWL files can easily be loaded using the Prolog Semantic Web Library (Wielemaker et al., 2003) and an OWL parsing library (Vassiliadis et al., 2009).

3.1.2 *Representing* RoboSherlock*-specific Knowledge*

To facilitate reasoning about objects of daily use, their visual appearance and the algorithms implemented in RoboSherlock, I have extended the base ontology of KnowRob with concepts that represent these. Figure 3.2 shows parts of the extended ontology, where different types of classes are color-coded, based on what they describe. Extending an already existing ontology has several benefits. Classes can be reused making implementational efforts simpler. Most importantly though, integrating with other software components of the robotic system that use the same base-ontology insures a common representation, enabling a smoother integration. For example, the class `RoboSherlockComponent` is defined as the child of the already existing `Algorithm` class, allowing all components of the robot system to reason about these if needed. `VisualAppearance` is introduced as a new class for representing the appearance of objects. A new class for the *type system*, `RSType` is introduced and classes for several objects of daily use are added. New properties are defined (e.g. `perceptualInputRequired` or `perceptutalOutput`) that are meant to connect subclasses of `RoboSherlockComponents` with subclasses of `RSType`, expressing pre- and post-conditions of RoboSherlock annotators in terms of the *type system*. Table 3.1 presents the most important classes that were manually added to the ontology.

While the main categories in the ontology are designed by hand, subclasses of the `RoboSherlockComponent` and `RSType`, as well as relations between these classes are automatically generated based on definitions in RoboSherlock. This is possible due to the fact that in RoboSherlock each implemented Primitive Analysis Engine (PAE) has a descriptor stored as a YAML file. The descriptor

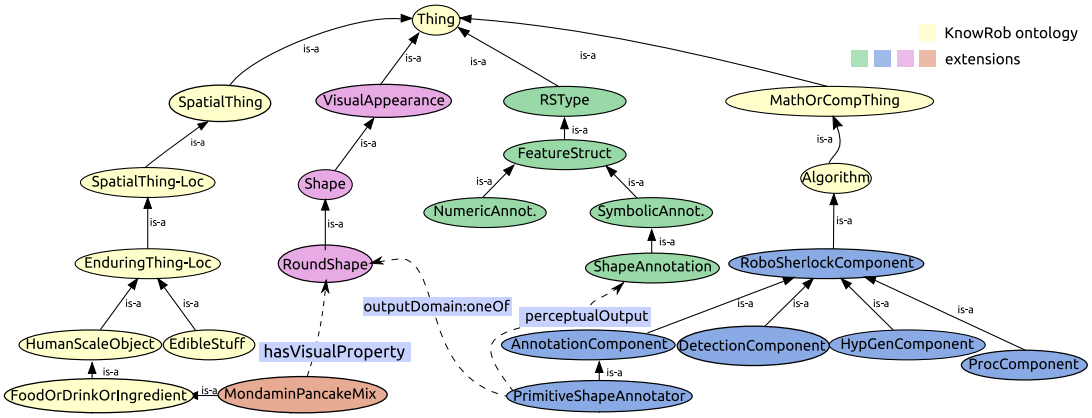


Figure 3.2: The extended ontology that is used to reason about perceptual capabilities and generate task- and object-dependent perception pipelines

contains information about the type of the expert, its input requirements and its output produced in terms of the *type system* as I have detailed this in Section 2.4.5. Based on these descriptors an OWL ontology containing the implemented PAEs is generated. The terms of the query language presented in Section 2.3.2 are also mapped to entities in the ontology, defining which types from the *type system* correspond to which terms of the language.

The currently implemented PAEs range from image or point cloud segmentation algorithms used for hypothesis generation to more complex model-based detectors, deep learning approaches or web-based annotators. I refer to the symbolic representation of these functionalities collectively as the *perceptual capabilities* of the framework, represented in the ontology as subclasses of the `RoboSherlockComponent` class. The specific algorithms implemented in RoboSherlock and used in experiments were detailed in Section 2.4. Based on the functionalities of algorithms, in addition to the two main categories of PAEs, outlined in Chapter 2 (hypothesis generators and annotators), another three categories are defined:

- **ProcessingComponents:** experts that are tasked with transforming sensory input and performing pre-processing (e.g. smoothing of 3D data, normal estimation, point cloud creation etc.),
- **DetectionComponents:** experts that wrap other perception frameworks, mostly model-based detection methods (e.g. BLORT (Mörwald et al., 2010),

Concept	Property	Description
`RoboSherlock-Component`		super type for all RoboSherlock components
`RSType`		parent class for all types, equivalent to the *uima.TOP* type from the UIMA type system
	`outputsType`	describes the output types produced by a PAE
	`inputTypeRequired`	specify the input types required by a primitive analysis engine
	`ouputTypeDomain`	set the value domain of the type that is returned
	`inputTypeRestriction`	set restrictions on the input values a type should accept

Table 3.1: Manually introduced concepts in the ontology that are used to express capabilities of the RoboSherlock framework

Simtrack (Pauwels and Kragic, 2015) or Line-Mod (Hinterstoisser et al., 2011)); These methods form a separate category since they can act as both annotation and hypothesis generation components,

- **ContinuousComponents:** perception experts that require a stream of images in order to solve a task (e.g. object tracking or multi-view perception).

For the purpose of this thesis I consider that each of the PAEs defined in RoboSherlock belongs to one of these five categories, although, if needed, new ones can easily be added (e.g. for further specializing the existing classes). As an example of a generated OWL class, I present the `PrimitiveShapeAnnotator` class. While knowledge in KnowRob is stored as OWL files using the XML RDF schema, in the following examples the more readable and less verbose Manchester syntax is used:

```
Class: PrimitiveShapeAnnotator
  SubClassOf : AnnotationComponent
    dependsOnCapability some Perc3DDepthCapability
    inputTypeRequired some RsPclNormalsCloud
    inputTypeRequired some RsAnnotationPlane
    inputTypeRequired some RsSceneCluster
    outputsType some ShapeAnnotation
```

This specifies that the annotator requires the surface normals of the input point cloud, a supporting plane and an existing 3D cluster, and it produces a `ShapeAnnotation` defined through the `perceptualOutput` object property. The fact that the robotic agent needs to be able to perceive depth information in order to use this module is encoded by the `dependsOnCapability` property.

The children of the `VisualAppearance` class can be used to describe objects through their visual properties. Using these descriptions, an object is described as:

```
Class: MondaminPancakeMix
        SubClassOf: FoodOrDrinkOrIngredient
        hasVisualProperty some MediumSize
        hasVisualProperty some MondaminLogo
        hasVisualProperty some YellowColor
```

Since visual properties of objects in the ontology are expressed using the same symbolic values as the output domains of expert algorithms (in Figure 3.2 exemplified using the dashed arrows), reasoning about which perception expert to run for different descriptions of objects is possible. At the moment visual properties of the objects found in the knowledge base are manually inserted, but parts of these can also be learned, as we have shown in previous work (Nyga et al., 2014) and will detail in Section 3.4.

The notions presented so far are part of the *TBox*, representing knowledge that is independent of a specific instance of a perception system created using RoboSherlock. While some properties of a Primitive Analysis Engine (PAE), such as the type of input data required, are static and depend on the implementation of the analysis engine, others can depend on the values of certain parameters. Because of this, during runtime instances of the delegate PAEs are created and reasoning about which is the appropriate PAE to execute is decided based on these. Knowledge that is specific to a certain system created using RoboSherlock, such as certain parameters of a PAE, or the list of delegate PAEs that are instantiated are asserted during runtime. Such assertions are done through a set of Prolog predicates that are defined for this purpose. For instance, creating an individual of a class happens through the use of the `owl_instance_from_class` predicate defined in KnowRob. The rule takes a class name from the ontology and creates a unique instance of it that has all

the properties the class has, with the benefit that additional relations can be defined. A typical situation where reasoning based on instances instead of classes is desired is when a specific parameter changes the output the PAE produces. A good example of this are primitive analysis engines that wrap supervised learning algorithms, such as classifiers. In the case of classifiers the dataset that they were trained on is important, as is the feature descriptor type used for training. To represent this knowledge, restrictions are added to the input and output types of PAEs, using rules defined in Prolog. Output domains for certain types can be set using the `set_annotator_output_type_domain` rule:

```
?-set_annotator_output_type_domain(Instance, Domain, Type):-
    owl_individual_of(Instance,'RoboSherlockComponent'),
    owl_individual_of(Instance,Annotator),!,
    compute_annotator_outputs(Annotator, Type),
    owl_restriction_assert(restriction(Type,
                           all_values_from(union_of(Domain)))
                          ,R),
    rdf_assert(Individual,outputTypeDomain,R).
```

The rule begins with sanity checks: is the specified instance a `RoboSherlockComponent` and if so, is the type we want to set a domain for among the output types of the annotator. If these conditions are met, a restriction R is created. Finally, the restriction created is connected to the individual using the `outputTypeDomain` relation. For example, setting that the `KNNAnnotator`'s output is one of the symbolic labels "Cup", "Bowl" or "Plate" is asserted as:

```
?-set_annotator_output_type_domain(
        rs_components:'KNNAnnotator_ASDRA',
        ['Cup', 'Bowl', 'Plate'],
        rs_components:'ClassificationAnnotation'
).
```

Similarly, the rule `set_annotator_input_type_constraint`, for adding restrictions on input types, is defined as:

```
?-set_annotator_input_type_constraint(Individual, Constraint, Type):-
    owl_individual_of(Individual,'RoboSherlockComponent'),
    owl_individual_of(Individual,Annotator),!,
    compute_annotator_inputs(Annotator, Type),
    owl_restriction_assert(restriction(Type,
                           all_values_from(union_of(Constraint))),
                           R),
    rdf_assert(Individual,inputTypeRestriction,R).
```

This rule also begins with a sanity check, the only difference is that instead of the `outputTypeDomain` property the `inputTypeRestriction` property is set to store the new relation. For example, given that the `KNNAnnotator` was trained using VFH 3D descriptors (Rusu et al., 2010), this type of feature needs to be present in order to run this primitive AE:

```
?-set_annotator_input_type_constraint(
    rs_components:'KNNAnnotator_ASDRA',
    ['VFH'],
    rs_components:'FeatureDescriptor'
).
```

Based on the presented ontology of objects and the perceptual capabilities of RoboSherlock I define Prolog rules to reason over the relation of these. While a subset of the rules are designed to be generic, such as reasoning about which PAEs result in certain annotations, others are more specific and tailored to a task or an object.

3.2 Generating a Perception Plan

A perception plan in RoboSherlock is an ordered list of Primitive Analysis Engines (PAEs) that hypothesize and annotate the raw data. The sequence of PAEs forms an Aggregate Analysis Engine, formally defined as AAE^I (see equation 2.6) that I commonly also refer to as a perception pipeline. In Unstructured Information Management (UIM) the AAE is controllable through a *flow-controller*. A *flow controller* gets its name from the fact that it is responsible for the *flow* of the data through the PAEs. Although flow controllers are basic building blocks of the UIM paradigm, in RoboSherlock they are re-implemented[1] and as such in the proposed system are referred to as AAE executors (Figure 2.2) to denote this difference.

The sequence of PAEs in RoboSherlock can either be predefined or generated based on the task description. When a query is received it is first parsed. Depending on the type of query (detect, inspect, etc.) different reasoning mechanisms are used. In the case of a *detect* query, the description of the object to

[1] the `uimacpp` library that RoboSherlock is built on top of does not support the implementation of custom flow controllers

be detected is parsed, key-value-pairs extracted and asserted into the knowledge base using the following Prolog rule:

```
?- parse_descriptions(Descripion, KvPs),
   assert_requested_key_value_pairs(KvPs).
```

The first set of rules infer which PAEs need to be run in order to answer a given perception task. This set of rules allow the reconfiguration of the processing pipeline, inferring missing experts and planning their execution sequence. They can take as an input either the name of an object defined in the knowledge base or a list of visual characteristics in the form of predicates. The logical rule that takes as input the object name is:

```
?-build_pipeline_for_object(Obj,Pipeline):-
    attributes_for_object(Obj,ListOfAttribs),
    enum_annotator_sets_for_pred(ListOfAttribs,Annotators),
    build_pipeline(Annotators,Pipeline).
```

This rule starts with the retrieval of the perceptual description of the queried object as a list of attributes, then searches for the set of annotators that have the desired outputs based on their `perceptualOutput` object property from the ontology. Finally, the order of the pipeline using the *build_pipeline* rule is deduced. Simplifying the rule by substituting the object with the list of attributes, the retrieval of these can be skipped and the PAEs can be searched for directly. For both cases the *build_pipeline* rule is the same and is detailed in Algorithm 1.

The algorithm takes as its input the query $Q^T(t)$ and the belief state containing the robot capabilities. Recall from Section 2.4.5 that each AAE defines a default ordering of PAEs for continuous processing, called the *fixed flow*. The algorithm first checks if the requested query terms are satisfied by the continuous processing component of the Aggregate Analysis Engine. If it does not, the missing annotators are retrieved, added to the list of PAEs to be executed and brought in the correct execution order based on their defined input and output conditions. The `robotMissesCap` rule used in the algorithm illustrates how the reasoning capabilities about robot hardware and the perceptual capabilities of RoboSherlock are integrated. The capability reasoning uses a model of the robot described using the Semantic Robot Description Language (SRDL), which includes information about the sensors. In the case of a PR2 robot, the perceptual capabilities that the robot possesses can be asked as:

Algorithm 1: Generation of the sequence of PAEs for a query

Data: $B^T(t)$, $Q^T(t)$
Result: AAE^I - ordered set of PAEs

$AAE^I \leftarrow$ getExpertsFromContinuousExec();
if ***not*** *fulfills(AAE^I, $\mathcal{Q}$)* **then**
  $AAE^I \leftarrow$ extendExpertsForQuery($Q^T(t)$, $B^T(t)$);
//initialize AAE^I with PAEs for the given query and robot capabilities;
for $i \in I$ **do**
  prec-experts $\leftarrow$ preconditionsOfExpert($AAE^I(i)$);
  if $\exists$ *prec-experts* $\wedge$ *robotMissesCap(prec-experts)* **then**
    return *An Error*
  end
  if $\exists$ *prec-experts* **then**
    prec-experts $\leftarrow$ prec-experts $\cup AAE^I$
  end
end
while *not completePipeline(AAE^I)* **do**
  $AAE^I \leftarrow AAE^I \cup$ getMissingExperts(AAE^I);
end
$AAE^I \leftarrow$ sortByDependencies(AAE^I);

```
?- perceptual_capabilities_on_robot(Cap,pr2:'PR2Robot1').
   Cap = rs_comp:'Perceive3DDepthCapability' ;
   Cap = rs_comp:'PerceiveColorCapability' ;
```

Verifying if a certain RoboSherlock analysis engine can be run on the robot can be done using the following rule:

```
?-action_feasible_on_robot(rs_comp:'Cluster3DGeometryAnnotator',
                           pr2:'PR2Robot1').
  true.
```

When planning a new perception pipeline the above rule is used to determine whether an expert can run or not on the current hardware. A complete example of a pipeline planning is presented in the experiments section of this chapter.

3.3 Reasoning About Scenes and Task Context

The use of KnowRob concepts and predicates ensures that we can make use of a large body of knowledge about objects, their properties and also robot

descriptions. All of the RoboSherlock component classes in the ontology have the object property `dependsOnCapability`, which creates a link between perception capabilities of the robotic agent and those of the perception framework. Using existing rules, we can check if a robotic agent has the necessary sensory capabilities to run a specific PAE:

```
?-expert_feasible_on_robot(Expert, Robot) :-
    missing_cap_for_action(Expert, Robot, _),
    missing_comp_for_action(Expert, Robot, _).
```

I further extend the pipeline planning rules to enable queries for a broader range of terms, using background knowledge about the objects. Each object defined in the knowledge base can have several parent classes. For example, a ketchup bottle is a child of the `Bottle` class as well as the `FoodOrDrinkOrIngredient` class. The first extension facilitates the recognition of all objects in a scene that are of a certain type. This allows the system to retrieve objects in a scene that are of type *food*, *drink*, *ingredient* or that are bottles or perishable items. An example for such a rule is:

```
?-build_pipeline_for_subclass(Obj, Pipeline, ChildObj) :-
    owl_subclass_of(ChildObj, Obj),
    build_single_pipeline_for_object(ChildObj, Pipeline).
```

Using this rule we can plan a perception pipeline for each of the subclasses of Obj, resulting in a list of $ChildObj, Pipeline$ pairs. The rule can be further extended by replacing the search for subclasses with the lookup of object properties of a class. In latter stages these properties that can either be visual ones (color, texture etc.) or affordances (has a lid, is graspable, etc.), are used to generate perception pipelines for the objects that posses them. Through the combination of the object representations and the extensions to the pipeline generation rules, RoboSherlock is able to answer queries based on descriptions that are not necessarily related to visual appearances.

Since the results are used to update the belief state, it is possible to examine them and formulate even more complex queries that require further reasoning about the object's properties. In most cases these fall in the category of *inspect*ion queries from the definition of the query language. From the name of a detected object we can infer certain properties that can inform the system to run a different set of PAEs to further analyze it. For example, if the detected object is a container

and its detected shape was cylindrical, the inspection pipeline might contain a cylinder model fitter to estimate its volume.

```
?-detect_volume(Obj,P):-
    owl_subclass_of(Obj,knowrob:'Container'),
    obj_has_predicate(cylindrical_shape, Obj),
    examine_pipeline_for_object(Obj,P).
```

Another example would be to search for objects that are most of the time found in combination with other objects but are harder to separate visually (e.g. cutlery can be found near plates, a pancake is on the pancake maker, etc.). Once we have the already perceived objects asserted we define the following rule to build special pipelines that take into consideration the interdependence between objects.

```
?-detect_if_individual_present(Obj,DependentObjectClass,P):-
    owl_individual_of(_, DependentObjectClass),
    build_pipeline_for_object(Obj,P).
```

Combining these logical rules enables interpreting scenes with the help of knowledge-based reasoning. Through reasoning we can, for example, infer that plates can be stacked or cups can be found on plates, which in turn informs the perception framework to take special care of these situations.

3.4 Handling Uncertainty

When planning a perception plan, it is possible that multiple PAEs that return the same type of information are inserted into the pipeline. An interesting issue is how can the system handle uncertainty, especially in the case where we have multiple PAEs of the same nature (e.g. model fitting or shape classifiers) returning contradictory results. To address this problem I present a component that is meant to handle the inconsistent, uncertain results of individual annotators. This component is implemented as a CAS Consumer. The main purpose of CAS Consumers is to analyze the hypotheses generated during a processing cycle and perform various post-processing steps on them, such as identity resolution, answer generation or handling contradictory data. A particularly powerful method for resolving inconsistencies in RoboSherlock is the application of first-order probabilistic reasoning, as described in our previous work (Nyga et al., 2014). We propose using Markov Logic Networks (MLNs) to leverage the power of the probabilistic

reasoning, learning models of which PAE performs best for different situations and use these as priors for choosing the PAEs that are most probable to detect the queried items. This work was conducted together with Daniel Nyga, where he contributed by providing the probabilistic framework (Nyga, 2017), designed the model and conducted the experiments, whereas I supplied the perceptual data for the experiments and integrated the results with RoboSherlock.

3.4.1 Overview

The key idea of the proposed method is as follows. The robot asks queries, such as: "is the category of the object hypothesis h_1 a cereal, given that the color of h_1 is yellow and the shape of h_1 is a box?", or in the query language: `(detect (an object (type cereal) (color yellow) (shape box))`. The query is transformed into relational conditional probability *P(category(h_1, cereal), color(h_1,yellow), shape(h_1, box) | E)* that takes the observed scene as its evidence. Treating uncertainty handling as a relational probabilistic reasoning problem has several advantages over alternative approaches. First, using perception algorithms wrapped in PAEs for collecting perceptual evidence rather than making decisions, makes the use of multiple specialized algorithms straightforward: they simply add their findings as annotations. Second, as inferences are drawn probabilistically on the basis of collected evidence, the system models inconsistent annotations. Third, the system can answer queries concerning all aspects contained in the probabilistic model under the given evidence. Fourth, the perception system can also exploit the regularities of the domain with respect to objects and their appearance and the occurrence/co-occurrence of objects in scenes. The approach is validated showing how the combination of elementary perception mechanisms significantly boosts recognition rates, while also demonstrating how our approach can be thought of as more than just a recognition framework.

Figure 3.3 gives a detailed overview of the components of our approach. It consists of three main components: (1) RoboSherlock as an image annotation component that generates object hypotheses and annotates these (2) a statistical relational learning system that learns joint probability distributions over annotated scenes, and (3) the probabilistic reasoning system that is wrapped as a CAS Consumer into RoboSherlock.

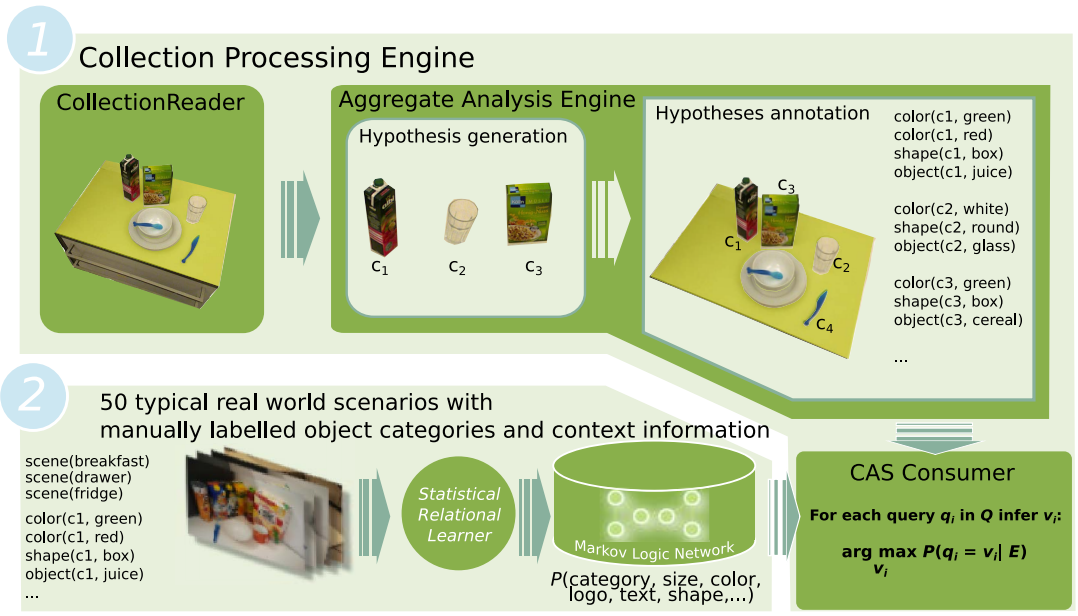


Figure 3.3: Architecture of the system: (1) generation of annotated hypotheses, (2) the statistical relational learning and (3) reasoning system as a CAS consumer. *(Adapted from previous work, by courtesy of Daniel Nyga)*

For the purpose of illustrating the method, a subset of the PAEs is used to generate and annotate hypotheses, details of which are presented in Section 3.5.5. Object hypotheses are annotated with shape and color information and compared to known object models using the Line-Mod algorithm (Hinterstoisser et al., 2011). Additionally, text and logo annotations are generated using the Google Goggles web service.

Given the annotations of objects AS, the probabilistic reasoning component of the perception system can be used to answer queries about any aspect of the respective probabilistic model. The probabilistic model is given by the joint probability distribution over the combination of the categories of objects and all possible annotations. The answer to the query Q is then $argmax_Q P(Q \mid AS)$.

The learning process of the joint probability distribution over annotated scenes is depicted in the lower part of Figure 3.3.

3.4.2 Information Fusion

(adapted from previous work by courtesy of Daniel Nyga)
Since most of the PAEs producing object hypotheses are applied independently of each other, their outputs are not guaranteed to be globally consistent and they typically do not take into account object interactions in the current scene. In fact, their annotations might even be incorrect or contradictory. Therefore, in order to come up with a final ensemble decision, a strategy for combining all the annotations is needed.

To this end, we apply state-of-the-art methods from the field of Statistical relational learning (SRL), a subfield of the machine learning discipline that has emerged and gained a lot of attraction in the recent couple of years. In SRL models, we can capture complex object interactions, represent and reason about object properties, their attributes and the relations that hold between them. Most notably, the ultimate strength of SRL models is their capability of allowing for reasoning about *all* observations simultaneously, taking into account interactions between objects and thus achieving a posterior belief that is guaranteed to be probabilistically sound and globally consistent.

In particular, we employ MLNs (Richardson and Domingos, 2006), a powerful knowledge representation formalism combining the expressive power of first-order logic (FOL) and the ability to deal with uncertainty of probabilistic graphical models. As opposed to most traditional machine learning approaches, learning and reasoning in MLNs is not restricted to a feature vector of fixed length, but is rather performed on whole databases of entities and relations. Since the chosen representation for our belief state, description logic, in general is a subset of FOL, going from one representation to the other is straightforward.

Maintaining a joint probability distribution over observations, their class labels and the robot's current task context and belief state, has several advantages over classical approaches and makes the system's reasoning capabilities go far beyond traditional classifier systems:

- **collective classification** MLNs are able to simultaneously take into account any arbitrary but finite number of objects for classification. This is an important feature for a perception system, since it captures interactions be-

tween objects in a scene. If a classification system is aware of the probability of jointly encountering two objects of particular types, this can tremendously boost the classification accuracy in real-world scenes. Encountering milk and cornflakes together on a table, for instance, is much more likely than finding cornflakes and ketchup.

- **confidence-rated output** Since the MLN for compiling annotations to a final decision is stacked upon the independent application of PAEs, such a probabilistic model is able to compensate for inconsistent annotations or uninformative features. If, for example, an annotator systematically confuses the shapes of clusters, the MLN will learn these erroneous hypotheses and treat them in a meaningful way.

- **generative models** An MLN representing a joint probability distribution can be used to infer answers to arbitrary queries about any aspect represented in the model. As our experiments will show, the MLN can also be used to reason about the most informative visual features when looking for a particular type of object in a scene.

- **ease of extension** integration of additional task-specific context information, or new specialized perception routines is straightforward. They just need to add their annotations to each of the object hypotheses and can be declaratively incorporated in the MLN.

More formally, an MLN consists of a set of formulas F in first-order logic and a real-valued weight w_i attached to each of those formulas F_i. The probability distribution over the set of possible worlds $\mathcal{X}$ represented by the MLN is defined as follows:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right), \tag{3.1}$$

where x is a complete truth assignment to all predicate groundings X (i.e. one possible world), $n_i(x)$ is the number of true groundings of formula F_i in x, and Z is a normalization constant.

From a logical point of view, the outputs of the feature annotators can be regarded as tables in a relational database and, thus, naturally correspond to

predicates in FOL, and the object hypotheses represent the domain of discourse of entities we wish apply probabilistic, logical reasoning to. Furthermore, we can think of the final class label, i.e. the object category we wish to predict, as an additional predicate. As an example, consider a scene of two objects c_1 and c_2, where the PAEs have identified c_1 being a yellowish box with a "Kellogg's" brand logo on it, and c_2 being a round, blue thing. We can capture such a scene in a relational database as follows:

$$\begin{aligned}
&\text{shape}(c_1, \mathit{Box})\\
&\text{color}(c_1, \mathit{Yellow})\\
&\text{logo}(c_1, \mathit{\text{“Kellogg's”}})\\
&\text{color}(c_2, \mathit{Blue})\\
&\text{shape}(c_2, \mathit{Round})\\
&\text{category}(c_1, \mathit{Cereal})\\
&\text{category}(c_2, \mathit{Bowl}),
\end{aligned}$$

where we have manually added information about object classes in the "category" predicate. In MLNs, it is straightforward to create a model putting object attributes into relation with their class labels, since they provide a simple, declarative template language for generating probabilistic models. If we assume, for instance, that we can infer an object's category given its shape, a set of weighted formulas such as

$$\begin{aligned}
w_1 &= \log(0.66) \qquad \text{shape}(?x, \mathit{Round}) \wedge \text{category}(?x, \mathit{Bowl})\\
w_2 &= \log(0.33) \qquad \text{shape}(?x, \mathit{Box}) \wedge \text{category}(?x, \mathit{Bowl})
\end{aligned}$$

can be added to the model, which naturally represent the rules "everything is a round bowl" and "everything is a box-shaped bowl" (by default, all variables are universally quantified in MLNs). Of course, the above rules do not hold for most of the entities we encounter in the real world and, in fact, they can be considered mutually exclusive. However, according to Equation (3.1), the probability distribution defined by this MLN indicates that any world in which we

encounter a round bowl is twice as likely as a world in which we find a box-shaped bowl (assuming all other aspects being identical). Following this, we add such abstract, coarse "rules of thumb" to the MLN, modeling connections between the PAEs and the final ensemble decision. The weight parameters of the resulting MLN can be learned in a supervised learning manner.

3.5 Experimental Analysis

Since the contributions presented in the last two chapters are neither individual algorithms nor a monolithic system, but a framework, and since they cover a considerably wider scope than previous work, it is hard to quantitatively assess the performance of the proposed approach. A quantitative evaluation of task adaptability (i.e. number of correct answers to different queries) would only be representative of the strength or weakness of the individual algorithms wrapped in RoboSherlock and not of the system's capabilities presented so far. It is also difficult to compare RoboSherlock to existing perception systems used in robotics, since it builds on top of these and offers developers the possibility to wrap them as Primitive Analysis Engines.

I therefore showcase the query-answering capabilities of RoboSherlock on three tasks where a robotic agent performs different experiments: (1) picking and placing of objects for a table setting (2) a robot performing pipetting in a chemical laboratory and (3) a robot operating in a supermarket. I evaluate the proposed system by formulating perception queries for each scenario, showing the variety of formulations and their descriptiveness. I exemplify the use of knowledge-based reasoning, demonstrating the broad range of queries the system can handle. I conclude the experiments with a quantitative analysis of the proposed uncertainty handling component.

3.5.1 Query Analysis

Before analyzing the scenarios, let us start by considering a simple query in the proposed query language and inspect the process of planning a new sequence of Primitive Analysis Engines step by step:

```
(detect (an object
            (type 'Container')
            (color 'White')))
```

The query specifies a detection task, where a white object that is a container should be found. As a first step the query is parsed and the key-value-pairs are asserted into the belief state:

```
?- assertValueForKey(color, 'White'),
   assertValueForKey(type, 'Container').
```

Assuming that the default pipeline of the Aggregate Analysis Engine cannot accomplish this task entirely and could only return *color* information, the knowledge base searches for RoboSherlock components that output information related to the *type* of an object. This is accomplished through a sequence of reasoning steps. Relations between classes in the knowledge base that make these steps possible are shown in Figure 3.4 in a simplified manner [2]. The mai reasoning steps that lead to a valid perception plan are as follows:

1. the classes representing the RoboSherlock *type system* are identified that correspond to the *type* keyword of the query language:

```
?- rs_type_for_predicate(type, T).
    T= rs_components:'ClassificationAnnotation'
    T= rs_components:'DetectionAnnotation'
```

2. Once theses classes that represents the RoboSherlock *type system* are identified, annotators whose outputs are of these types are searched for (Figure 3.4a):

```
?- annotator_outputs(A,rs_components:'ClassificationAnnotation').
    A = rs_components:'KnnClassifier',
    A = rs_components:'RandomForrestClassifier'.
```

3. After finding the annotators, the one that can produce the desired value is identified. This is accomplished through checking whether the output restrictions of the annotator contain the queried value or a parent class of it:

[2] although reasoning happens on the level of individuals, for simplicity the figure only shows classes

```
?- compute_annotator_output_type_domain(
      rs_components:'KnnClassifier',
      rs_components:'ClassificationAnnotation',
      DList).
    DList = ['WhiteBowl','NesquikCereal']
?- member(D, Dlist), owl_subclass_of(D, 'Container').
    D = 'WhiteBowl'
```

In the example ontology from Figure 3.4b the `KnnClassifier` can recognize a *WhiteBowl*, which is a subclass of the *Container* class, hence it meets the requirements.

4. After finding a suitable annotator to detect the requested *type* of object, the AAE needs to be planned. For this, the required inputs of the found annotator and possible restrictions of it are requested (3.4c):

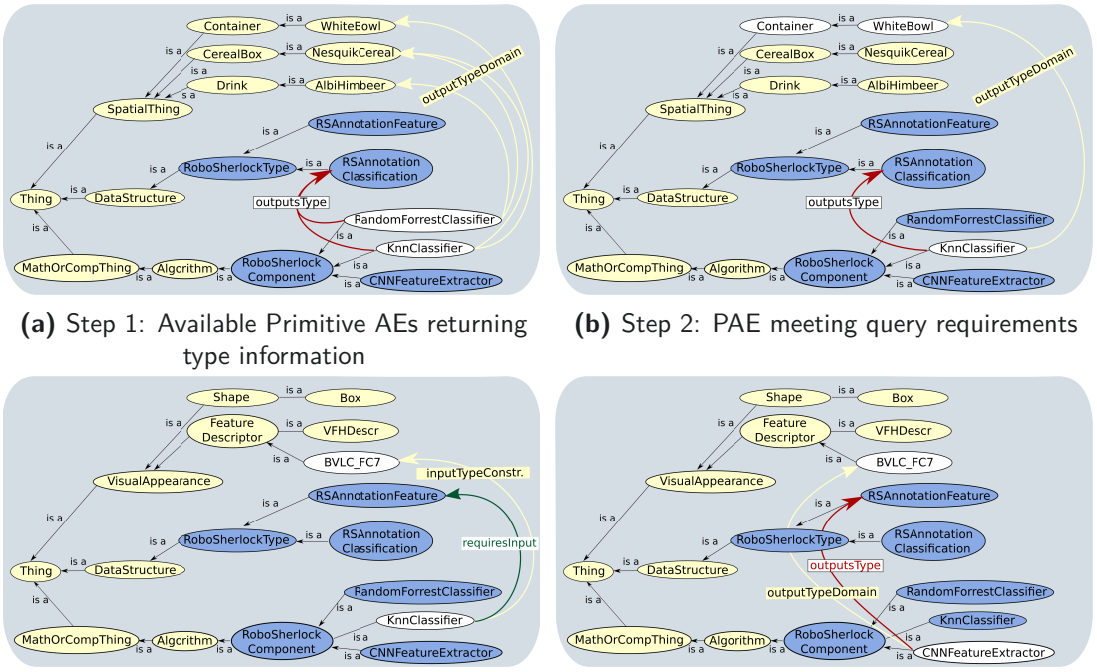
```
?- compute_annotator_inputs(rs_omponents:'KnnClassifier',T),
   compute_annotator_input_type_restrictions
      (rs_components:'KnnClassifier', T, R).
    T = 'rs_components:'RSObjectHypothesis'
    R = [],
    T = 'rs_components:'RSAnnotationFeature'
    R = ['BVLC_FC7']
```

The `KnnClassifier` has two preconditions requirements: object hypotheses need to exist that have a specific feature descriptor already extracted (BVLC_FC7 Krizhevsky et al. (2012)). This latter constraint is a direct result of the parameterization of the `KnnClassifier` in its descriptor file.

5. We now have identified the immediate requirements of the `KnnClassifier` and need to find PAEs that return `RSAnnotationFeature` annotation and `RSObjectHypothesis` (Figure 3.4d). This is the exact same reasoning step that was performed at the beginning but this time with different values, so we apply the reasoning rules from the first step. This process continues, until all annotators in the planned AAE have their input conditions satisfied by the PAEs in the pipeline.

After the input conditions of all primitive AEs are met the PAEs are ordered based on their pre- and post-conditions resulting in the following AAE:

```
?- pipeline_from_predicates([type, shape,], AAE).
AAE = [
```



(a) Step 1: Available Primitive AEs returning type information

(b) Step 2: PAE meeting query requirements

(c) Step 3: Input requirements of the found PAE (KnnClassifier)

(d) PAE producing the input required by KNnnClassifier

Figure 3.4: Visual representation of the relations about entities in the ontology when picking the correct classifier that can return a container. Results of the reasoning step are highlighted with a white background.

```
rs_components:'PlaneAnnotator',
rs_components:'PointCloudClusterExtractor',
rs_components:'NormalEstimator',
rs_components:'PrimitiveShapeAnnotator',
rs_components:'CNNFeatureExtractor',
rs_components:'KNNClassifier',
rs_comp:''
].
```

Once this pipeline is executed the result interpreter will find the object hypotheses that correspond to the description from the query, generate an answer and assert them in the belief state.

This example highlights how the query is interpreted and reasoned about in order to plan a perception pipeline based on object descriptions. This is just one way a perception plan can be generated. We can also model objects in the knowledge base, which allows us to query the system about the known properties

Figure 3.5: Results from of the same scene with different pipelines planned. Left: detecting handles. Right: searching for a pancake.

of an object:

```
?-predicates_for_object('PancakeMaker',P)
    P = [color,shape,'LinemodModel'].
```

The outputs of this query can be used to plan a perception pipeline that detects a pancake maker. When modeling an object in KnowRob, besides the visual characteristics, we can also state that we have an object model of it required by a specific expert (e.g. a CAD model or a model for Line-Mod, as in the previous query). Since there is no unique representation that can be handled by all the model fitting algorithms (e.g. BLORT uses a CAD model and the extracted key-points but Line-Mod and Moped use their own format for storing learned objects), separate classes in the ontology for each model fitting component are defined. Once an object has been recognized and is asserted into the belief state it is possible to infer other perception experts that further examine it. An example of this would be to check whether there is a pancake on the previously detected pancake maker:

```
?- detect_if_individual_present('Pancake','PancakeMaker',P)
 yes
 P =[rs_comp:'CollectionReader']
   .
   .
    rs_comp:'PancakeAnnotator']
```

In the same scene we can also query for the handles of the drawers. The generated pipeline, instead of hypothesizing about objects lying on a supporting plane, will apply a special expert designed for detecting the furniture parts.

Results of these queries are shown in Figure 3.5. From an algorithmic perspective these queries are handled by custom, task-specific PAEs that are only executed once task-conditions are met. Figure 3.7 shows more results of queries where previously asserted objects in the knowledge base are examined. Using background knowledge about these objects, the system is able to generate further hypotheses representing newly found objects or parts of objects. An example of this is having a spoon in a cup that is on a plate. Initially the system only found one hypothesis, and by labeling it as a cup it triggered specialized experts for hypothesizing about objects in its surrounding [3].

An additional advantage of modeling the objects in a common knowledge base is that the background knowledge available can be used to increase the domain of answerable queries. We can formulate queries where pipelines are generated only for objects that possess a certain property. Example of such a case would be to detect all objects that are defined as having a lid:

```
?-owl_subclass_of(Obj,'HumanScaleObject'),
  class_properties(Obj,'hasVisualProperty','ObjectPartLid'),
  build_pipeline_for_object(Obj,P).
```

In the above query the property 'ObjectPartLid' can be replaced by any kind of object property that is defined in the knowledge base. Another interesting type of query that results out of having background knowledge is to find objects that are of a certain type, e.g. *FoodDrinkorIngredient* or a *Utensil*.

```
?- build_pipeline_for_subcl('FoodOrDrinkOrIngredient',P,C).
    P = [....] C = 'KetchupBottle';
    P = [....] C = 'MondaminPancakeMix';
    P = [....] C = 'Pancake';
```

This makes it possible to query for objects through properties that are not directly visible. The queries presented exemplify how the background knowledge about objects and perceptual capabilities can generate processing pipelines that detect these. Let us now look at the three different scenarios and the particular perception tasks that can be solved using the knowledge-based reasoning techniques, illustrating the task adaptability of the framework.

[3] e.g. by running different types over-segmentation algorithms depending on the asserted object types

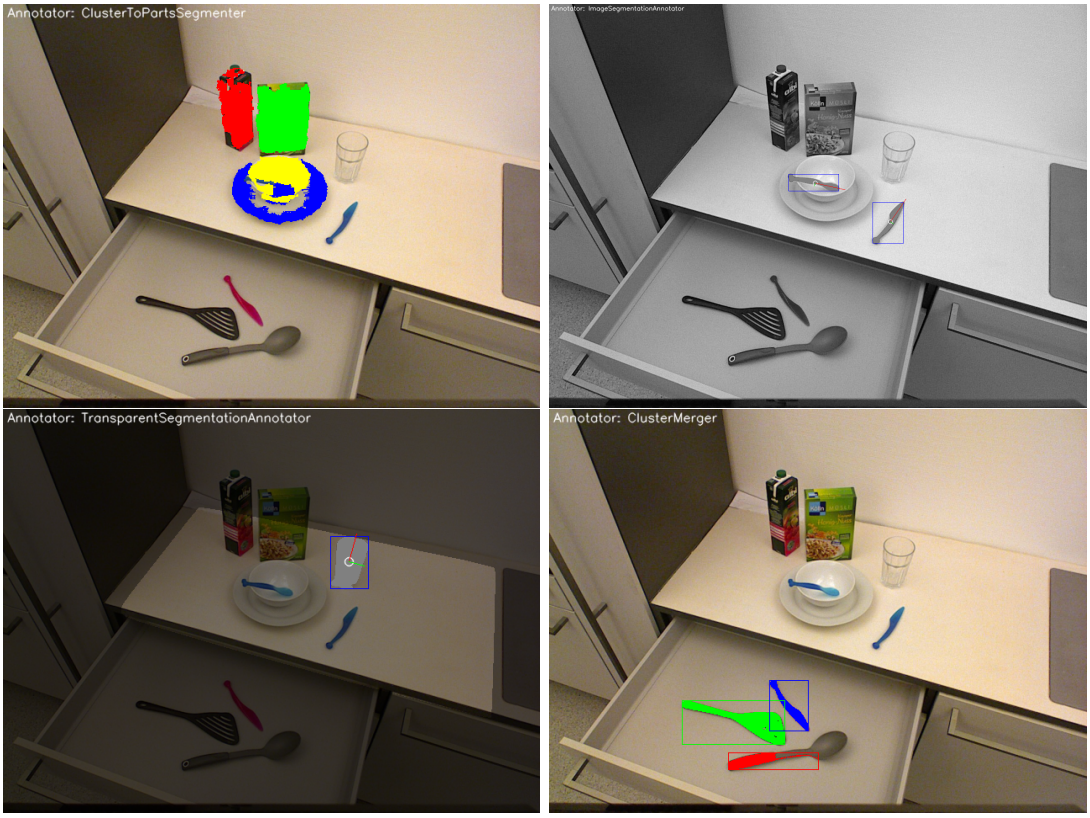


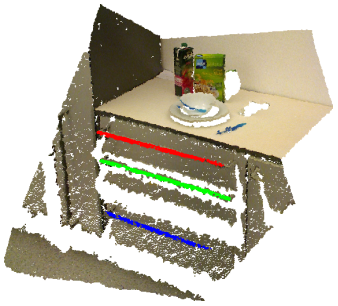
Figure 3.6: Results of various hypothesis generation algorithms, using the visualization from RoboSherlock. Top-left: 3D region growing, Top-right: binary segmentation, Bottom-left: transparent segmentation, Bottom-right: specific semantic region

3.5.2 Task 1: Setting a Table in a Kitchen

The first task I present is that of a household robot setting and cleaning a table, performing pick and place tasks. Perception tasks will be exemplified on the scene from Figure 3.6.

Objects of daily use are not the only ones of interest in this scene. If we want the robot to put away the clean objects, it needs to open drawers and look into them. The control system can direct RoboSherlock to do find the handles of a piece of furniture by issuing the query:

```
(detect ( an object (
   (type 'Handle')
   (part-of (an object
                ( type 'IAI-Drawer'))
))))
```



The system deduces that it needs to switch to a set of PAEs that were specially tailored to find the handles in our kitchen and execute those. This is done by joint reasoning about the attributes the query is asking for (*type*, *part-of*) and the values of these attributes ("Handle", "IAI-Drawer").

The default hypothesis generation only finds a subset of the objects. By querying for specific objects we can task the framework to look for the transparent glass or the knife. Similarly, by specifying which semantic location we want the robot to look at we can find the objects in the drawer. Results of the hypothesis generation are shown in Figure 3.6. As it can be observed, transparent or very flat objects would not be found with the default table top segmentation, but having additional experts that analyze the observations all objects in the scene are identifiable.

Before grasping any of the objects, the robot might want to take a closer look at them. There can be multiple reasons for doing so. Either we want to check the state of the object being manipulated (is the plate dirty, is there something in the bowl) or get a better pose estimate using, for example, an additional camera mounted on the wrist of the robot. These cases can be handled by issuing inspection queries for objects.

```
(inspect (an object (obj-id #uid))
         :pose,:obj-part)
```

Because the objects perceived using the detection queries are asserted into the belief state, these inspection queries are highly dependent on the background knowledge we have stored about them. If, for instance, a hypothesis was detected as being an object of container type, the system can run over-segmentation algorithms

on the hypothesis in order to examine them. Results of queries where previously asserted objects in the belief state are inspected are shown in Figures 3.7a to 3.7f. In these examples, when inspecting the plate and cup the system finds additional objects through over segmenting them, since the knowledge base specifies these types of objects as ones that possibly hold other objects. When looking at the pitcher, the system checks if it is open or not by searching for a flat surface on top of the object (assuming it stands up-right) belonging to the lid, since the object is marked as a child of the container class in the ontology.

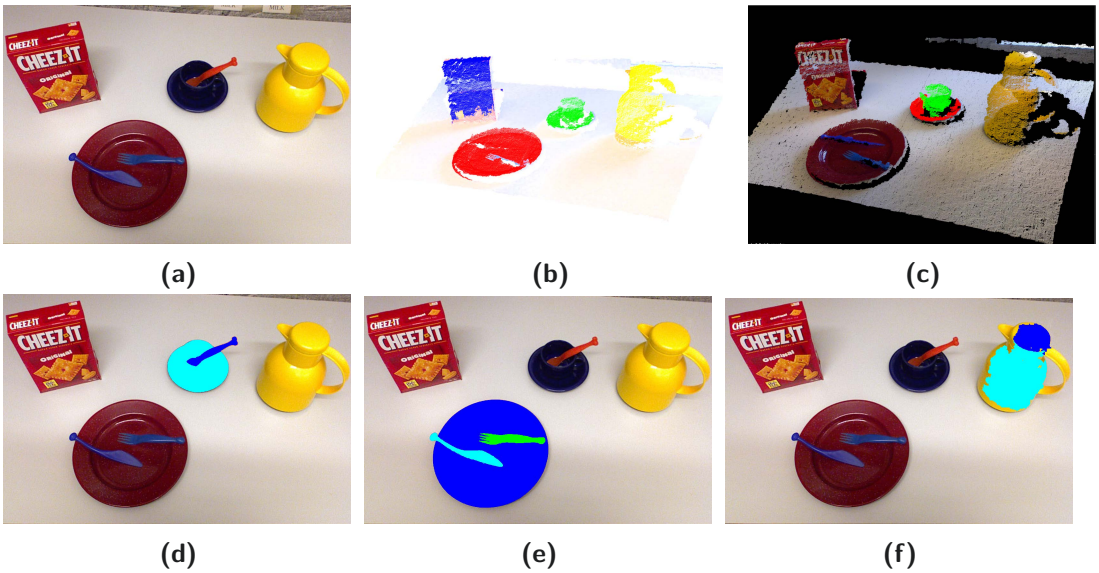
We should also be able to detect all similar objects based on some common property. For instance, all cutlery or all perishable items. This helps optimize the sequence of steps a robot has to take in order to clean the table. Using the query language several similar queries that would yield the same result are formulated as:

```
(detect ( an object(
   (shape box) (color green)))
(detect ( an object(
   (class 'KnusperHonig'))))
(detect ( an object(
   (type 'Food')))
```

These queries also illustrate how the query language can be used to express the same thing in several different ways.

3.5.3 Task 2: Chemical Experiment

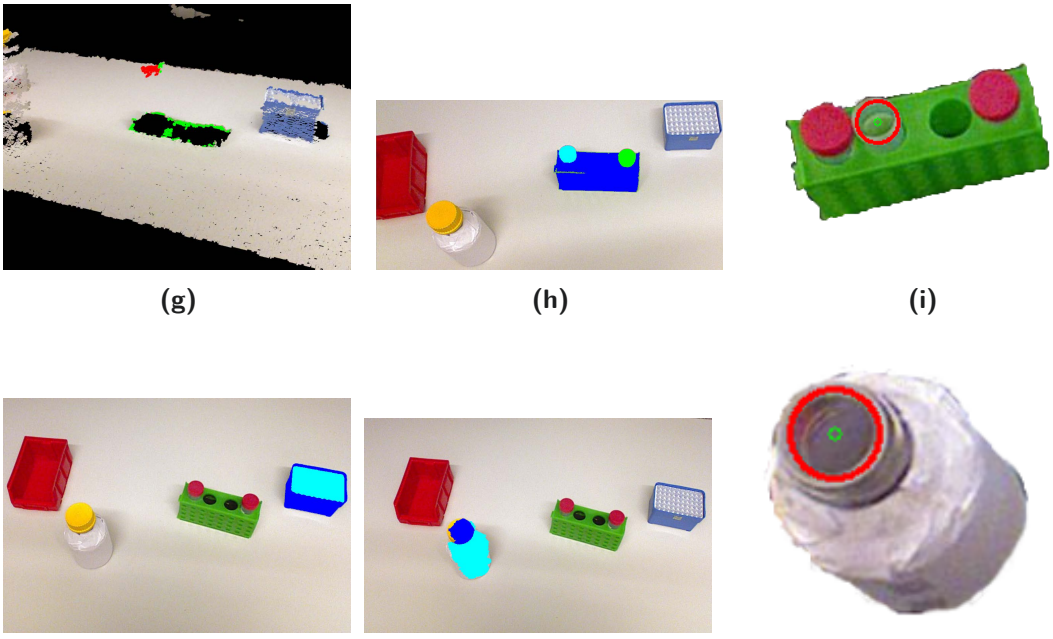
For this scenario, consider a mobile robot that has to help out human workers in a chemistry laboratory in their day-to-day jobs. Since performing some parts of chemical experiments is a time-consuming and mostly repetitive task, robot automation lends itself naturally. Solutions like Adam the chemist robot (King, 2009) obtained promising results, but it requires special deployment and setting up. One such task, part of a DNA extraction process, was demonstrated in our previous work by Lisca et al. (2015), where we showed that mobile robots are usable in these scenarios (Figure 1.2c). The robot's task is to pick up the pipette,



(a) **(b)** **(c)**

(d) **(e)** **(f)**

Inspection tasks in a breakfast scene: original image **(a)**, segmentation results **(b)**, hypotheses about: a cup on a plane **(c)**, spoon in a cup **(d)**, cutlery on plate **(e)**, the lid of the pitcher **(f)**.



(g) **(h)** **(i)**

(j) **(k)** **(l)**

Inspection tasks performed in a chemistry lab scenario(top down, left to right): raw point cloud data of the tubes rack **(g)**, hypotheses representing tube caps **(h)**, opening of a tube **(i)**, hypotheses about the opening of the pipette tips container **(j)**, lid of a bottle **(k)** and opening of a bottle **(l)**

Figure 3.7: Result of inspection tasks using background knowledge about the asserted objects in a breakfast scene and in a chemical lab scene.

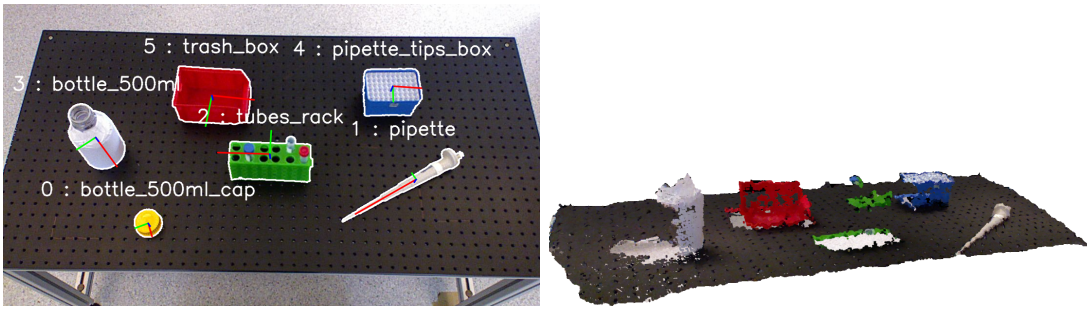


Figure 3.8: Pipetting scene as seen by the robot. *Left: RGB, Right: PointCloud*

mount a tip on it, extract some solution from the bottle and release it into one of the tubes found in the rack. Finally, the robot has to release the contaminated pipette tip into the trash box.

From the point of view of object perception, the challenges of chem-lab scenario are: (1) not all objects can be perceived using RGB-D sensors (see Fig. 3.8), hence a combination of color and point cloud segmentation is used and (2) some of the needed perception tasks are based on commonsense knowledge: the opening of a container is the top part of an object, tubes can be found in a tube-rack, may be open or closed, etc. As an example, test tubes are small transparent objects that are usually found in tube racks and can be open or can have a small lid on top, can be empty or can hold some amount of substance in them. To add a small amount of substance to this test tube all of these need to be detected first. Although in this task a closed world assumption holds, i.e. all objects are known beforehand, the objects used and the variety of relationships between these makes the application highly challenging (transparent objects, objects with (re)movable parts, with buttons, etc.). It serves as a great example of how the high-level control program can steer perception in the right direction through semantic querying.

Queries like *detect the bottle with acid in it*, or *detect the Erlenmeyer flask on the heater that can hold 500ml of liquid* are possible due to the representation of objects in the knowledge base and the flexible re-planning capabilities of the system.

There is no unique solution to perceive all of the objects and their parts, but having several expert algorithms combined with knowledge processing significantly

increases the success rate of finding the relevant items on the table. We need to precisely identify where the pipette tip needs to enter the bottle, or find the tubes in the rack. Accomplishing this task is possible through formulating rules like:

```
fitCircle(Obj,Radius) :-
  category(Obj,'container'), object-part(Obj,Opening),
  geom-primitive(Obj,'circular'),
  radius(Opening,Radius),
```

which deduces the radius of the circle to be found in order to identify the opening on top of a container, or the holes in the rack where tubes can be found.

The reasoning mechanisms can enable the detection of objects where there is little to no, or very noisy sensory input. For instance, the floating small segment (highlighted in red in the first picture of the third row from Figure 3.7) signals the system to search for hypotheses in image space, hence we find the tube rack. Asserting this, informs the system that it should search for tubes in it, leading to a final detection of the tubes, by applying a specialized expert. Since tubes can be open or closed the system uses two task-specific annotators: one for finding tubes that are closed (looking for the colored caps) and one for searching for open tubes, using the previously showcased logical rule and a more general geometric primitive fitting to find the circular opening. Several other examples similar to this are shown in Figures 3.7g to 3.7l.

3.5.4 Task 3: Object Counting in a Supermarket

One of the most promising fields to profit from robotic automation is that of logistics (EU-MAR, 2016). Competitions like the Amazon picking challenge (Eppner et al., 2017) are meant to push the state of the art in this field the same way the DARPA Grand Challenge (Thrun et al., 2006) did for autonomous cars. Mobile robots operating in supermarkets and performing inventory checking are receiving increasing attention from both industry and research. Compared to the kitchen and the chem-lab scenario, the world is much more structured, allowing for simplifications of perception tasks through the encoding of these known structuring elements. It is, thus, ideal to investigate how generic knowledge about the environment and object arrangement aids the process of perception.

The last task I present is that of building and maintaining a semantic object map in a retail environment. Figure 3.9 shows an example scene from a store

replica. The perception tasks that are needed in order to successfully manage a semantic map vary greatly. This scenario also highlights the need for adaptability to a task during run time, especially the need for changing between continuous and on-demand modalities of the framework.

A store consists of several modular elements that can often change positions. Shelf systems are considered to be static objects, but shelf floors, barcodes on the shelf facings, product separators and obviously the products themselves can change location often. The robot is tasked with scanning the shop every night and rebuilding a semantic object store. The perception system of such a robot is responsible for detecting all of the movable parts of the shop. It is evident that the detection of these parts can only be addressed with very different algorithmic solutions and that if we want a truly autonomous system, adaptability to a task is a key concept. Detecting shelf floors, for instance,is handled by fitting lines using RANSAC or Hough transform, if the necessary conditions are met (camera and robot position, viewing angles, etc.). On the other hand a volumetric counting algorithm using background knowledge about the objects is used to estimate the number of objects of a certain type in a facing. Running any of these algorithms in the wrong context would yield erroneous or no results at all.

Some of the perception tasks in this scenario are what I referred to in Sec-

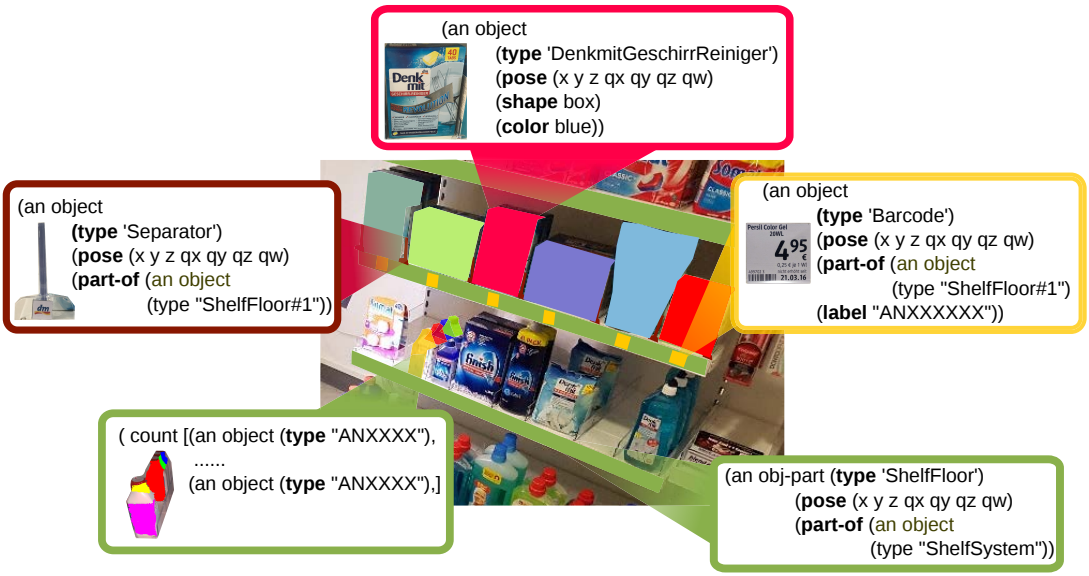


Figure 3.9: Semantically rich description of a scene from the retail store

tion 2.3.2 as continuous tasks, and need a stream of observations in order for the robot to correctly hypothesize about object locations. The compound querying capabilities of RoboSherlock are used to state such tasks. For instance, scanning for the shelf floors is formulated as:

```
(scan (for object
        (detect ( an object (
            (type 'Shelf')
)))))
```

This command will have as direct effect the exchange of the continuous component with a perception pipeline that contains the expert for shelf scanning. Using the encapsulated detection query we can specify what kind of object we want the system to scan for. A similar compound query is used to count objects of a certain type:

```
(count (object
    (detect ( an object (
        (type 'ANXXXX')
        (pose (x y z qx qy qz qw))
        (width 0.05)
)))))
```

The retail scenario is highly knowledge intensive. Stores usually have large databases where information about objects useful to the perception system is kept. This knowledge can help in greatly simplifying the requirements of the perception system. Information such as a product images can help verify if the objects we are perceiving are placed at the correct location or information about the size of an object can help filter the raw data as well as facilitate volumetric counting.

3.5.5 Result Merging

Although perception plans are adapted to the perception task the robot is supposed to execute, the individual components that form the plan can often have erroneous results. The main goal of integrating an MLNs based reasoning system into RoboSherlock as a CAS Consumer is for handling uncertainty after the execution of the AAE, which can manifest itself through contradictory annotations of object hypotheses. To demonstrate how this works, we conducted experiments that I will present in the following, with the purpose of showing that:

1. the proposed method is robust towards inconsistent annotations, which can be treated in a meaningful way,

2. the system's capabilities go far beyond traditional classifier systems, which are mainly given by discriminant functions with dedicated in- and output variables.

(adapted from previous work by courtesy of Daniel Nyga)
We arranged and recorded 50 realistic scenes, each comprising of 5-10 instances of 21 different object categories, which can generally be found in typical kitchen scenarios. We discern four different kinds of scenarios: a breakfast table, a cooking scenario, a view into a refrigerator and a view into a kitchen drawer. The types of scenarios have been incorporated into each data set and can be regarded as task-specific knowledge about the current context of an activity. This is a reasonable presumption, since the location the robot is currently looking at can be assumed to be known from e.g. a map of the environment, and co-occurrences of objects are highly correlated in real-world scenarios. The object categories for each object have been labeled manually.

For training the MLN and obtaining a final ensemble decision of experts, we used the logical predicates described in Table 3.2, which naturally correspond to the annotator outputs in the system. Two additional predicates are used for specifying knowledge about the current context (i.e. the type of scenario) the perceptual task is performed in and for assigning a class label to each of the clusters in the scene at hand:

Primitive AE	**Condition**	**MLN Predicate**
ColorAnnotator	Always	color(cluster, color)
Cluster3DGeometry	Always	size(cluster, size)
Goggles	If Google Goggles returns text or logos	logo(cluster, logo) text(cluster, text) texture(cluster, t)
PrimShapeAnnotator	Always	shape(cluster, shape)
LineMod	Confidence that c is one of the objects looked for exceeds threshold	linemod(cluster, category)

Table 3.2: PAEs used for training and the conditions under which they work and the predicate declarations in the MLN.

- scene(*scene*): represents knowledge about the current context in which the perceptual task is being performed. possible contexts are dom(*scene*) = {*breakfast*, *cooking*, *drawer*, *fridge*}

- category(*hypothesis*, *object*!): assigns a class label to each hypothesis in the scene. In our experiments, we distinguished 21 different object categories.

In the MLN syntax, the "!" operator in a predicate declaration specifies that this predicate is to be treated as a functional constraint for the respective domain, meaning that each hypothesis requires exactly one object category association. Since a particular hypothesis or entity cannot be of two different categories at a time, we argue that this model constraint is a reasonable assumption.

The following MLN has been designed in order to model correlations between annotator outputs and the object classes:

$$
\begin{aligned}
w_1 \quad & \text{size}(?c, +?sz) \wedge \text{shape}(?c, +?sp) \\
& \qquad \wedge \text{color}(?c, +?cl) \wedge \text{category}(?c, +?obj) \\
w_2 \quad & \text{linemod}(?c, +?ld) \wedge \text{category}(?c, +?obj) \\
w_3 \quad & \text{logo}(?c, +?logo) \wedge \text{category}(?c, +?obj) \\
w_4 \quad & \text{text}(?c, +?text) \wedge \text{category}(?c, +?obj) \\
w_5 \quad & \text{scene}(+?s) \wedge \text{category}(?c, +?obj) \\
w_6 \quad & \text{category}(?c_1, +?t_1) \wedge \textit{category}(?c_2, +?t_2) \wedge ?c_1 \neq ?c_2,
\end{aligned}
$$

Ground Atom	Cereal	Chips	Cup	Pot
color(c,yellow)	**0.4264**	**0.3484**	**0.4422**	0.2936
text(c,VITALIS_A)	**0.6230**	0.0000	0.0000	0.0004
logo(c,Kellogg's)	**0.3734**	0.0000	0.0000	0.0008
linemod(c,Popcorn)	**0.7392**	0.0006	0.0000	0.0010
linemod(c,Pot)	0.0008	0.0004	0.0004	**0.9994**
linemod(c,PringlesSalt)	0.0002	**0.4986**	0.0010	0.0006
shape(c,box)	**0.4806**	0.3870	0.2810	0.3556
shape(c,cylinder)	0.3722	**0.4540**	0.4010	0.4266
shape(c,round)	0.3176	**0.4092**	**0.5182**	**0.4068**
size(c,big)	**0.3680**	**0.3442**	**0.3768**	**0.3292**

Figure 3.10: (Partial) probabilities for different queries about visual features conditioned on the object class. Excerpt from previous work Nyga et al. (2014)

where the "+" operator specifies that the respective formula will be expanded to one individual formula for every value in the respective domain.

To determine the weights, we used pseudo-log-likelihood learning with a Gaussian zero-mean prior of $\sigma = 10$, which serves regularization purposes.

The system was first evaluated as a classifier, where it achieved F_1-scores significantly above 70% for all objects except for the cutlery in 10-fold-cross-validation. For a detailed description of the classification results readers are kindly referred to the original publication (Nyga et al., 2014). We will now turn our attention to some of the advantages of having a joint probability distribution over objects and their annotations.

Inferring the most probable categories given the observed properties of each object is only one possible kind of query the system can answer. Indeed, the learned joint probability distribution on objects and their attributes allows reasoning about *arbitrary* queries with respect to any variable that is contained in the model. This approach can also be used to reason about the perceptual features to be expected when looking for a particular object in a scene. If the robot is supposed to find a box of cereals on a breakfast table, for instance, the following query can be formulated in order to retrieve the most informative features for distinguishing the cereal box from other objects:

$$P\left(\begin{array}{l}\text{shape}(c,?sh),\text{color}(c,?c),\\ \text{size}(c,?s),\text{logo}(c,l?),\text{text}(c,?t)\end{array}\;\middle|\;\begin{array}{l}\text{scene}(\textit{breakfast}),\\ \text{category}(c,\textit{Cereal})\end{array}\right).$$

Figure 3.10 shows an excerpt of the probability distribution computed for the above query, where the most probable solution is printed bold. According to the most probable solution, we can deduce symbolic descriptions of expected visual properties of different kinds of objects: e.g. the cereals are expected to be a big, yellow and red box, on which we can read the text "VITALIS_A", and the Line-Mod annotator would consider it popcorn. This latter example is a good example of how the MLN compensates for the errors of individual experts: it has learned that the Line-Mod algorithm systematically generates false positive results for this object. The same happens with systematic errors of other annotators as well. Based on the deduced symbolic values, we can use the trained MLN during the perception plan generation step as an additional step for filtering the results. This leads to being able to answer queries only about attributes and objects used during the training phase. A possible solution is to retrain the network for new attributes and objects. While the approach is very powerful, it does need a considerable amount of data which is hard to acquire since all expert algorithms need to be run on the same raw data and manual labeling is necessary. Further experiments using MLNs for merging results will be shown in Chapter 6, also describing a proof of concept approach for automatically acquiring training data.

3.6 Related Work

Many knowledge-based vision systems have been proposed that are mainly focused on interpreting the raw sensor data with knowledge at different levels of abstraction. Formal representations of image semantics and low-level image processing features have started emerging in the early days of content-based image retrieval systems, when researchers started focusing on addressing the well-known semantic gap problem (Smeulders et al., 2000) in image annotations. Content-based image retrieval systems address a problem that is very similar to the computational problem this thesis addresses, i.e. given a description of something find all images that satisfy this. In order to relate the description to the extractable information from images, formal representations of the semantics are needed. Bannour and Hudelot (2011) present an excellent roadmap for using ontologies in

image retrieval and annotation, presenting a survey of recent works towards using semantic hierarchies and ontologies for the purpose of image annotations. The most common use-case is that of using some hierarchical representation which allows easy retrieval of similar concepts. This would be the case for the well known ImageNet (Deng et al., 2009) data set, built on top of WordNet (Fellbaum, 1998). Bannour and Hudelot (2011) identify four important fields where ontologies are most commonly used as representations: low-level features, visual description ontology, knowledge description and semantic mapping. They conclude with highlighting the importance of reasoning and knowledge in the process of image understanding.

In a work preceding this analysis Hudelot (2005) created an ontology with signal features used in image processing and included knowledge about processing algorithms that produces this data. However, the work lacks a unified knowledge representation, different sources of information being stored and processed in separate knowledge bases. This makes reasoning over the whole of the represented knowledge tedious. A lot of work has also been done on developing different visual primitives (Town, 2006; Hudelot, 2005), which are domain-independent descriptions of visual data.

In robotics one of the earlier perception frameworks that aimed at runtime reconfigurability is RECIPE (Arbuckle and Beetz, 1998). RECIPE was built on several of the criteria that RoboSherlock also embraces: extensibility, resource management, reuse of image processing tools, standard library support and runtime reconfigurability, to name a few. Although it proposes no central language to interact with it, or a common way of representing knowledge, the RECIPE framework recognizes the task dependent nature of robotic perception and offers an early prototype for addressing this.

COP-MAN (Beetz et al., 2009) is another interesting approach for bringing knowledge processing and robotic perceptual capabilities closer together. In some aspects, i.e. integration of knowledge processing, COP-MAN can be considered a predecessor to RoboSherlock. While it was specifically designed to perceive the environment and objects in it during typical kitchen pick-and-place task, RoboSherlock is easily usable in different domains. There has also been a lot of work towards knowledge representation in robotics (Tenorth and Beetz,

2013) and combination of knowledge processing and perception (Pangercic et al., 2010). All of these previous works address only parts of the problem. Perhaps the most relevant statement from recent literature that addresses knowledge-based perception is formulated by Fiorini and Abel (2010). In their review about knowledge-based computer vision the authors argue that the greatest challenge lies in the appropriate representation of visual knowledge needed for perception tasks, emphasizing the need for standardization. Although I do not claim that the representations presented in this work should be considered as a standard I offer a unique solution for having perceptual capabilities and visual knowledge tightly coupled together serving as a basis for standardization efforts. Hager (2014) identifies two main future challenges of sensing for robots. The design of modular sensing, organized around ontologies and information structure rather than the data itself and the realization of context-relevant question-answering systems through "information API", challenges that are embraced and addressed by the task adaptable perception realized with the help of the RoboSherlock framework.

3.7 Summary and Discussions

In this chapter I have presented how knowledge processing is integrated into the RoboSherlock framework, how it is used to generate context-specific perception plans that invoke specialized perception experts as plan steps and how it allows for richer description of perception tasks. The benefits of using background knowledge were highlighted through several examples, where having additional information encoded about an object or a task can greatly simplify detecting it.

Even though parts of the knowledge base can automatically be generated (e.g. RoboSherlock implementation specific knowledge), most of it is still based on manual entries. This happens either through directly editing the OWL files that store the knowledge or as meta information stored in the descriptor files of RoboSherlock annotators that get later converted into OWL files. This requires users of the framework to be familiar with the inner workings of the pipeline planning and query answering process as well as to adopt a different mindset when developing the perception components for a new task, environment or set of

objects. By giving knowledge processing and knowledge-based reasoning a central role in the process of perception, the framework forces developers to formally describe the expert knowledge that they possess. It also requires developers to implement annotators whose input and output constraints can be expressed using the existing *type system* or if this is not possible to come up with extensions to the *type system* that allow this. I believe that it is only through this process of formally describing what is being implemented that we can achieve a system that truly understands its capabilities, and can automatically adapt to new situations.

The formal description of perceptual capabilities also opens the door for further automated testing and learning. Given several similar perception algorithms, automated tests can be created to model the performance of these and results used to choose the best possible algorithms when trying to perform a task. As a simple example, consider choosing between image features for detecting a specific object. A feature descriptor that works well for one objects might not perform as good for a different one. The formal description of these feature descriptors, the objects and the environment allow for easy deduction of processing pipelines for testing. Automating these tests is ever more possible thanks to the recent advances in game engine technology and realistic real-time off-screen rendering technology.

Now that the basic inner mechanics of perception pipelines and query answering in RoboSherlock were introduced, recall the three categories of questions that I have identified as important for robots performing everyday manipulation tasks to answer (Table 1.1). The query language introduced so far and the interpretation of these queries and generation of answers using knowledge-based reasoning enables a robot to answer the questions about the present, i.e. what is currently visible, and in a limited way about the past, through the inspection queries. In the remainder of the thesis I will present parts of the proposed system that realize a pervasively operating question-answering system possible, enabling answering questions about past and future states of the world.

CHAPTER 4

Generating and Using Perceptual Episodic Memories

With recent technological advances, robotic agents are increasingly capable of performing ever more sophisticated manipulation tasks. These robots need to be able to adapt their perception systems to the wide variety of tasks they are required to perform. Remembering what a robot has seen, what the rationale was behind the decisions it took or how it ended up understanding the world as it did are important questions if perception systems are to scale towards real-world manipulation tasks. Most of the perception problems a robot encounters during its operation have several solutions, but the choice of which approach to use when is usually based on expert knowledge explicitly encoded by a developer. Furthermore, especially during the development phase, erroneous results often occur without noticing, either due to the lack of supervision or simply because results do not get propagated to the robot control program and are not noticed during runtime. Achieving a pervasively operating perception system that enables robotic agents to fluently perform everyday manipulation tasks requires robots to have access to past percepts and use these to optimize their processing components. To address this **I propose to enable robots with the capability of recording perceptual episodic memories and the ability to actively access these.**

Using RoboSherlock I achieve this through the implementation of a new CAS Consumer that generates, stores and retrieves perceptual episodic memories. To allow easy retrieval of these memories, I introduce a Domain Specific Language

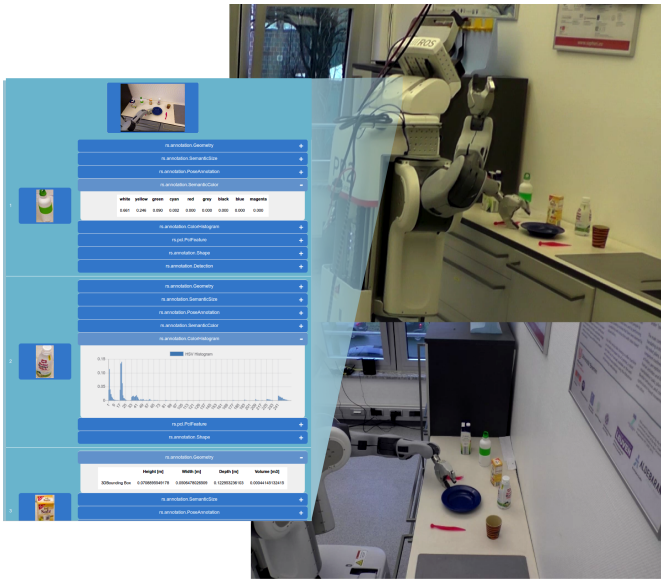


Figure 4.1: PR2 picking up a plate and visualization of a frame from the associated perceptual memory using the web-inspection tool proposed in this chapter

(DSL) for object and scene description that serves as a layer of abstraction between the structure of the perception logs and the semantic interpretation of these. I demonstrate how the description language is used programmatically to retrieve specific parts of the generated episodic memory and adapt the object recognition modules of a robotic agent. Besides this, the proposed system offers a web-interface through the use of which the memories can be queried for and inspected during or after task execution. There are multiple reasons for the proposed solution: to enable online retrospection, to enable users of the RoboSherlock framework to interactively explore results and to specialize perception routines through the use of supervised learning techniques. The chapter is mostly based on previous work covered in Bálint-Benczédi et al. (2017), extending this with further experiments demonstrating the benefit of using the perceptual episodic memories for adapting object recognition modules.

4.1 Literature Review

When studying the role of memory in robotic systems, one inadvertently comes across the topic of building artificial cognitive systems. Vernon (2014) states, that

in a cognitive architecture:

> “ ”memory should be thought of not as a storage location [... but a] pervasive facet of the complete cognitive system, fully integrated into all aspects of the cognitive architecture.” ”
>
> *David Vernon*

The capability of recording and accessing episodic memories proposes achieving this through a query interface that enables pervasive integration of memory in the perception processing pipeline.

The way human memory works has eluded scientists to this day. Nevertheless, most researchers commonly agree on a differentiation between long-term and short-term (often referred to as working) memory. The long-term memory can further be categorized as being declarative or procedural. Declarative memories can be further split into semantic (encyclopedic) and episodic memories. Episodic memories have an especially important role in robotics (Vernon et al., 2015; Winkler et al., 2014), since they enable robots to ”relive” past events and learn from them. Perceptual memory is a long-term memory for visual, auditory and other perceptual information. In this chapter we will look at the storage and retrieval of perceptual episodic memories, that is, of visual data generated throughout the execution of a manipulation task.

Blodow (2014) presented the first concepts of memory storage in RoboSherlock. In his work, he identifies the Common Analysis Structure (CAS) as the short-term memory of a running process, and a storage component, called the “robot object store” as the long-term memory. In a preliminary analysis he presented what the system could do using the logged memories, identifying three major use-cases for them:

- for later evaluation of results,
- knowledge source for later processing *and*,
- a developer tool for inspecting results.

His implementation offered a proof-of-concept for the memory system but it lacked the necessary query-answering capabilities for easy and efficient retrieval and

the challenges and possible benefits of long-term logging were not addressed and evaluated. In addition to the three use-cases I will also focus on optimizing the storage of long-term memories through the use of background knowledge.

With recent advances in mobile robotics that allow for research of integrated robotic systems, the storage of perceptual memories (and not just) has been the focus of wide-ranging research. Surprisingly though, very little effort has been invested in the retrieval of these and the creation of a Domain Specific Language for describing perception tasks or objects of daily use. Nordmann et al. (2014) provide a comprehensive survey of DSLs in the field of robotics. Browsing through the on-line collection[1], it is striking how perception and sensing are under-represented. In one of the few works addressing this, Hochgeschwender et al. (2014) introduce RPSL (Robot Perception Specification Language) for specifying the integral parts of perception architectures using explicit models. The focus is mainly on modeling the implicit design decisions which make most of the current perception architectures inflexible. This is achieved through the introduction of meta-models and the language to describe these. A perception graph meta-model enables the composition of elements and the custom generation of perception pipelines. While the method resembles the task adaptability proposed in this thesis, offering a high-level DSL for specifying perception tasks, answering queries about past percepts is not addressed.

Storing and learning from perceptual memories is most prominently addressed by Niemueller et al. (2013a,b). Lifelong learning of training data and perception method parametrization is the main focus of their work, with emphasis on the storage of perceptual data and (re-)detection of objects based on this. In another interesting work Oliveira et al. (2014) is concerned with grounding the object category symbols into the perception of known instances of categories. They do this by using two separate memory locations, one for semantic memory and one for perceptual memory. While both of the previously mentioned approaches resort to storing perceptual memories, semantically querying these is not addressed.

[1] `http://corlab.github.io/dslzoo/`

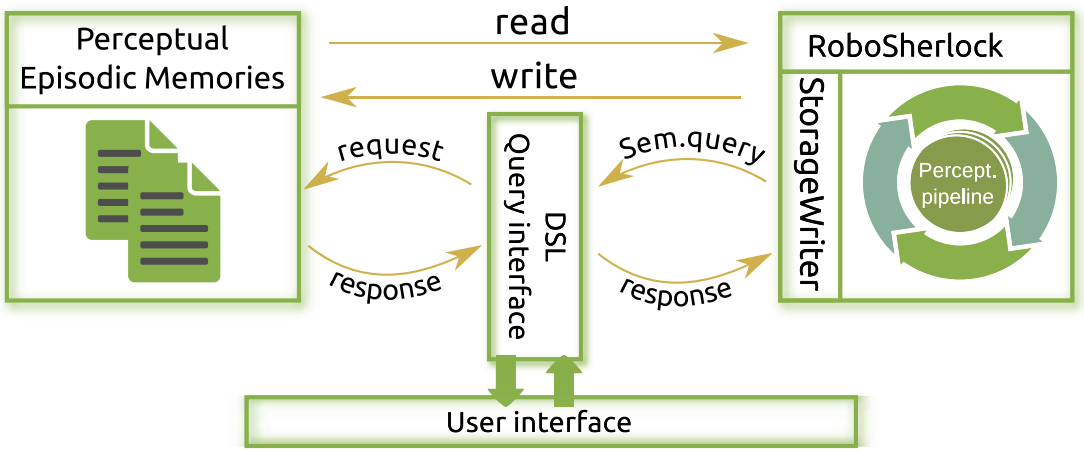


Figure 4.2: Components of the proposed sub-system.

4.2 System Architecture

The generation of perceptual episodic memories is implemented as a CAS Consumer in RoboSherlock. As we have previously seen, built upon the principles of Unstructured Information Management (UIM) (Ferrucci and Lally, 2004), RoboSherlock enables the realization of scalable perception systems for robotic agents. Among other properties, it allows for implementing perception systems that (i) can be equipped with ensembles of expert perception algorithms, (ii) can be tasked, and (iii) can enhance perception with knowledge and reasoning.

The defined RoboSherlock *type system* serves as a blueprint for the internal data structures and allows communication between components of the framework through the CAS. The CAS is the central data structure in RoboSherlock, acting as a whiteboard architecture (Boitet and Seligman, 1994), allowing components to post and retrieve data solely based on the defined type. At any given time, components need to know only about the data types they are supposed to receive and send to the CAS. Raw data and other modalities of this (gradient- or normal-images) are referred to as *views*, or Subjects of Analysis (SofAs). The name *view* is preferred because of its more descriptive nature, since *views* depict the same reality, but from a different point of view. One essential *view* of the data is the *scene*. A *scene* in the type system is a high-level representation of the underlying semantic structure of the raw data, made up of object hypotheses and annotations of these, believed to be true at a given timestamp. During execution

the CAS gets reset every time a new processing loop is started. This is why it can be considered analog to a short-term memory of the system. Before resetting, a storage component maps the CAS to a data base. This is achieved through a low-level, internal data structure to database document serialization, which allows bi-directional conversion. This means that the CAS can also be read out from the database and specialized Primitive Analysis Engines (PAEs) can inspect the results in it. The description language and the query interface proposed in this chapter act as a semantic bridge between the execution engine and the data structures in the database, offering a common interface to inspect results in a programmatic manner and through a web-interface. Figure 4.2 presents these major components of the proposed system and the interaction between them, with the query interface acting as a semantic interface between the execution engine and the storage place for episodic memories.

4.3 Storing Memories

In RoboSherlock we opt to use MongoDB (Chodorow, 2013) a schemaless database for storing the perceptual episodic memories. The basic unit in a MongoDB is called a *document*. Unlike in relational databases (e.g. SQL) the structure of a document is not enforced, thus, documents can have a varying number of fields and data types stored in them. Documents are stored in the *BSON* format, the name coming from the combination of the words *Binary* and *JSON* (JavaScript Object Notation). A database is made up of several *collections*, where a *collection* is simply a grouping of several documents, again, without any preexisting schema. The fact that no underlying common structure is enforced fits perfectly the paradigms of UIM, since there is no guarantee that all hypotheses will be annotated in the same way, or that every execution loop generates the same *views*. The choice of MongoDB is further strengthened by the benefits of using schemaless databases in robotic applications shown by many before (Blodow, 2014; Niemueller et al., 2012; Winkler et al., 2014; Oliveira et al., 2014). Furthermore, as of recent versions (3.0+), MongoDB introduces default compression of BSON data, which makes it more efficient for storing big data.

The first version of storing memories for RoboSherlock was presented

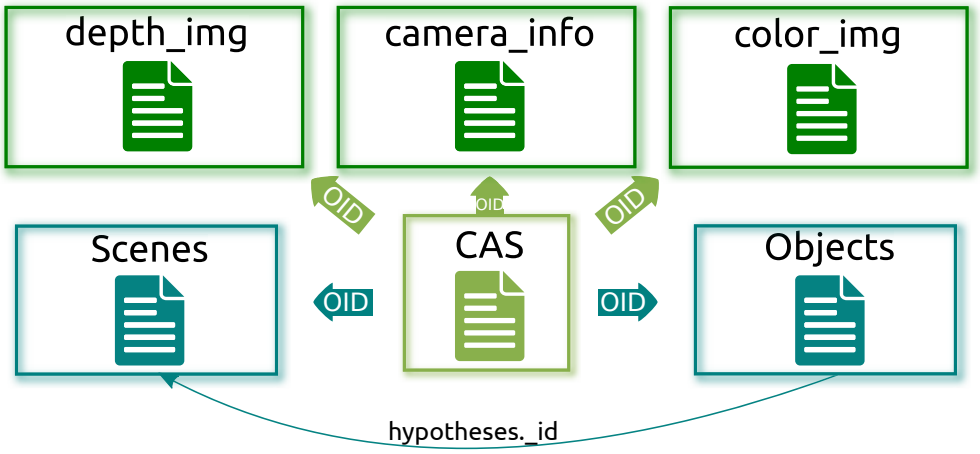


Figure 4.3: Top level structure of the storage schema in MongoDB. Relation between collections

by Blodow (2014), focusing on dynamic object stores for robots performing everyday activities. In his implementation only a part of the data resulting from processing components was stored in a single collection. In order to give the data more structure and allow bi-directional compatibility with the internal data structure of RoboSherlock, this has been changed such that each *view* is stored as a separate collection in the database and a central collection contains pointers to these views (Figure 4.3). For example, a document in this central collection, conveniently named the CAS collection, has the following structure:

```
{
  "_id" : ObjectId("5874f179be3f7a7ad0ed5d99"),
  "_timestamp" : NumberLong(1482229380595892427),
  "scene" : ObjectId("5874f179be3f7a7ad0ed5b6b"),
  "color_image_hd" : ObjectId("5874f179be3f7a7ad0ed5fa3"),
  "camera_info_hd" : ObjectId("5874f179be3f7a7ad0ed5fa4"),
  "camera_info" : ObjectId("5874f179be3f7a7ad0ed5fa7"),
  "depth_image_hd" : ObjectId("5874f179be3f7a7ad0ed5faa")
}
```

This means that, if we have a new *view* of our data, say a new sensor on the robot, integrating the data from it in the memory system is straightforward. All of this happens through automatic serializations of RoboSherlock types from the *type system* to entries in the database. The structure of the database, depicted in Figure 4.3, contains a collection called *"Objects"*. The *"Objects"* collection stores part of the belief state representing the objects that are thought to be present in the environment after entity resolution. These entries are connected to the

individual hypotheses through the IDs of the object hypotheses that are located in the *Scenes* collection. The collections are indexed using the timestamps of the incoming data and the aforementioned unique IDs are automatically generated by MongoDB. The process of solving entities and building a perceptual belief state of objects will be detailed in Chapter 5.

As I have already mentioned, the *Scenes* collection contains the data about how the system has perceived the world at a given timestamp. A stripped down version of a document from this collection has the following structure:

```
{
  "_id" : ObjectId("56058051d609ea15c7128ea5"),
  "_parent" : ObjectId("5841f694812388cb317a6e8c"),
  "_type" : "rs.core.Scene",
  "timestamp" : NumberLong(1443201105410070624),
  "viewPoint" : {
    "_id" : ObjectId("5841f694812388cb317a6e8d"),
    "_parent" : ObjectId("56058051d609ea15c7128ea5"),
    "_type" : "rs.tf.StampedTransform",
    "rotation" : [ ... ],
    "translation" : [ ...],
    "frame" : "map",
    "timestamp" : NumberLong(1443201105442234497),
    "childFrame" : "head_mount_kinect_rgb_optical_frame"
  },
  "hypotheses" : [  ],
  "annotation" :[  ]
}
```

Since this entry is automatically generated it is a mere replica of the *scenes* type from the RoboSherlock *type system*. Most notably, a document from this collection contains:

- a text field containing the type name from the *type system*; this allows easy conversion between database entries and RoboSherlock data structures (`_type`),
- a `viewPoint`, which is a timestamped transformation from the robot's camera frame to a predefined location (e.g. this is a map frame if the robot is localized),
- MongoDB ObjectID of the parent document in the CAS collection (`_id`),
- a list of scene specific `annotations` (annotations that are common to all hypotheses, such as a supporting plane)

- a list of the `hypotheses` that were found in the current views.

The list of hypotheses contains regions of the raw data and their respective annotations that are the result of hypothesis generation and merging. The symbols used are part of the RoboSherlock *TBox* and as such are represented in the central knowledge base (as described in Chapter 3). A common entry for every sub-part of a document (e.g. list elements, arrays, etc.) is the "_type" field, which contains the name of the type from the RoboSherlock *type system*. A hypothesis entry from the database has the following structure:

```
"hypotheses" : [
  {
  "_id" : ObjectId("5841f68a812388cb317a6a2f"),
  "_parent" : ObjectId("56058051d609ea15c7128ea5"),
  "_type" : "rs.scene.Cluster",
  "annotations" : [
    {
      "_id" : ObjectId("5841f68c812388cb317a6a47"),
      "_parent" : ObjectId("5841f68a812388cb317a6a2f"),
      "_type" : "rs.annotation.Detection",
      "source" : "MLN",
      "name" : "spatula",
      "confidence" : 1.0000000000000000
    },
    {...}
   ]
  }
]
```

To optimize storage space, instead of storing a complete object hypothesis, the meta data about it is stored, i.e. instead of storing the image or the point cloud of an object hypothesis, regions of interests or vectors of indices are stored. This reduces storage space considerably, while still enabling later reconstruction of results. Another important mechanism for reducing the amount of data that needs to be stored is through the use of filters that use task and background knowledge. The presented modifications to the storage component of RoboSherlock were joint work with Thiemo Wiedemeyer.

4.3.1 Filtering Based on Background and Task-knowledge

(*by courtesy of Thiemo Wiedemeyer*)
Background and task-knowledge are valuable sources of information. With a

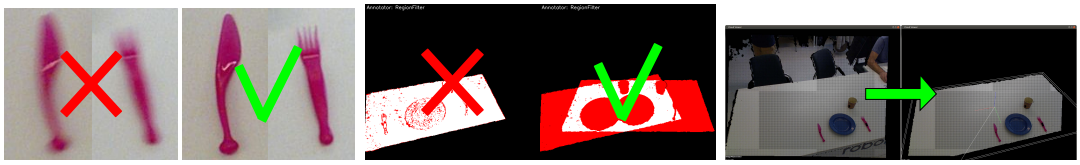


Figure 4.4: Filtered images, left to right blurred images, change detection and region filtering

semantic map of the environment (Pangercic et al., 2012) and a localized robot, it is trivial to filter out any part of the observations that are not in scope of the current task, e.g. if during a pick-and-place task only the source and destination regions are of interest. This reduces the chance of false detections and computational effort.

Besides this, other filtering techniques are available that improve performance and reduce the amount of redundant data. In many robotic tasks the scenes are mostly static, successive frames are often similar and lead to no information gain, therefore, skipping them offers more processing time for other tasks and storing them is handled by pointing to a previous observation.

Another source for false detections are motion blurred images. Depending on the task being executed, this can happen if the camera or something in view is moving fast. We can opt to not store these images, depending on how we want to use the stored data or store only the raw data but not process them. The latter can be useful if we want to have a preparatory perception that hypothesizes about regions while the robot is moving towards its goal position. Examples of the effects of the filters are shown in Figure 4.4.

4.4 Perception-log Query Language

The goal of storing the perceptual episodic memories is to enable robots to relive experiences, to learn from them, and to enable debugging and retrospection for advanced robotic systems. To do this, efficient and expressive ways of recovering memories or parts of these are needed. MongoDB offers a powerful JavaScript-based query language, but it requires in-depth knowledge of the underlying data structure. Also, through the query language offered by MongoDB we can access only the explicit knowledge stored in the perception logs. By creating

⟨*description-list*⟩ ::= ⟨*bl*⟩⟨*key-value-pair*⟩[⟨*separator*⟩]+ ⟨*el*⟩

⟨*separator*⟩ ::= ‘,’|‘;’

⟨*bl*⟩ ::= ‘[’ ⟨*el*⟩ ::= ‘]’

⟨*key-value-pair*⟩ ::= ⟨*kvp*⟩ | ⟨*comparison-kvp*⟩ | ⟨*key*⟩⟨*assignment*⟩⟨*description-list*⟩

⟨*kvp*⟩ ::= ⟨*key*⟩⟨*assignment*⟩⟨*value*⟩

⟨*comparison-kvp*⟩ ::= ⟨*key*⟩⟨*comparator*⟩⟨*value*⟩

⟨*assignment*⟩ ::= ‘:’

⟨*comparator*⟩ ::= ‘<’ | ‘>’ |’=’

⟨*key*⟩ ::= ‘shape’| ‘color’ |‘size’|‘type’| etc.

⟨*value*⟩ ::= ⟨*word*⟩ | ⟨*number*⟩

Figure 4.5: Extended Backus-Naur Form of the proposed description language for filtering the episodic memories generated by RoboSherlock

a higher level query language that acts as an abstraction layer between the low-level data structure and a symbolic knowledge base, existing knowledge and logging infrastructures can be leveraged that enhance the general query-answering capability of a robotic agent.

Conceptually the perception-log query language that I introduce, is similar to the perception task language from Section 2.3.2. The major difference is that when accessing image logs, it is necessary to express relations between low-level data structures that are specific to the implementation of the framework. In contrast, the query language for perception tasks is not meant to describe low-level data structures, and its purpose is the semantic description of the entities to be perceived.

The grammar of the proposed perception-log query language, expressed in the extended Backus-Naur Form (BNF), is shown in Figure 4.5. Using this syntax, scenes, objects or object hypotheses can be described through easy-to-write nested key-value pairs. If the assignment operator is used, values in the description are considered to be literals or a list of key-value pairs. If on the other hand one of the comparator operators is used they are considered to be numerical. The description

Rule	Description
Base predicates	
scenes(S, D)	return all scenes (S) that fit the description (D)
hypotheses(H, D)	return all hypotheses (H) that match description (D)
object(O, D)	return all objects (O) that match the description (D)
Relational predicates	
objectsInScene(O, S)	return all objects (O) that were found in the scenes (S).
hypothesesInScene(H, S)	return all hypotheses (H) that were found in scenes (S).
hypothesesForObject(H, O)	return the hypotheses (H), corresponding to the objects (O)
scenesWithHypotheses(S, H)	find all scenes (S) that contain the list of specified hypotheses (H)
scenesWithObjects(S, O)	find all scenes (S) that contain the list of specified objects (O)
Useful SWI-Prolog predicates	
intersect(L_1, L_2, L_3)	L_3 is the intersection of L_2 and L_1
union(L_1, L_2, L_3)	L_3 is the union of L_2 and L_1

Table 4.1: Predicates for querying the logs using the description language as a parameter

language can than be used, through a set of predefined Prolog predicates, to query the perceptual episodic memories. These predicates are presented in Table 4.1. They can be categorized based on the type of data we want to retrieve. We can either ask for scenes, hypotheses, objects from the belief state or a combination of these. For instance, the following query will yield all object hypotheses that have a property shape with the value *flat* and a property color with the value *black* attached:

hypotheses(Hyp, '[shape:flat, color:black]').

Timestamps are the global index of episodic memories (not just the perceptual ones), therefore it is important that scenes can be retrieved based on them:

```
scene(Scene, '[ ts < 1482401877855847376,
                ts > 1482401675823958606]').
```

Using existing predicates from Prolog as well as the relational ones from Table 4.1, the previous two rules can be concatenated as:

```
hypotheses(Hyp1, '[ shape:flat, color:black]'),
scene(Sc, '[ ts < 1482401877855847376,
             ts > 1482401675823958606]'),
hypothesesInScene(Hyp2, Sc),
intersection(Hyp1,Hyp2,Res).
```

First the hypotheses that match a given description and scenes that were created in between the two timestamps are asked for. This is followed by asking for the list of hypotheses in the resulting scenes, and an intersection gives the final result.

Annotations can also be described in more detail, specifying for instance the confidence value or the source of an annotation:

```
object(Obj, '[shape:[value:flat, confidence<0.7]
              type:['ElectricalDevice'],
              class-label:[value:PancakeMaker,confidence>0.8,
                           source:KNNClassifier],
              distance < 1.5 ]').
```

There are two interesting keywords that are worth taking a closer look at: `type` and `distance`. Both of these keywords go beyond what is stored in the perceptual episodic memory, and entail computations. In the case of `type`, knowledge-based reasoning is performed by accessing the ontology to see if any of the class labels attached to the resulting objects is a match. For `distance`, a data processing sub-module is triggered, which in this case calculates the distance of each resulting object from the robot camera.

4.5 Experimental Analysis

A quantitative analysis of logging using MongoBD in robotics and specifically logging perceptual events has been performed by many before in the literature (Niemueller et al., 2012, 2013a). MongoDB has proven itself to be a fast and reliable non-relational database when it comes to accessing speeds and it is capable of scaling to large amounts of data. Instead, the focus of this analysis will be on the use-cases of the proposed DSL: as a retrospection tool and how it enables learning and adaptation of perception skills based on episodic memories.

4.5.1 Retrospection and Debugging

To show the retrospection capabilities, the perceptual memories during a pick-and-place task, executed by a PR2 robot, were logged. The robot was tasked with setting the table, and for simplifying execution it had to bring only a plate and cutlery from one counter top to another (Figure 4.7).

```
[A]
db.getCollection('scene').aggregate(
 {$project:{_id:0,identifiables:1}},
 {$unwind:'$identifiables'},
 {$match:{$and: [{'identifiables.annotations':{$elemMatch:{shape:"round"}}},
                 {'identifiables.annotations':
                   {$elemMatch: {confidence: {$gt:0.8},
                                 source: "DeCafClassifier"}}}
]}})

[B]
db.getCollection('scene').aggregate(
 {$match:{$and: [{timestamp: {$gt:1482401800959899335}},
                 {timestamp: {$lt:1482401886643145862}}]}},
 {$project:{_id:0,identifiables:1}},
 {$unwind:'$identifiables'},
 {$match:{$and: [{'identifiables.annotations':{$elemMatch: {shape: "round"}}},
                 {'identifiables.annotations':
                     {$elemMatch: {confidence: {$gt:0.50},
                                   {source:"DeCafClassifier"}}}
]}})
```

```
[A]
hypotheses(Hyp, '[shape:round,
                  detection:[confidence>0.8, source:DeCafClassifier]]').

[B]
hypotheses(Hyp1,'[shape:round,
                  detection:[confidence>0.8,source:DeCafClassifier]]'),
scenes(Sc, '[ts<1482401886643145 86,
             ts>1482401800959899335]'),
hypothesesInScenes(Hyp2,Sc),
intersect(Hyp1,Hyp2).
```

Figure 4.6: Queries (denoted A and B) in the native MongoDB JavaScript style query language and their counterparts in the proposed perception-log query language.

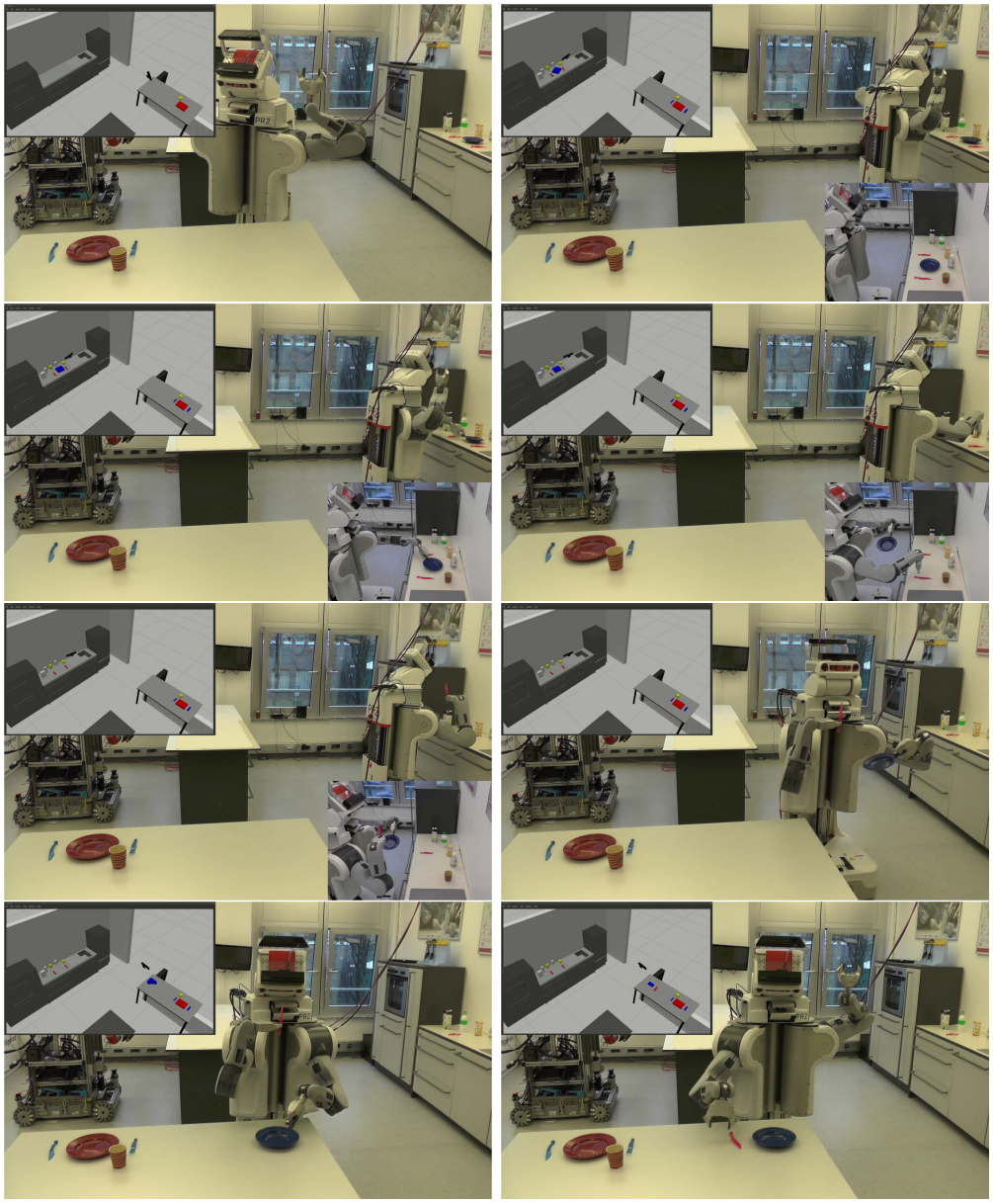


Figure 4.7: PR2 setting the table. Upper left corner of each image shows what the robot believes to be true about the environment

We have already seen in the previous section how using the query language we can ask for more than just what is stored in the memories, by attaching processes or knowledge-based reasoning queries to certain keywords. To show the usefulness of a custom query language I will focus on demonstrating through example queries how complexities of the native query interface can be hidden and a debugging,

introspection tool for advanced robotic systems is realized. Figure 4.6 shows two queries. The top part is in the MongoDB native query language while on the bottom their counterparts expressed in the query language proposed in this chapter. Beside being less verbose and easier to read, a clear benefit is that users do not need to know about the exact internal structure of the MongoDB, and can thus bypass the use of special MongoDB operators that search in arrays or perform other data manipulation tasks (e.g. aggregation, projection, etc.).

Depending on what was asked for (hypotheses, objects or scenes) the results of the queries will be multiple documents or parts of documents, containing the full description of the matching entities. The results of queries can be visualized using the web-interface shown at the beginning of the chapter (Figure 4.1), which shows all the low-level percepts that the system has calculated and offers a way for attaching ground truth labels to the data.

4.5.2 Learning From Episodic Memories

The main benefit of generating the perceptual episodic memories is that they allow retrospection and adaptation of components to the tasks that a robot executes. To demonstrate this, I present a proof of concept of how asking semantic queries can generate data for new supervised learning problems that improve the performance of the initial algorithm without human intervention.

For this experiment I take as baseline an instance-recognizer, trained on a dataset that was recorded identically to that of the popular RGB-D Object dataset (Lai et al., 2011). The newly recorded dataset consists of partial views of objects, CAD models of these objects, as well as sequences of images recorded from a robotic agent that is performing table setting or simple pick and place tasks. The details of this dataset are presented in Appendix B, as subsets of it will also be used in further experiments. A baseline classifier, a linear Support Vector Machine (SVM) (Cortes and Vapnik, 1995)), is trained to recognize instances of objects using the data recorded from the turntable. It is trained using the nowadays popular DeCAF method (Donahue et al., 2014), where the outputs from a layer of a convolutional neural network are used as feature descriptors. Specifically, the output of the 7th, fully convolutional, layer of a modified AlexNet (Krizhevsky et al., 2012) implementation is used that comes with the Caffe deep learning

framework (Jia et al., 2014) and is trained on the ImageNet dataset (Deng et al., 2009). The base classifier is trained on 21 object instances and achieves a 99.5% accuracy on ten-fold cross validation when using random two-to-one splittings of the turntable data into training and testing sets.

	Turntable	Adaptation set($E4$)	$E1$	$E2$	$E3$
nr. of obj. types	21	21	7	12	17
nr of views.	4526	699	117	302	647

Table 4.2: Details of the data used for experimentation, showing number of object instance types in the episode and the number of partial views of objects

To demonstrate how querying of memories using the proposed language can be achieved, four episodic memories were recorded during a pick-and-place scenario performed by a PR2 robot, containing subsets of the 21 objects the baseline SVM was trained on. A Primitive Analysis Engine that implements the baseline SVM, *SVMAnnotator*, was used during the robot execution for classifying the objects into one of 21 instance classes. One of the episodes that contains all 21 objects, is referred to as the *adaptation set*, whereas the rest of the episodes that contain a subsets of these objects, are referred to as $E1, E2, E3$.

The scenes in the four episodes are similar to the ones shown in Figure 4.7, where objects are clearly separable and are found on supporting surfaces. The perceptual episodic memories were recorded using the storage components presented in this chapter, using the knowledge-based filters from Section 4.3.1. For the purpose of this experiment the Aggregate Analysis Engine in RoboSherlock was set manually, such that every object hypothesis that is generated gets annotated with the results from the base SVM and the confidence in the result. Details of the dataset used are presented in Table 4.2. The hypotheses in the episodes were annotated with ground truth using the proposed web-tool, ground truth in this case being one of the 21 object instance labels. Once the ground truth was available, the base SVM was evaluated on the episode data. The confusion matrix of this is shown in Figure 4.8.

It is immediately evident from the confusion matrix that classification results are substantially worse than those that resulted from the ten-fold cross-validation on the turntable dataset. Accuracy, recall and f1-score are all around 80% while

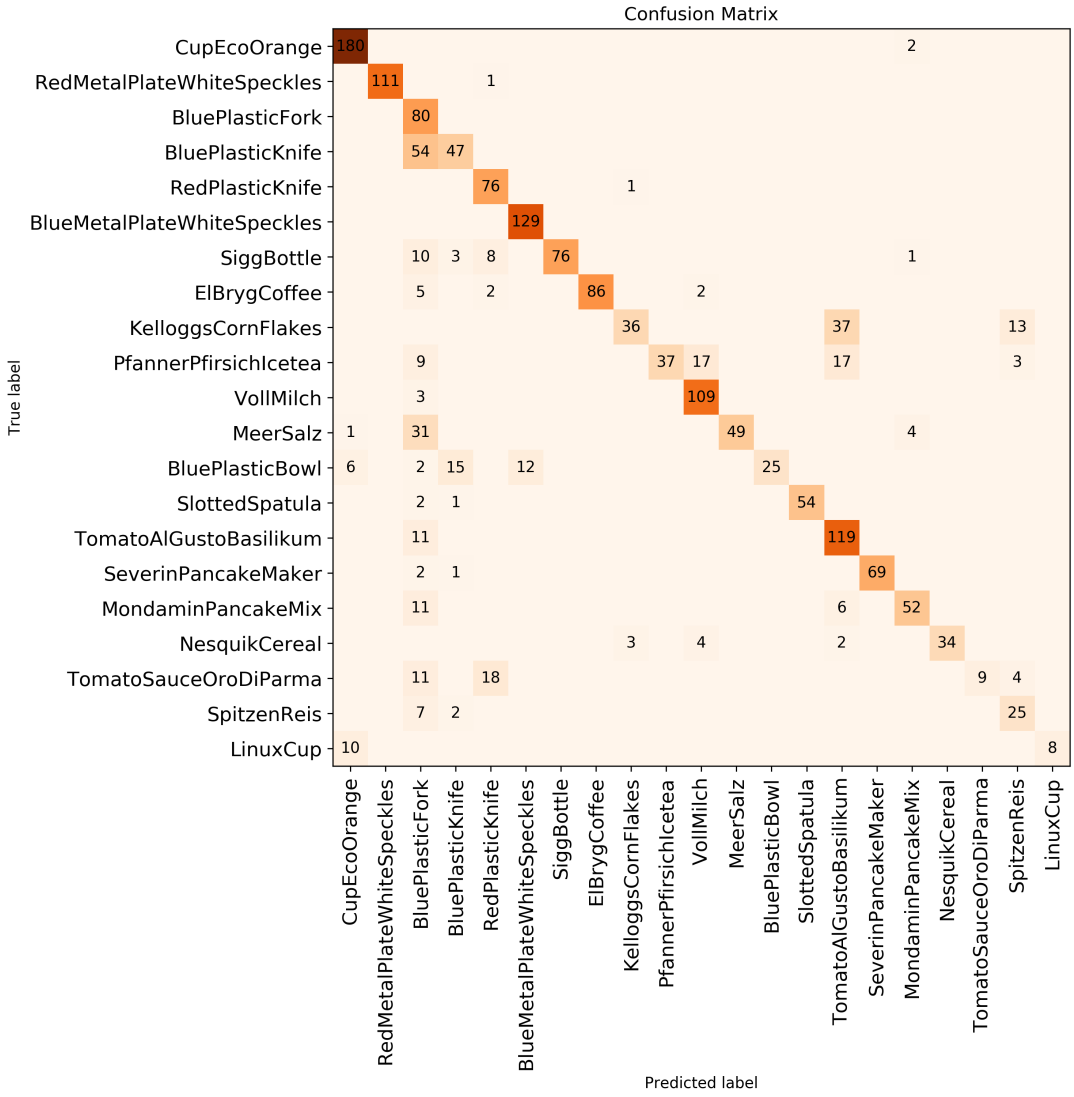


Figure 4.8: Confusion Matrix of the baseline SVM tested on the data from the four episodes

precision is slightly better at 86%. In conditions where the exact same objects and identical camera system were used during both the experiment with the robot and the recording from the turntable this drop in performance is significant. These results are not surprising though, since during execution of a pick-and-place task images of objects can be out of focus, lighting conditions can change, objects can be cut-off in the images, etc. Examples of such images are shown in Figure 4.9, next to images of the same object from the turntable as a comparison.

The goal of the conducted experiment is to demonstrate that the data from

Figure 4.9: Examples of object instances from the turntable data and the same objects from the episodic memories

the perceptual episodic memories are useful for training an instance recognizer for the objects that are encountered by the robot, such that we get more accurate results without the additional need for manual labeling. The query language is used to gather additional training data in a programmatic manner, by taking for each classified instance the results that are considered to be *confident enough*. The notion of *confident enough* is determined by a confidence threshold. If this threshold is too high, the instances chosen from the episodes are too similar to the ones in the turntable data. On the other hand, as the chosen threshold is decreasing, the chance of adapting the classifier with wrongly labeled data increases. In order to find the best possible value, the confidence of the baseline SVM is analyzed on the adaptation set. The process of choosing the best confidence threshold is described in Algorithm 2.

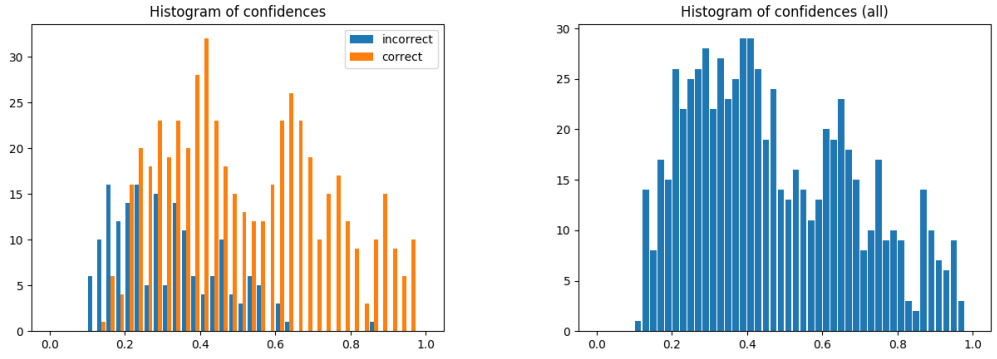
In a first step, all instances of objects from the episode are classified using the baseline SVM. Based on the resulting confidence values and the available ground truth data the distribution of confidence values along correctly and incorrectly classified object instances is analyzed. Histograms of the confidence level are shown in Figure 4.10. Since it is not desired to have any human intervention during the adaptation of the classifier, a confidence threshold needs to be chosen that minimizes the amount of falsely classified instances considered for retraining. We can see from the histograms that this value is somewhere between *0.2* and *0.4*, where the number of false detection peaks and starts decreasing but the number of correctly labeled instances is still high.

To find the exact value the classified instances are split into two groups, based on a confidence threshold. Those instances whose confidence is higher than the threshold are added to the turntable dataset and a new, adapted, SVM is trained.

Algorithm 2: Accuracy analysis of episode data with the help of the proposed query language

Data: episodeData, turnTableData, baselineSVM, confidenceValues = [0.05...0.95]
Result: adaptedSVM

```
accuracies = {};
for conf ∈ confidenceValues do
    confHypMap={};
    nonConfHypMap = {};
    for i ∈ baslineSVM.Labels do
        conf-hyps = (hypotheses(Hyp,
            [class-label:[value:i,confidence>conf,source:BaslineSVM]]));
        confHypMap[i] = conf-hyps;
        non-conf-hyps = (hypotheses(Hyp,
            [class-label:[value:i,confidence<conf,source:BaslineSVM]]));
        nonConfHypMap[i] = non-conf-hyps;
    end
    testingSVM = trainLinearSVM(turnTableData+confHypMap);
    acc = testLinearSVM(testingSVM, nonConfHypMap);
    accuracies[conf] = acc;
end
[newData] = splitEpisodes(max(accuracies, episodeData));
adaptedSVM = trainLinearSVM(newData)
```



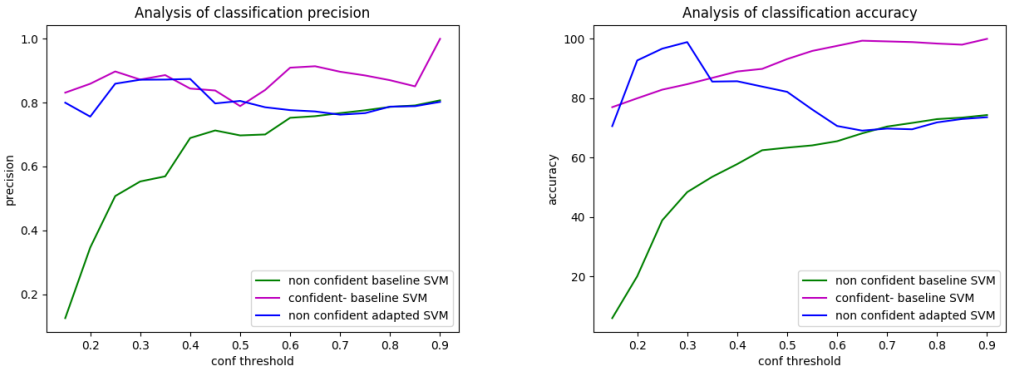
(a) Histogram of confidences for wrongly and correctly labeled instances

(b) Histogram of confidence values for all instances

Figure 4.10: Confidence histograms

The rest of the data, where the confidence was lower than the selected threshold, is used to test the newly trained, adapted SVM. The process is repeated for a

predefined set of threshold values, ranging from 0.05 to 0.95. The query interface is used to retrieve the data from the episodic memories, asking for all hypotheses that were classified as a certain object instance and where the classifier confidence was lower than the given threshold value. Results of the analysis in the form of threshold versus average accuracy and threshold versus average precision are plotted in Figure 4.11.

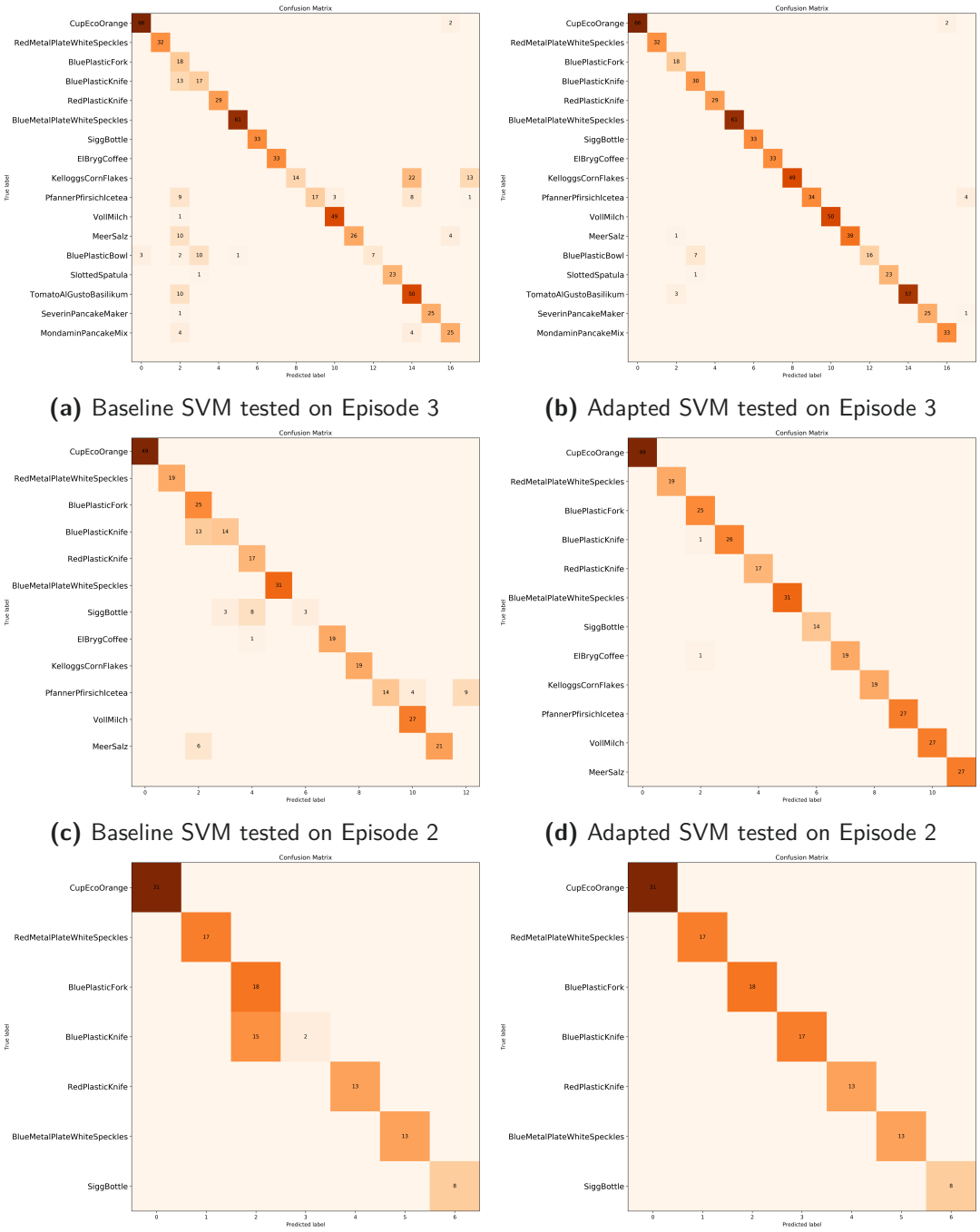


(a) Precision vs. confidence threshold **(b)** Accuracy vs. confidence threshold

Figure 4.11: Confidence analysis of the base classifier performed on data from the adaptation set

The figure shows the results from three different cases: the base SVM tested on the instances from the adaptation set that were confident, the base SVM tested on instances from the adaptation set that are considered non-confident and the adapted SVM also tested on the non-confident instances. The non-confident test data is the relevant one, since it contains previously unseen instances of the objects. We can notice that both accuracy and precision of the adapted SVM have an inflection point around the confidence value of *0.3*, and converge to the results of the baseline SVM towards higher values of the confidence. This is the expected behavior as we have just argued earlier. The more confident the data that we use to adapt our classifier, the more similar it is to the original images and bears no extra discriminative information.

Once the threshold value is chosen the performance of an adapted SVM is analyzed on the remainder of the episodes. This is done by choosing the confident results from three of the episodes to adapt the baseline SVM and testing this on the fourth episode. Specifically, there are three distinct cases: adapting using



(a) Baseline SVM tested on Episode 3

(b) Adapted SVM tested on Episode 3

(c) Baseline SVM tested on Episode 2

(d) Adapted SVM tested on Episode 2

(e) Baseline SVM tested on Episode 1

(f) Adapted SVM tested on Episode 1

Figure 4.12: Performance analysis of adapted classifiers: confusion matrices of baseline and adapted classifiers for the three episodes

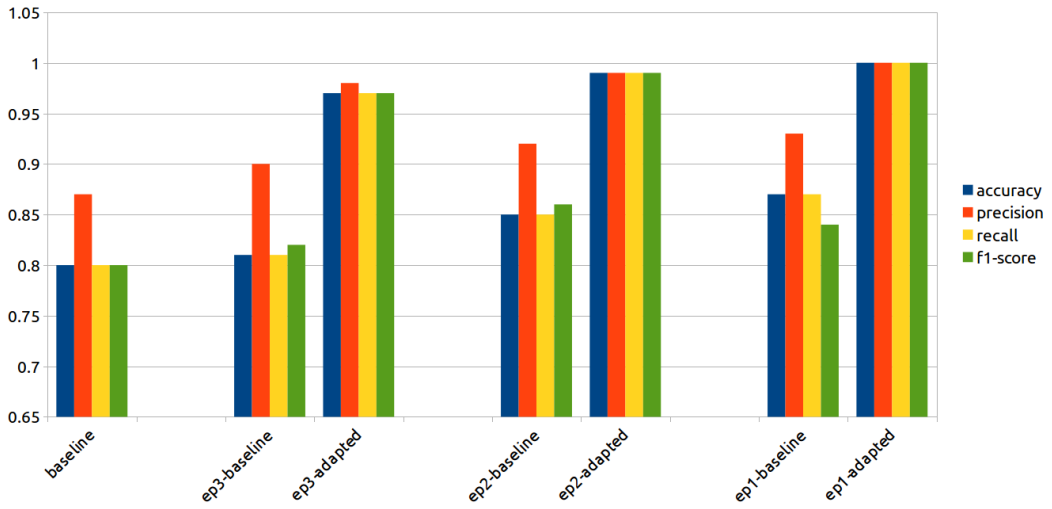


Figure 4.13: Classification metrics when adapting the baseline SVM, trained on data from a turntable, with data from the episodic memories

results deemed confident enough from $E1, E2, E4$ and testing on $E3$, adapting using $E1, E3, E4$ and testing on $E2$ and adapting using $E2, E3, E4$ and testing using $E1$. For each of these three cases the resulting confusion matrix is presented in Figure 4.12. Episode 4 is not tested this way, since it was used for finding the confidence threshold.

The performance metrics of the adapted SVM compared to the baseline tested on the object instances from the same episode are shown in Figure 4.13. From the bar chart we can conclude that using the adapted classifier a significantly better result on all metrics is achieved. On average there is a 14% increase in accuracy, recall and f1-score, and an 8% increase in precision. The example shown here serves only as a demonstrator for the possibility of using the query language as a means for retrieving data for supervised learning problems. Obviously, if the goal is to enable a lifelong learning approach using the described mechanisms, a more thorough analysis is required. Nonetheless, the experiments shown in this section shed light on one of the typical problems of object recognition in robotic application, that is the mismatch between available image datasets and the requirements of real robotic applications. A possible solution is proposed through the use of episodic memories. An in-depth analysis of the problem of transfer learning, and adapting object classifiers was presented in a related

publication by Durner et al. (2017), which is complementary to the previous work (Bálint-Benczédi et al., 2017) this chapter is based on.

4.6 Conclusions

This chapter presented a system for generating perceptual episodic memories and a query interface that allows retrieval of subsets of these. In the context of this thesis, the work presented in this chapter relates to enabling robots to answer general questions about their environment, specifically about past percepts, and towards an effort of enabling robotic lifelong learning through transfer learning.

By offering a way to retrospect perception results both programmatically and through user interaction, a significant step is taken towards enabling robots with awareness about their environment. This is especially the case, when we take into account that the proposed query language is compatible with the more general query interface of the openEASE (Beetz et al., 2015c) framework. Using the openEASE infrastructure, queries can be conditioned not just on information generated by the perception system, but also on actions performed by the robot allowing the execution of queries that ask for all images where the robot was performing a manipulation task. Such queries can allow the specialization of perception algorithms even more.

The experiments in this chapter demonstrate that supervised learning can be used to improve the performance of object recognition. If we are able to generate images that are very specific to a particular manipulation task (e.g. slicing a bread or pouring water), then even more specialized object recognition modules could be trained that would yield better results than their general counterparts in situations where their context requirements are fulfilled. The case for such a system is even more evident when we look at current deep learning trends, where highly accurate object detectors and recognizers can be trained for precisely defined problems.

Another aspect of the work presented in this chapter is that of enabling retrospection during runtime. Enabling robots to access the knowledge about what perception routines they ran, how confident these were in their decisions, opens new possibilities in developing lifelong perception approaches, where self-adjustment

based on previous precepts is important. Although the results of the experiments conducted are promising, as we have already seen, further investigation would be necessary to create a truly lifelong learning architecture. One of the issues that needs to be addressed is how much the newly adapted classifiers over fit the data from the experiments and what strategies are the best at circumventing this.

The perceptual episodic memories recorded by RoboSherlock do not only serve as sources for answering questions about what the robot has seen in the past, but also enable the refinement of answering queries about what the robot sees in the current scene and allow the anticipation of scenes that a robot might see in the future. These two topics are the subject of the chapters to follow.

CHAPTER 5

Amortized, Preparatory Perception for Long-term Manipulation

Robotic agents that are to perform long-term manipulation tasks in realistic environments have to be equipped with sophisticated, robust and accurate object and scene perception capabilities that go beyond the analysis of individual observations. Consider, for example, a robotic agent that is to perform house chores, such as prepare meals and serve them. To do this, it needs to set or clean the table, load and unload the dishwasher, place objects in cupboards, etc. The perception system of such a robot, should be able to quickly answer queries like '*where did I see an object out-of place*' or '*is there anything left on the table*' without the extra effort of navigating to an object's location and applying direct perception. To competently answer such queries having an active memory system, such as the one presented in the previous chapter is necessary but it is often not enough to recall past perception tasks independently. Robots need to be equipped with means for tracking and maintaining these results over time and ways for using them to simplify or solve possible future perception tasks. To address this, in this chapter **I investigate and realize a preparatory, amortized perception component that spreads the act of perceiving through the life-time of the robotic agent** and improves the query-answering efficiency of systems built using RoboSherlock. The two components contribute towards the pervasive operation of the perception system proposed in this thesis.

In the context of long-term manipulation tasks, a pervasively operating per-

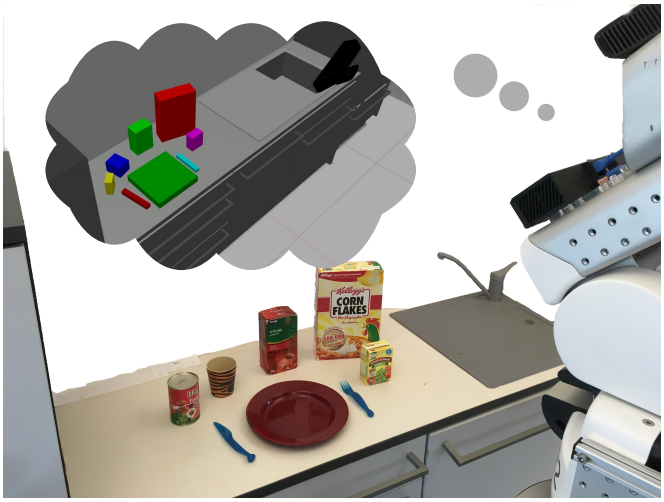


Figure 5.1: PR2 looking at a table scene, and visualization of its internal representation of the sub-symbolic belief state

ception system is aware of the perception tasks that are expected to be issued by the high-level robot plans and *continuously* and *opportunistically* collects information whenever possible making the execution of perception queries easier. The observations of the world, besides being used for knowledge discovery and decision making, become part of a collection of feedback loops that update and refine the object belief state, while interacting with the other software components of the robotic system. For example, the robot considers the objects on a table to be objects in use and therefore candidates for later fetch and place tasks. Therefore, it maintains a belief state containing the objects in use. Whenever possible, the robot automatically performs a perception task to validate, refine and maintain the respective part of the belief. In a query-driven perception system I propose to achieve this through two components:

1. a preparatory perception component that *continuously* analyzes the environment, preparing the perception system for future tasks to come, gradually building a low-level internal representation of the environment (Figure 5.1)

2. and an amortized perception component that *opportunistically* refines the internal representation using results of past perception tasks.

The term *amortized* is most often used in the domain of probabilistic reasoning, as in *amortized inference*, which stands for learning from past inferences, such

that future ones run faster (Gershman and Goodman, 2014). A robotic agent's perception system needs to behave in a similar fashion. Amortization also refers to spreading an asset's cost over its useful lifetime. In this sense the perception tasks need to be spread over the entire operational lifetime of the robotic agent to learn from the past inferences. This means that perceptual processes should be active even when perception is not directly needed by the robot control system such that the utility of past percepts is maximized. For accessing the past percepts the proposed approach relies on the use of semantic logs introduced in Chapter 4. The central question in realizing the amortization effect is how to maximize the information gain from the logged images such that the amount of correctly answerable queries is increased.

The preparatory perception component corresponds to a continuously executed perception pipeline that analyzes observations and generates object candidates. The perception components that make up the continuous execution pipeline are dependent on the task a robot is performing and prepare the system to answer future queries more efficiently by gradually building and maintaining an object belief state. Part of this process involves solving object identities and maintaining a time series of observations that belong to a single object.

The chapter is based on previous work described in Bálint-Benczédi and Beetz (2019). The entity resolution component was joint work together with Thiemo Wiedemeyer, his contribution being the similarity measure based identity resolution as described in our previous work (Wiedemeyer et al., 2015).

5.1 Motivation and Related Work

The proposed amortization effects, as well as the preparatory perception can only be achieved through the management of a belief state about the objects a robot encounters. There are several reasons why computing and maintaining an object belief state in an amortized manner is feasible. First, many environments (such as an apartment, a factory or a surveillance area) change slowly compared to the frequency in which robots capture images of parts of them. Second, during operation, robots typically capture multiple images from different views of the same scenes and therefore can exploit the images from different views to improve results.

Third, when the robot acts in the environment it often has spare computational resources, in particular when it navigates from one place to another. In these cases it can perform perception tasks on previously captured images without delaying robot activity. Fourth, if the robot has already a good model of a scene, it can drastically simplify perception tasks. For example, instead of answering a query by computing the information from the image, it can test whether the captured scene matches its previously estimated model and answer the query based on the model. Often detecting the difference between an expected and captured image is much easier than computing the respective information from scratch.

A large proportion of the most commonly used object detection and recognition systems in robotics, as the likes of LineMod (Hinterstoisser et al., 2013) or more recently deep learning based approaches like YOLO (Redmon and Farhadi, 2016), process images in a one-shot manner. Other popular perception systems enable visual servoing, tracking low-level features in an image (e.g. SimTrack (Pauwels and Kragic, 2015)). However, they are oriented towards the short-term tracking of objects during individual manipulation tasks. Tracking objects over prolonged periods of time and remembering their location in the world is similar to the problem of Simultaneous Localization and Mapping (SLAM). Most of the SLAM approaches however are concerned with capturing the world around the robot as accurately as possible and localizing the robot in it, less focus being put on the semantic interpretation of the data. In a thorough survey of SLAM approaches, Cadena et al. (2016) identify several open problems among which the need for high-level, rich representations. Gemignani et al. (2016) also argue for a deep understanding of a robot's environment and propose a semi-autnomous approach for acquiring the semantic maps with the help of human interaction. In another recent work Deeken et al. (2018) propose SEMAP, a framework for semantic map representation that enables qualitative spatial reasoning about objects in a robots surrounding, but they also mostly consider larger-scale objects, such as furniture pieces. The object belief states built using the approaches presented in this chapter are complementary to these semantic object maps.

A comprehensive work on object belief states, for robots performing everyday manipulation tasks, was done by Blodow (2014), laying the ground work for what is necessary to create and manage such representations. In an earlier work, Blodow

et al. (2010) present a Markov Logic Network (MLN) based framework for dynamically resolving the entities of objects. The idea of entity resolution is further developed by Wiedemeyer et al. (2015), where a pervasive 'calm' perception component for RoboSherlock is built, which acts as a preparatory perception routine.

Another interesting approach is taken by Lee et al. (2014b). They argue for the importance of a correct belief management and highlight connections to biological systems. Milliez et al. (2014) and Warnier et al. (2012) present SPARK, a framework for belief management, similar to the work presented here, but with a focus more on spatial reasoning and the knowledge component. Parts of SPARK consist of managing an object belief state, but since emphasis is put on spatial reasoning, the scenes and objects used in the experiments are idealized.

The argument for building belief states in an amortized preparatory manner, is excellently phrased by Gershman and Goodman (2014), where authors state that the:

> " brain operates in the setting of amortized inference, where numerous related queries must be answered [..] in this setting, memoryless algorithms can be computationally wasteful. "
>
> *S.J. Gershman and N.D. Goodman*

5.2 System Overview

The extension to RoboSherlock proposed in this chapter manages and updates beliefs about objects in the environment during the entire execution of a long-term manipulation activity, such as pick and place tasks. Preparatory and amortized perception are implemented using a custom Aggregate Analysis Engine (AAE) executor, generating sub-symbolic and symbolic data by separate computational blocks, but maintaining a common belief state. Symbolic and sub-symbolic data are treated separately since parts of the information that constitutes the belief state (e.g. labeling objects with symbols) are derived on demand through the query answering mechanism of RoboSherlock, but other parts (e.g. resolution of object entities) are handled by using sub-symbolic information that can be

generated at a higher frequency. Another reason for differentiating between these two types of data is that most of the state-of-the-art algorithms used in object recognition and scene understanding are not able to handle the real-time (or close-to real-time) requirements imposed by the needs of service robotics. If they do (e.g. SimTrack (Pauwels and Kragic, 2015)), they can only handle a subset of the objects one might encounter. The problem with not processing observations fast enough is that we risk loosing valuable information and that we are delaying execution on the robot by waiting for computationally expensive algorithms. By maintaining an object belief state based only on sub-symbolic information we can have low-level percepts at higher frequencies about the objects in the environment, while maintaining the option of having semantic information where needed.

In order to build the proposed system there are two main components that need to be detailed. First, how symbolic and sub-symbolic information about the objects in the world are stored in a unified manner. Second, how the computational processes that generate these interact with each other and the robot's high-level control system.

The main data structure in RoboSherlock is the CAS, which is based on a whiteboard architecture, allowing the independent annotation of arbitrary regions in the raw data from different information sources. This enables a coherent integration of symbolic and numeric (sub-symbolic) data and universal access for all computational components. Let's recall from Chapter 2 the definition of the CAS. Let $\mathcal{P}(\mathcal{H})$ be the power set of all hypotheses and $\mathcal{O}$ the set of all observations. The space of all CASs is defined as:

$$\mathcal{CAS} := \mathcal{O} \times \mathcal{P}(\mathcal{H})$$

A hypothesis represents a certain region in our observations and a set of annotations attached to this region. A region in an observation is a set of indices of the vector of observations, e.g. pixels in an image or 3D points in a point cloud. An annotation is an $ABox$ representing a fragment of the belief state. Let $\mathcal{P}(\mathbb{N}^+)$ be the power set of pixels of an image or points of a cloud. The space of all hypotheses $\mathcal{H}$ is then defined as

$$\mathcal{H} := \mathcal{P}(\mathbb{N}^+) \times ABox.$$

Using description logic, an exemplary hypothesis of a box-like red object which has a pose and a 3D feature descriptor would have the form:

$$Thing(h_1) \wedge shape(h_1, box) \wedge color(h_1, red) \\ \wedge pose(h_1, [x, y, z, r, p, y]) \wedge vfh(h_1, [...])$$

As we can observe, the properties of the hypothesis are either numerical values or a symbol, depending on the type of *relation*. During the long-term execution of a robot experiment a trajectory of CASs is is generated:

$$CAS_T : T \longmapsto \mathcal{CAS}$$

where $T = \{0, \dots\}$ is a set of globally unique timestamps.

Objects that a robot encounters during the execution of a task bear various visual characteristics that can only be recognized using different algorithmic approaches. For this reason, processing the raw data in an ensemble-of-experts approach is beneficial. This means that processing is split into several special purpose modules that generate object hypotheses in the RGB-D image captured by the robot or annotate these hypotheses.

Figure 5.2 depicts an overview of the proposed execution cycle. Recall from Chapter 2 that the definition of an Aggregate Analysis Engine (AAE) contains the list of expert algorithms (PAEs) that should be run during a task, also referred to as the list of delegates (Section 2.4.5). This is the pool of PAEs that is used for pipeline planning. Besides these, a *fixed flow* (referred to also as the low-level perception pipeline) is given, which is a sequence of PAEs that defines the continuous execution block. The AAE executor is extended such that the PAEs from the *fixed flow* are continuously run and are responsible for generating object hypotheses and the sub-symbolic annotations of these, preparing the system for future perception tasks. At the end of each processing cycle a CAS_t is generated that contains the interpretation of the observations and a CAS Consumer resolves the identities of the hypotheses to existing beliefs that make up the sub-symbolic belief state (see Section 5.3.1). The continuous execution analyzes input images at a high frequency (5-10Hz) and raw data that

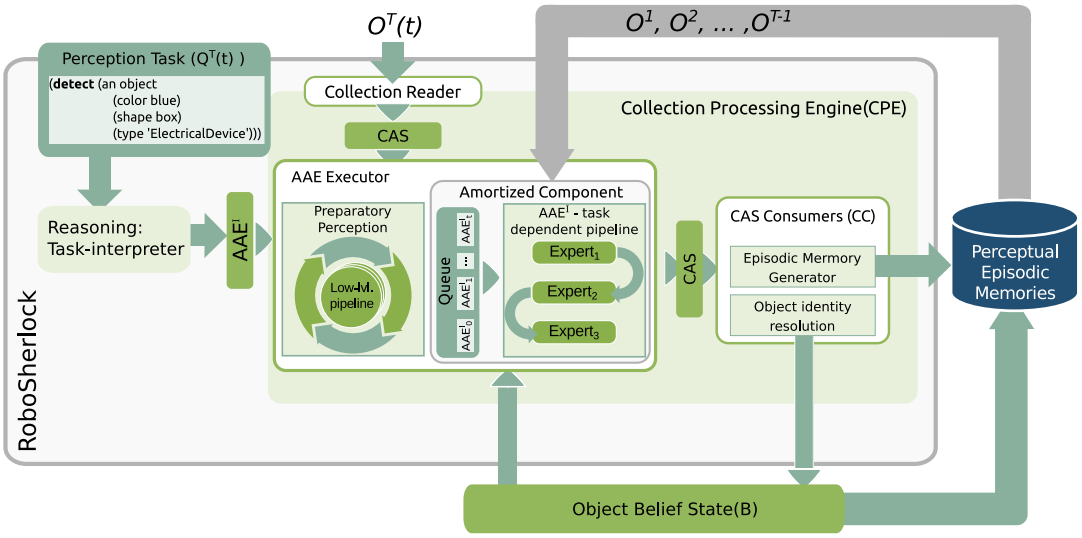


Figure 5.2: System overview with the interaction of all major components.

gets processed is logged for later processing. During the execution of a task, the robot control program sends queries ($Q^T(t)$) formulated using the query language introduced in Section 2.3.2. These queries get interpreted and a tasked pipeline is planned (AAE^I) and executed, resulting in a set of annotated hypotheses. The resulting hypotheses are then matched to the objects in the sub-symbolic belief state refining these using the symbolic information that result from the query.

At this stage, symbolic results get generated only for the objects that are in the scene at the time of the query. This is equal to answering a query through performing one-shot perception. To achieve the desired amortization effect and spread the cost of perception throughout the execution of a task, perception plans generated are queued and executed on the previously logged images. This is done in parallel and in the background at times when the robot is idling or moving from one place to another. This way, the belief state gets updated with symbolic data that results from object hypotheses found in previous scenes and the representation of objects in it becomes richer. Perception plans from the queue are processed in a first in, first out manner, executed on the most recent images. The image filters described in Section 4.3.1 assure that there is a good degree of data throttling so that we do not waste resources on processing redundant images that would yield little or no information gain.

5.3 Implementation

The preparatory, amortized perception components are implemented using RoboSherlock and a custom AAE executor. The executor implements the logic of switching between low-level and tasked sequences of PAEs. The storage component described in Chapter 4 is used to log the stream of images and results. The low-level, continuous pipeline detects objects in each frame obtained by the sensors. Each frame is only a snapshot of the current scene. By default, the objects detected in one frame are independent from the ones detected in previous or future frames. To connect each of these observations, object identity resolution based on the low-level percepts is run.

5.3.1 Generating Sub-symbolic Representations

(adapted from previous work, by courtesy of Thiemo Wiedemeyer)
The sub-symbolic representation of objects consists of numerical annotations acquired by the continuous execution component. Initially, the system is aware only of hypotheses of objects that have visual descriptors and low-level geometric information: e.g. estimated pose in the world and oriented 3D bounding box. These low-level percepts, the position in the world and the timestamps help disambiguate between objects of similar characteristics. Each object hypothesis contains a number of annotations, computed by the continuous pipeline. Annotations needed by the entity resolution framework are directly returned by PAEs of the low-level pipeline. For the similarity measurement, a set of low-level annotations are used: geometry, color histograms and image features. To compute the similarity between annotations of the same type, a distance function $dist(a, b) \rightarrow [0 \dots 1]$ is defined for each of them. When solving the identity of objects, the distance function takes two annotations of the same type as parameter and computes a normalized distance d. Each distance d is weighted by the factor w, which is heuristically defined for each type. If we consider $\tau = \{\tau_0, \tau_1, \dots\}$ to be the set of low-level annotations used, then the overall distance for an object a to an object b is the sum of all weighted distances normalized to $[0 \dots 1]$, defined as:

Features used for sub-symbolic matching	
Visual Feature	Description
Color features	HSV color histogram
3D features	VFH, C-VFH or OUR-VFH
Generic descriptors	$DeCAF_7$ trained on ImageNet2012 (Donahue et al., 2014)
2D-Keypoints/Descriptors	BRISK/FREAK
6DOF Pose	3D estimate from centroids or 2D estimate from Hu-moments
Symbolic annotations	
color	symbolic color based on HSV color distribution
shape	primitive shape, inspired by Goron et al. (2012)
class label	k-NN classifier trained on $DeCAF_7$ descriptors of partial views of objects from a turntable

Table 5.1: Examples of symbolic and sub-symbolic perceptual features and annotators

$$dist(a,b) = \frac{\sum_{i=0}^{|\tau|} w_i * dist_i(a_i, b_i)}{\sum_{i=0}^{|\tau|} w_i}$$

The weighting is applied to prioritize certain annotations that are more reliable and discriminating than others. For example, shape histograms are similar for many objects while color histograms and image features differ more. The weight for each annotation is chosen based on empirical results with respect to their reliability and discrimination.

The visual features extracted in the continuous execution during the conducted experiments are presented in the upper half of Table 5.1. For the geometry annotation, which contains a minimal bounding box, the distance is the normalized sum of the distances in each dimension (width, depth, height) of the bounding box:

$$dist_{geo}(a,b) = \frac{dist_d(w_a, w_b) + dist_d(d_a, d_b) + dist_d(h_a, h_b)}{3}$$

The distance $dist_d$ for each dimension is the absolute difference divided by the minimal length, limited to one. Every difference that is greater than or equal to

the doubled minimal distance is mapped to 1, everything lower is linearly mapped from 0 to 1:

$$dist_d(a,b) = \min(1, \frac{|a-b|}{\min(a,b)})$$

The distance between two color histograms, which are normalized, is the sum of absolute differences per bin divided by two:

$$dist_{color}(a,b) = \frac{\sum_{i=0}^{len(color_hist)} |a_i - b_i|}{2}$$

All image features and descriptors from one object are matched against the ones from the other object using a brute force matching algorithm, which gives back a minimal distance d_i for each descriptor. These distances are summed up and divided by the product of the number of descriptors n and a constant $maxDist$ which defines the maximal distance between descriptors that is taken into account. The result is limited to 1, if higher:

$$dist_{feature}(a,b) = \min(1, \frac{\sum_{i=0}^{len(descriptor)} d_i}{n * maxDist})$$

The maximal distance $maxDist$ can be defined lower than the real maximal distance between descriptors. Image descriptors can be considered as dissimilar even if they are not near the maximal distance. This increases the resolution for the similarity measurement for near descriptors and reduces the limit of dissimilarity that is inspected.

During object identity resolution, for each hypothesis found in the scene the similarity to each known object from the belief state is computed and together with their pose passed to the entity resolution framework, which returns for each cluster the probability of belonging to a certain object. Based on the results, the hypotheses are either assigned to the best object candidate or get added as new objects. Objects from the belief state that do not have a hypothesis associated from the current scene are marked as out of view.

The resolution of identities is done by comparing descriptions of object hypotheses in the current scene with the description of objects from the belief state. A fast-matching between objects that had been seen in the last frame and should still be in view is performed with the object hypotheses from the scene. The

fast-match quickly finds candidate pairs, based on the distance between the known and perceived object pose and the time that had passed since it was last seen. For each pair the similarity is computed and if it exceeds a predefined threshold, the two get merged.

5.3.2 Symbolic Belief Management and Amortization

In order to be competent at answering complicated queries, robots need to have semantic information about the objects in the environment. Most of the state-of-the-art algorithms used in object recognition and scene understanding are not able to handle the (close-to) real-time requirements of mobile robots, or if they do, they only handle a subset of the objects in the environment. For example SimTrack (Pauwels and Kragic, 2015) can detect and track several objects in real time simultaneously but only if they have enough texture. Because of this, symbolic data in the system is introduced through asynchronous query answering. Queries are formed using the query language described in Section 2.3.2, which offers the necessary flexibility and descriptiveness. A very simple query that asks for a flat shaped object that is black is shown below:

```
(detect (an object
            (shape flat)
            (color black)))
```

The symbolic information maintained in the belief state depends on the queries that are issued by the high-level control system of the robot. The right side of Figure 5.3 depicts the link between the symbolic and sub-symbolic beliefs. In addition to the low-level percepts stored in the sub-symbolic belief state, symbolic information, as the likes of color, shape, class label or pose of a CAD-model, are linked to these beliefs.

The sources of symbols are presented in Table 5.1. In this chapter only one source of information for each type of symbol is considered (shape, color, type), and the focus is on how the correctness of these effects query answering and to what extent does amortization improve results. For each object in the belief state a histogram of the resulting symbolic values is maintained, where each bin of

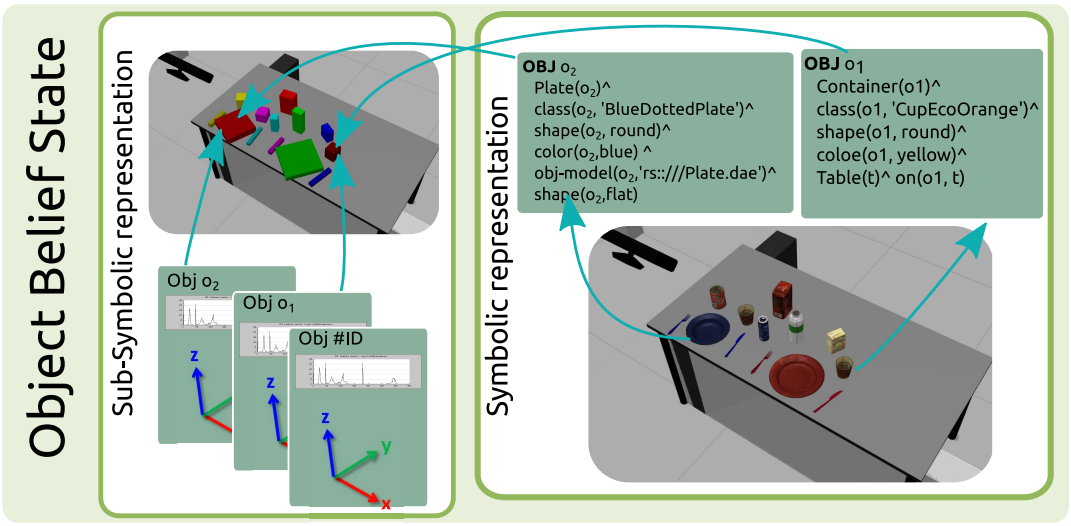


Figure 5.3: Example of symbolic and sub-symbolic beliefs about objects in the belief state

the histogram corresponds to one of the symbols. These histograms get updated through the amortization effect, namely the execution of the buffered perception plans on the logged data. When answering a query the symbolic value that has the most votes in the histogram is considered to be the correct answer.

5.4 Experimental Analysis

Amortized operation improves the query-answering capabilities of a robotic agent by integrating symbolic results from past images into the decision-making process, while the preparatory perception allows for a slow tracking of objects in the environment. In order to demonstrate this, I analyze the correctness of the sub-symbolic belief state and the effects of using the background- and task-knowledge enabled filters, evaluating the preparatory perception system. This is followed by a qualitative and quantitative analysis of amortization effects.

5.4.1 Sub-symbolic Management of the Belief State

To evaluate the low-level pipeline and the identity resolution of the objects, the experimental set-up we described in Wiedemeyer et al. (2015) is followed. In comparison to previous work the complexity of scenes used in this evaluation is increased by having more objects and object locations, and by performing pick

Figure 5.4: Initial state of the two tabletop scenes in episode 2.

and place tasks. The experiment consists of four different episodes with increasing difficulty, where the task of the robot is to move objects from one supporting surface to another. The episodes are the same as the ones used in the previous chapter and were additionally hand labeled to contain ground truth about the symbolic information of the objects (shape, color and class).

In all cases the experiments were conducted using a localized[1] PR2 robot that was performing pick-and-place tasks in a kitchen environment[2]. The objects chosen for the experiment are typical household items that possess varied visual characteristics. Some are flat and textureless, while others are textured and well visible (see Figure 5.4 for example). The objects are clearly separated with few occlusions, since we are only interested in analyzing the effects of the preparatory, amortized perception on the annotations. The hypotheses throughout the experiments are generated using the combination of 3D Euclidean clustering with a threshold based binary color segmentation, useful for small flat objects that otherwise would not be found. For classifying the objects k-NN, detailed in Table 5.1, trained on the 21 object instances was used.

Details of the four runs, such as duration, number of pick and place tasks performed (*Nr. of p.p. tasks*), number of objects at the end of execution in the belief state and number of total object hypotheses in the episode (*# of Hyp*) are presented in Table 5.2. Although this is the same data as in Chapter 4, the

[1] we use adaptive Monte-Carlo localization (`wiki.ros.org/amcl`)

[2] Actual picking and placing of the objects was handled by a human, in order to reduce the complexity

	Ep.1		Ep.2		Ep.3		Ep.4	
Nr. of objects	9		15		20		25	
Duration	277(s)		328(s)		510(s)		520(s)	
Nr. of *Hyp.*	130		353		694		759	
Nr. of p.p. tasks	3		4		6		10	
Filters	Off	On	Off	On	Off	On	Off	On
Nr. of objects in b.s.	15	9	26	20	49	28	54	31

Table 5.2: Summary of episodes of the conducted experiments

number of objects in the second row of the table is higher than the ones from the previous chapter reported in Table 4.2. This is because some object types appear multiple times in the episodes (e.g. there are two identical cups) and for entity resolution each individual object matters as opposed to the object classification tasks investigated in Section 4.5.

The weights of sub-symbolic annotations that are taken into account in the entity resolution part were set to be equal, so $w_k = 1$ for all percepts. Doing so introduces no bias towards any type of annotation. The threshold for fast matching that only compares poses of objects from last frame and the hypotheses in the current frame was empirically set to $2.5cm$, allowing for small errors in robot localization and compensating for noise of the RGB-D sensor. The number of actual objects in the environment compared with the number of objects in the belief state at the end of a pick and place task gives an insight into how well the system is able to track objects over time. The last row of Table 5.2 highlights the difference in results when using the knowledge-based filters described in Section 4.3.1. It would be difficult to build the desired amortization effect on top od the identity resolution without these filters (blur, motion, change detection), since the identity resolution performs significantly worse without them. Since the main contribution of the chapter is the realization of the amortization effect it would be out of scope to further investigate the implemented entity resolution. The heuristic rule-based entity resolution together with the knowledge-based filters is deemed satisfactory for analyzing the amortization effects. For a probabilistic approach on solving object entities, I kindly refer interested readers to the previous work of Blodow (2014).

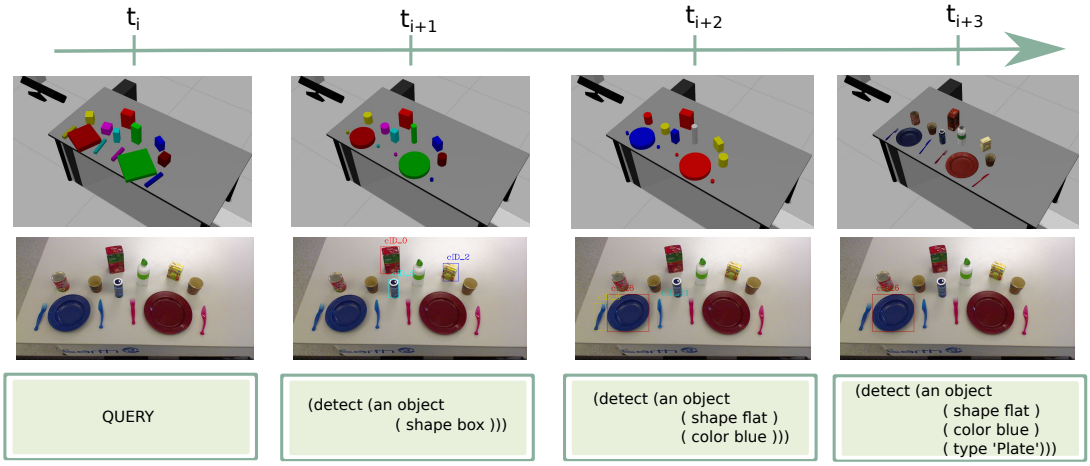


Figure 5.5: Updating the symbolic beliefs about objects in a static scene. First row: the rendered belief state. Second row: Highlighted results of the queries in the original image. Third row: query asked

5.4.2 Symbolic Updates of the Belief State

To showcase the interaction between the sub-symbolic and symbolic belief states, a detailed description of the updating process for a static scene and for one of the pick and place episodes from the previous subsection is presented.

Let us look at how the beliefs about a static scene get updated over time as new symbolic information is computed. Figure 5.5 depicts a typical breakfast scenario as it gets added to the belief state. At the very beginning (t_i) the environment is populated with the objects from the sub-symbolic beliefs generated by the continuous component, consisting of the pose and the estimated 3D bounding box. After querying for the shape of an object at t_{i+1}, the belief about the shape of all visible objects gets updated. At t_{i+2} symbolic color information is added (we choose the top color of each object for visualization), while at t_{i+3} due to a query asking for the types of objects, each belief gets replaced by a CAD model of the object instance it gets classified as (if a CAD model exists).

Figure 5.6 shows another example where the the evolution of the belief state happens while a robot is performing a pick and place task. In this episode the robot was moving between the table and the counter top, and setting a table was simulated. The symbolic beliefs only get updated as queries are being answered and are attached to the sub-symbolic beliefs, so when objects change locations,

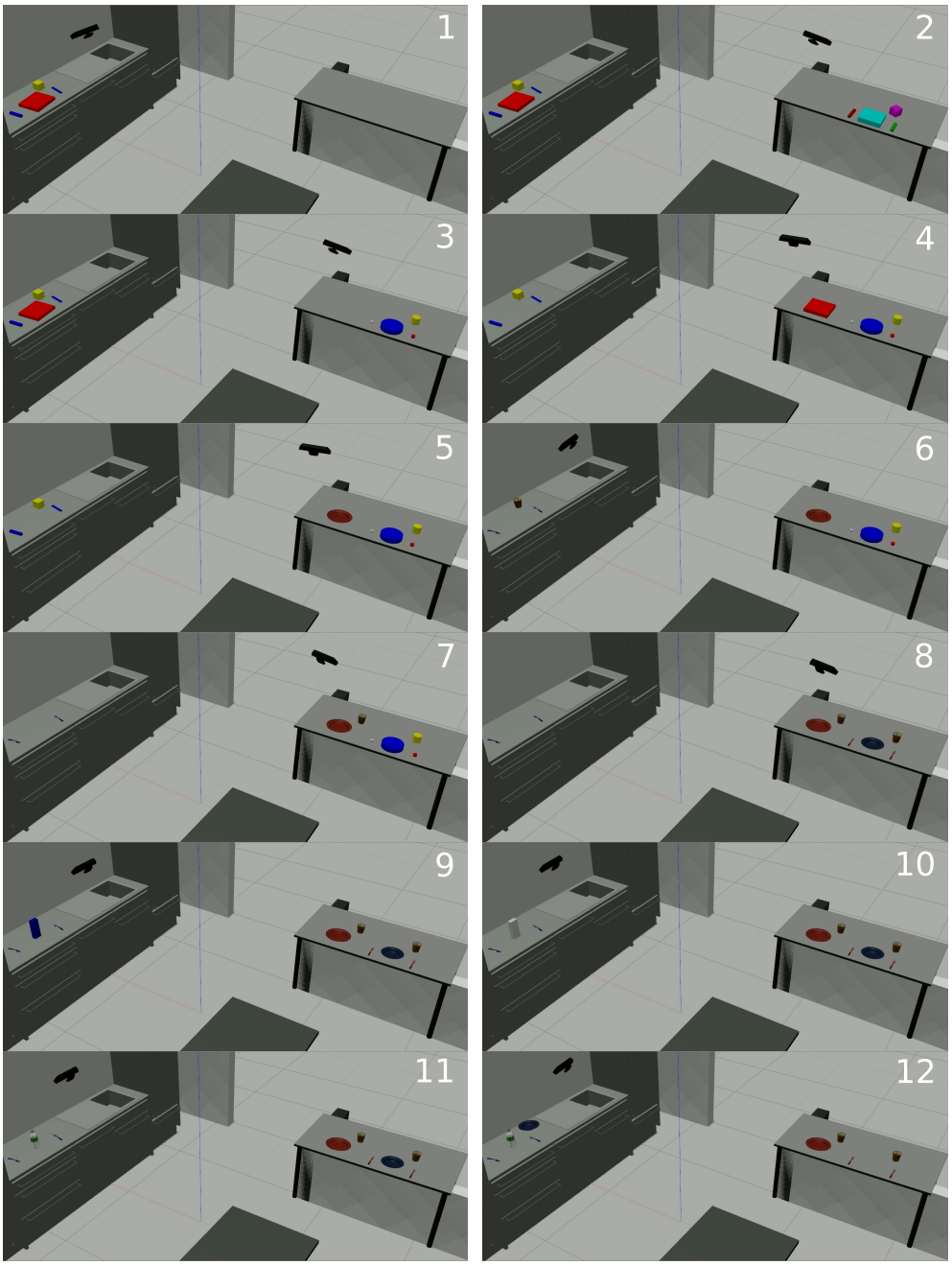


Figure 5.6: Evolution of the belief state, as queries get answered during a pick-and-place operation for episode 1. (1) low-level percepts and color analysis of left table, (2) low-level percepts of right table, (3) shape analysis for objects on right table, (4) red object moved to right table, (5) classification of the red object, (6) classification of the objects on the left table, (7) cup moved to right table, (8) classification of the other objects on the right table, (9) new object on left table, (10) color analysis for new object, (11) classification of new object, (12) blue plate has been moved to left table.

id	Object	# obj.hyp	#classifications	% correct
3	blue knife	127	81	**30.8** [%]
5	red plate	83	81	97.5 [%]
8	sigg bottle	83	**7**	100 [%]
10	yogurt	62	25	100 [%]
11	cup	75	60	86.6 [%]
12	ice tea	176	**51**	100 [%]
13	soy milk	176	176	**57.9** [%]
15	salt	176	**30**	100 [%]
16	blue fork	90	85	83.5 [%]

Table 5.3: Analysis of object hypotheses per object in the belief state. Gray rows: few successful classifications; Cyan rows: low precision

we do not loose the existing beliefs, e.g. between frames 6 and 7 where the cup gets picked up from the counter top and placed on the table. The symbolic beliefs persist even when we are not looking at a certain scene.

5.4.3 Benefits of Amortization

Through amortization the results from past queries executed on previously seen images are integrated into the belief state of the robot. Answering a query correctly means having the correct symbolic data deduced from the current scene. In the remainder of this chapter I refer to this as one-shot perception. With amortization, the deduction of these symbols happens through integrating the results from the past instances of the same object. Quantifying the correctness of query-answering is equivalent to measuring the correctness of symbolic labels for each of the objects at each occurrence during a task.

Let us first look at this problem from a qualitative perspective on objects selected from *Episode 2*. A query that asks for a specific object at a moment when this is falsely classified or the classification result is not confident enough would not yield a correct answer. The type of an object in RoboSherlock is typically deduced using a classifier trained on a set of predefined objects in advance. In this experiment *KnnAnnotator*, trained on turntable data, is used for the baseline performance with a confidence threshold of the k-NN classifier set to 0.6. Confidence in the case of k-NN is simply one minus the distance of

the normalized feature descriptors. The colored rows in Table 5.3 highlight two distinct cases where a query for a specific type of object would not be answered correctly based only on the evidences from the current scene. First, the "blue knife" and the "soy milk" are falsely classified in a lot of the scenes. In all of those cases, a query about those objects would not yield a correct results. Second, the case of the "sigg bottle", "salt" and "ice tea" are worth taking a closer look at. Based only on the results from the continuous execution, all three objects are successfully tracked by the entity resolution, having a high number of hypotheses attached to them. On the other hand all three have a low number of classification results, all of which are correct. This means that for the scenes in which the hypotheses do not get classified we would not be able to answer queries about these objects.

In these cases, in order to benefit from the amortization effect, two things need to happen: there needs to be a perception plan in the buffer that returns the symbolic value we are interested in (*type* in the case of our example) and enough time needs to pass such that the buffered plan can execute on a logged image where the object's type can be detected. I look at these two conditions for the "sigg-bottle" and "salt" in Figure 5.7. The X-axes of the figure represents indices of hypotheses generated over time for the objects. The blue bars represent the duration between two consecutive hypotheses in seconds. The red bars mark when the hypothesis was correctly classified. In the case of these two objects there were no false detections. Whenever there is a large time gap, the robot is either moving or one of the filters is active, allowing for the buffered perception plans to execute.

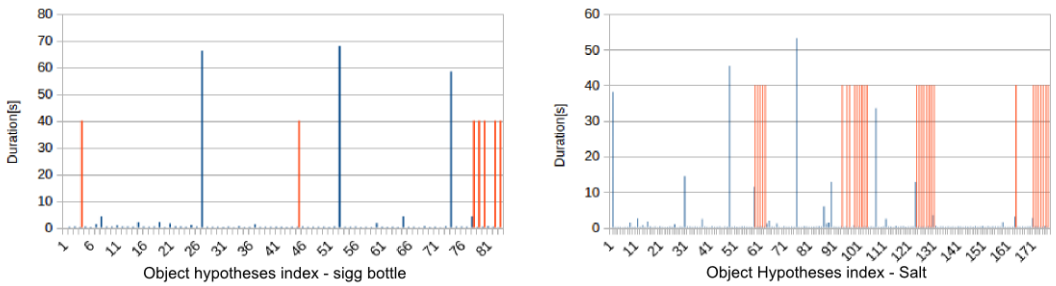


Figure 5.7: Results of classification for the hypotheses of the *sigg bottle* and *salt*. Blue bars represent the time elapsed between hypothesis generation, red bars mark the hypotheses that are correctly classified. Amplitude of red bars is only for ease of viewing.

		Ep.1			Ep.2			Ep.3		
		A	**P**	**R**	**A**	**P**	**R**	**A**	**P**	**R**
Shape	One-shot	0.68	0.82	0.69	0.65	0.67	0.65	0.66	0.67	0.66
	Amort.	0.76	0.83	0.76	0.68	0.71	0.69	0.73	0.75	0.73
Color	One-shot	0.87	1.0	0.87	0.91	0.95	0.92	0.82	0.9	0.83
	Amort.	0.89	1.0	0.89	0.98	0.99	0.99	0.86	0.93	0.87
Class	One-shot	0.93	0.95	0.93	0.98	0.98	0.98	0.95	0.96	0.95
	Amort.	0.99	0.99	0.99	0.96	0.96	0.96	0.92	0.94	0.92

		Ep.4			Average			
		A	**P**	**R**	**A**	**P**	**R**	
Shape	One-shot	0.63	0.64	0.64	0.65	0.7	0.66	
	Amort.	0.68	0.68	0.68	0.71	0.74	0.71	
Color	One-shot	0.84	0.92	0.84	0.86	0.94	0.86	**Cov. one-shot**
	Amort.	0.85	0.91	0.85	0.89	0.95	0.9	82.2 %
Class	One-shot	0.96	0.97	0.96	0.96	0.96	0.96	**Cov. amortized**
	Amort.	0.89	0.90	0.90	0.92	0.92	0.93	94.3 %

Table 5.4: Accuracy (**A**), Precision(**P**) and Recall(**R**) for shape, color and class annotations of object hypotheses with and without the amortization (**Amort.**) effects.

For the "sigg bottle", between the two hypotheses that match the object at index 26 and 27 there is a time gap of more than 60 seconds. If in this time the system manages to process enough frames to reach the logged image where hypothesis number 4 was found (where the red bar marks a successful classification), the result can be propagated to all future hypotheses that get added to this object. As a second example, in the case of the *salt* one would need to wait until the second big time gap to find past results where the hypotheses can be correctly annotated. Once a result is found, queries would get answered correctly for the scenes containing the rest of the instances.

These observations empirically demonstrate that amortization can be beneficial during the life-cycle of a robotic agent. Let us now quantify the effects of amortization. To do this I look at the extreme case of the robot asking a query about an object's type, shape and color at every frame, independently of each other. Results of this analysis are shown in Table 5.4. There are two measures to consider when analyzing the results: the performance metrics of correctly annotating a hypotheses and the number of hypotheses that are annotated (referred to as

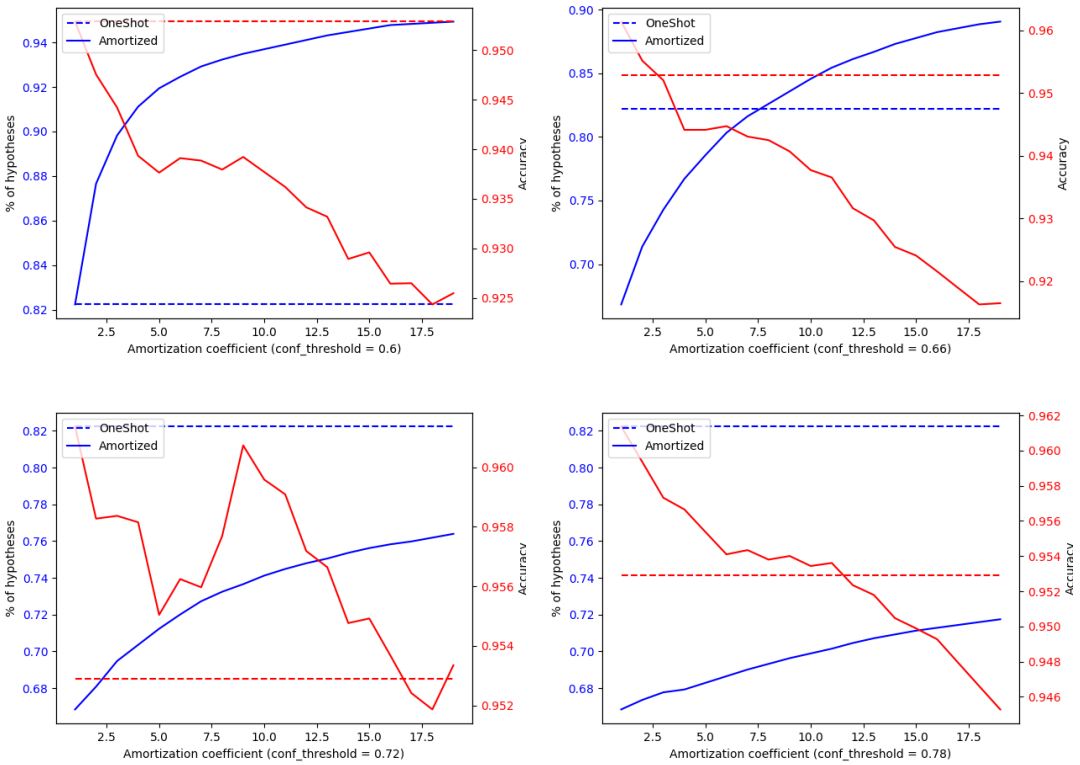


Figure 5.8: Trade-offs between hypothesis coverage and accuracy when varying confidence thresholds and the amortization coefficient

coverage) with and without amortization.

The last entries of the table report the average performance measures over all four episodes with two additional metrics: coverage of data with and without amortization. When considering scenes in isolation, even though the accuracy and precision of the classification is high, only 82.2% of all hypotheses are annotated. In the rest of the cases the confidence of the k-NN classifier does not meet the classification threshold. This is due to the idealized nature of the turntable data used for training the *KnnAnnotator*, in which objects are always centered and clearly visible. It is these two measures (accuracy and coverage of hypotheses) that need to be jointly maximized through amortization in order to increase the number of queries that are correctly answered.

The amortization process depends on two parameters:

- an *amortization coefficient (ac)*: defining how far into the past to go when

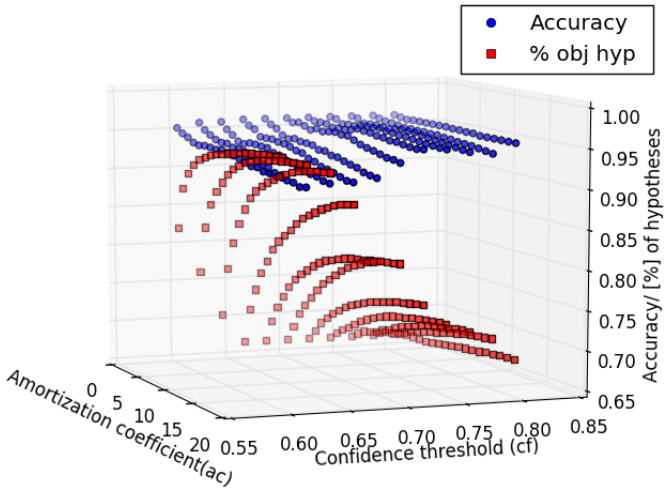


Figure 5.9: Relationship of coverage, accuracy of class annotation, confidence threshold and amortization coefficients

processing hypotheses,

- and a *confidence threshold (cf)*: defining how confident the result from the past has to be, to be taken into consideration.

A larger amortization coefficient allows the integration of more results from the past at the cost of classification performance. On the other hand, a higher confidence threshold yields a better performance but it also requires more images to be processed from the past. This trend is visible in Figure 5.8 which illustrates the inverse proportional relation of the accuracy and coverage with $cf = 0.6, 0.66, 0.78$ and $ac = 1 : 20$.

In order to find the best combination of the parameters, grid search is performed. This is visualized as a scatter plot in Figure 5.9. The best choice of parameters is along the line of intersection of the two resulting manifolds, given by $max(ac + cf)$ and is found to be at a slightly higher confidence threshold than the one-shot classification (0.62) and at an amortization factor of 13. In practice any similar value would yield satisfactory results, the differences being only minor.

Table 5.4 presents the performance measures on the four episodes. Overall, in the case of shape and color a general increase in all measures can be noticed, while in the case of classification there is a slight decrease, but with the added benefit of a 10% increase in coverage. Since the changes in accuracy, precision or

recall for the k-NN are small, from the perspective of maximizing the number of answerable queries it is a better choice to select parameters where these measures are slightly worse, but the coverage increases as much as possible.

Considering that queries contain more than one key at a time, the increase of the number of queries that can be correctly answered using amortization is considerably larger.

5.5 Discussion

In this chapter I presented a system that spreads the perception task throughout the lifetime of a robotic agent and maintains a joint symbolic and sub-symbolic belief state for a mobile robot performing pick and place tasks in a human environment. This system allows robot perception to operate in a pervasive manner. Available task and background knowledge play a key role in the successful application of the system. The dynamic belief state can be correctly managed in a timely manner, without hindering robot action, thanks to the knowledge-based filters. The benefits of the approach lie in the way the semantic query capabilities of the robot are extended so that questions not only about what it currently perceives in the current scene are answerable, but also about past percepts.

In the current implementation, the system uses its filters to decide when to run pervasive querying on the logged images (e.g. when several consecutive frames are not processed). Furthermore, there is no ranking of the perception plans added to the queue. As future prospect, the episodic memories could be used to analyze the performed tasks in order to estimate how much time certain robot operations might take, in order to better manage resources and to schedule tasks in a less invasive manner. Queries could also be ranked depending on how much information there is to gain from executing them, allowing for sorting and merging of pending perception tasks.

An interesting part of the proposed system, which we will look at more closely in the next chapter, is the spawning and maintenance of the belief state in the Gazebo physics simulator (Koenig and Howard, 2004). Although in the current chapter it is used purely as a visualization platform for the global belief state (images in the figures were generated from Gazebo), it served as an entry point for

future research on the possibilities that arise from having a constant connection between how a scene looks like and how a robot "imagines" it to look like. Recent advances in 3D engines that aim at creating more realistic renderings (Weichao et al., 2017) allow for off-screen renderings of the belief state to be used for adapting perception components of robotic agents.

CHAPTER 6

Variations of Episodic Memories for Prospecting

Roadmaps for the future development of robotics (US Roadmap, 2016; Bischoff and Guhl, 2009) present a variety of different application tasks, in a variety of different environments and with a variety of different robots. One of the big challenges raised by these technology roadmaps is the question of how the necessary programming for all different combinations of robots, tasks and environment can be accomplished. Being inspired by human learning, a promising approach is to enable robots to transfer knowledge between tasks. Humans are impressively capable at doing this, that is, we identify relevant knowledge from previous tasks, transfer this knowledge to new tasks and do adaptations, if necessary.

Adapting the object recognition and detection modules of mobile robots to new objects or new environments can be a time consuming task. In order to alleviate this process, in this chapter **I propose enabling robots with the ability to relive variation of past experiences and automatically adapt their object detection and recognition capabilities through this process.** The proposed ability can be seen as a way towards achieving prospection. Prospection is described by Vernon (2014) as one of four forms of *internal simulation* that are directly related to key roles of memory in cognition and he defines it as:

> “ the brain’s ability to experience the future by simulating what it might be like. ”
>
> *Artificial Cognitive Systems, David Vernon*

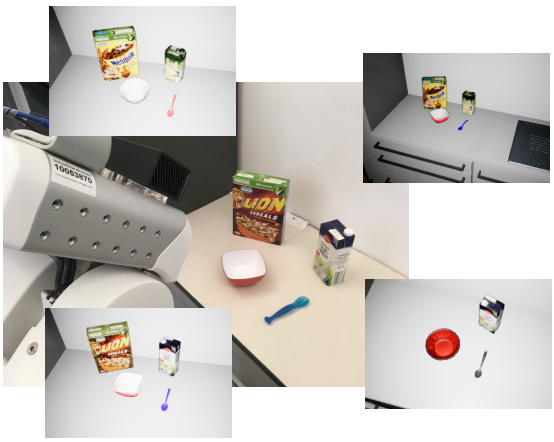


Figure 6.1: Real scene and variations of it rendered using a game engine

This means that achieving prospection entails the realization of an internal simulation process, which requires mechanisms for perceptual imagery (Vernon et al., 2015). Furthermore, Vernon describes a strong link between episodic memories and prospection, that is, when re-living past experiences, cognitive agents apply slight variations to these. It is this latter observation that forms the basis of this chapter. By enabling a robotic agent to relive the past experiences and simulate possible future variations of these its object recognition and detection modules are adapted such that queries about unseen objects and constellations thereof are answerable.

Generating the perceptual imagery that depicts how future scenes might look like is realized with the use of state-of-the-art simulators that are capable of generating high-quality image data. The constellation of scenes that serve as the source for generating these images are taken from episodic memories of a robotic agent, gathered during the execution of a task. Variations of the task and the scenes perceived are generated based on background knowledge about the objects, and data is gathered with the purpose of learning new models for detection and recognition. For practical purposes we can also manually define how a scene should look like. In this case the object constellations are given by the application developer. In both cases the images generated represent possible future states of a world the robot is operating in. Example images that are variations of what the robot currently sees are exemplified in Figure 6.1.

In this chapter I will detail how the episodic memories are used to render new scenes and use these to improve its object recognition and detection modules. Specifically, memories of a PR2 robot performing pick-and-place tasks in a kitchen are used to automatically generate task-specific training data with ground truth annotation for new environments and new objects. The approach can be summarized as follows: robotic agents recreate scenes from the robot memory as

photo-realistic environments in a game engine. This representation allows for varying the scenes by removing objects, adding others, repositioning and substituting them. In order to generate scenes that are more realistic, background knowledge is used to substitute objects by semantically similar ones. The gathered images are used to train new supervised learning algorithms that are then applicable in the real world. The chapter is based on work originally presented in Bálint-Benczédi and Beetz (2018).

6.1 Motivation

Consider a household service robot that has learned to perform a task in a specific environment and has mastered it to the level that it is ready to be deployed in application-specific scenarios. The task could be anything from the assembly of furniture pieces (Knepper et al., 2013) to table setting in a home (Beetz et al., 2011). The question immediately arises: how do we ensure that the robot copes with the wide variety of objects it will encounter? In most cases, even for small changes in an object's visual appearance or in that of the environment, the perception routines have to be adapted in order to compensate for these. This is not desired since, depending on the perception component, such a process can take a prolonged period of time. One possible way of addressing the complexity introduced by a high number of classes is through recognizing object affordances (Varadarajan and Vincze, 2012) based on various geometric features and functional parts of objects. We can go as far as learning how to substitute these objects with similar ones (Abelha et al., 2016), based on a task we want to execute.

Recent advances in computational power have enabled researchers to use simulation and 3D rendering more prominently to teach robots about objects and their appearance. Two related approaches are presented by Mozos et al. (2011) and Wohlkinger et al. (2012). In both of these the authors use CAD models from the world wide web and 3D renderings of these in order to learn to distinguish between geometric classes of objects. Another interesting approach is taken by Abelha and Guerin (2017), where they use the Gazebo simulator in order to find substitute objects for certain manipulation tasks.

Another way to address the issue of transferring perception from one task to

another is through collecting massive databases of images and applying learning algorithms to disambiguate between a large number of object classes (Deng et al., 2009). Although these latter approaches are promising and have made huge strides towards human-level recognition rates, they obviously do not address the full spectrum of problems that robot perception is facing. Unfortunately, there is a discrepancy between the data that is used to train these classifiers and the images that robots see. We have seen in previous chapters and recent studies have also shown (Durner et al., 2017; Reza Loghmani et al., 2018) that there is a considerable drop in the performance of learning algorithms when deployed on mobile robotic platforms. The problem lies in the fact that a lot of the available datasets are biased towards how we as humans take photographs (in focus, object of interest centered, clearly visible, etc.), whereas in case of a mobile robotic platform the visual data is often out of focus, objects are seldom positioned in the center of view, can be cut off at the edges, etc.

In this chapter I aim at combining the two latter approaches. Recent advances in game engine technologies have made it possible to render photo-realistic images of an environment, and this realism of the game environments is becoming better and better. Weichao et al. (2017) introduced UnrealCV, a plugin for the Unreal game engine that allows easy collection of ground truth data and realistic renderings of the world. The plugin returns off-screen renders of the RGB, depth and object mask images, such that it is straightforward to find objects in the raw data. In addition, as part of the RobCoG project (Haidu et al., 2018) the necessary software tools for grabbing images and interfacing with ROS are continuously developed. Through the combination of these technologies, the perceptual episodic memories gathered through the mechanisms presented in Chapter 4 are offloaded to a game engine and scenes that are similar to how robots see the world are generated and used to adapt the internal models of the agent to recognize novel objects that it has never encountered in the real world before.

6.2 System Overview

An overview of the steps the proposed approach consists of is depicted in Figure 6.2. First, perceptual episodic memories are collected using a mobile robotic agent while

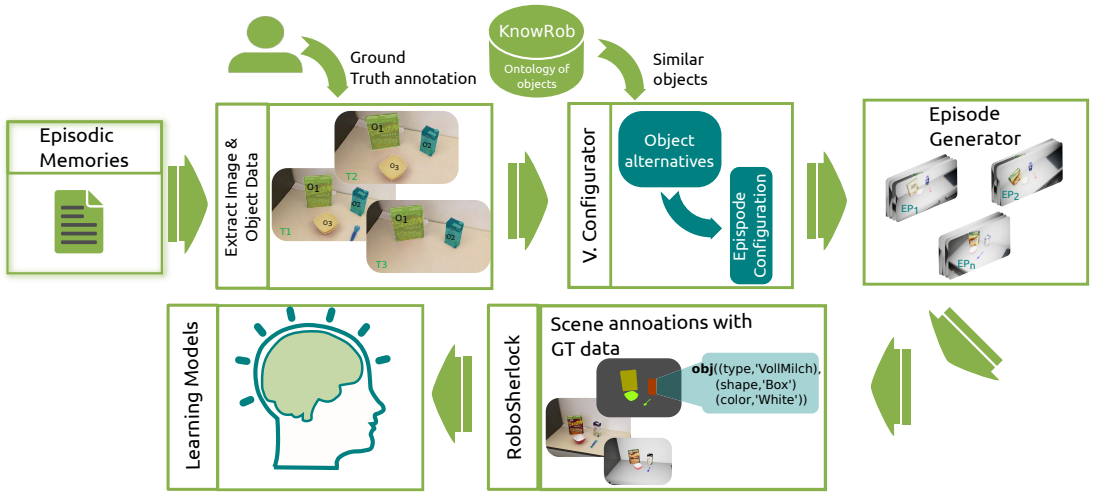


Figure 6.2: Overview of the system architecture generating variations of episodic memories, showing main components and the flow of data between them

it is performing a task. This is achieved using RoboSherlock and the storage component detailed in Chapter 4. The perceptual episodic memories consist of the images, the objects found in these images and the metadata associated with these. It is important that during task execution a perception pipeline that asserts symbolic knowledge about the world is run. Using a visualization tool, as the likes of openEASE (Beetz et al., 2015c) or the one presented in Chapter 4, these results are inspected and erroneous results corrected. Generally speaking this correction can happen in multiple ways. Durner et al. (2017) for instance present a system which can perform semi-automatic labeling of robot data, where the system prompts the user for validating its beliefs after execution. Mechanisms like this can easily be integrated into a robotic system's control framework. Once the logs are corrected, the system looks at the objects that the specific task involved, finds alternative replacements for them and generates a set of variation configurations. Based on these configurations, images are off-screen rendered in order to generate variations of the original episodic memory. These variations are logged using the same methodology as with the real robot, but this time the ground truth is "for free", the labels coming from the models that were loaded in the game engine. Once the data has been generated, supervised learning techniques are applied in order to improve or adapt the robotic agent's perception algorithms. Each of these components will be presented now in detail.

6.2.1 Logging and Correcting Logs

Recording the perceptual episodic memories of mobile robots during the execution of various tasks has been presented in Chapter 4. Let us briefly recapitulate what was introduced. The RoboSherlock framework offers a web based inspection interface with a semantic query language that allows easy retrieval and inspection of perception results. The perceptual episodic memories contain all the information that is necessary to replay perception events that occurred while performing a task, and are indexed based on a global timestamp. This information includes robot positions, raw images, features that were extracted, objects that were classified, regions of interest where objects were found, etc. The metadata about each image is stored in a collection called *scenes*. A scene consists of an array of object hypotheses, a list of believed objects in the raw data at a particular timestamp. Each object hypothesis has a list of annotations storing symbolic and numeric information.

The web-based inspection interface allows the addition of ground truth data to the list of annotations and the possibility to correct this ground truth in the case of automatic labeling. In this case, under ground truth data, unique labels for each individual instance of an object that was found during task execution are understood. As we will see later it is important that these labels are grounded in a symbolic representation or at least can be easily mapped to a symbol from the knowledge base.

The episodes are stored in a schema-less database (MongoDB), in a way that it is easy to serialize and de-serialize from native RoboSherlock data structures to these.

6.2.2 Generating New Variations

In order to generate new variations of a task the set of objects that were used during the task have to be retrieved. If the robot was maintaining a belief state about the objects, this is straight forwards. Otherwise, the corrected ground truth labels are used and the list of objects is created manually.

The first step in generating the variations is to find a set of similar objects that can be used to replace the ones in the recorded memory. Representing objects

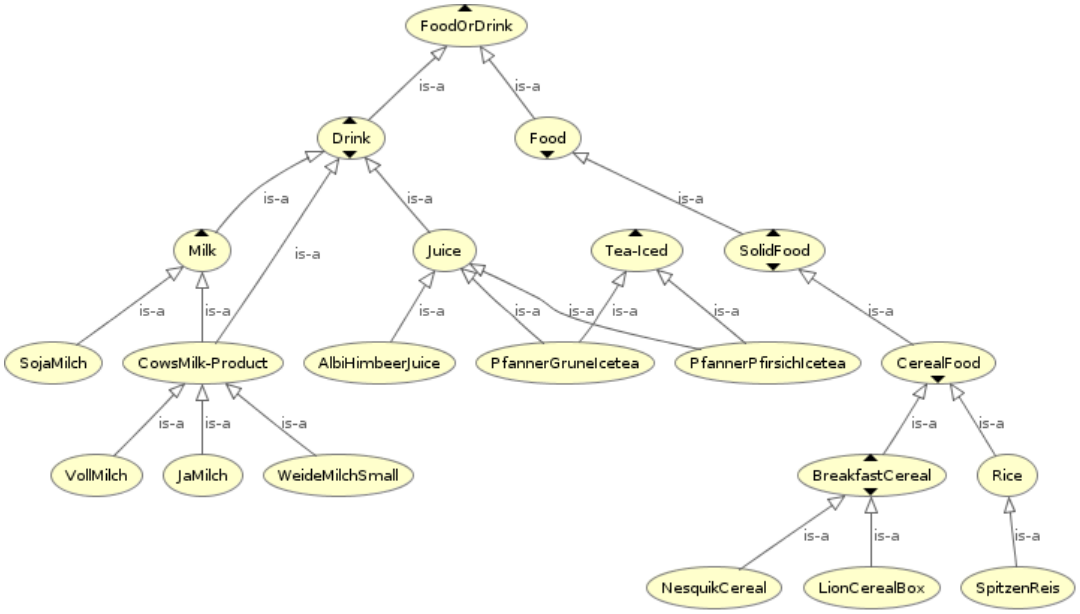


Figure 6.3: Part of the object ontology used in our experiments

in a taxonomy is beneficial in this case. Since in RoboSherlock objects are represented in the ontologies defined in KnowRob (Tenorth and Beetz, 2013), as it was presented in Chapter 3, similar objects are found by searching these. An example of some of the objects from the ontology is presented in Figure 6.3. Generating replacement objects based on the ontology ensures that the variations are more realistic, since they are selected based on a semantic similarity measure. The semantic similarity is calculated using the Wu-Palmer Similarity (Wu and Palmer, 1994), more commonly referred to as the *wup* similarity, defined by:

$$wup(a,b) = \frac{2 * depth(lcs(a,b))}{depth(a) + depth(b)}$$

where *lcs* stands for the lowest common subsumer of a and b. Based on this similarity, a pool of objects is retrieved that are good candidates for replacing the ones from the episodic memory. Choosing *wup* similarity instead of alternative semantic similarity measures, such as shortest path, ensures that objects that are deeper in the ontology (under the same *lcs*) are more similar to each other. Table 6.1 shows the *wup* distances for the objects from the part of the ontology shown in Figure 6.3. Depending on the similarity threshold chosen, greater or lower variety of scenes can be allowed. For instance, 'DairyMilk' products can

Object pair	Distance
wup(JaMilch, WeideMilchSmall)	0.873
wup(SojaMilch, WeideMilchSmall)	0.823
wup(AlbiHimbeerJuice, PfannerGruneIcetea)	0.875
wup(JaMilch, PfannerGruneIcetea)	0.75
wup(SojaMilch, PfannerGruneIcetea)	0.75
wup(JaMilch, LionCerealBox)	0.636
wup(LionCerealBox,SpitzenRice)	0.75

Table 6.1: *wup* distance examples from the knowledge based

be replaceable by soy milk, but not by juices. In the experiments conducted, the threshold has been chosen such that only the closest object alternatives are considered.

Algorithm 3: Generating a set of variation configurations

```
Data: SeedEpisodes, Ontology, nrOfVars
Result: variation = []
for episode ∈ SeedEpisodes do
    obj-list ← getListOfObjectsFromEpisode();
    obj-alternatives = ;
    for obj ∈ obj-list do
        obj-alt = getObjectAlternatives(Ontology);
        obj-alternatives[obj] = obj-alt
    end
    idx = 0;
    while nrOfvars > idx do
        current-config = ;
        for obj ∈ obj-alts do
            current-config[obj] = randomSamp(obj-alts[obj]);
        end
        Variations[idx++] = current-alternatives
    end
end
```

When generating a new episode configuration, new objects are randomly sampled out of the possible alternatives, with the additional possibility that objects can also be removed. This is shown in Algorithm 3.

6.2.3 Generating New Episodes

The actual generation of the new episodes based on the variation configurations happens using the game engine utilized by the RobCoG (Haidu et al., 2018) project and RoboSherlock. The objective of RobCoG is to equip robots with commonsense and naive physics knowledge using games with a purpose. In the games, users are asked to execute various mundane tasks in different scenarios. One of these scenarios is a realistic kitchen environment with objects of daily use, which makes it a perfect fit for the tasks investigated in this chapter. RobCoG offers extensions to communicate through ROS[1] and also offers a wrapper for UnrealCV[2], in order to stream images over the network. These two extensions are essential for the generation of the variations using RoboSherlock.

The episode generation is partly implemented in RoboSherlock and partly as a plugin for the RobCoG system. Modules in RoboSherlock are responsible for generating the variation configurations described in the previous subsection, for reading the data from episodic memories and communicating these to the generator module. The data consist of positions of objects, and position of the camera. In order to increase the diversity of the variations, a small amount of noise is added to the poses of objects. Additionally, random positions of the camera are also generated. Based on the location of the robot's camera from the logs, a set of possible locations on the surface of a sphere that is defined by the center of supporting plane and camera location are generated. Each time the camera position is sent to the simulator, a location is randomly sampled from this sphere.

The plugin developed for the RobCoG system, offers basic interfaces for spawning and moving objects and the virtual camera. These interfaces are implemented as standard ROS communication protocols. The images from the virtual camera are streamed back to the RoboSherlock framework, and are written to the CAS using a *Collection Reader* specifically created for this purpose. Logging these variations happens exactly the same way as it does with real data, except that in this case the ground truth encoded in a mask image is additionally stored.

[1] `https://github.com/robcog-iai/UROSBridge`
[2] `https://github.com/robcog-iai/URoboVIsion`

Once it is consider that enough variations of a scene were generated, the newly logged rendered images can be retrieved and supervised learning techniques applied to them in order to adapt the robotic agent's object recognition and detection methods.

6.3 Experimental Analysis

To analyze the benefits of prospection, I examine two distinct cases. First, I look at how the episodic memories enable the adaptation of perception skills to new objects. Second, I will present how the proposed extensions can be used to generate images of manually defined scenes and learn probabilistic models from them.

6.3.1 Learning from Episodic Memories

In order to evaluate the proposed approach, two $(E1, E3)$ plus two $(E2, E4)$ episodic memories of a PR2 robot picking and placing objects from one tabletop to another were collected. The episodes contain 18 different instances of objects with class labels and segmentation masks as ground truth. Data was acquired using the Kinect Xbox v1 sensor that is the standard sensing device on the PR2. Episodes $E3$ and $E4$ are real world variations of $E1$ and $E2$, respectively. On average there are 35 scenes per episode containing around four objects per scene (during the task itself, objects disappear and reappear in consecutive scenes). To simplify the collection of data, picking and placing of objects was performed by a human. The robot was localized using adaptive Monte-Carlo localization[3] in a previously acquired 2D map of the environment. Taking the original two episodes as source, the recognition models of the robot are adapted to detect and recognize the objects in the real-world variations. Specifically two perception techniques that are popular in robotics are detailed:

- recognition of objects on tabletop scenes, using classifiers trained from turntable data,
- semantic segmentation of the scenes using convolutional neural networks.

[3] `wiki.ros.org/amcl`

(a) Episode 1 **(b)** Episode 2

Figure 6.4: Pool of objects generated from the source episodes

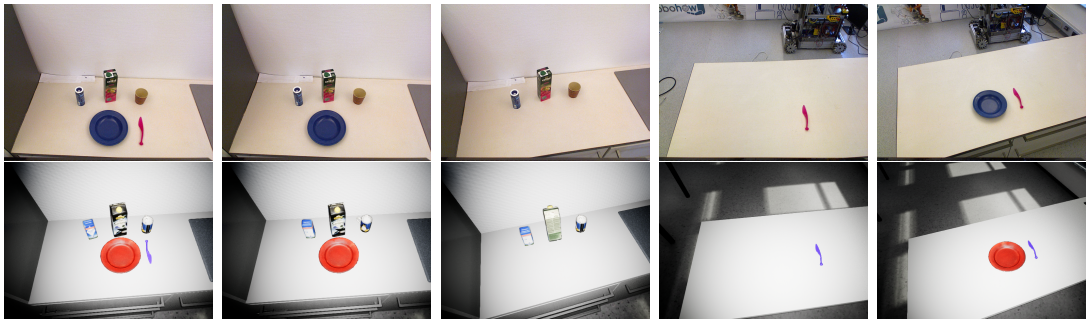


Figure 6.5: Pick-and-place task performed by the robot: view from camera mounted on the robot (1st row), variation of the task rendered from the belief state in Unreal (2nd row)

For both of these experiments variations of the seed episodes need to be generated. $E1$ and $E2$ contain in total nine objects of various shapes and visual characteristics. These objects are all represented in the KnowRob ontology. Performing a similarity search based on the *wup* distances, conditioned on the existence of CAD models, yields a list of 31 candidate objects (this includes the original nine from $E1$ and $E2$). A list of the alternative objects is shown in Table 6.2 and their respective CAD models rendered using Unreal are visible in Figure 6.4.

A variation consists of randomly picking an alternative for each object from the episode with the option that objects can also be missing from the scene. A total number of 100 variations (50 for each episode) were generated based on $E1$ and $E2$, keeping $E3$ and $E4$ for testing purposes. This results in a total of 3887 scenes containing 8970 views of the objects. One such variation is shown in Figure 6.5.

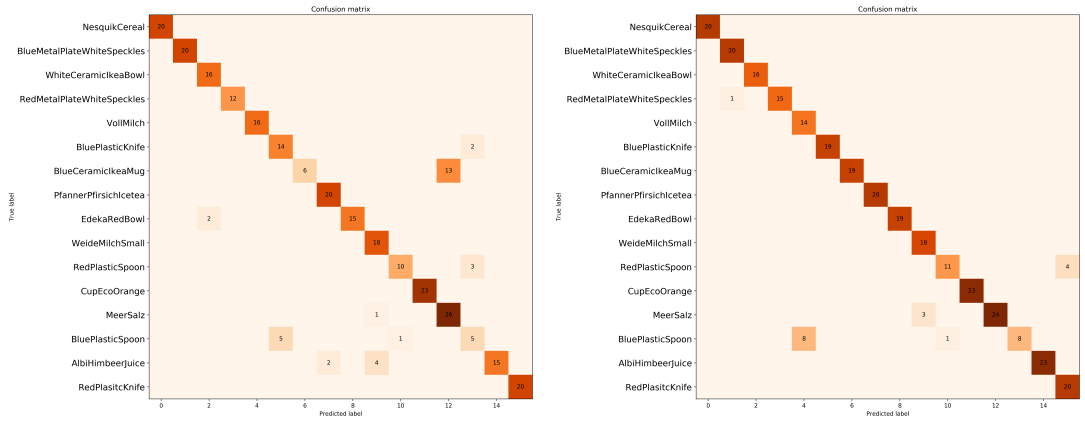
Object in E[1-2]	Alternatives
EdekaRedBowl	WhiteCeramikIkeaBowl RedMetalBowlWhiteSpeckles
VollMilch	WeideMilchSmall JaMilch SojaMilch
BluePlasticSpoon	RedPlasticSpoon LargeGreySpoon
LionCerealBox	NesquikCereal KelloggsToppasMini KoellnKnusperHonig KnusperSchokoNuss KelloggsCornFlakes
AlbiHimbeerJuice	PfannerGruneIcetea PfannerPfirsichIcetea
BlueMetalicPlateWhiteSpeckles	YellowCeramicPlate RedMetalPlateWhiteSpeckles
RedPlasticKnife	BluePlasticKnife
CupEcoOrange	BlueIkeaMug LinuxCup RedMetalCupWhiteSpeckles
MeerSalz	JodSalz MarkenSalz

Table 6.2: Objects from $E1$ one and $E2$ and their alternatives

Object Classification

One of the most common approaches for recognizing objects in robot perception is a bottom-up approach where first candidate objects are generated using segmentation algorithms, feature descriptors are extracted for these object candidates and finally based on these descriptors class labels are attached to the candidate objects (Rusu and Cousins, 2011). For this experiment the second step of this process is of interest (hence the non-cluttered scenes), more precisely, how does the same classifier behave if we train it with data obtained solely from the variations compared to data obtained with the help of an RGB-D sensor and a turntable.

To be able to compare results, partial views of the objects from Episodes [1-4] used in the experiments have been collected using a similar setup to that of the



(a) Trained on data from variations **(b)** Trained on turntable data

Figure 6.6: Confusion matrices of k-NN trained on different datasets, tested on data from episodes

Universityof Washington RGB-D dataset Lai et al. (2011). Data was recorded using an Asus Xtion RGB-D sensor positioned at around one meter distance from a turntable at three different angles. The angular step of the turntable was set to 5 degrees. Description of the data and its acquisition are detailed in Appendix B.

For comparison, two k-NN classifiers were used with $DeCAF_7$ features trained on ImageNet2012 (Donahue et al., 2014). One, using the data from the turntable, and another one, using the 8970 views of objects rendered from unreal. Testing in both cases was done on the partial views of the 18 objects from the four episodes collected from the robot. Performance metrics of the classifiers are reported in Table 6.3 and the confusion matrices shown in Figure 6.6. While results using only the data from the variations are not as good as the ones achieved on the real data, they are within a reasonable margin. These results could further be improved by using different features, better classification schemes or adding noise to the artificial data, compensating for the difference in color space. The results demonstrate that training robotic agents with the data resulting from the variations alone is feasible and by fine-tuning parameters better performance could be achieved. Note that the metrics and confusion matrices are calculated for 16 object instances, since for two of the objects (‘LionCerealBox‘ and ‘JodSalz‘) due to the bad quality of the CAD models none of their instances in the episodes got recognized, which highlights the necessity of high quality CAD models for this approach to be applicable.

Data	Precision	Recall	Accuracy	F-Score
Turntable	0.96%	0.94%	0.94%	0.94%
Unreal Variations	0.90%	0.83%	0.83%	0.84%

Table 6.3: Performance metrics of the k-NN classifier using data collected from an Asus Xtion RGB-D camera and a turntable and the data from the variations in Unreal

Semantic Segmentation

In a second experiment I investigate how end-to-end learning on the rendered images can be applied to real world robot data. In recent years deep learning has been dominating the trends in computer vision research. One approach favored by many is called YOLO (Redmon and Farhadi, 2016), because of its simplicity, good results on standard datasets and most prominently, for its fast inferencing. This latter makes it applicable in the close to real-time demands of mobile robotics. YOLO is a semantic segmentation algorithm, so it infers both bounding boxes and semantic labels at the same time.

To retrain the network, the initial weights were took from a model that was trained on ImageNet (Deng et al., 2009) and is supplied with the framework. The data generated through the variations was split in two: 80 variations are used for training, while 20 are used for testing. In addition to these, several experiments in varied configurations of the training and testing data using the four episodes from the robot were conducted.

The results of these different configurations in terms of IoU (intersection of union) and recall are reported in Table 6.4. Since this study server as a proof of concept, the amount of training data was kept low. For this reason the same metrics are reported at different epochs, in order to illustrate that after a certain number of iterations the network simply over fits the model to the data. The general trend is that both IoU and recall decrease as the number of epochs decreases. We observe that adding the two episodes used as seeds for the variations to the training data, increased the performance of the system. This is expected since the average mean of the images, calculated during the training process, will take into consideration the color space of the real images as well.

The mean average precision per object in the test data is reported in Table 6.5. In the case of the first test the model was trained with 80 unreal variations and

Training Data	Testing Data	Epochs	IoU	Recall
80 Var	20 Var	45k	89.17 %	98.86 %
		20k	86.11 %	99.26 %
		10k	85.39 %	99.13 %
		1k	71.30 %	91.32 %
	4 Robot Episodes	45k	80.83 %	99.42 %
		20k	80.41 %	97.98 %
		10k	78.16 %	97.69 %
		1k	67.88 %	88.76 %
80 Var + 2 RE	20 Var	45k	89.68 %	98.82%
		20k	88.39 %	99.13 %
		10k	85.82 %	98.99 %
		1k	79.11 %	97.18 %
	2 Robot Episodes	45k	84.47 %	99.37 %
		20k	83.85 %	100 %
		10k	81.44 %	100 %
		1k	74.6 %	98.1 %

Table 6.4: IoU and Recall of YOLO trained with different data sets: Variations of the original two scenes (Var), and variations combined with Robot Episodes (RE)

tested with the four robot variations. During the second test, the model was trained with 80 unreal variation plus episodes one and two and tested on $E3$ and $E4$. In both cases the same behavior is observable. Although the good mAP scores (mean average precision) using only the training data from unreal are not negligible, performance metrics improve when using real images during the training process.

These two experiments clearly show that it is not only possible to use the photo-realistic renderings from modern game engines to (re-)train perceptual models of robotic agents, but that it is possible to achieve results that compete with real-world datasets.

6.3.2 Probabilistic Scene Understanding

As a last experiment I investigate how an ensembles of expert system can be trained to recognize objects using a manually defined set of scenes and rendered images

Object	Test1 ap [%]	Test2 ap [%]
BlueCeramikIkeaMug	98.27 %	100 %
BluePlasticKnife	100 %	100 %
JodSalz	69.7 %	100 %
NesquikCereal	100 %	100 %
PfannerPfirsichIcetea	99.57 %	100 %
RedMetalicPlateWhiteSpeckles	100 %	100 %
RedPlasticSpoon	77.92 %	98.93%
WeideMilchSmall	100%	100%
WhiteCeramikIkeaBowl	98.4 %	100%
mAP	**93.76%**	**99.88%**

Table 6.5: Average precision of objects after 10k epochs for two test cases.

thereof. We have seen how probabilistic understanding of scenes is of particular interest when presenting the knowledge-based query answering component in Chapter 3, since Primitive Analysis Engines can return often contradictory results. A proposed solution is a CAS Consumer that uses Markov Logic Networks (MLNs), to handle uncertainty and merge results from multiple sources (Section 3.4). As with most supervised learning algorithms, one of the challenges in training an MLN is the availability of good quality data. This is especially the case for the proposed probabilistic approach here. Since it is an ensemble based learning problem, it requires data that has been annotated by the same algorithms that are going to be used during run time, in order for it to capture the behavior of each expert. If an MLN trained on data obtained from expert algorithms that run on synthetic images is transferable to real images, the training (and therefore applicability) of the probabilistic models becomes significantly easier to accomplish.

The experiments presented here were conducted jointly with Dominik Dieckmann (Dieckmann, 2018) as part of his bachelor thesis under my guidance. I start by detailing how the data was generated, followed by an experimental analysis of the MLNs trained using this data. Contrary to the previous two experiments, the MLN is trained to detect object classes instead of object instances. In order to analyze the plausibility of using off-screen rendered images to train the classifiers the performance of the trained model using several combinations of synthetic and

Figure 6.7: Three images of the same scene rendered from different viewpoints

real training and testing data will be analyzed.

Data Generation

Two similar sets of images, depicting typical kitchen scenes were recorded, one using a custom plugin for generating synthetic images from the Unreal game engine and one using the PR2 robot for gathering real images.

Unreal Images Synthetic images were recorded similarly to the experiments before, using custom plugins developed for the Unreal game engine. The same methodology was used as for the variations of episodes, except that the object constellations in the scenes were manually arranged. Using this setup 114 scenes were created, each containing two to five objects and for each scene images were rendered from five different viewpoints. Example of a single scene from different viewpoints is shown in Figure 6.7. Additionally each scene is labeled as one of three types: *breakfast*, *cooking* or *fridge*, following the example from Section 3.4. Scene type is a valuable additional source of information for identifying objects, since objects tend to be grouped together based on the scene they are found in. For example one would not expect to find a toaster in a refrigerator and finding a plate in a scene can signal that there might be cups and cutlery in it, as well.

Real Images In order to test the approach, images of real scenes were also recorded using a PR2 robot operating in a kitchen environment. The objects are the same as the ones used for generating the synthetic scenes. We have arranged 114 distinct scenes and recorder images from three different viewpoints for each. One of the scenes is shown in Figure 6.8. The images of the scenes were manually

Figure 6.8: Example of a real scene

annotated with ground truth using the web tool from Chapter 4.

Training a Markov Logic Network

Annotator	MLN-predicate	Domain
Cluster3DGeometry-Annotator	size(cluster, size)	small, medium, large
PrimitiveShape-Annotator	shape(cluster, shape)	box, round, flat
ClusterColor-HistogramCalculator	color(cluster, color)	red, yellow, green, cyan, blue, magenta, white, black, gray
GogglesAnnotator	goggles_Logo(cluster, logo) goggles_Text(cluster, text) goggles_Product(cluster, product)	variable
RfAnnotator	shape(cluster, shape)	box, cylindrical, disk, flat, sphere, other
SVMAnnotator	instance(cluster, i)	Instance classification
UnrealGTAnnotator	object(cluster, object)	Class name

Table 6.6: Overview of annotators and their respective generated predicate

First the synthetic scenes are used to train and test an MLN. This is done by running a custom perception pipeline on the generated synthetic data. The logical predicates generated by Primitive Analysis Engines (PAEs) are presented in Table 6.6. Most of the annotators used for generating the predicates are the same as the ones presented in Section 3.4 Table 3.2, the only difference being that

LineMod has been replaced by the *SVMAnnoator* for labeling object instances and an additional classifier (*RFAnnotator*) is used to detect the shape of an object (trained on a subset of the RGB-D dataset (Lai et al., 2011) using VFH features (Rusu, 2009)).

Additionally to the predicates in the table, the *scene* predicate is defined, specifying one of the three types of scenes we have defined: $dom(scene) = \{breakfast, cooking, fridge\}$. Ground truth is automatically added with the help of the *UnrealGTAnnotator*, using the *object* predicate. The *object* predicate is defined as follows: $object(cluster, object!)$ and specifies the true type of an objects. The ! operator specifies a functional constraint, specifically that there can always be only one atom that is true for the object predicate. This makes sense, since an object should not belong to two classes at the same time.

The following formulas define the relations between the predicates and object classes:

$$
\begin{aligned}
w_1 \quad & shape(?c, +?sha) \wedge object(?c, +?obj) \\
w_2 \quad & color(?c, +?col) \wedge object(?c, +?obj) \\
w_3 \quad & size(?c, +?size) \wedge object(?c, +?obj) \\
w_4 \quad & instance(?c, +?inst) \wedge object(?c, +?obj) \\
w_5 \quad & goggles\_Logo(?c, +?comp) \wedge object(?c, +?obj) \\
w_6 \quad & goggles\_Text(?c, +?text) \wedge object(?c, +?obj) \\
w_7 \quad & goggles\_Product(?c, +?prod) \wedge object(?c, +?obj) \\
w_8 \quad & scene(+?s) \wedge object(?c, +?obj) \\
w_9 \quad & object(?c1, +?t1) \wedge object(?c2, +?t2) \wedge ?c1 = / = ?c2
\end{aligned}
$$

Training of the MLN is done using the `pracmln` framework[4]. The procedure follows the same setup as in Section 3.4. The MLN is trained using *Discriminative Pseudo-likelihood Learning with Custom Grounding*. The discriminative variant of the learning algorithm is chosen because of its faster learning times. As a requirement, when asking the MLN for results, predicates must either be part

[4] www.pracmln.org

of the evidence or the query. Since classifying the objects is the main goal, the queries contain only the *object* predicate. Thus, for testing, the conditional probability $P(object \mid E)$ needs to be calculated. The evidence E is generated from the scene using the same sequence of primitive AEs, as it was done for training. Inferencing is done using the *WCSP* algorithm. WCSP calculates the *Most Probable Explanation (MPE)*, so it will not calculate the full conditional probability, but it will estimate the most probable solutions, given the evidences. WCSP transforms the instantiated MLN into a *weighted Constraint Satisfaction Problem*.

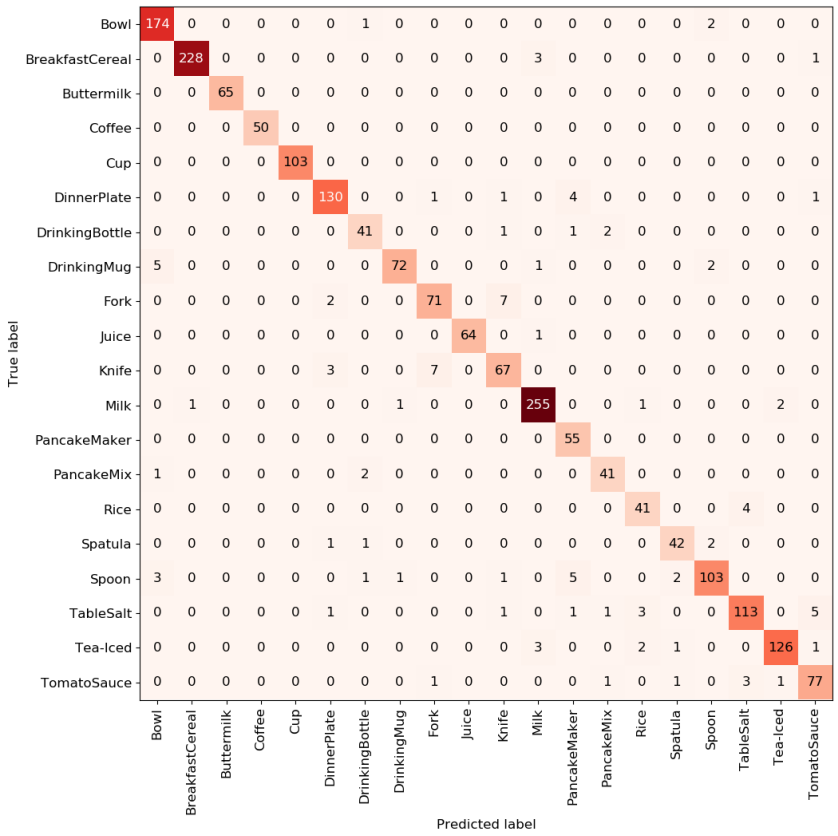


Figure 6.9: Confusion matrix of the ten-fold cross-validation using the synthetic data

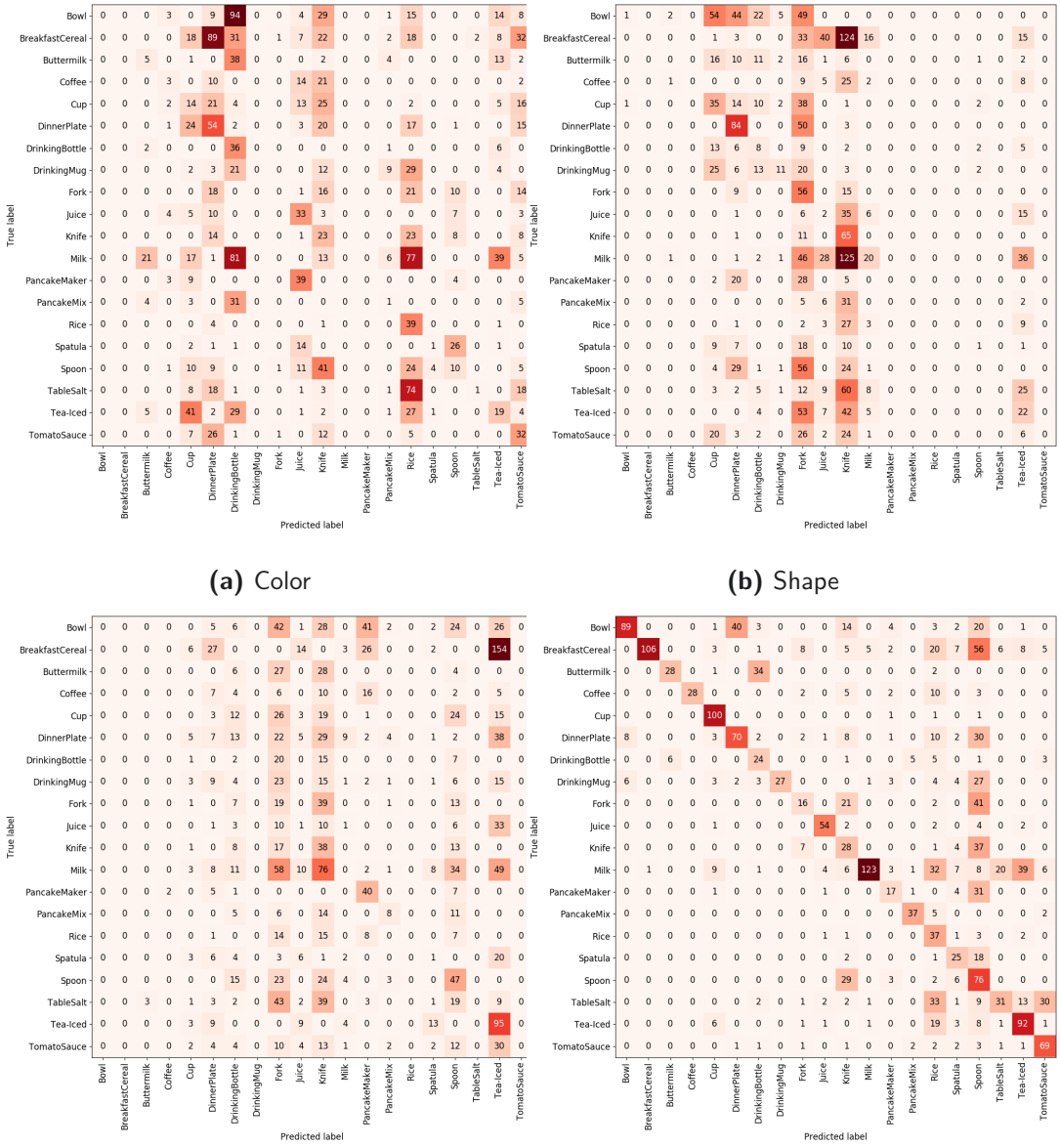
To evaluate the trained model ten-fold cross-validation is performed. Results of the cross-validation are shown in Figure 6.9 and Table 6.7. Overall a high accuracy is achieved, but also values of over 90% for precision, recall and f1-score.

Oject	**accuracy**	**precision**	**recall**	**f1score**
Bowl	0.99	0.95	0.98	0.97
BreakfastCereal	1.0	1.0	0.98	0.99
Buttermilk	1.0	1.0	1.0	1.0
Coffee	1.0	1.0	1.0	1.0
Cup	1.0	1.0	1.0	1.0
DinnerPlate	0.99	0.95	0.95	0.95
DrinkingBottle	1.0	0.89	0.91	0.9
DrinkingMug	0.99	0.97	0.9	0.94
Fork	0.99	0.89	0.89	0.89
Juice	1.0	1.0	0.98	0.99
Knife	0.99	0.86	0.87	0.86
Milk	0.99	0.97	0.98	0.98
PancakeMaker	0.99	0.83	1.0	0.91
PancakeMix	1.0	0.91	0.93	0.92
Rice	0.99	0.87	0.91	0.89
Spatula	1.0	0.91	0.91	0.91
Spoon	0.99	0.94	0.89	0.92
TableSalt	0.99	0.94	0.9	0.92
Tea-Iced	0.99	0.98	0.95	0.96
TomatoSauce	0.99	0.91	0.92	0.91
Total	**0.95**	**0.95**	**0.95**	**0.95**

Table 6.7: Classification results per object class for the ten-fold cross-validation trained and tested on synthetic data

To better illustrate the way the combination of results from multiple expert algorithms improve performance the same ten-fold cross-validation is performed with individual predicates in isolation. Results of these are shown in Figure 6.10. These results were generated by training and testing, using only one of the predicates in addition to the *scene* predicate.

We can notice that for the predicates *color*, *shape* and *size* certain object classes are preferred. The number of preferred classes strongly correlates with the size of the domain for each predicate. For example, there are nine values the predicate *color* can take, and we can notice from the confusion matrix that most objects are classified as one of nine to ten classes. For *shape* and *size* the same can be noticed. Overall these results show how the annotators are complementary and that only through combining their results can we achieve a better output.



(a) Color **(b)** Shape

(c) Size **(d)** Instance

Figure 6.10: Confusion matrices of the ten-fold cross-validation performed on rendered images with each predicate considered in isolation

Experiment	accuracy	precision	recall	f1score
Random Forest	0.94	0.67	0.6	0.57
MLN ten-fold cross-validation	0.95	0.95	0.95	0.95
MLN tested with real images	0.98	0.78	0.76	0.76
Mixed training of MLN	0.98	0.91	0.9	0.9

Table 6.8: Summary of the resulting performance metrics from each experiment

Testing with Real Images The goal of the experiments is to study how perception modules adapted or re-trained using images gathered through prospection behave when put to test on the real robot. As a next step the MLN trained using only the synthetic data is tested using the images of the real scenes. Results of the classification are shown in the confusion matrix from Figure 6.11. The accuracy for all classes is over 90% and the values of precision, recall and f1-score are also mostly above 70%. Results are slightly worse than in the case of ten-fold cross-validation performed on the synthetic data, the classifier confusing some of the object classes. This is to be expected since there is a noticeable difference between the rendered and real images and some of the CAD models of objects used for rendering are either of poor quality or do not resemble the real objects. Nevertheless the results are promising and are comparable to the ones reported in previous work (Nyga et al., 2014) where the experiment was conducted using only real images.

As a last experiment, in order to improve the performance, a part of the real images is used during the training of the MLN. Specifically, one third of the images recorded using the robot were randomly chosen, totaling a number of 684 images in the training set, with a remainder of 228 images for testing.

The results of this last test, as well as all others before are shown in Table 6.8. Accuracy is slightly improved, but more importantly we can notice a significant improvement in all of the other metrics. Most sources of confusion are eliminated through the mixed training set, for example for objects where the 3D model was significantly different from the real object.

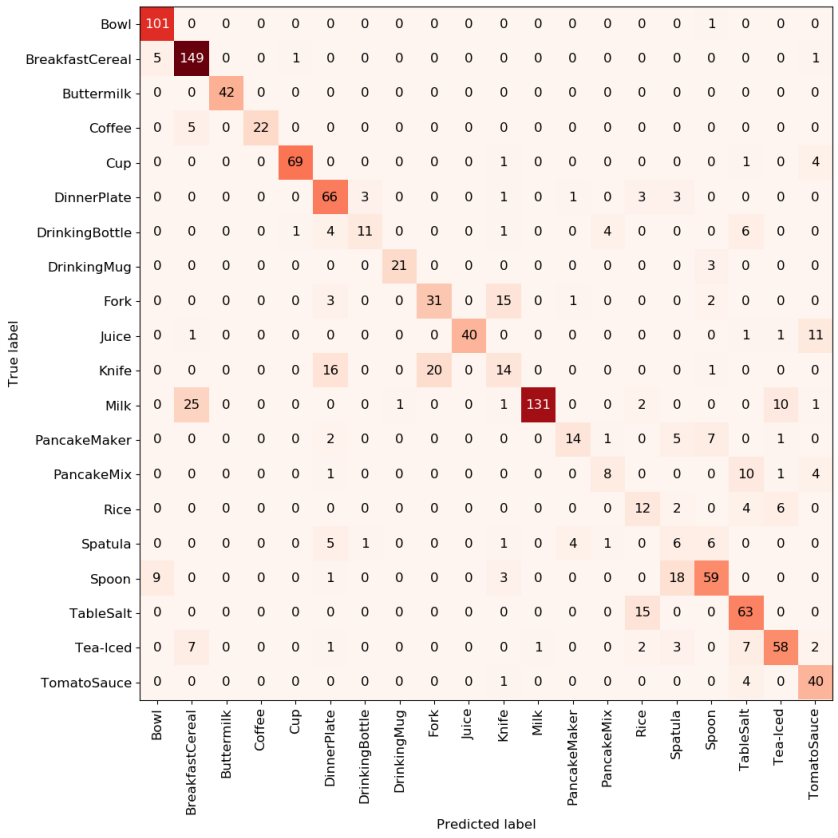


Figure 6.11: Confusion matrix of object classification using an MLN trained on data from generated images and tested on the real images

6.4 Conclusion

In this chapter I have presented a system that enables robotic agents with the capability of *imagining* future possible scenes in order to adapt their object recognition and object detection capabilities. This is achieved by taking the perceptual episodic memories of a robotic agent and off-screen rendering photo-realistic, plausible variations of these. Experiments conducted demonstrate that the data generated through these variations is usable for learning new object detection models that achieve good results on real-world data. The work does not only open new possibilities for training robots how to recognize objects in new scenarios, but can also be considered a first step towards learning common sense knowledge about the objects. Generating vast amounts of data for challenging perceptual

problems where new objects appear or drastically change visual appearance during task execution are a real possibility, being limited only by what is readily available in the game scenarios. Introducing game engines in the way we teach robots about our environment will radically change the way robotic applications are developed.

The prospection capabilities discussed in this chapter are still limited to simple object recognition tasks. There are several improvements that one can identify. Neither were the capabilities of the unreal engine used to its fullest potential, nor were the consequences of different approaches for variations thoroughly investigated. Renderings of the images can be made better, object models made more realistic, different methodologies of how to introduce variations investigated. For instance, how would semantic segmentation perform if instead of the objects we start varying the environment. Common-sense knowledge could be automatically learned by introducing the actions of the robot in the loop, for example, learning the perceptual consequences of actions performed on certain objects (e.g. pouring, opening a container, stacking, unstacking objects, etc.). The contributions of this chapter open up a vast possibility for future research in robotic perception systems.

From a query-answering perspective the generated images are usable as direct input for a perception system created using RoboSherlock. This enables answering any kind of query about the imagined scenes. By running the variation generation live during the operation of the robot question formulated in the proposed query language are answerable. The query language was also extended to give control over the synthetic scenes to the outside users. Example of this will be presented in the concluding chapter.

CHAPTER 7

Conclusion and Future Work

In this thesis I proposed to treat visual perception for robots performing everyday manipulation tasks as an open question-answering problem. In this context visual perception problems that robots encounter during their operation, are formulated as questions that need to be answered and are a function of three major components: the observations, the query and the current beliefs about the world the robot is operating in (Equation 2.1). In order to formulate perception for everyday manipulation tasks as a question-answering problem, a task adaptable, pervasive perception framework was proposed that relies on relevant knowledge about the tasks and the environment to be explicitly represented. To analyze the solutions proposed in the thesis, in this chapter I examine the contributions along two lines of thought: first, how the requirements of perception systems identified in Chapter 1 were met and, second, how these serve the purpose of realizing an open question-answering system.

7.1 Retrospect

In the introductory chapter of the thesis I identified several requirements that robot perception systems should fulfill, if we are to have robots that scale towards complex human-level manipulation tasks. We have already looked at how subsets of these conditions are met in the individual chapters, so let us now summarize these. Perception systems created using the RoboSherlock framework, when used for service robots performing everyday manipulation tasks, are:

R1: ***task adaptable;*** Adaptability of the system to the current perception task is achieved through the combination of multiple modules. First and foremost, during operation, the query interface introduced in Chapter 2 serves as a means for conveying what the high-level control system of the robot expects to perceive. Questions are formulated using the proposed query language that covers a large variety of perception tasks, such as detection, inspection or tracking and is meant to be extensible if needed. The query interface is a key part in creating a task-adaptable system, but it is not the only one. The ensemble of experts approach (R3) is also an integral part, since it allows for the development of specialized perception routines. So is the fact that these experts are represented in a symbolic knowledge base, and as such the system is able to reason about when to execute which expert algorithm (R2).

R2: ***enhance perception with knowledge and reasoning;*** Chapter 3 presented in detail how knowledge representation and reasoning are an integral part of the framework. Description logic is proposed as the means for formal representation of knowledge about the objects, the environment and most importantly the perceptual capabilities of RoboSherlock. Since this representation is not standalone but part of a bigger framework, namely KnowRob, it allows for reasoning not just about objects and algorithms that detect these but also about robot capabilities, semantic locations or object class properties. The mandatory configuration files in the RoboSherlock framework ensure that all perception algorithms that get implemented are represented in the knowledge base and can be reasoned about.

R3: ***equipped with ensembles of experts;*** As we have seen throughout the examples in this thesis, when perceiving the environment in which the robot operates multiple algorithmic solutions, referred to as experts, are needed to perceive different visual characteristics. Since RoboSherlock is based on Unstructured Information Management (UIM), using ensembles of expert approaches is one of the core features of the framework. The proposed probabilistic fusion of results using Markov Logic Networks (Sections 3.4

and 6.3.2) offers a powerful method for combining the results of these expert algorithms. With the recent advances in deep convolutional neural networks more and more specialized object detectors can be trained. The expert based framework presented in this thesis offers a general way for integrating these into robotic perception systems.

R4: ***incorporate beliefs maintained about the world;*** The preparatory amortized perception system, described in Chapter 5, proposes a promising way for incorporating past percepts into the decision making process. The belief state, together with the recording of perceptual episodic memories from Chapter 4 are both meant to ease the development of novel methods that take into account the past percepts of a robot. I have shown three examples of how beliefs are used to enhance perceptual capabilities: the aforementioned preparatory amortized perception, the use of logged memories to adapt perception experts and the use of recorded beliefs to enable prospection by generating possible future scenes and learning from them. These subsystems exemplify the importance of belief states in general and demonstrate their central role in RoboSherlock.

R5: ***modular and extensible;*** Since RoboSherlock follows the processing logic of UIM, it is highly modular. New sensors can be introduced by defining new Collection Readers, any existing algorithm can be wrapped as an Analysis Engine and CAS Consumers can easily be added, extending current functionality. Since the focus of the research presented in the thesis was mostly concerned with the development of components that use results from existing state-of-the-art algorithms the framework was designed such that extending it with new capabilities is straight-forward. The fact that RoboSherlock is implemented as part of the ROS ecosystem makes it easy to interface to and to split into several software modules.

The contributions of the thesis that address these requirements demonstrate the potential that lies in treating robot perception for everyday manipulation tasks on a system's level. As we will see in the discussion about future work (Section 7.3), a lot of the proposed modules in this thesis are enablers for further research in

an integrated manner. Before we look at future prospects though, let us see how the open question-answering capabilities of the RoboSherlock framework are realized.

To examine the question-answering capability of RoboSherlock, recall the scenario from Chapter 1 and the minimum set of questions a robot performing everyday manipulation tasks needs to be able to answer, identified in Table 1.1. The questions there are split in three categories: answering questions about the past, the present and the future. In order to take a closer look at these types of questions and see how they can be formulated using the query language introduced in this thesis let us look at the breakfast scenario described in the first paragraphs of Chapter 1.

Queries about the Past Thanks to the logging infrastructure and the extensions of the query language described in Chapter 4, questions about what the robot has seen can be issued to the perception system before actively searching for an object. When fetching objects in a breakfast scenario this means that the robot has to search less and can rely on what it has seen before to find the objects to be placed on the table. Two features of the log-based query answering extend the types of questions that can be handled: the custom keywords that trigger computations and the knowledge integration. The latter allows for asking the knowledge base for properties of objects (e.g. is the milk that I saw on the table the right type, i.e. whole milk instead of soy). The computation-triggering keywords allow for performing extra checks on the objects that were not computed when the robot has seen them (e.g. is the object the right size, color etc.)

Queries about the Future Perhaps the most interesting aspect of the three types of queries are the ones where a robot is supposed to answer questions about the future. A truly cognitive system can only be achieved if it is capable of anticipating the future states of the world. The prospection capabilities of the framework, presented in Chapter 6, have the purpose of enabling this. The main contributions are focused around rendering scenes off-line and adapting object recognition and detection modules through simulation. Nevertheless the same components are used to enable the spawning of the object belief state in the game

engine environment and render images on demand by issuing queries using an extended query language. For example the query:

```
( render ( a scene
           ( replace #objID1)
           ( cam-pose (stamped-pose (...)))))
```

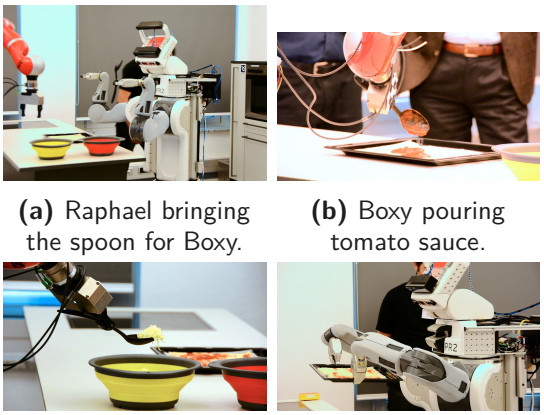
tells the system to render a new image of a scene where one of the objects gets randomly replaced and the virtual camera position is given. This example also illustrates how the query language is extensible in order to afford new use cases. As we will later discuss extending the query language for catering for prospection tasks enables promising future possibilities.

Queries about the Present When it comes to robotic applications, queries to the perception framework about what the robot currently sees are by far the most frequent. These questions describe what the robotic agent is currently expected to perceive and the answers enable the robot to manipulate the objects and environment in which it is operating. The proposed query language from Section 2.3.2 allows for a wide range of queries to be formulated, from detecting functional parts of objects (e.g. lid of a contained) to tracking an object while a manipulation action is performed (e.g. track the pouring of a liquid). The descriptiveness of the language lets the high-level control system specify the results that it wants to receive, reducing the number of false positives in the process. The proposed query-answering interface is not without its downside. Descriptiveness and flexibility come at the price of being more complex than simple static interfaces, such as ROS services or actions. Many of the perception queries used during current robot manipulation task are simple descriptions of the object that needs to be manipulated and as such they could easily be replaced by static interface definition. However, some of the objects, object parts or scenes are more challenging and this is where having a more complex query interface is a clear benefit since describing what the perception system should look for introduces valuable background knowledge that simplifies the perception task. Since the perception tasks are of various difficulties and the goal is to create a perception system that can be deployed in diverse environments, declaring

new static interfaces for each use case would quickly become difficult to manage. Instead, extending the language with new terms or simply using the existing terms to describe the problem keeps the interface unified.

7.2 Discussion

The presented capabilities of RoboSherlock have been tested in several application scenarios where robot demonstrations took place, mostly within the context of research projects. RoboSherlock was one out of many other robotics software components (navigation, manipulation, planning, etc.), as part of an integrated cognitive robotic system. These demonstration scenarios were very diverse, and as such, developing the perception component offered a unique opportunity to test the different parts of the framework. The development processes contributed greatly towards the current state of RoboSherlock, helping to identify weaknesses, missing features or software bugs. They were also a great opportunity to present the framework and get valuable feedback. Since a lot of the lessons learned while developing these applications shaped the contents of this thesis, I will now give a brief overview of the most significant projects and how RoboSherlock was used in them.



(a) Raphael bringing the spoon for Boxy.

(b) Boxy pouring tomato sauce.

(c) Boxy bringing cheese to the tray.

(d) Raphael bringing the tray to the oven.

Figure 7.1: RoboHow demonstrator scenario: parts of the pizza preparation experiment as described by Beetz et al. (2016)

RoboHow The *RoboHow* project (Beetz et al., 2016) investigated ways of enabling robots to competently perform everyday human-scale activities in human living and working conditions. During the project, everyday cooking scenarios were investigated (such as pancake making or pizza preparation) as well as a general-purpose fetch-and-place tasks that are necessary for such activities. The use-case in the *RoboHow* project served as the main test scenario for developing the query interface and the first iterations of the

framework. Task adaptability and the need for special purpose algorithms was highlighted during meal preparation tasks, and first versions of object belief state management and pervasive, preparatory perception systems were proposed for the fetch and place tasks. The ideas of the *RoboHow* project are currently investigated as part of a larger effort for understanding everyday activities inside the EASE [1] collaborative research center. At the time of writing, RoboSherlock is the main perception framework used in the project for the demonstrator scenario (setting and cleaning a breakfast table). The use of RoboSherlock in *RoboHow* and EASE greatly influenced the current query-answering capabilities of the system, since most perception algorithms were developed for objects and the kitchen environment. The *RoboHow* project also offered a great opportunity for deploying the framework on different robotic platforms, namely the PR2 and a custom built service robot, called Boxy (Figure 7.1).

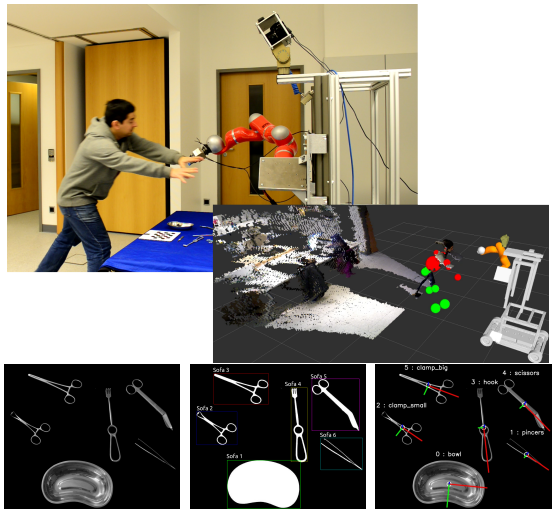


Figure 7.2: Demonstrator scneario of the SAPHARI project (Beetz et al., 2015b)

SAPHARI The *SAPHARI* project (Beetz et al., 2015b) was about safe human robot collaboration. In *SAPHARI* a robot is operating in a hospital scenario and manipulating surgical tools. RoboSherlock was used to detect the surgical utensils that needed to be picked as well as the human operators, signaling the robot's plan if a human is too close to the robot (Figure 7.2). The scenario drove the development of aggregate analysis executors, and highlighted the need for parallel execution of perception pipelines and support for multiple camera systems.

During *SAPHARI* RoboSherlock was also used for investigating ways of integrating human and object perception into a single perception pipeline, experimenting with parallel executions of aggregate analysis engines. The results,

[1] `www.ease-crc.org`

presenting the detection of human intentions and actions, was described in previous work (Worch et al., 2015).

ACAT In the *ACAT* project RoboSherlock enabled a PR2 robot to perform complex manipulation tasks in a chemical laboratory (Lisca et al., 2015). The task of the robot was to perform parts of a DNA extraction process (Figure 7.3). The knowledge-enabled reasoning mechanisms that allowed the detection of various object parts, as well as the core idea of task specific perception experts were essential in the success of this demonstration. The detection of transparent objects such as bottles or test tubes highlighted the need for several specialized perception experts that work together in an ensemble. We have seen several examples of perception tasks from this scenario in Section 3.5. The ensembles of expert approach that RoboSherlock builds on and the knowledge-based reasoning mechanisms of the framework greatly eased the development process. A good example of this was the detection of openings of different sized containers, through a special purpose primitive analysis engine that, depending on the inspected object type, parameterized its detectors accordingly.

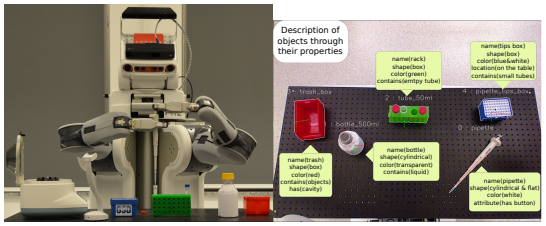


Figure 7.3: PR2 pipetting as part of a DNA extraction sequence

SHERPA During the final presentation of the *SHERPA* project a simulation-based rescue scenario was proposed (Yazdani et al., 2018). As part of this scenario the first prototype for processing images generated in real time by a game engine was developed and integrated into RoboSherlock. This prototype served as a starting point for enabling the system

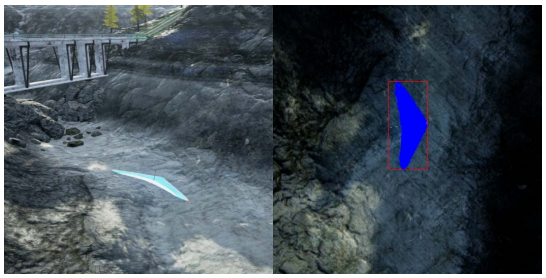


Figure 7.4: Result of a query executed during the SHERPA demonstrator scenario

to interface with game engines and learn from photo realistic renderings of these. As part of the demonstration scenario RoboSherlock was used as the perception engine, using the query interface to detect man made objects in a mountainous are. The detection of a crashed hang-glider is shown in Figure 7.4. By further developing the ideas from this scenario and populating the object belief state in the game engine, answering queries about future states of the world was made possible.

Lessons Learned Through the development of the necessary perception capabilities of the robots in these scenarios valuable feedback was gained as to how the interfacing to other components should be realized. One of the interesting realizations was that a flexible query interface is not always wanted or welcome. This was specifically the case at the beginning, when the query language was still under constant development. This is of course something that hinders integration efforts. To address this a static interface was defined for the most common questions that the robotic applications need (e.g. detect the pose of an object). This static interface is then internally converted into a query in the proposed language. This offers a better, more stable way for the development process, with the added benefit that the query-language-based interface is still there for more complex tasks.

Developing using RoboSherlock comes with the overhead of specifying metadata about the algorithms in configuration files. When developing the perception system for a new application a fair amount of knowledge engineering is necessary. Expert knowledge about the existing algorithms and type definitions is essential so that the correct perception plans are planned for the queries. Currently it is also necessary to have insights about the inner working of the query answering process, since it eases the process of debugging faulty queries. In the future it is envisaged that these complexities are hidden from users of the framework.

Another observation is the importance of reusable algorithms. The Primitive Analysis Engines (PAEs) of RoboSherlock are meant to wrap existing perception algorithms and develop new ones if needed. When developing a PAE it is important to take into consideration which parts of it might be useful for other applications as well and consider splitting implementation into two PAEs. On

the other hand, having too many PAEs can introduce different problems, such as slow pipeline times. One way of reducing the number of PAEs is by exposing parameters of the algorithm that affect its behavior. To illustrate with an example, consider the case of a 3D plane detection algorithm. Setting a parameter to look for horizontal planes allows finding supporting surfaces, but changing this to vertical planes can help find doors or walls. Similarly, when finding the opening of a container we try fitting a circle to the upper part of the container, where the radius of the circle depends on the size of the object we are inspecting and can be retrieved from the knowledge base.

Perhaps the most important observation is the one that is most obvious: when developing a robotic application, to date, even when we consider the constrained domain of operating in the kitchen, every perception task that the robot has to perform needs to be investigated in depth and separately. Robust algorithms that solve these perception tasks depend on a number of conditions that if not met make the application of the specific algorithm unfeasible. Explicitly encoding some of these conditions not only improves the robustness of the perception systems created using RoboSherlock, but also helps non-experts to better understand strengths and weaknesses of the algorithms.

7.3 Future Prospects

The example scenarios show that the ideas presented in this thesis are applicable in a wide variety of tasks. The contributions of the thesis enable some interesting research opportunities that are worth an in-depth investigation. To conclude the thesis I will now outline some of the next steps that I believe would further contribute towards achieving the ultimate goal of having a robotic perception system comparable to that of humans.

In recent years convolutional neural networks have become the standard in some of the problems computer vision addresses. Specifically object classification, semantic segmentation and image captioning systems are nowadays unimaginable without some deep learning parts. Combining the discriminative power of these approaches by using the mechanisms proposed in this thesis and enabling robotic systems to exploit the strengths of the deep learning approaches can greatly boost

the perceptual capabilities of robots. While deep learning is excellent at solving very specific problems, the knowledge-based expert system of RoboSherlock is a perfect fit for meaningfully combining these.

The RoboSherlock perception task language enables researchers to investigate how the perception algorithms of a robotic system behave when instructed to perform different tasks. Combining this with the logging of perceptual memories serves as a starting point for learning generalized perceptual capabilities. Specifically the logged memories allow for replaying of what the robot has seen while it was performing a task and what questions did the high level ask of it. This allows for offline analysis of the methods and ranking of expert algorithms that ran. One of the difficulties when developing a new perception system for a robotic application is the choice of algorithms to use in the case of multiple similar approaches. The same difficulty arises when planning a perception pipeline. Given multiple options, which one should the planner choose, or which result should be trusted. These difficulties are handled with the proposed Markov Logic Network based probabilistic reasoning component, offering a solution for contradictory symbolic values. When it comes to pose estimations, for example, currently it is left at the discretion of the application developer to choose the best approach. The re-execution of perception routines on logged episodic memories and evaluation of these would allow for the modeling of algorithmic behavior and the ranking of expert algorithms based on experience data. The querying of perceptual episodic memories, when taken together with the execution logs enables the retrieval of image data that is specific to a certain type of action. This opens the possibility to train specialized detectors, that work well only when task assumptions are met.

A common way for evaluating a system's or algorithm's performance is done through the use of standard datasets. While this is a good way for finding the best algorithmic solution for a specific perception problem, as of now there are no standard datasets for robotic query answering. Most datasets consist of a collection of images (RGB and depth) or point clouds created with the purpose of evaluating a standard robotic perception approaches: segmentation in cluttered scenes, classification problems, SLAM, pose estimation, etc. As this list also demonstrates, these data were recorded with a specific purpose in mind and as such, they don't contain information about what task the robot is supposed

to execute, what is it supposed to do with the objects. Most of the time the datasets are also biased towards the way we, as humans, see the environment. The experiments from Chapter 4 demonstrate that once deployed on a robotic agent the performance of general object recognition algorithms trained on these datasets drops significantly. With the use of RoboSherlock gathering of large scale realistic perception datasets is possible and would be a worthwhile endeavor.

Perhaps the most interesting of all future prospects is that of enabling the robot system to dream and conjure up new scenarios with the purpose of learning from them. We have seen that models trained in virtual realities are transferable to real world scenarios. Thus, it is reasonable to believe that off-screen rendered images of the belief state can reliably be used to create a truly cognitive architecture where past, present and future beliefs about the environment are all combined together in order for the robot to better understand the world.

APPENDIX A

List of Scientific Publications

Journal Articles and Book Chapters

Ferenc Bálint-Benczédi, Jan-Hendrik Worch, Daniel Nyga, Nico Blodow, Patrick Mania, Zoltán-Csaba Márton and Michael Beetz, "RoboSherlock: Cognition-enabled Robot Perception for Everyday Manipulation Tasks", *In Arxivx.org, revision under review for the International Journal of Robotics Research, 2019*

Jan-Hendrik Worch, Ferenc Bálint-Benczédi, Michael Beetz, "Perception for Everyday Human Robot Interaction", *In KI - Künstliche Intelligenz, Springer Berlin-Heidelberg, vol. 30, no. 1, pp. 21-27, 2015.*

Karol Hausman, Dejan Pangercic, Zoltán-Csaba Márton, Ferenc Bálint--Benczédi, Christian Bersch, Megha Gupta, Gaurav Sukhatme, Michael Beetz, "Interactive Segmentation of Textured and Textureless Objects", *In Handling Uncertainty and Networked Structure in Robot Control, Springer International Publishing, Cham, pp. 237-262, 2015*

Michael Beetz, Ferenc Bálint-Benczédi, Nico Blodow, Christian Kerl, Zoltán-Csaba Márton, Daniel Nyga, Florian Seidel, Thiemo Wiedemeyer, Jan-Hendrik Worch, "RoboSherlock: Unstructured Information Processing Framework for Robotic Perception", *In Handling Uncertainty and Networked*

Structure in Robot Control, Springer International Publishing, Cham, pp. 181-208, 2015.

Zoltán-Csaba Márton, Ferenc Bálint-Benczédi, Oscar Martines Mozos, Nico Blodow, Asako Kanezaki, Lucian-Cosmin Goron, Dejan Pangercic, Michael Beetz, "Part-Based Geometric Categorization and Object Reconstruction in Cluttered Table-Top Scenes", *In Journal of Intelligent and Robotic Systems, Springer Netherlands, pp. 1-22, 2014.*

Zoltán-Csaba Márton, Florian Seidel, Ferenc Bálint-Benczédi, Michael Beetz, "Ensembles of Strong Learners for Multi-cue Classification", *In Pattern Recognition Letters (PRL), Special Issue on Scene Understandings and Behaviours Analysis, 2012.*

Conference Papers

Ferenc Bálint-Benczédi, Michael Beetz, "Amortized Object and Scene Perception for Long-term Robot Manipulation", *In Arxiv.org, 2019*

Jianxiang Feng, Maximilian Durner, Zoltán-Csaba Márton, Ferenc Bálint--Benczédi, Rudolph Triebel, "Introspective Robot Perception using Smoothed Predictions from Bayesian Neural Networks", *In International Symposium on Robotics Research (ISRR), Hanoi, Vietnam, 2019.*

Fereshta Yazdani, Gayane Kazhoyan, Asil Kaan Bozcuoglu, Andrei Haidu, Ferenc Bálint-Benczédi, Daniel Beßler, Mihai Pomarlan, Michael Beetz, "Cognition-enabled Framework for Mixed Human-Robot Rescue Team", *In International Conference on Intelligent Robots and Systems (IROS), Madrid, Spain, 2018.*

Manuel Brucker, Maximilian Durner, Zoltán-Csaba Márton, Ferenc Bálint--Benczédi, Martin Sundermeyer, Rudolph Triebel, "6DoF Pose Estimation for Industrial Manipulation based on Synthetic Data", *In International Symposium on Experimental Robotics (ISER), 2018. accepted for publication*

Ferenc Bálint-Benczédi, Michael Beetz, "Variations on a Theme:'It's a poor sort of memory that only works backwards'", *In International Conference on Intelligent Robots and Systems, 2018.*

Maximilian Durner, Simon Kriegel, Sebastian Riedel, Manuel Brucker, Zoltán Csaba Márton, Ferenc Bálint-Benczédi, Rudolph Triebel, "Experience-based Optimization of Robotic Perception", *In ICAR 2017 - 18th International Conference on Advanced Robotics, IEEE, 2017.* ***Finalist for Best Paper Award***

Ferenc Bálint-Benczédi, Zoltán-Csaba Márton, Maximilian Durner, Michael Beetz, "Storing and Retrieving Perceptual Episodic Memories for Long-term Manipulation Tasks", *In Proceedings of the 2017 IEEE International Conference on Advanced Robotics (ICAR), Hong-Kong, China, 2017.* ***Finalist for Best Paper Award***

Jan Winkler, Ferenc Bálint-Benczédi, Tobias Fromm, Christian- A. MLler, Narunas Vaskevicius, Andreas Birk, Michael Beetz, "Knowledge-Enabled Robotic Agents for Shelf Replenishment in Cluttered Retail Environments", *In Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Singapore, 2016.*

Michael Beetz, Daniel Beßler, Jan Winkler, Jan-H. Worch, Ferenc Bálint--Benczédi, Georg Bartels, Aude Billard, Asil K. Bozcuoglu, Zhou Fang, Nadia Figueroa, Andrei Haidu, Hagen Langer, Alexis Maldonado, Ana-Lucia Pais, Moritz. Tenorth, Thiemo Wiedemeyer, "Open Robotics Research Using Web-based Knowledge Services", *In International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016.*

Ferenc Bálint-Benczédi, Patrick Mania, Michael Beetz, "Scaling Perception Towards Autonomous Object Manipulation — In Knowledge Lies the Power", *In International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 2016.*

Jan Winkler, Ferenc Bálint-Benczédi, Thiemo Wiedemeyer, Michael Beetz, Narunas Vaskevicius, Christian A. Mueller, Tobias Fromm, Andreas Birk,

"Knowledge-Enabled Robotic Agents for Shelf Replenishment in Cluttered Retail Environments: (Extended Abstract)", *In Proceedings of the 2016 International Conference on Autonomous Agents ; Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 1421-1422, 2016*

Gheorghe Lisca, Daniel Nyga, Ferenc Bálint-Benczédi, Hagen Langer, Michael Beetz, "Towards Robots Conducting Chemical Experiments", *In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015.*

Michael Beetz, Georg Bartels, Alin Albu-Schäffer, Ferenc Bálint-Benczédi, Rico Belder, Daniel Beßler, Sami Haddadin, Alexis Maldonado, Nico Mansfeld, Thiemo Wiedemeyer, Roman Weitschat, Jan-Hendrik Worch, "Robotic Agents Capable of Natural and Safe Physical Interaction with Human Coworkers", *In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015.*

Michael Beetz, Ferenc Bálint-Benczédi, Nico Blodow, Daniel Nyga, Thiemo Wiedemeyer, Zoltán-Csaba Márton, "RoboSherlock: Unstructured Information Processing for Robot Perception", *In IEEE International Conference on Robotics and Automation (ICRA), Seattle, Washington, USA, 2015.* ***Best Service Robotics Paper Award***

Thiemo Wiedemeyer, Ferenc Bálint-Benczédi, Michael Beetz, "Pervasive 'Calm' Perception for Autonomous Robotic Agents", *In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagen Systems, ACM, Istanbul, Turkey, 2015.*

Daniel Nyga, Ferenc Bálint-Benczédi, Michael Beetz, "PR2 Looking at Things: Ensemble Learning for Unstructured Information Processing with Markov Logic Networks", *In IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014*

Moritz Tenorth, Stefan Profanter, Ferenc Bálint-Benczédi, Michael Beetz, "Decomposing CAD Models of Objects of Daily Use and Reasoning about their

Functional Parts", *In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo Big Sight, Japan, pp. 5943-5949, 2013*

Karol Hausman, Ferenc Bálint-Benczédi, Dejan Pangercic, Zoltán-Csaba Márton, Ryohei Ueda, Kei Okada, Michael Beetz, "Tracking-based Interactive Segmentation of Textureless Objects", *In IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 2013,* ***Finalist for Best Paper Award****d*

Zoltán-Csaba Márton, Ferenc Bálint-Benczédi, Florian Seidel, Lucian Cosmin Goron, Michael Beetz, "Object Categorization in Clutter using Additive Features and Hashing of Part-graph Descriptors", *In Proceedings of Spatial Cognition (SC), Abbey Kloster Seeon, Germany, 2012.*

Ferenc Bálint-Benczédi, Zoltán-Csaba Márton, Michael Beetz, "Efficient Part-Graph Hashes for Object Categorization", *In 5th International Conference on Cognitive Systems (CogSys), 2012*

Workshop Papers

Ferenc Bálint-Benczédi, Thiemo Wiedemeyer, Moritz Tenorth, Daniel Beßler, Michael Beetz, "A Knowledge-Based Approach to Robotic Perception using Unstructured Information Management", *In Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Istanbul, Turkey, 2015.*

Zoltán-Csaba Márton, Ferenc Bálint-Benczédi, Oscar Martinez Mozos, Dejan Pangercic, Michael Beetz, "Incremental Object Categorization in Clutter", *In 2nd Workshop on Robots in Clutter, in conjunction with RSS2013, Berlin,Germany, 2013.*

Lucian Cosmin Goron, Ferenc Bálint-Benczédi, Zoltán-Csaba Márton, Michael Beetz, "Incremental Modeling and Object Classification based on the Segmentation from Cluttered Settings", *In Workshop on Low Cost 3D Sensors, Algorithms and Application, Berlin,Germany, 2012.*

APPENDIX B

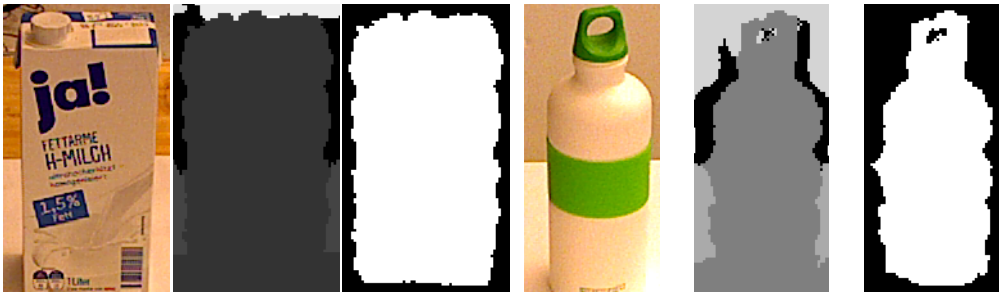
RGB-D dataset

This appendix offers a description of image data consisting of color (RGB) and depth images, collected using RGB-D sensors mounted on a mobile robotic platform that was used in some of the experiments described in this thesis. The data, also made public on the RoboSherlock project website[1], is made up of multi-view images collected using a turntable and kitchen scenes collected with the use of a PR2 robot. The data can be of particular interest for robotics researchers interested in solving perception tasks in the household domain and interested in multi modal learning.

It consists of RGB-D (color and depth) images of objects of daily use, CAD models of these objects, tabletop scenes in a kitchen and a taxonomy of these objects that builds on top of the KnowRob ontology. The dataset is split into four parts, one for each modality. In the following I will describe each of these modalities separately.

Partial views The first part of the data consists of partial views of objects in the form of color and depth images and the respective segmentation masks. Figure B.1 shows examples of these for two of the objects. The partial views consist of 105 object instances, with an average of 200 views per object. The naming convention of these files is as follows: `<object_name>_<view_angle>_<idx>-_<specifier.ext>`. Viewing angle is one of three predefined positions of the

[1] `www.robosherlock.org`



(a) RGB **(b)** Depth **(c)** Mask **(d)** RGB **(e)** Depth **(f)** Mask

Figure B.1: Example images recorded using a turn table for two typical objects

camera (see the Section about acquisition) and `specifier.ext` is one of the following:

- `depthcrop.png` → the cropped depth image
- `crop.png` → the cropped color image
- `mask.png` → the binary segmentation mask
- `loc.txt` → pixel location in the original raw image

In addition, data for some of the objects is marked `_lr`, this is shorthand for low-resolution and its significance will be detailed in the next section.

CAD models CAD models are an important part of the dataset. Although not all objects have a corresponding model, almost all textured objects do have one as well as some of the none textured ones. The Collada (*dae) format was chosen due to its popularity, and not least the possibility to store materials and textures. Currently 48 of the objects that have partial view data also have a CAD model model. The rest of the objects pose difficulties when it comes to scanning them, either because of their small size (e.g. AA batteries) or their visual characteristics (e.g. shiny forks). The dataset homepage offers additional information about the modalities for each data.

Tabletop Scenes Scenes gathered from the robot are an important part of the data. Classifiers trained with data gathered only from the turntable have

(a) Arrangement 1 **(b)** Arrangement 2 **(c)** Arrangement 3

Figure B.2: Examples of table-top scenes from the perspective of the robot

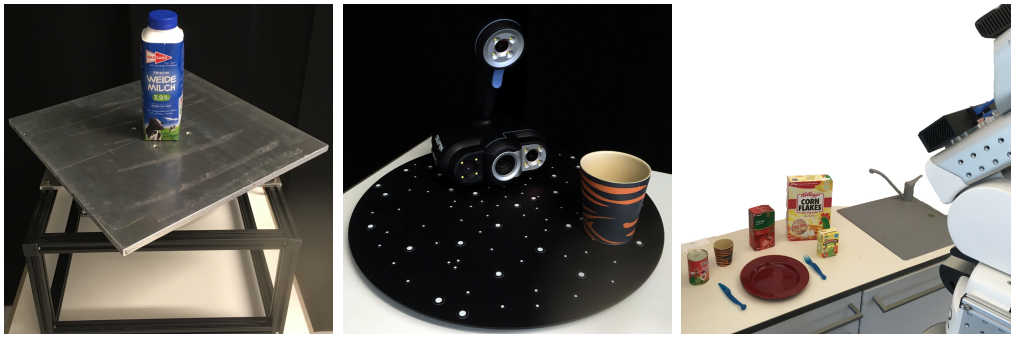
a significant drop in performance when applied to data coming from a mobile platform. Currently seven different scenarios are released. Four of these seven scenarios were recording while performing pick-and-place tasks with a robot. The remaining three are split into distinct breakfast scenes (Figure B.2) with increasing complexity, and contain objects that also have partial views data.

This data has been post processed and table top scene segmentation has been run, in order to extract the objects. In some cases this segmentation is erroneous, and this is intended. Using mobile robotic platforms, one cannot guarantee that objects will be successfully segmented. The naming convention for the data is as follows: `<ground_truth>_<idx>_<specifier>_<ts>.png`, where `ground_truth` is the label, specifier is the same as previously described for partial views and `ts` is the timestamp.

Object Taxonomy The taxonomy of objects is an important part of the data. The names of the objects are grounded in the KnowRob ontologies. Having the objects represented in a ontology allows users to query based on object super types and to easily generate subsets of the data. For instructions on how to install KnowRob and use the ontology, interested readers are kindly referred to the KnowRob project website[2].

Acquisition The different parts of the dataset were acquired with different modalities. Figure B.3 shows the most important hardware systems used during acquisition.

[2] `www.knowrob.org`



(a) Turn table **(b)** CREAFORM GoScan 50 **(c)** PR2 in a kitchen

Figure B.3: The different modalities of acquiring the data

Acquisition of the partial views was done using a custom built turntable (Figure B.3a) and an ASUS Xtion RGB-D camera. The RGB sensor of the camera was set to high-resolution (1280 × 960 pixels), with depth registration turned on. For objects marked with "_lr" the resolution of the RGB image was set to VGA. The acquisition itself was conducted in a similar fashion to the WRGB-D dataset, performing one complete rotation of the object while positioning the camera at around one meter distance from the object, at three different viewing angles: thirty, forty-five and sixty degrees. The big difference is that, while in the WRGB-D dataset the angular resolution is one degrees, in the case of the data presented here a view every five degrees was recorded. It has been shown that the extra views resulting from a recording with a one degree turning do not result in a high information gain, since the resulting images are very similar.

The 3D CAD models were recorded using a commercially available hand held 3D scanner, the GoScan 50 (Figure B.3b). While currently not all objects have a CAD model, the dataset is continuously being updated and new models will be added as they are available.

The scene data was collected using an Xbox Kinect sensor mounted on a PR2 mobile robotic platform. Using the objects from the turn-table data scenes of increasing complexity were set up on a counter top (see Figure B.2 and Figure B.3c) while the robot was driven around, pointing its camera at the scene. The collected images were processed using the RoboSherlock framework in order to extract the objects on the table, and to annotate them with ground truth data.

In order to keep the size of the dataset manageable point cloud data is not

provided directly. Instead python and bash scripts are provided to generate these from the depth and RGB images.

Acronyms

AAE Aggregate Analysis Engine. 27, 35–39, 41, 42, 44, 53, 71–73, 83, 95, 119, 133, 135, 137

AE Analysis Engine. 27, 37, 38, 41, 52, 70, 83, 95, 172

API Application Programming Interface. 100

CA Content Analytics. 24, 26

CAD Computer Aided Design. 85

CAS Common Analysis Structure. 27, 35–42, 44, 46, 49, 59, 75–77, 95, 103, 105–110, 134, 135, 161, 168, 181

CPE Collection Processing Engine. 35, 39, 40

DL Description Logic. 28–30, 64

DSL Domain Specific Language. 11, 103, 104, 106, 115

MLN Markov Logic Network. 75, 78–80, 94–96, 98, 132, 168, 170–172, 175, 189

OWL Web Ontology Language. 64–66, 68, 100

PAE Primitive Analysis Engine. xv, 27, 29, 37, 39, 41, 42, 44, 46–49, 51–53, 64, 66–81, 83, 85, 87, 95, 108, 119, 135, 137, 168, 170, 187, 188

PR2 Personal Robot 2. 14, 73, 118, 130, 142, 154, 162, 169, 185

RDF Relational Data Format. 65, 68
ROS Robot Operating System. 43, 45–47, 54, 156, 160, 161, 181, 183

SLAM Simultaneous Localization and Mapping. 132, 189
SofA Subject of Analysis. 36, 37, 43, 44, 48, 107
SRDL Semantic Robot Description Language. 73
SRL Statistical relational learning. 77, 78
SVM Support Vector Machine. 118, 119, 121, 122, 125, 126

UIM Unstructured Information Management. 9, 19, 24–27, 51, 59, 71, 107, 108, 180, 181
UIMA Unstructured Information Management Architecture. 25, 43–45, 68

References

P. Abelha and F. Guerin. Learning how a tool affords by simulating 3d models from the web. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 4923–4929, 2017. doi: 10.1109/IROS.2017.8206372. URL `https://doi.org/10.1109/IROS.2017.8206372`.

P. Abelha, F. Guerin, and M. Schoeler. A model-based approach to finding substitute tools in 3d vision data. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2471–2478, May 2016. doi: 10.1109/ICRA.2016.7487400.

A. Aldoma, Z.-C. Marton, F. Tombari, W. Wohlkinger, C. Potthast, B. Zeisl, R. B. Rusu, S. Gedikli, and M. Vincze. Tutorial: Point Cloud Library – Three-Dimensional Object Recognition and 6 DoF Pose Estimation. *Robotics & Automation Magazine*, 19(3):80–91, September 2012a.

A. Aldoma, F. Tombari, and M. Vincze. Supervised learning of hidden and non-hidden 0-order affordances and detection in real scenes. In *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, USA, May 14–18 2012b.

T. Arbuckle and M. Beetz. RECIPE - a system for building extensible, run-time configurable, image processing systems. In *Proceedings of Computer Vision and Mobile Robotics (CVMR) Workshop*, pages 91–98, 1998.

F. Bálint-Benczédi and M. Beetz. Variations on a theme: 'it's a poor sort of memory that only works backwards'. In *International Conference on Intelligent Robots and Systems*. IEEE, 2018.

F. Bálint-Benczédi and M. Beetz. Amortized object and scene perception for long-term robot manipulation. In *Arxiv.org*, 2019. preprint.

F. Bálint-Benczédi, P. Mania, and M. Beetz. Scaling perception towards autonomous object manipulation — in knowledge lies the power. In *International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, 2016.

F. Bálint-Benczédi, Z.-C. Marton, M. Durner, and M. Beetz. Storing and retrieving perceptual episodic memories for long-term manipulation tasks. In *Proceedings of the 2017 IEEE International Conference on Advanced Robotics (ICAR)*, Hong-Kong, China, July 2017. Finalist for Best Paper Award.

F. Bálint-Benczédi, J.-H. Worch, D. Nyga, N. Blodow, P. Mania, Z.-C. Márton, and M. Beetz. Robosherlock: Cognition-enabled robot perception for everyday manipulation tasks. In *Arxiv.org*, 2019. preprint, revision under review for the International Journal of Robotics Research.

S. Banker. The autonomous mobile robot market is taking off like a rocket ship, 2019. URL `https://www.forbes.com/sites/stevebanker/2019/03/11/the-autonomous-mobile-robot-market-is-taking-off-like-a-rocket-ship/#1bacbf501603`. Accessed: 2019-06-22.

H. Bannour and C. Hudelot. Towards ontologies for image interpretation and annotation. In *2011 9th International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 211–216, June 2011. doi: 10.1109/CBMI.2011.5972547.

S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL Web Ontology Language Reference, 10 2004.

M. Beetz, N. Blodow, U. Klank, Z. C. Marton, D. Pangercic, and R. B. Rusu. CoP-Man – Perception for Mobile Pick-and-Place in Human Living Environments. In *Proceedings of the 22nd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on Semantic Perception for Mobile Manipulation*, St. Louis, MO, USA, October 11-15 2009. Invited paper.

M. Beetz, U. Klank, A. Maldonado, D. Pangercic, and T. Rühr. Robotic roommates making pancakes - look into perception-manipulation loop. In *IEEE*

International Conference on Robotics and Automation (ICRA), Workshop on Mobile Manipulation: Integrating Perception and Manipulation, pages 529–536, May, 9–13 2011.

M. Beetz, F. Bálint-Benczédi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Marton. RoboSherlock: Unstructured Information Processing for Robot Perception. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA, 2015a. Best Service Robotics Paper Award.

M. Beetz, G. Bartels, A. Albu-Schäffer, F. Bálint-Benczédi, R. Belder, D. Beßler, S. Haddadin, A. Maldonado, N. Mansfeld, T. Wiedemeyer, R. Weitschat, and J.-H. Worch. Robotic Agents Capable of Natural and Safe Physical Interaction with Human Co-workers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015b.

M. Beetz, M. Tenorth, and J. Winkler. Open-EASE – a knowledge processing service for robots and robotics/ai researchers. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA, 2015c. Finalist for the Best Cognitive Robotics Paper Award.

M. Beetz, D. Beßler, J. Winkler, J.-H. Worch, F. Bálint-Benczédi, G. Bartels, A. Billard, A. K. Bozcuoglu, Z. Fang, N. Figueroa, A. Haidu, H. Langer, A. Maldonado, A.-L. Pais, M. Tenorth, and T. Wiedemeyer. Open robotics research using web-based knowledge services. In *International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, 2016.

R. Bernardi, R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Muscat, and B. Plank. Automatic description generation from images: A survey of models, datasets, and evaluation measures (extended abstract). In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4970–4974, 2017. doi: 10.24963/ijcai.2017/704. URL `https://doi.org/10.24963/ijcai.2017/704`.

R. Bischoff and T. Guhl. Robotic visions to 2020 and beyond – the strategic research agenda for robotics in europe, 01 2009.

N. Blodow. *Managing Belief States for Service Robots: Dynamic Scene Perception and Spatio-temporal Memory*. PhD thesis, Intelligent Autonomous Systems Group, Department of Informatics, Technische Universität München, 2014.

N. Blodow, D. Jain, Z.-C. Marton, and M. Beetz. Perception and Probabilistic Anchoring for Dynamic World State Logging. In *10th IEEE-RAS International Conference on Humanoid Robots*, pages 160–166, Nashville, TN, USA, December 6-8 2010.

J. Bohren, R. B. Rusu, E. G. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mosenlechner, W. Meeussen, and S. Holzer. Towards autonomous robotic butlers: Lessons learned with the pr2. In *ICRA*, Shanghai, China, May 2011.

C. Boitet and M. Seligman. The "whiteboard" architecture: a way to integrate heterogeneous components of NLP systems. *CoRR*, abs/cmp-lg/9411010, 1994. URL `http://arxiv.org/abs/cmp-lg/9411010`.

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

R. Brooks. Steps toward super intelligence 4: Things to work on now, 2018. URL `https://rodneybrooks.com/forai-steps-toward-super-intelligence-iv-things-to-work-on-now/`. Accessed: 2019-07-12.

C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. ISSN 1552-3098. doi: 10.1109/TRO.2016.2624754.

K. Chodorow. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. " O'Reilly Media, Inc.", 2013.

E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970. ISSN 0001-0782. doi: 10.1145/362384.362685. URL `http://doi.acm.org/10.1145/362384.362685`.

D. Coldewey. Robo-logistics company magazino raises $25m for its warehouse bots, 2018. URL `https://techcrunch.com/2018/02/27/`

robo-logistics-company-magazino-raises-25m-for-its-warehouse-bots/. Accessed: 2019-07-21.

A. Collet, M. Martinez, and S. S. Srinivasa. The moped framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research*, 30(10):1284–1306, 2011. doi: 10.1177/0278364911401765.

C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, Sept. 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411.

H. Deeken, T. Wiemann, and J. Hertzberg. Grounding semantic maps in spatial databases. *Robotics and Autonomous Systems*, 105:146 – 165, 2018. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2018.03.011. URL http://www.sciencedirect.com/science/article/pii/S0921889017306565.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

D. Dieckmann. Probabilistic scene understanding using virtual reality and markov logic networks, 2018.

M. Dogar, R. A. Knepper, A. Spielberg, C. Choi, H. I. Christensen, and D. Rus. Multi-scale assembly with robot teams. *The International Journal of Robotics Research*, 34(13):1645–1659, 2015. doi: 10.1177/0278364915586606. URL https://doi.org/10.1177/0278364915586606.

J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference in Machine Learning (ICML)*, 2014.

J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39 (4):677–691, 2017. ISSN 0162-8828. doi: 10.1109/TPAMI.2016.2599174. URL https://doi.org/10.1109/TPAMI.2016.2599174.

M. Durner, S. Kriegel, S. Riedel, M. Brucker, Z. C. Marton, F. Bálint-Benczédi, and R. Triebel. Experience-based optimization of robotic perception. In *ICAR 2017 - 18th International Conference on Advanced Robotics*. IEEE, July 2017. URL `http://elib.dlr.de/113534/`. Finalist for Best Paper Award.

A. Ecins, C. FermÃijller, and Y. Aloimonos. Cluttered scene segmentation using the symmetry constraint. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2271–2278, May 2016. doi: 10.1109/ICRA.2016.7487376.

C. Eppner, S. HÃűfer, R. Jonschkowski, R. MartÃŋn-MartÃŋn, A. Sieverling, V. Wall, and O. Brock. Lessons from the amazon picking challenge: Four aspects of building robotic systems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4831–4835, 2017. doi: 10.24963/ijcai.2017/676. URL `https://doi.org/10.24963/ijcai.2017/676`.

EU-MAR. Robotics 2020 multi-annual roadmap, 2016. URL `https://www.eu-robotics.net/cms/upload/downloads/ppp-documents/Multi-Annual_Roadmap2020_ICT-24_Rev_B_full.pdf`. Accessed: 2019-07-22.

E. A. Feigenbaum. Expert systems: Principles and practice, 1992.

C. Fellbaum. *WordNet: an electronic lexical database*. MIT Press USA, 1998.

P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9), 2010.

D. Ferrucci and A. Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty. Building Watson: An overview of the DeepQA project. *AI Magazine*, pages

59–79, 2010. ISSN 0738-4602. URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/2303.

S. R. Fiorini and M. Abel. A review on knowledge-based computer vision, 2010. URL http://www.inf.ufrgs.br/~srfiorini/wp-content/uploads/Sandro-Fiorini-Mara-Abel-Review-Knowledge-Based-Computer-Vision.pdf.

M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

G. Gemignani, R. Capobianco, E. Bastianelli, D. Bloisi, L. Iocchi, and D. Nardi. Living with robots: Interactive environmental knowledge acquisition. *Robotics and Autonomous Systems*, 78:1–16, 2016. doi: 10.1016/j.robot.2015.11.001. URL http://robertocapobianco.com/publications/RAS15.pdf.

S. Gershman and N. Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

L. C. Goron, Z. C. Marton, G. Lazea, and M. Beetz. Segmenting cylindrical and box-like objects in cluttered 3D scenes. In *7th German Conference on Robotics (ROBOTIK)*, Munich, Germany, May 2012.

G. D. Hager. Life in a world of ubiquitous sensing. In *Conference Keynote at IROS*, 2014. URL http://www.cs.jhu.edu/~hager/Talks/IROS-2014-Sensing-Keynote.pdf.

A. Haidu, D. Beßler, A. K. Bozcuoglu, and M. Beetz. Knowrob-sim âĂŤ game engine-enabled knowledge processing for cognition-enabled robot control. In *International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018. IEEE.

K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL `http://arxiv.org/abs/1703.06870`.

S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Proceedings of the 11th Asian Conference on Computer Vision - Volume Part I*, ACCV'12, pages 548–562. Springer-Verlag, 2013. ISBN 978-3-642-37330-5.

S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

N. Hochgeschwender, S. Schneider, H. Voos, and G. K. Kraetzschmar. Declarative specification of robot perception architectures. In *Simulation, Modeling, and Programming for Autonomous Robots - 4th International Conference, SIMPAR 2014, Bergamo, Italy, October 20-23, 2014. Proceedings*, pages 291–302, 2014. doi: 10.1007/978-3-319-11900-7_25. URL `http://dx.doi.org/10.1007/978-3-319-11900-7_25`.

J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016. URL `http://arxiv.org/abs/1611.10012`.

C. Hudelot. *Towards a Cognitive Vision Platform for Semantic Image Interpretation; Application to the Recognition of Biological Organisms.* PhD thesis, Université Nice Sophia Antipolis, 2005.

S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-

0716-1. doi: 10.1145/2047196.2047270. URL http://doi.acm.org/10.1145/2047196.2047270.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *CoRR*, abs/1412.2306, 2014. URL http://arxiv.org/abs/1412.2306.

R. D. King. The automation of science. *Science*, 324(5923):85–89, April 3, 2009.

R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *2013 IEEE International Conference on Robotics and Automation*, pages 855–862, 2013. doi: 10.1109/ICRA.2013.6630673.

N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154, 2004. doi: 10.1109/IROS.2004.1389727.

L. Kolondy. *Meet Flippy, a burger-grilling robot from Miso Robotics and CaliBurger*, 2017. URL https://techcrunch.com/2017/03/07/meet-flippy-a-burger-grilling-robot-from-miso-robotics-and-caliburger/. Accessed: 2019-06-28.

D. Kragic and M. Vincze. Vision for robotics. *Foundations and Trends in Robotics*, 1:1–78, 01 2009. doi: 10.1561/2300000001.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *2011 IEEE International Conference on Robotics and Automation*, pages 1817–1824, 2011. doi: 10.1109/ICRA.2011.5980382.

D. D. Lee, P. A. Ortega, and A. A. Stocker. Dynamic belief state representations. *Current Opinion in Neurobiology*, 25:221 – 227, 2014a. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2014.01.018. URL http://www.sciencedirect.com/science/article/pii/S0959438814000348. Theoretical and computational neuroscience.

D. D. Lee, P. A. Ortega, and A. A. Stocker. Dynamic belief state representations. *Current Opinion in Neurobiology*, 25:221–227, Apr. 2014b. ISSN 09594388. doi: 10.1016/j.conb.2014.01.018. URL http://dx.doi.org/10.1016/j.conb.2014.01.018.

D. Leidner, A. Dietrich, F. Schmidt, C. Borst, and A. Albu-SchÃďffer. Object-centered hybrid reasoning for whole-body mobile manipulation. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1828–1835, May 2014. doi: 10.1109/ICRA.2014.6907099.

C. Li, J. Bai, and G. D. Hager. A unified framework for multi-view multi-class object pose estimation. *CoRR*, abs/1803.08103, 2018. URL http://arxiv.org/abs/1803.08103.

T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

G. Lisca, D. Nyga, F. Bálint-Benczédi, H. Langer, and M. Beetz. Towards Robots Conducting Chemical Experiments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7354110.

W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

P. Loncomilla, J. R. del Solar, and L. MartÃŋnez. Object recognition using local invariant features for robotic applications: A survey. *Pattern Recognition*, 60:499 – 514, 2016. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.05.021. URL http://www.sciencedirect.com/science/article/pii/S0031320316301054.

D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision, Kerkyra, Corfu, Greece, September 20-25, 1999*, pages 1150–1157, 1999. doi: 10.1109/ICCV.1999.790410. URL https://doi.org/10.1109/ICCV.1999.790410.

I. Lysenkov, V. Eruhimov, and G. Bradski. Recognition and Pose Estimation of Rigid Transparent Objects with a Kinect Sensor. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

J. Maitin-Shepard, M. Cusumano-Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315, 2010. doi: 10.1109/ROBOT.2010.5509439.

Z.-C. Marton, F. Bálint-Benczédi, O. M. Mozos, N. Blodow, A. Kanezaki, L.-C. Goron, D. Pangercic, and M. Beetz. Part-based geometric categorization and object reconstruction in cluttered table-top scenes. *Journal of Intelligent and Robotic Systems*, pages 1–22, 2014. ISSN 0921-0296. doi: 10.1007/s10846-013-0011-8. URL http://dx.doi.org/10.1007/s10846-013-0011-8.

G. Milliez, M. Warnier, A. Clodic, and R. Alami. A framework for endowing an interactive robot with reasoning capabilities about perspective-taking and belief management. In *IEEE RO-MAN 2014*, page FrAT2.4, Edinburgh, United Kingdom, Aug. 2014. doi: 10.1109/ROMAN.2014.6926399. URL https://hal.archives-ouvertes.fr/hal-01064546.

T. Mörwald, J. Prankl, A. Richtsfeld, M. Zillich, and M. Vincze. Blort- the blocks world robotic vision toolbox. In *"Best Practice Algorithms in 3D Perception and Modeling for Mobile Manipulation Workshop" - CD (in conjunction with the IEEE ICRA 2010)*, 2010.

O. M. Mozos, Z. C. Marton, and M. Beetz. Furniture Models Learned from the WWW – Using Web Catalogs to Locate and Categorize Unknown Furniture Pieces in 3D Laser Scans. *Robotics & Automation Magazine*, 18(2):22–32, June 2011.

M. Muja, R. B. Rusu, G. Bradski, and D. Lowe. Rein - a fast, robust, scalable recognition infrastructure. In *ICRA*, Shanghai, China, 09/2011 2011.

T. Niemueller, G. Lakemeyer, and S. S. Srinivasa. A Generic Robot Database and its Application in Fault Analysis and Performance Evaluation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems 2012*, Vilamoura, Algarve, Portugal, 2012. IEEE/RAS.

T. Niemueller, N. Abdo, A. Hertle, G. Lakemeyer, W. Burgard, and B. Nebel. Towards Deliberative Active Perception using Persistent Memory. In *Proc. of IROS 2013 - Workshop on AI-based Robotics*, Tokyo, Japan, 2013a.

T. Niemueller, S. Schiffer, G. Lakemeyer, and S. Rezapour-Lakani. Life-long Learning Perception using Cloud Database Technology. In *Proc. of IROS 2013 - Cloud Robotics Workshop*, Tokyo, Japan, 2013b.

A. Nordmann, N. Hochgeschwender, and S. Wrede. A survey on domain-specific languages in robotics. In *Simulation, Modeling, and Programming for Autonomous Robots - 4th International Conference, SIMPAR 2014, Bergamo, Italy, October 20-23, 2014. Proceedings*, pages 195–206, 2014. doi: 10.1007/978-3-319-11900-7_17. URL `http://dx.doi.org/10.1007/978-3-319-11900-7_17`.

D. Nyga. *Interpretation of Natural-language Robot Instructions: Probabilistic Knowledge Representation, Learning, and Reasoning*. PhD thesis, University of Bremen, 2017. URL `http://nbn-resolving.de/urn:nbn:de:gbv:46-00105882-13`.

D. Nyga, F. Bálint-Benczédi, and M. Beetz. PR2 Looking at Things: Ensemble Learning for Unstructured Information Processing with Markov Logic Networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, May 31-June 7 2014. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6907427`.

K. Okada, M. Kojima, S. Tokutsu, T. Maki, Y. Mori, and M. Inaba. Multi-cue 3D object recognition in knowledge-based vision-guided humanoid robot system.

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)., pages 3217–3222, 2007.

M. Oliveira, G. H. Lim, L. S. Lopes, S. H. Kasaei, A. M. TomÃĺ, and A. Chauhan. A perceptual memory system for grounding semantic representations in intelligent service robots. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2216–2223, Sept 2014. doi: 10.1109/IROS.2014.6942861.

OpenCyc. OpenCyc, 2009. URL `www.opencyc.org`.

D. Pangercic, M. Tenorth, D. Jain, and M. Beetz. Combining Perception and Knowledge Processing for Everyday Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1065–1071, Taipei, Taiwan, October 18-22 2010.

D. Pangercic, M. Tenorth, B. Pitzer, and M. Beetz. Semantic object maps for robotic housework - representation, acquisition and use. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vilamoura, Portugal, October, 7–12 2012.

K. Pauwels and D. Kragic. Simtrack: A simulation-based framework for scalable real-time object pose detection and tracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hamburg, Germany, 2015.

R. Platt, R. Tedrake, L. Kaelbling, and T. Lozano-Pèrez. Belief space planning assuming maximum likelihood. In *Proceedings of Robotics: Science and Systems*, 06 2010. doi: 10.15607/RSS.2010.VI.037.

J. Prankl, A. Aldoma, A. Svejda, and M. Vincze. Rgb-d object modelling for object recognition and tracking. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 96–103, Sept 2015.

A. Pronobis, O. M. Mozos, B. Caputo, and P. Jensfelt. Multi-modal semantic place classification. *The International Journal of Robotics Research (IJRR), Special Issue on Robotic Vision*, 29(2-3):298–320, feb 2010. doi:

10.1177/0278364909356483. URL http://www.pronobis.pro/publications/pronobis2010ijrr.

J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

M. Reza Loghmani, B. Caputo, and M. Vincze. Recognizing objects in-the-wild: Where do we stand? In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2170–2177, May 2018. doi: 10.1109/ICRA.2018.8460985.

M. Richardson and P. Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, 2006. ISSN 0885-6125. doi: http://dx.doi.org/10.1007/s10994-006-5833-1.

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, Nov 2011. doi: 10.1109/ICCV.2011.6126544.

S. Rudolph. *Foundations of Description Logics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-23032-5. doi: 10.1007/978-3-642-23032-5_2. URL https://doi.org/10.1007/978-3-642-23032-5_2.

J. Ruiz-del-Solar, P. Loncomilla, and N. Soto. A survey on deep learning methods for robot vision. *CoRR*, abs/1803.10862, 2018. URL http://arxiv.org/abs/1803.10862.

S. J. Russell and P. Norvig. *Artificial Intelligence — A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

R. B. Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Technische Universität München, 2009.

R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4, Shanghai, China, May 9-13 2011.

R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 2010.

S. Salti, F. Tombari, and L. di Stefano. Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, 2014.

M. Scheutz. ADE - steps towards a distributed development and runtime environment for complex robotic agent architectures. *Applied Artificial Intelligence*, 20 (4-5), 2006.

L. Skelly and S. Sclaroff. Improved feature descriptors for 3-d surface matching - art. no. 67620a. *Proceedings of SPIE - The International Society for Optical Engineering*, 09 2007. doi: 10.1117/12.753263.

D. Skočaj, A. Vrečko, M. Mahnič, M. Janíček, G.-J. M. Kruijff, M. Hanheide, N. Hawes, J. L. Wyatt, T. Keller, K. Zhou, M. Zillich, and M. Kristan. An integrated system for interactive continuous learning of categorical knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, 28(5):823–848, 2016. doi: 10.1080/0952813X.2015.1132268. URL `https://doi.org/10.1080/0952813X.2015.1132268`.

A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, Dec 2000. ISSN 0162-8828. doi: 10.1109/34.895972.

R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng. Grounded compositional semantics for finding and describing images with sentences. *TACL*, 2:207–218, 2014.

M. Sridharan, J. Wyatt, and R. Dearden. Planning to see: A hierarchical approach to planning visual actions on a robot using pomdps. *Artificial Intelligence*, 174 (11):704 – 725, 2010. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.

2010.04.022. URL http://www.sciencedirect.com/science/article/pii/S0004370210000664.

Y. Sun, L. Bo, and D. Fox. Attribute Based Object Identification. In *IEEE International Conference on on Robotics and Automation*, 2013.

M. Tenorth and M. Beetz. KnowRob – Knowledge Processing for Autonomous Personal Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4261–4266, 2009.

M. Tenorth and M. Beetz. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. *Int. Journal of Robotics Research*, 32(5):566 – 590, April 2013. URL http://ijr.sagepub.com/content/32/5/566.short.

M. Tenorth, S. Profanter, F. Bálint-Benczédi, and M. Beetz. Decomposing CAD Models of Objects of Daily Use and Reasoning about their Functional Parts. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5943–5949, Tokyo Big Sight, Japan, November 3–7 2013. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6697218.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, 2005.

S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the DARPA grand challenge. *Journal of Field Robotics*, 2006. Foresight Report.

F. Tombari and L. D. Stefano. Object recognition in 3d scenes with occlusions and clutter by hough voting. *Proceedings of the 4th Pacific-Rim Symposium on Image and Video Technology (PSIVT 2010)*, 0:349–355, 2010.

C. Town. Ontological inference for image and video analysis. *Machine Vision and Applications*, 17(2), 2006. URL http://link.springer.com/article/10.1007/s00138-006-0017-3.

L. Ulanoff. You can now order pizza from a robot in singapore, 2017. URL https://mashable.com/2016/05/24/pepper-robot-pizza-hut/?europe=true. Accessed: 2019-06-28.

US Roadmap. A roadmap for us robotics: From internet to robotics. http://jacobsschool.ucsd.edu/contextualrobotics/docs/rm3-final-rs.pdf, 2016.

K. M. Varadarajan and M. Vincze. Afrob: The affordance network ontology for robots. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1343–1350, Oct 2012. doi: 10.1109/IROS.2012.6386232.

V. Vassiliadis, J. Wielemaker, and C. Mungall. Processing OWL2 ontologies using Thea: An application of logic programming. In *OWLED, CEUR Workshop Proceedings*, volume 529, 2009.

D. Vernon. *Artificial Cognitive Systems: A Primer*. MIT Press, 2014. ISBN 0262028387, 9780262028387.

D. Vernon, M. Beetz, and G. Sandini. Prospection in cognition: The case for joint episodic-procedural memory in cognitive robotics. *Frontiers in Robotics and AI*, 2:19, 2015. ISSN 2296-9144. doi: 10.3389/frobt.2015.00019. URL https://www.frontiersin.org/article/10.3389/frobt.2015.00019.

M. Warnier, J. Guitton, S. Lemaignan, and R. Alami. When the robot puts itself in your shoes. managing and exploiting human and robot beliefs. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 948–954, Sept 2012. doi: 10.1109/ROMAN.2012.6343872.

Waymo. On the road to fully self-driving, 2017. URL https://storage.googleapis.com/sdc-prod/v1/safety-report/waymo-safety-report-2017.pdf. Accessed: 2019-06-28.

Q. Weichao, Z. Fangwei, Z. Yi, Q. Siyuan, Z. Xiao, K. Tae Soo, W. Yizhou, and Y. Alan. Unrealcv: Virtual worlds for computer vision. *ACM Multimedia Open Source Software Competition*, 2017.

T. Wiedemeyer, F. Bálint-Benczédi, and M. Beetz. Pervasive 'calm' perception for autonomous robotic agents. In *Proceedings of the 2015 International Conference*

on Autonomous Agents and Multiagent Systems, Istanbul, Turkey, 2015. ACM. URL http://dl.acm.org/citation.cfm?id=2773264.

J. Wielemaker, G. Schreiber, and B. Wielinga. Prolog-based infrastructure for rdf: Scalability and performance. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003*, pages 644–658, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39718-2.

J. Winkler, M. Tenorth, A. K. Bozcuoglu, and M. Beetz. CRAMm – memories for robots performing everyday manipulation activities. *Advances in Cognitive Systems*, 3:47–66, 2014.

W. Wohlkinger, A. Aldoma, R. B. Rusu, and M. Vincze. 3dnet: Large-scale object class recognition from cad models. In *2012 IEEE International Conference on Robotics and Automation*, pages 5384–5391, May 2012. doi: 10.1109/ICRA.2012.6225116.

J.-H. Worch, F. Bálint-Benczédi, and M. Beetz. Perception for everyday human robot interaction. *KI - Künstliche Intelligenz*, 30(1):21–27, 2015. ISSN 1610-1987. doi: 10.1007/s13218-015-0400-1. URL http://dx.doi.org/10.1007/s13218-015-0400-1.

Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics Morristown, NJ, USA, 1994.

J. L. Wyatt, A. Aydemir, M. Brenner, M. Hanheide, N. Hawes, P. Jensfelt, M. Kristan, G.-J. M. Kruijff, P. Lison, A. Pronobis, K. Sjöö, D. Skočaj, A. Vrečko, H. Zender, and M. Zillich. Self-understanding and self-extension: A systems and representational approach. *IEEE Transactions on Autonomous Mental Development*, 2(4):282 – 303, December 2010.

F. Yazdani, G. Kazhoyan, A. K. Bozcuoglu, A. Haidu, F. Bálint-Benczédi, D. Beßler, M. Pomarlan, and M. Beetz. Cognition-enabled framework for mixed human-robot rescue team. In *International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018. IEEE.