Institut für Informatik
Technische Universität München

Managing Belief States for Service Robots: Dynamic Scene Perception and Spatio-temporal Memory

Dissertation

Nico Blodow

TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Informatik IX
Intelligente Autonome Systeme

Managing Belief States for Service Robots: Dynamic Scene Perception and Spatio-temporal Memory

Nico G. S. Blodow

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende: Univ.-Prof. Gudrun J. Klinker, Ph.D.

Prüfer der Dissertation:

1. Univ.-Prof. Michael Beetz, Ph.D.,
   Universität Bremen
2. Univ.-Prof. Dr.-Ing. Darius Burschka
3. Ao. Prof. Dr. techn. Markus Vincze,
   Technische Universität Wien / Österreich

Die Dissertation wurde am 26.09.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 14.05.2014 angenommen.

Abstract

Unstructured information management (UIM) has proven itself as a powerful paradigm for scaling intelligent information and question answering systems towards real-world complexity. Complexity in UIM is handled by identifying (or hypothesizing) pieces of structured information in unstructured documents, by applying ensembles of experts for annotating information pieces, and by testing and integrating these isolated annotations into a comprehensive interpretation of the document.

In this thesis the paradigm of unstructured information management is applied to the problem of robot perception of realistic scenes containing objects of daily use. In this view, the documents in UIM correspond to camera and depth images taken by the robot, the structured information pieces such as furniture pieces and objects to be manipulated to object hypotheses, and the ensembles of experts to sets of object perception methods. We believe that the application of the UIM principle to robot perception can achieve similar scaling effects as it did for intelligent information systems.

We will present RoboSherlock, an open source software framework for unstructured information processing in robot perception and sketch a feasibility study of a perception system built on top of the framework that indicates the potential of the paradigm for real-world scene perception. A large number of perception methods have been implemented or integrated into RoboSherlock that span detection of different kinds of objects, annotators that compute appearance or spatial feature descriptors, shape classification, and object reconstruction as well as tracking and entity resolution methods. Web services have been incorporated to provide additional, visual or non-visual information such as text and logo recognition, or product matches from online stores.

The object descriptions obtained with RoboSherlock show a richness and versatility that we believe to be unrivaled in the field and is demonstrated within experiments over numerous scenes containing arrangements of objects of varying kinds.

Kurzfassung

Das UIM-Prinzip[1] zur Verarbeitung unstrukturierter Informationen hat sich auf dem Forschungsgebiet der natürlichen Sprachverarbeitung als mächtiges Werkzeug zur Skalierung intelligenter, informationsverarbeitender und Fragen beantwortender Systeme hin zur Bewältigung der Komplexität der realen Welt erwiesen.

Um komplexe Probleme zu lösen, werden strukturierte Informationen in unstrukturierten Dokumenten identifiziert oder hypothetisiert, indem sog. Expertenensembles informationstragende Segmente erkennen, markieren und kommentieren. Die darauf folgende Analyse und Integration dieser Annotationen führt zu einer umfassenden und konsistenten Interpretation des Dokuments.

In der vorliegenden Arbeit wird das UIM-Paradigma auf das Problemfeld der Roboterwahrnehmung angewandt, insbesondere für realistische Szenen mit Objekten des täglichen Gebrauchs. So werden Kamerabilder, Tiefenkarten und Punktwolken als Dokumente im UIM-Sinne aufgefasst; Objekthypothesen über beispielsweise Möbelstücke oder Objekte, die vom Roboter manipuliert werden sollen, entsprechen strukturierten Informationen; und Objektperzeptionsalgorithmen werden als Expertenensembles modelliert. Wir sind der Überzeugung, dass die Verwendung des UIM-Prinzips einen ähnlichen Skalierungseffekt für die Roboterwahrnehmung erzielen kann wie in der Vergangenheit für intelligenten Informationssysteme.

Im Verlauf dieser Arbeit wird RoboSherlock vorgestellt, ein quelloffenes Softwareframework zur Verarbeitung unstrukturierter Informationen im Rahmen eines Wahrnehmungssystems für Roboter. Außerdem wird ein Perzeptionssystem auf Basis von RoboSherlock implementiert, das das Potential des UIM-Prinzips für realistische Szenen verdeutlicht. Zu diesem Zweck wurden zahlreiche Wahrnehmungsmethoden in

[1] Englisch: unstructured information management

RoboSherlock entwickelt und integriert, die vielseitige Themengebiete abdecken, wie Objekterkennung mittels verschiedener Messprinzipien und Algorithmen, Methoden zur Berechnung von Geometrie- oder Erscheinungsmerkmalen, und die Klassifikation dieser Merkmale mittels Methoden des maschinellen Lernens. Ferner wurden Methoden zur Verfolgung und Identitätsbestimmung von Objektobservationen, sowie Webdienste, die zusätzliche, visuelle und nicht-visuelle Informationen bieten können, wie beispielsweise Texterkennung, Logoerkennung oder Produktseiten von Online-Shops, erarbeitet und einbezogen.

Die Objektbeschreibungen innerhalb RoboSherlocks weisen dabei eine Vielfalt und Fülle auf, die wir auf diesem Gebiet für unerreicht halten, was anhand von Experimenten demonstriert wird, die zahlreiche Szenen mit Anordnungen verschiedenster Objekte enthalten.

Acknowledgements

This thesis would not have been possible without the support of certain people.

First and foremost, I would like to thank my thesis advisor Prof. Michael Beetz who continually supported me by giving me challenging topics to tackle and recognizing and optimizing the potential of my ideas. I am especially grateful for the opportunity to spend several months at exciting institutes over seas. One could not wish for a more insightful, visionary, driving and relaxed boss.

The second crucial influence in my career was Dr. Radu Rusu, with which I had the pleasure and honor to work for many years. Since his mentorship for my undergraduate theses, and later as a colleague and friend, I have never had as much fun, work or success as when working at his side.

Obviously, there were numerous other colleagues who influenced my academic career or my daily life in the office. Specifically, I would like to thank Mihai Dolha and Zoltan-Csaba Marton for being the best office mates and creating a concentrated yet humerous atmosphere, and Dominik Jain, Alexis Maldonado, Lorenz Mösenlechner, Thomas Rühr and Florian Seidel for fascinating discussions concerning life, the universe and everything. I am especially grateful to Christian Kerl for his contributions and Ferenc Balint-Benczedi, who has taken the responsibility to continue the research put forth in this thesis, and I wish him the best of luck and success for this challenge.

Last but not least I would like to thank my family for getting me to the point where I could start my dissertation, and my wife Steffi for getting me to finish it. Without her continuous support and love, you would be holding an empty document right now.

Thanks to all of you.

Contents

List of Resources

Figures

Tables

Listings

Chapter 1

Introduction

Autonomous robots are increasingly conquering households, and will likely continue to do so in the future. Currently, these robots are limited to specific tasks such as *e.g.* vacuuming, wiping floors or cutting grass, but over the next years and decades, they will evolve more advanced skills, develop higher degrees of flexibility, and continually support and assist us more. At present, these more advanced robots already exist in academic research environments, but during our lifetime, we will see them leave the labs and find their place in health care, the public and private sector, as well as personal homes. Mobile manipulation, *i.e.* the ability to navigate and pick up and use objects, will become an indispensible skill in being able to assist the elderly, the physically impaired and all those who want a robot to alleviate their daily chores.

Figure 1.1.: *TUM-Rosie and TUM-James, two of our robots, rely on a large collection of algorithms when preparing sandwiches and popcorn in our lab kitchen environment. A multitude of individual perception methods allow manipulating various objects such as pots, cutlery, or slices of cheese or toast.*

Consider an assistant robot such as our PR2[177] TUM-James or our in-house developed robot TUM-Rosie depicted in Figure 1.1 that performs its tasks in a typical household. Its duties include manipulation of objects, such as cleanup, table setting tasks or even meal preparation. Within the robot control program, the following, naturally specified queries and tasks can arise:

- Bring me my tea cup.
- Put all perishable items in the fridge.
- Stow away the items in the shopping basket.
- Set the table for breakfast.
- Clean up the kitchen.
- Is there anything on the couch table?
- Where did I leave my tea cup?
- When was the last time you saw it?
- Is there any cereal on the kitchen table?
- What objects are standing on the counter?
- What do we keep in this drawer?
- Do we have milk in the fridge?

In order to be able to solve these problems, it is quintessential to provide perceptual capabilities that encompass a wide array of research topics, performing functions such as object detection, object recognition, categorization and classification, object localization and reconstruction.

The computer vision research community has primarily focused on solving these individual tasks using spezialized algorithms, custom-tailored data representations and often ad-hoc inclusion in control software. This is merely an observation, not a critique of other work, since it is obviously extremely important to explore the possibilities and potential solutions to hard and difficult problems. It does however mean

that there is no integrated system where flexibility, generality and semantic depth is being achieved on a large scale.

Taking one step back, the problem of perception for robotics is often posed very narrowly, and disregards important issues that need to be addressed to be able to answer the queries given above: vision systems should be more than merely algorithms to match a percept to a label or name, possibly to grasping points and 3D models, but intended usage, associations to semantic concepts and embedding into a task context should be inherently engrained within these system. In our opinion, a robot should be able to autonomously navigate an environment, and see, infer and manipulate it in an informed manner.

Carrying out these activities of daily life requires a deep and thorough understanding of various aspects of the environment, reaching beyond environment maps and object models.

One of the most important resources that enable such robots to perform these tasks reliably, efficiently, and competently is an understanding of the environment in which they are operating. Robots that are to perform everyday manipulation tasks in human living environments must maintain a belief state concerning an environment model that is much more comprehensive and richer than the models that have been employed by robots so far. A representation merely of the spatial layout of the environment with perceptual descriptions of locations, as used for *e.g.* navigation methods, is not sufficient for more complex tasks such as setting tables, shopping groceries, preparing meals or cleaning a room.

On the other hand, while having full knowledge of all possible pieces of information is an ultimate goal of robots performing on par with humans, it is generally not possible to perform sufficient analysis on sensor data and knowledge sources due to constrained time and computing resources. Also, this is not strictly necessary in most cases, *e.g.* to pick up a glass of water, the price of the glass is irrelevant, so a complete reconstruction of all aspects is usually not required to perform a given task.

Often, queries such as the couch table question given above, are posed at a point where the robot could *in theory* already answer it. While vacuuming, the robot might have gone past the couch table several times, so when the question is asked, the robot should simply “remember” what objects were placed on it, even if he is currently in

another part of the house. This means that the robot might not know at the point of perceiving a scene what queries will be asked later. Plus, it is simply not possible to reconstruct all possible information at the moment of observing a certain object. For these open-ended questions, unknown at the time of data acquisition, proactive percept collection abilities are required that exceed the current state of the art.

This means that if a task is given to an agent, that task defines the subset of information that needs to be gathered in order to carry it out successfully. However, there is an almost infinite number of different kinds of information, ranging from *visible* (*e.g.* geometric or physical) to *hidden* aspects such as ingredients (*e.g.* allergens), safety considerations (*e.g.* burning candles), ownership (*e.g.* *my* cup") or hygiene (*e.g.* which dish brush to use for cleaning the dog's food bowl). Generally speaking, not all of these descriptions need to be discovered in a single task context, so their generation needs to be moderated by appropriate algorithm selection and orchestration. Also, a likewise open object description is necessary that can be enriched on demand with required information.

This leads to another important aspect of such a supposed system, which is that of algorithms and data flow. Confronted with this large variety of contexts, tasks, objects and their respective properties, it seems highly unlikely that there is a single algorithm that can achieve the generality, robustness, scalability and attention to detail required for competent and informed decision making in these environments. In human environments, almost every object is designed for certain actions and purposes. As a consequence, there are strong correlations between objects, their appearance, their locations, their state, and the activity context. These correlations must be known to the robot in order to perform the right action on the right object in the right way. As other fields of research have shown, ensemble-of-experts systems can be created that combine the specificity and confidence of individual "expert" algorithms in a flexible and modular framework that enables solving incredibly hard problems such as natural language processing and question answering, as demonstrated *e.g.* in the IBM Watson project [38], the Siri agent [1] or Google Now [2]. Ensembles of experts have been applied successfully in machine learning [104, 112, 123], and in the field of robotics to *e.g.* place classification [117] and furniture categorization [97].

[1] http://www.apple.com/ios/siri
[2] http://www.google.com/now

1.1 Scenario

Consider the following scenario: A robot is deployed in an average household and has to fulfill typical duties, such as cleaning up, emptying shopping bags and stowing groceries, fetching items, setting the table and so forth. As it enters the kitchen the first time, it needs to gather a lot of knowledge concerning the structure of the kitchen, storage containers and how to open them, storage location for food items, dishes and glasses, identification of these objects, and so forth.

In the initial orientation or mapping phase, the robot needs to acquire the neccessary know-how to be able to successfully navigate, locate regions of interest for subsequent tasks, *e.g*. tables, appliances and containers, which it represents in a semantic map, along with related information, such as articulation models for operating doors and drawers and typical storage locations.

After this initial phase, the robot is ready to perform tasks that will most frequently be concerned with identification and manipulation of objects. Apart from detecting and classifying objects, there are several important challenges that the robot needs to be able to deal with.

Multiple instantiations of a class of objects, *e.g*. a set of indiscernible mugs or plates, must not pose a problem for the robot. Simply recognizing that a certain object is a glass is not sufficient if the robot is to fetch mine. Imagine a human seeing a counter with three cups in a row. If the middle one is brought to the table while he is looking away, the human observer would be able to look at the remaining two and infer that since they have not moved, the one on the table was most likely taken from the center position, even if they all look the same.

An immensely important skill such an agent requires to solve these problems is the ability to perform *temporal reasoning*, *i.e*. the ability to relate current observations with past observations and establishing object identity in the presence of partial data. Object identity in this context does not neccessarily mean knowing exactly which object the robot perceives, but also the (reduced) inference that the object is most likely the same as another one seen previously. In the above example, the cups' absolute identities are irrelevant, *e.g*. after a dish washer cycle they could be permutated and given to different people to drink from.

These object tracking capabilities also help to accumulate an understanding of an object through multiple multi-modal observations over time, as more information is gathered due to different view points or sensing modalities. Object model reconstruction is one direct application, where multiple partial scans are merged into a complete model. However, there are many more potential consequences from this setting. From the front, a can of chips might have a distinctive logo which can be reliably used for redetection, while on the back there could be a barcode which can lead to very detailed product information such as the kind, name, manufacturer or ingredients of the product, through online registries.

Identity resolution thus allows the system to enrich perceptual data (appearance and geometrical cues) with additional, non-visual information, such as names, categories, product manufacturer, ingredients and other concepts. This is neccessary to allow successful mapping of (initially anonymous) objects to named entities in a user command, or to map a task execution plan to a real-world environment by anchoring required objects in perceptual data. In the chips example, the resulting object model for the chips contains the 3D geometry, a visually distinctive logo, a barcode and the manufacturer and product names so a user can refer to “chips” or “can of Pringles” without manual labeling or training.

This makes it also possible to steer away from purely database-driven approaches, where features are mapped to objects which directly state grasp points or other relevant information. This is simply not scalable, since it is generally impossible to rely on knowing all objects in a deployment environment beforehand. There are potentially millions of different objects a robot could encounter, be it new dishes, visitors’ shoes or ever changing designs of food or household products. In contrast, including various and diverse properties in object descriptions can help planning and reasoning components to identify aspects of interest without computing and storing every aspect such as grasp points, motion constraints and intended object use in advance.

The overall vision for such a robotic household agent is that it is able to navigate, perceive, reason and manipulate successfully in presence of all the ambiguities and implicit assumptions that humans can deal with so effortlessly.

1.2 Conceptual Apparatus

Explicitly, we expect the following from a system capable of delivering the aforementioned capabilities.

- The system must be able to *proactively collect percepts*, since at acquisition time, it might not be known what will be asked of the robot later.
- The system must provide information collecting capabilities incorporating a *wide and extensible set* of specialized algorithms. This set must span multiple dimensions:
  - The system must provide several *kinds* of algorithms: segmentation, object detection, classification and reconstruction methods. Reconstruction is meant to include any kind of information that can be gathered to describe an object observation in more detail, *i.e.* on top of shape-based reconstruction, it includes *e.g.* text recognition, brand/logo recognition, association of an object to a product in online stores, ontologies, ingredients or weight.
  - For every of these categories, it is necessary to have potentially *multiple* equivalent or similar algorithms that, while providing the same high-level functionality, are able to exhibit different strenghts. The system must be able to *combine* these methods and their data analysis representations as to maximize the robustness, reliability and confidence of the extracted information.
  - On top of general algorithms, it sometimes neccessary to include very fine-tuned, *specialized algorithms*, such as *e.g.* a detector for translucent objects or even a single product. This would be especially desirable when it comes to potential safety issues such as hot stoves or food preparation, where the cost of developing and executing specific solutions is far outweighed by the gain in confidence and reliability.
- Object descriptions – or rather descriptions of object observations – must be *open and extensible* in the sense that other modules within and outside the perception domain can access and augment these descriptions.

- Looking at scene dynamics, *tracking* functionality is required to be able to observe an object over time. This can be extended to a more general problem of *object identity resolution*, where the temporal aspect of tracking is disregarded. This way, object observations from very different points in time can be compared and used for various purposes, *e.g.* merging of different partial views or collecting and combining object aspects reconstructed from different partial views. Additionally, connections to human tracking or activity recognition fall within this domain, *e.g.* to annotate an object with the information "was put down by person X".
- Apart from object-centric information scene or context related information (*e.g.* object life cycles, states, ownership or typical storage locations) must be representable and processable.

1.3 Proposed Solution

One of the biggest challenges for robot perception is scaling towards real-world scene and task complexity. Future household robots, for example, will have to detect, recognize, categorize, localize, and reconstruct textured objects, form-characterized objects, translucent ones, fluids, cloths, dirt, objects that are cut into pieces, ingredients that are mixed into other ones, *etc*. Objects and furniture have to be perceived in clutter, inside cupboards, and under different lighting conditions.

RoboSherlock, the project described and implemented in this thesis, investigates the representation, acquisition and use of environment and object models that provide the aforementioned kinds of information. The basic idea of the RoboSherlock project is as follows: The system is embedded in the well-known perception-action loop (*i.e.* within our other robot processes for planning and reasoning), but can act in several *roles*: RoboSherlock can contain processing pipelines that work *passively* on sensor streams, attempting to reconstruct a world belief state as new data becomes available to the system. Additionally, there are RoboSherlock pipelines that can be triggered as required and thus act in a more *active* way. As an example, consider a robot observing a table surface as it is performing some task. Over time, various partial views of an object on this table are collected passively. At some point in time, a decision

making process can trigger a processing pipeline that merges these partial views into a complete 3D object model for storage and reuse.

In this thesis, we lay out both an extensible *framework* that strives to provide the architectural underpinning required for a system capable of such functionality, and provide a *system implementation* within this framework that deals with the problem scenarios that arise in the given context. Due to the high goals and thus large scope of the framework, we must leave certain claims concerning framework capabilities unassessed if they do not overlap with issues arising in our system implementation.

1.4 Contributions

The key contributions of this dissertation are as follows:

1. The design and implementation of a framework based on the premise of treating perception as an Unstructured Information Management task. This includes ensemble-of-experts methods, infrastructure for the orchestration and execution of these methods, and well-defined, open and extensible object descriptions.

2. Demonstrating the applicability of the UIM principle, which has been successfully used to solve hard problems such as natural language processing and open-ended question answering, to a novel domain.

3. A type system that is both specific and general enough to join these various processing modules and their data representation in an integrating and extensible manner. This includes “raw” data types and more powerful concepts such as object hypotheses.

4. Furthermore, we detail a set of various singular “experts”/algorithms for specific tasks, such as semantic real-time segmentation methods, point cloud interpretation methods, interfaces to web services, and object identity resolution models.

1.5 Outline

The basic principle behind RoboSherlock is the treatment of perception in a service robotics context as an *Unstructured Information Processing* (UIP, also Unstructured Information Management, UIM) problem, as described in Chapter 2, where we also give a system overview describing the details of the proposed RoboSherlock framework. An integral aspect of our framework is the design of a meta type system that allows the representation of heterogenous, structured metadata that results from data analysis within the various expert methods. It allows mostly independent development of components and enables communicating information between modules. A detailed description can be found in Chapter 3.

For the scope of this dissertation, we define *objects of use* that are relevant to the robot to be subject to two constraints:

- Objects can be held and manipulated using the robot end effector;
- Objects are supported by horizontal support surfaces.

These constraints are not overly limiting, since they allow objects of various sizes, from cutlery to cutting boards and cereal boxes, as well as a wide range of storage locations, such as tables, counter tops, shelfs, cupboards and drawers.

Within the framework, algorithms ("experts") are also called *Annotators*, since they annotate specific information related to an artifact to be analyzed (*e.g.* a point cloud). In the system implementation, we identified three main categories of Annotators:

1. *Object Hypotheses Generators* are modules that can hypothesize the presence of an object of interest given various sources. This covers mainly segmentation algorithms, that analyze *e.g.* input point clouds or camera images under some assumption of "objectness". Objectness has been defined in literature to include *e.g.* convexity, symmetries, surface smoothness, or texture smoothness (see Collet Romea [25] for their definition of "structure discovery"). However, the type system allows more general notions of an object hypothesis, such as a locations in space generated by attention mechanisms or a human motion tracker observing a put-down event. These Annotators are detailed in Chapter 4.

2. *Object Annotators* use these possible object locations to compute various additional pieces of information. This class of Annotators likely makes up the largest collection of experts within our system as the scope of object annotations is very broad. In fact, there are annotators that rely on the OpenCV library[19], others rely on the Point Cloud Library (PCL)[130], reuse existing annotations from previous experts, or make use of existing web services to collect additional, semantic information. Chapter 5 lists the various Annotators available in our system.

3. The third category of Annotators contains *Tracking, Information Fusion and Storage* modules and attempts to *condense* and *merge* the rich object information obtained up to this point to bridge the gap from *object observations* to *objects*. Individual scenes are merged into *belief states* of all currently and previously perceived objects in the environment and stored in a database for later reuse. For details, please refer to Chapter 6.

Some of the methods described in this thesis rely on a structured, semantic 3D map of the environment. The robot follows an autonomous indoor exploration and 3D mapping strategy to acquire full room models through careful sensor placement, scan acquisition and registration. This is followed by geometric functional reasoning, where meaningful components such as kitchen cabinets, drawers, counters, walls, doors *etc*. are identified. This generates a hierarchical, semantically labeled 3D environment model with articulated doors and drawers that can be easily used to define regions of interest – horizontal supporting structures – that can be processed by respective modules. This system is currently implemented outside of our framework as it predates RoboSherlock. However, we describe this fundamental component in Chapter 7 for completeness.

There is another component that has originally been developed in a different context, dealing with known structure segmentation, where the existing environment map and the geometric robot model (along with internal joint sensor information) is used to separate the *known* (*e.g*. robot arm or tables) from *unknown* data points using a GPU-shader based real-time virtual rendering method. While not technically a RoboSherlock Annotator as of this writing, it was placed with the other segmentation algorithms in Chapter 4 due to thematic context.

In order to assess the validity, information richness and performance of our proposed system, we conducted experiments on a PR2 robot[177] as well as our group's in-house developed robotic platform, TUM-Rosie. Experimental setups and results are given in the respective annotators' sections for individual methods, and in Chapter 8 for the system implementation.

We conclude in Chapter 10, where we also provide an outlook into future work.

Chapter 2

Perception as Unstructured Information Processing

Work on robot perception has so far mainly focused on individual algorithms that typically work on objects with specific characteristics, *e.g.* point features for 3D opaque objects [126], SIFT [83] for textured or analysis of missing data points [84] for translucent objects. There are as of now no methods to perceive "everything".

While there are some systems which attemp to widen the scale – most authors strive for robustness and generality of their solution – most of them are focusing on a particular problem set. Consider the following leading edge systems for the detection and localization of objects for robots: The blocks world robotics vision toolbox [96] can reliably detect 3D objects from single camera images using a combination of edge based detection of shapes, tracking and SIFT features for learning and recognizing appearance models. It is however currently limited to cuboid and cylindrical objects. Collet Romea et al. [26] describe MOPED, an object detection and localization system based on per-point appearance features and their geometric relationships which is obtained from rigid 3D models of objects. The range of objects that can be reliably detected are thus textured and three-dimensional objects. A third popular object perception system is LINEMOD [57], a template-based approach which can deal with non-textured objects.

Obviously, algorithms that have been specialized to more specific perception problems have achieved much higher accuracy, robustness and efficiency as long as the assumptions and constraints of the specialized perception problem are met [58]. A key observation is that the combination of methods with lower accuracy can outperform single algorithms in particular in cases where the perception failures of the

individual methods are independent of each other [161, 143]. Also, research in various fields, including algorithm theory [16] has shown that the average resource cost can be drastically reduced through algorithms that "hypothesize" the results and test them afterwards. This is the case because in many problem classes the immediate generation of correct results is computationally much harder than checking an hypothesis for correctness.

This evidence suggest that it is more promising to aim for scalability towards real-world complexity at a system level rather than through the mere improvement of individual algorithms and development of new ones.

System support in robot perception has so far mainly been provided through middleware frameworks, *e.g.* ROS [119], that aim at encapsulating algorithms and methods in order to make the interfaces and data structures that they use compatible. Examples of perception libraries and frameworks are ECTO[1], PCL [130], and OpenCV [19], some of which were used in the development of this thesis (*cf.* Section 9.5). So far these frameworks have targeted the ease of program development but the problem of boosting perception performance through more powerful method *combination* has received surprisingly little attention.

On the other hand, we have seen in recent years several intelligent systems that have impressively scaled towards real-world task complexity: Using the Siri agent, which is a descendant of the CALO project [98], it is possible to instruct a phone to call a taxi, reserve a table at a restaurant or to provide the weather forecast for tomorrow. Siri uses the temporal and spatial task context in order to infer the true meaning hidden in the task's verbal formulation. Watson [39], another intelligent system with remarkable scaling success, has won the US quiz show *jeopardy!* against the champions of the show demonstrating an enormous breadth of knowledge and the crucial ability to correctly judge it's own competence in answering a particular questions.

In the problem domain of collaborative filtering, the NetFlix Prize challenged researchers to predict user preferences based on their and their friend's movie ratings. The participants quickly realized that competitive performance can be attained by joining efforts. Ensemble learning has seen much popularity and success [148],

[1] http://plasmodic.github.io/ecto/

analogously to the open-domain question answering problem addressed by the Watson system. We wish for a similar development in robot perception, as combining various approaches with complementary strengths might alleviate their respective weaknesses and lead to better generalization to novel scenarios [74].

We believe that for the scaling of perception to real world complexity a lot can be learned from the afforementioned approaches to real world intelligent systems. Let us consider the Watson system as an example. In a nutshell, the recipe for success of the system lies in a combination of the following system level principles: (1) acquisition of a knowledge base for question answering is considered as an "unstructured information management" task; (2) the employment of ensembles of expert methods, (3) information interpretation as annotation; (4) parallelization through mostly independent annotators; (5) simplified collaboration through annotation based meta type system.

Since we believe the need for robots to possess the *collective perceptual and cognitive capabilities* of ensembles of expert routines, we propose in this dissertation to build scalable perception systems using UIMA (Unstructured Information Management Architecture)[2], the open source software architecture used for the realization of the Watson system. As UIMA has been primarily designed for mining textual information in the web and creating vast knowledge bases[3], various extensions and changes have to be made to make it applicable to robot perception. Foremost, it is important to replace the text centric data structures with perceptual ones. We also created convenient wrappers for fundamental data types of existing perception libraries (*e.g.* PCL, OpenCV) to integrate with them effortlessly and make them directly usable. We converted the top-level operation of the system from a batch processing into an event-triggered version that can run within the perception-action loops of autonomous robots. In RoboSherlock, Annotators can be written in C++ and Java without constraints, whereas Collection Readers, Flow Controllers and CAS Consumers are only supported in Java as of now (*cf*. Figure 2.3 and Section 2.3 for terminology).

Applying RoboSherlock to an existing robot perception system requires three steps: First, declaring the data structures of the perception system in the RoboSherlock type hierarchy, if the provided ones are not sufficient. Second, wrapping perception

[2]http://uima.apache.org/
[3]https://cwiki.apache.org/UIMA/powered-by-apache-uima.html

routines as RoboSherlock hypothesis generators and annotators. Third, specifying the control flow for complex perception tasks in terms of RoboSherlock pipelines. RoboSherlock already provides type hierarchies and wrappers for PCL and ROS data structures and modules. The control flow specified in RoboSherlock pipelines are then applications on top that can offer very flexible and task adaptive processing for semantically rich problem formulations.

A short note on RoboSherlock types: Throughout this thesis, existing RoboSherlock types will be written in a special `typeface` and are explained in more detail in Chapter 3. Annotator names are written the same way as their distinction will be intuitively clear, but they are described as well in Chapters 4, 5, and 6. In both cases we believe their semantic meaning will be clear from their name or the context.

2.1 Example

Let us explain the principles underlying RoboSherlock using an illustrative example. The robot is looking at the kitchen counter and acquiring an RGB-D scan of the scene with its Kinect sensor (*cf*. Figure 2.1(1,2)).

This scan is supplied to RoboSherlock Analysis Engines (AE), which can themselves be aggregations of other Analysis Engines or "primitive" (*i.e*. singular) Annotators (*cf*. Figure 2.1(3)). AEs can therefore be as simple as a gaussian blur filter or a complete object reconstruction pipeline. The original input data and all reconstructed analysis meta data is stored within a central, cross-referenced data structure called Common Analysis Structure (CAS).

The availability of a Semantic 3D Map of the environment enables us to presegment data points into 3 distinct kinds of data components: All points which coincide with the 3D model of the environment can be treated as *known* structure and labeled or filtered accordingly. Furthermore, the specification of *regions of interest* in terms of parts within the map enables filtering of data points which fall within these regions. Examples include segmenting objects on top of all counter and table tops, or locating handles in front of drawers. In both examples, spatial volumes are expressed in coordinates relative to the various parts in the map, which can even change in world coordinates if the respective parts are articulated. Last, all remaining points can be

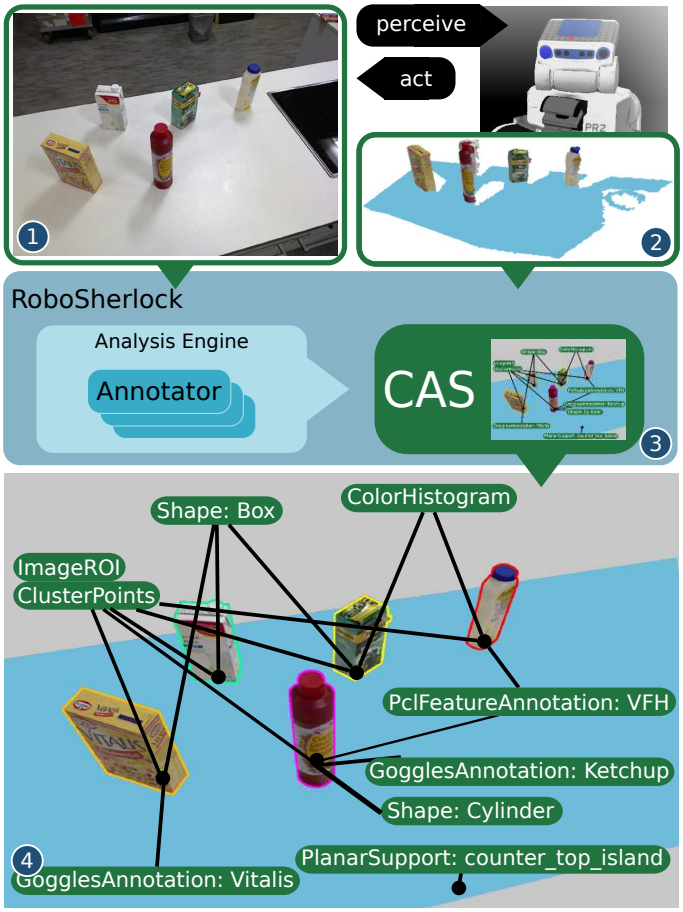


Figure 2.1.: *Perception as Unstructured Information Management: Common Analysis Structure (CAS) holds the original high-res camera (1) and depth (2) images, which act as input documents for our* RoboSherlock *system (3). The CAS (4) also contains information concerning known structure (such as the 3D semantic map, light blue), and irrelevant regions (gray), as well as Annotations referencing various parts of the document (green labels).*

treated as *irrelevant* and subsequently disregarded. These three kinds of components that make up a scene are depicted in Figure 2.1(4). Regions irrelevant to the task such as background clutter, floors *etc*. are shown in gray and are an immediate function of the specified regions of interest and is thus not interpreted by RoboSherlock. Known structure already contained in the static environment model is shown in blue. Objects within the region of interest are shown in their original texture, outlined with colors depending on their cluster association.

A variety of annotators are then employed to analyze the various object observations, and the results are stored in the CAS, referencing the respective segments of input data. In our example, irrelevant and known regions are marked (together with a `PlanarSupport` annotation for the counter top), and each object observation has one or more annotations associated, which range from the segmented data (*i.e*. `ImageRoi` and `ClusterPoints`) to simple metrics (*e.g*. `ColorHistogram`), 3D shape descriptors (`PclFeatureAnnotation`), web service result (`GogglesAnnotation`) and classification (`Shape` Annotation obtained through VFH `PclFeatureAnnotation`).

This makes for a very rich description of the objects of use in our scene, and it is able to capture very different aspects of information in an organized, reusable and extensible manner.

2.2 RoboSherlock Principles

We believe a proposal for a perception system as outlined in this thesis constitutes a paradigm shift in approaching perception systems. We identified 3 principles that form the foundation of this novel paradigm.

Principle 1: Perception as unstructured information processing

The notion of unstructured information has its origin in document content analysis. It is defined as information without an underlying a-priori data model or which cannot be easily expressed in a relational table, and has usually been associated with textual data.

Analogously we consider sensor data to be an "unstructured information document". An RGB-D scan can be regarded as a large set of raw data points and low-level features which are a function of latent underlying structure in the real world, such as furniture or other objects.

In general, RoboSherlock considers an RGB-D scan to consist of three kinds of subcomponents: (1) components that are already present in the static model of the environment and can therefore be used to structure the sensor data, (2) clutter or generally regions which are of no interest to the task, and most importantly, (3) information which is vital to solving the given problem. In a first approximation, we consider clusters of RGB-D points that are supported by horizontal planes of interest, which are tables, kitchen counters, shelves, *etc*. These clusters are assumed to be the objects of daily use that are to be manipulated by the robot. However, the RoboSherlock framework is not limited to this view of the computational problem.

Principle 2: Annotation of object hypotheses through ensembles of experts

In RoboSherlock, Annotators are Analysis Engines, which can either be primitive (consisting of a single processing block) or aggregate, meaning multiple (primitive or aggregate) engines orchestrated by a Flow Controller. They examine a snippet of the raw document or other annotations and enrich the CAS with new annotations relating to their field of "expertise".

Three examples are shown in Figure 2.2, which depicts a point cluster of a cereal carton, for which a feature annotator creates a `PclFeatureAnnotation`, which in turn gets used by the classification framework. It creates a `ClassificationAnnotation` of type box, with confidence 88.9%.

On the left hand side, the Goggles annotator fetches the corresponding image region from the higher-resolution camera, uploading it for analysis by Google Goggles (*cf*. Sec. 5). As can be seen, the resulting `GogglesAnnotation` contains interesting information such as a brand logo and a product link.

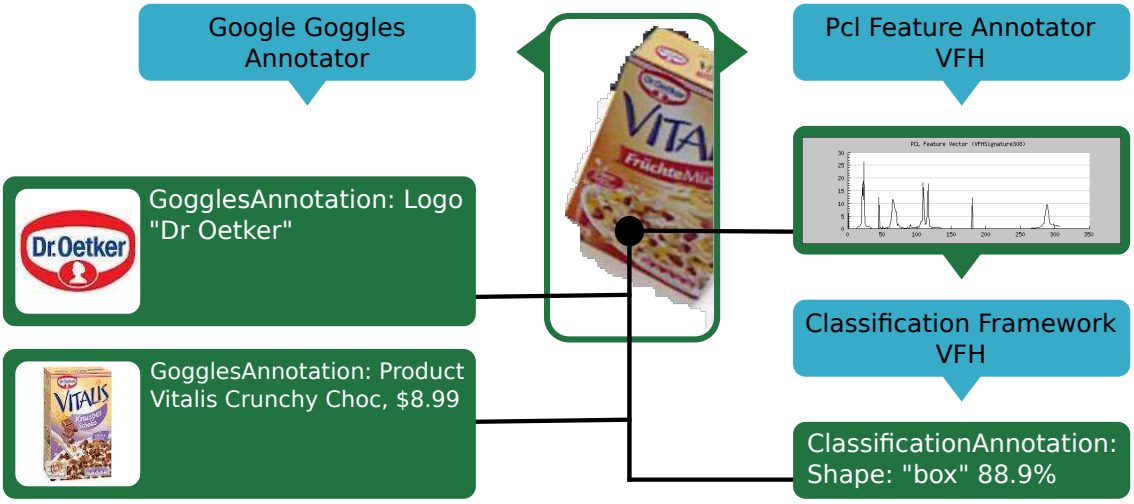


Figure 2.2.: *Annotators (blue) take an artifact, in this case a cluster of a cereal box, from the CAS. They create Annotations (green), which reference the artifact and can be used in different Annotators. The example shows the VFH feature vector, a shape classification, as well as two Goggles results.*

Principle 3: Information fusion and hypothesis selection

Results obtained from complementary or competing methods need to be merged and filtered in order to arrive at a final decision. This is essential for combining strengths and mitigating weaknesses, thus improving performance over the individual methods.

The mechanism enabling this is self-analysis of the system's competences and abilities. This is made possible by attaching confidences and other quality measures to both processing modules (Annotators) and their statements (Annotations) by collecting respective statistics. These can also be used in algorithm selection by reasoning over available computing resources and expected returns [46].

Examples include the synergistic effects of drawing on multiple heterogenous object descriptions, but also the fusion of different segmentations.

Another important manifestation of this principle is the *a priori* selection of algorithms, *i.e.* not only merging available object Annotations, but choosing the subset of Annotators to execute beforehand. This can be done using Flow Controllers that *e.g.* can stack multiple Annotators in a decision tree, where early decisions can prune

entire trees of processing steps. If an Annotator can decide early on that the object in question is very flat, *e.g*. a piece of paper, computing various 3D point descriptors is a waste of resources.

One can think of this 3rd principle from a different angle as well: instead of deciding which pieces of information to keep, the question could be formulated to decide which pieces of information to *forget*, which can be far easier to answer.

2.3 System Overview

The basis for our proposed system consists of the Apache UIMA open source project for Unstructured Information Management[4]. UIMA is a software framework for the analysis of unstructured data such as text documents, video or audio data. It is a pluggable architecture allowing collaboration and modularization, and has been proven to be a powerful tool for unstructured information processing. Existing UIMA modules center mostly around text and natural language processing, but the structural building blocks are independent of the problem domain. We propose to treat point cloud and camera data in robot systems as unstructured information sources, to leverage the existing infrastructure of UIMA and to extend it with perception-centric data structures and building blocks.

At the core of UIMA is a programming-language-independent meta-type system which describes the format of input documents and the metadata that results from analysis. We will outline our proposed type system for object perception in Chapter 3.

Documents, or sensory input data in our case, are called Subjects of Analysis (SofA) and are stored in the CAS. Additionally, the CAS holds a graph of Annotations to these documents, which are registered in Index Repositories. These annotations usually reference a subset of the SofA, such as a region of text or a mask in an image.

Concerning the terminology, there are several kinds of processing modules defined within UIMA which are aggregated in a Collection Processing Engine (CPE), as shown in Figure 2.3: *Collection Readers* form an interface to a collection of documents that need to be analyzed. They know how to parse a particular data source and create CASes with documents that are in turn processed by *Analysis Engines* (AEs). An Analysis Engine can either be a single Annotator or an aggregate of multiple Annotators, and analyzes the CAS contents, enriching it with additional information. Annotators – experts – can filter the contents to obtain the data relevant for their tasks, create new SofAs, annotations and index repositories. A special module is the *CAS Consumer* which converts relevant information from the enriched CAS into an application specific data structure such as a search index, a database or an ontology. The actual data flow and algorithm scheduling is orchestrated by a *Flow Controller*, which may decide to invoke AEs serially like a pipeline or dynamically based on the contents of the CAS.

[4] http://uima.apache.org/

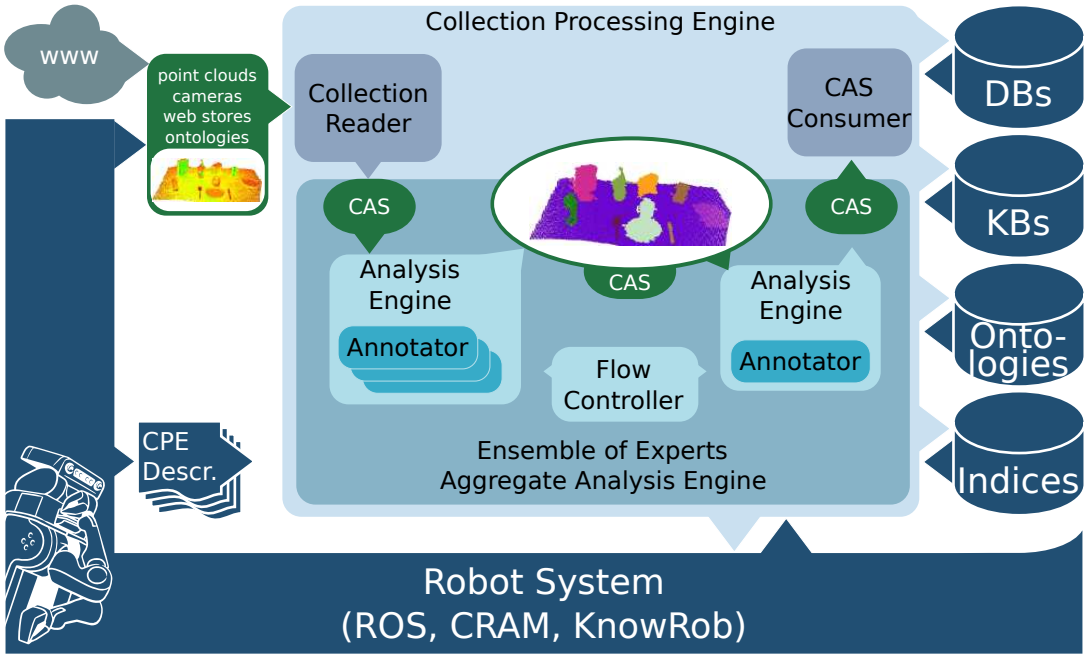


Figure 2.3.: *Schematic overview of our UIMA-based object processing framework. Unstructured information from the robotic process or online sources is processed by Collection Processing Engines described in a CPE Descriptor, utilizing an ensemble of experts to annotate unstructured data. Results are stored in the CAS and made persistent in databases, knowledge bases, ontologies and indices.*

In contrast, data flow in *e.g.* ROS is very rigid, and message channel ("topic") connectivity encodes semantic meaning of data. Addition or modification of information to messages can be cumbersome.

The type system defined for an application is the basis for the CAS. The CAS is an object-based data structure that is able to represent objects and their properties as defined by the type system. The basic structure is a graph of objects, instances of thusly defined types, which is organized by Index Repositories. Each SofA holds one or more Indexes which provides the only way for other Annotators to locate existing Annotations. We will provide more detail concerning the CAS and Index Repositories in Chapter 3, where we also describe the type system we developed for our perception system.

One can think of the CAS as a mixture between message passing and black board systems. An interesting aspect is that Annotators can also act as CAS multipliers, *e.g.* an annotator can collect percepts of the same object from different views over time, and generate an object-only CAS for storage in the object database describing the instances of things found in a given environment, or to enable CAD model reconstruction, feature-based machine learning or online CAD model fitting.

Another important concept in UIMA and RoboSherlock is that of a *Collection Processing Engine* (CPE). A CPE wraps a Collection Reader (a source of artifacts to be analyzed), a set of Analysis Engines, and a CAS Consumer into a complete unit that can be instantiated and executed by a Collection Processing Manager (CPM). While it is possible to construct such CPEs programmatically, we used CPE descriptors (an XML-based description format) that reference the various descriptors for type systems, Annotators and so forth to be included, and declares how they are combined in a CPE. A CPE factory can read this description and instantiate and deploy the required classes, and the CPM takes care of the execution in a (potentially distributed) work flow, CAS management, error handling and collection of statistical information.

In our scenario, a typical Collection Processing Engine can be sketched as follows: First, a CAS is created that holds the Kinect sensor data as well as other interesting internal information from the robot (*e.g.* localization data). This CAS is processed by Annotators which create hypotheses where objects could be located (*cf*. Sec. 4.1). Those potential objects are then processed and annotated, creating rich feature representations (*cf*. Sec. 5), which are then analyzed in total by entity resolution and tracking modules (*cf*. Sec. 6.5), which strive *i*) to explain the sensory input semantically; *ii*) to find consistent belief states about which objects are located where; and *iii*) to collect data about the environment, such as to reconstruct more complete object models, find object lifecycles or common storage locations. In the end, the resulting belief states are collected in a data base for convenient visualization, reuse in later processing iterations, and to serve as data collection for different research scenarios (*cf*. Sec. 6.6).

This system allows bottom-up programming to extract primitive object annotations, then matching it with richer descriptions from *e.g.* online knowledge bases, which yield priors on how to proceed with processing (top-down). This combination of

top-down and bottom-up processing is what is assumed to be done by humans as well [42].

Combining these principles makes the construction of powerful mechanisms possible, extending these data-driven object descriptions with descriptions of object properties, such as volume, more specific names, prices, ingredients, product images, or even online search queries for additional resource retrieval:

- text recognition (OCR) on a bottle label can be linked to ingredients list, container volume or price from an online store (*cf*. Sec 5);
- barcodes identifying an product very specifically can be used in publicly available product registries;
- through the use of ontologies (OWL-DL [155]), the concept of perishability of milk can be grounded in the observation of a specific brand of milk.

Note that in the current state of the project, CPE descriptors are manually created to construct different processing pipelines. In the future RoboSherlock should be extended to allow a higher-level reasoning module to construct them on-the-fly to custom-tailor a high-performance pipeline to a certain object retrieval, reconstruction or scene analysis task.

Since in this work we make use of a large amount of pre-existing algorithms in the areas of image understanding and 3D data processing, we rely on open source libraries such as OpenCV [19] and the Point Cloud Library (PCL) [130]. For a full list of open source software we use, we refer to Section 9.5.

Chapter 3

A Type System for Service Robotics Perception

In the past, many robotics middleware frameworks have been developed and published in the community. Examples include Player [48], YARP [91], and ROS [119]. For all their differences, there are several core principles these systems have in common:

- Modularity through a component-based structure allows interoperability of modules from different origins, reusability of functionality, and portability across platforms.
- Abstraction of hardware through drivers with generic interfaces enables the implementation of functionality more or less independent of the actual embodiment in a physical system.
- Communication middleware is a central aspect in the implementation of these principles to facilitate communication between sensors, actuators and processing modules and for distributed deployment on multiple networked machines.

While the aforementioned middleware systems vary greatly in their implementation details, communication is in general modeled after a client-server (or sender-receiver) paradigm, with usually long-lasting communication links. A more detailed comparison of existing middlewares is given in Chapter 9.

Let us consider ROS as an example. ROS uses a powerful but somewhat rigid communications paradigm where each “node” (a processing module that encapsulates one module of functionality with a clearly defined interface) communicates with other nodes through named and typed pipes called “topics”. Message types have to be

defined at compile time, and topic connections are created at run-time through a publisher-subscriber model. Since this model has strict sequential initialization requirements, starting a network of nodes can be tricky. Additionally, it is impossible to convey additional information that was not envisioned at compile time. Take as an example a segmentation node that publishes a point cloud topic. If that node were to include additional information, *e.g.* a semantically meaningful string, it would have to create a new message type that encapsulates the point cloud and a string. This is not a major concern if one has control over the code of all involved nodes, however this is in general not the case and directly contrary to the modularity principle as implemented in ROS. Hence, many nodes express a certain level of semantics through topic names, which is obviously neither flexible nor powerful enough for complex situations, or the system includes some form of *topic relays*, where various versions of similar information content is present in the system, leading to redundancy and increased bandwith requirements.

In contrast, in RoboSherlock, a potentially very high number of Annotators analyze the same artifact, so *information sharing* is a central aspect of communications. We therefore use the communication model employed by the IBM Watson project, built around the Common Analysis Structure. The CAS can be thought of as a blackboard system, which collects all information in a central data structure and is being passed between Annotators (as opposed to message passing, where all information is sent and received in a decentralized and distributed manner[1]).

The CAS is hence the central data structure within RoboSherlock (and UIMA in general). It is shown schematically in Figure 3.1 and consists of the following elements:

- The CAS contains one or more *SofAs* (Subjects of Analysis), which in our case usually contain sensory input data, *i.e.* we can have an image SofA, a depth image or point cloud SofA, or a normal map SofA, where each "pixel" contains the normal vector of the corresponding point in the point cloud SofA.

[1]Note that we do not consider one paradigm better in general than the other. Message-passing based middlewares such as ROS or YARP work perfectly well for a general-purpose robotics system. At a high abstraction level, these systems are concerned with processing various streams of sensor or control data, whereas RoboSherlock's processing model is centered around *document analysis*.

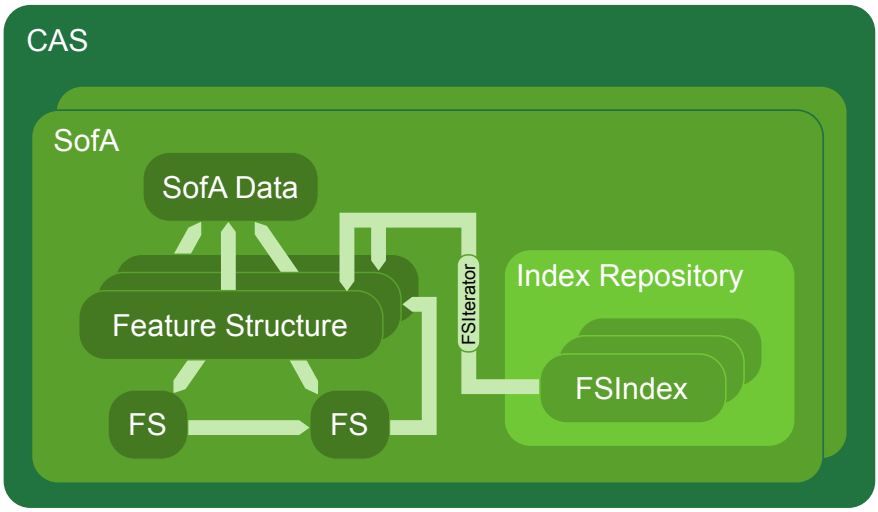


Figure 3.1.: *The Common Analysis Structure (CAS) and its components.*

- Each SofA has an object graph of *FeatureStructures* that represent Annotations of the document stored in the SofA. They are instantiated types defined in the type system, can have internal fields or properties and reference other FeatureStructures or a segment of the SofA data.
- When an Annotator creates a FeatureStructure, it is registered in one or more *FSIndexes* within the Sofa Index Repository. Each index allows retrieval of Annotations of a certain type (or subtypes) by providing a (potentially sorted) iterator over all FeatureStructures registered in that index. One could have an index that contains all objects in a scene, sorted by their size, and another one storing only translucent ones. Additionally, it is possible to apply filters to *e.g.* retrieve all Annotations of a specific subtype.

As can be seen, the CAS can represent complex object graphs and is a general and flexible data base with the ability to store arbitrary data structures. Its *application* or specialization to a problem domain is defined by a *type system* that specifies a taxonomy of application-relevant types that can be instantiated as FeatureStructures. In essence, the type system serves as an object schema for data stored in the CAS.

Hence, the type system is a fundamental component in RoboSherlock, allowing independent development of components while enabling very heterogeneous data rep-

resentation. At the same time, it defines a common language to allow Annotators to *represent* and *share* their results.

The type system is defined in a set of XML files, which makes it programming language agnostic. UIMA provides among others C++ and Java bindings to wrap or convert these meta types into native types. Note that the same CAS can be shared between both C++ and Java Annotators, and both can read and write to it through their native wrappers.

UIMA defines a set of primitive types for integer and floating point numbers of various sizes, and strings. It furthermore supports composite structures, as well as fixed-sized arrays and dynamically-allocated linked lists of them, similar to ROS message types or in fact most type systems.

3.1 RoboSherlock Convenience Wrappers to UIMA Interfaces

Before we will describe the actual type system in the next section, we want to present a set of wrappers and programming practices that we created to simplify working with the interfaces available in UIMA.

Often when working with UIMA data structures, Annotators follow a certain pattern to access data. As an example, consider the relatively simple problem of obtaining the timestamp at which a `Scene` object in the CAS was acquired. The following code snippet shows a typical way of doing this using existing UIMA programming interfaces:

Listing 3.1 Example of inconvenient interfaces for accessing the CAS.

```
1 CAS* view = 0;
2 Type SceneType = tcas.getTypeSystem().getType(UnicodeString("ias.uima.
    scene.Scene"));
3 Feature timestamp = SceneType.getFeatureByBaseName("timestamp");
4
5 if (!tcas.getView("cloud", view)) {
6   throw std::exception("View not present");
7 }
```

```
8
9  FSIndexRepository &index_rep = view.getIndexRepository ();
10 FSIndex index = index_rep.getIndex("general", SceneType);
11 FSIterator it = index.iterator();
12
13 FeatureStructure fs;
14 if (it.isValid()) {
15   fs = it.get();
16 } else {
17   throw std::exception("FSIterator not valid");
18 }
19
20 if(!fs.isValid()) {
21   throw std::exception("FeatureStructure not valid");
22 }
23
24 long time = fs.getLongValue(timestamp);
```

The code can be broken down into several blocks: Lines 1 to 3 deal with retrieving type and feature objects from the type system, lines 5 to 7 accesses the CAS to retrieve the SofA, and the code in lines 9 to 11 queries the Index Repository to retrieve an iterator over the Scene FeatureStructures. This is followed by lines 13 to 22, where the first Scene object is retrieved and tested for validity. Finally, in line 24, we can read the timestamp value from the Scene FeatureStructure.

There are several observations that we would like to point out:

- An obvious source of errors is the use of string identifiers in line 2 to specify the Scene type, in line 3 to denote the Feature name, and in line 10 for the FSIndex name. This means that changes to the typesystems have wide repercussions, as all occurrences of these identifiers need to be updated accordingly.
- Another observation is the fact that we need to write a lot of preparatory and error handling code which creates clutter and obscures the intention of what we are trying to achieve, *i.e.* reading the Scene timestamp value.

- Due to the low-level access to the CAS, we need to create temporary or auxiliary variables for intermediate results, *e.g.* in lines 1,2,3,9,10,11, and 13.
- Finally, the last line shows that the FeatureStructure offers differently named getter (and setter) methods for accessing struct members of differing types.

To facilitate dealing with data access in RoboSherlock, we therefore implemented several C++ wrappers that offer both convenience in reduced typing and code readability, but also robustness to common sources of errors, such as advanced type safety at compile time, and more transparent exception handling.

3.1.1 FeatureStructureProxy

The central machanism we introduced is the notion of a templated FeatureStructureProxy class, which manages the underlying FeatureStructure object and offers simple and intuitive interface dealing with memory allocation within the CAS, uniform getters and setters, and iterators for sequence types. It also contains exception handling internally to reduce repetitive code.

The mapping of these proxy objects to objects in the type system is performed using type *traits*, which means that all string identifiers are only specified once (where the type trait is defined). This transforms all lookups using string identifiers at run time to compile-time type checks performed by the C++ compiler.

Using the FeatureStructureProxy mechanism, we can transform the previous code listing into a much more compact and readable piece of code:

Listing 3.2 The FeatureStructureProxy interface offers convenient and expressive access to the CAS.

```
1 ias_uima::SceneCas scas(tcas);
2 ias_uima::Scene scene = scas.getScene();
3 long time = scene.timestamp.get();
```

Again we show how to retrieve the Scene object from the CAS and read the timestamp field. In line 1, we create a RoboSherlock wrapper for the CAS which we called SceneCas. This class simply acts as a facade for the CAS, and multiple similar

wrappers can be created for the same CAS, each offering a unique "view" on the CAS that is custom-tailored to the application. In this case, the SceneCas wrapper offers a scene-centric view onto the CAS, with direct accessors into the relevant feature structures, encapsulating the complexity of type conversion and feature structure graph traversal. Member functions therefore bear semantic meaning and include *e.g.* `setDepthImage()`, `addCluster()` or `getScene()`.

In line 2, we retrieve the Scene object in the SceneCas, which contains FeatureStructureProxy members for each Feature (*e.g.* timestamp) stored in a Scene. We can therefore access the timestamp field using a uniform `get()` method which internally calls the correct function (*e.g.* `getLongValue()`) to read the requested value.

3.1.2 FeatureStructureConversion

To facilitate conversion between native ROS message types (in fact, all possible native types, *e.g.* PCL or application-specific ones) and RoboSherlock types, we created a templated *conversion* interface that again leads to more concise and less repetitive code. The `FeatureStructureConversion` class is templated on the native type and has two functions, `from`, which takes a RoboSherlock type and returns the corresponding native type, and `to`, which performs the inverse operation. The two functions take care of all allocation and memory management issues, as well as conversions between the two representations.

As an example, consider the conversion interface for the OpenCV image type `cv::Mat`. The function `to` gets passed the image and a reference to the View the FeatureStructure should reside in. The image is converted to the corresponding CAS representation by allocating memory in the CAS and copying over the image data and all metainformation necessary to invert the transformation. This includes image dimensions, number of channels *etc*. The inverse function `from` reads these image properties from the CAS, creates and initializes an appropriate `cv::Mat` structure, and copies the pixel data over. As a result, the interpretation of data stored in the CAS in terms of application data types only happens at a single location in code, at the definition of the conversion interface for that type. Again, the same advantages as in the case

of FeatureStructureProxy can be observed: reduced complexity, higher robustness, readability and less sources for error.

Here is an example of storing a `cv::Mat` (named "input")in the CAS ("cas") that can then be referenced and accessed via the corresponding FeatureStructureProxy subclass `ias_uima::Mat` ("mat"):

Listing 3.3 Data conversion from and to CAS structures can be tedious.

```
Type type = cas.getTypeSystem().getType(UnicodeString("ias.uima.cv.Mat"
    ));
FeatureStructure fs = cas.createFS(type);

ias_uima::Mat mat = ias_uima::Mat (fs);

std::vector<char> data(input.rows * input.step);

int rowsize = input.elemSize() * input.cols;
for (int i = 0; i < input.rows; i++) {
  memcpy((void*) &data[i * rowsize],
      (void*) (input.data + i * input.step), rowsize);
}

mat.data(data);
mat.rows(input.rows);
mat.cols(input.cols);
mat.type(input.type());
mat.step_size(input.step);
```

Using the conversion interface, more specifically the `to` function, we can replace this code with the equivalent:

Listing 3.4 The FeatureStructureConversion mechanism provides a much more compact interface.

```
// typedef FeatureStructureConversion<cv::Mat> MatConversion;
ias_uima::Mat mat = MatConversion::to(cas, input);
```

The typedef in the first line resides in a shared header file. The implementation of the conversion interface obviously relies on the type trait for specifying the type, as mentioned above.

The effect of the two concepts – proxy and conversion interfaces – is significantly reduced code complexity and verbosity, which leads to considerable improvements in code development and readability. Also, once the wrapper objects are verified to be correct (*e.g.* through unit testing), we gain type safety by not strictly relying on string objects to identify types anymore.

The obvious benefits of the approach come at a slight cost in that each type defined in the XML-based type system needs a class that defines the corresponding FeatureStructureProxy object, as well as a type trait definition which ties the proxy object to the type's string identifier. Without this, we would have to resort to the original UIMA interfaces. There is however a clear one-to-one mapping between the two representations, so the proxy definitions can be created automatically from the XML definition of the type. The only time where this automatic process needs to be augmented manually is in cases where the proxy object is supposed to offer additional functionality. As a simple example, a temperature FeatureStructureProxy might offer methods to set and get temperatures in Fahrenheit or Centigrade with automatic conversion between the two scales.

We consider these RoboSherlock improvements to the C++ programming interface an important contribution to UIMA, as the aforementioned benefits help greatly in being able to manage the overwhelming complexity inherent to a project as ambitious as RoboSherlock.

3.2 Type System

With the aforementioned improved interfaces in mind, we will now go over the various types that make up the type system defined within RoboSherlock.

The type system in RoboSherlock is designed along two dimensions: We strive to be able to represent all possible types we can encounter in the libraries we rely on, *i.e.* ROS, PCL and OpenCV. We consider these to be basic or raw types that repre-

sent a particular piece of data. On the other hand, we need to be able to express higher level types that represent a semantic concept in our problem domain, which is mainly focused on perception of objects of use in every day environments. We identified three major high-level concepts: object hypotheses, annotations as a result of analyzing an object hypothesis, and scene annotations which encapsulate multiple object hypotheses and their analysis results.

We will highlight each in the following sections.

3.2.1 Basic (Raw) Data Types

In this section, we will give details on how we can incorporate external data types in RoboSherlock. We consider these to be "lower-level", data-centric information, as opposed to "higher-level" concepts that bear more semantic characteristics. Specifically, we are able to manage data from various open-source libraries and projects that we rely on for the development of RoboSherlock: ROS, the Robot Operating System, PCL, the Point Cloud Library, OpenCV, the Open Source Computer Vision library, the *tf* transformation library, Bullet physics and the Eigen linear algebra library (*cf*. Section 9.5).

3.2.1.1 ROS Message Definitions

One of the most important aspects of a perception system for assistive robots is the integration within a larger context of robot control. As ROS has in recent years taken a lead role as a robotics framework with support for a wide range of robotics platforms and a strong community with numerous contributions in terms of functionality and behaviour, RoboSherlock provides direct support for ROS message types.

We see it as an important building block to be able to represent and manipulate all information present in the robot system within RoboSherlock. Since information in ROS is communicated via typed topics, this amounts to mapping the native ROS message types to RoboSherlock types. As a convention, we consider it sensible to mostly manipulate native data types (as opposed to creating our own, non-compatible representations), since we can make use of client libraries written for them, *e.g*. manip-

ulating coordinate system transformations using the *tf* library, or the abundance of PCL algorithms for dealing with point cloud data.

It must be noted that the message definition of a ROS type follows similar patterns as types in UIMA: They are defined in programming-language-agnostic files and support similar primitive types and similar composability thereof in arrays and structs. For this reason, we created an automatic code generator that can collect and convert all ROS message types defined in a given ROS installation to their respective isomorphous representation in RoboSherlock.

The code generator uses core ROS functionality, the python module `rosmsg`, to find and iterate all message types defined in a ROS package hierarchy. It consists of the message collector, a UIMA type system writer, and a FeatureStructureProxy writer.

The *message collector* can be parameterized to either collect all available ROS types, or a specific (sub)set of message types. In the latter case, it will recursively scan the message types of interest, identify dependencies and incorporate these dependencies as well, computing in essence the transitive closure of message type dependency. As an example, one could specify to be interested in joystick and point cloud messages, and would obtain the following list of types to be converted into RoboSherlock types: `sensor_msgs/PointField`, `std_msgs/Header`, `sensor_msgs/PointCloud2`, and `sensor_msgs/Joy`, where the former two types are dependencies of the latter two. Along with each of these types, the message collector stores the message definitions (types and names of each field defined in them).

The *UIMA type system writer* takes the list of message definitions produced by the first step and converts them into a XML type system file (specified by the user) which can then be included in the RoboSherlock type system. To this effect, it relies on a predefined table containing equivalence mappings of primitive types between ROS and UIMA, *e.g.* "`float64` → `uima.cas.Double`". There is a small set of ROS message types requiring special treatment since they were treated as built-in types in early ROS releases: `duration`, `time`, and `header`, so there are custom mappings for these as well, *e.g.* "`time` → `ias.uima.ros.std_msgs.Time`".

The *FeatureStructureProxy writer* takes care of prividing the corresponding FeatureStructureProxy C++ headers for each type, as well as the FeatureStructureConversion. Together, the three components of the code generator make it completely trans-

parent to access, create and process ROS data structures within UIMA as well as being able to store them and retrieve them from the CAS via the conversion interfaces. In our scenario, we found the most important ones to be sensor messages (*e.g.* images, point clouds, joystick messages *etc.*) and *tf* messages which communicate transformations and kinematic chains for all coordinate frames in a ROS system.

The net result of automatic type translation combined with the proxy and conversion interfaces is almost complete transparency for the user: An annotator can freely subscribe to and publish ROS topics, create and access ROS messages and data types, make use of existing libraries, and store any wanted information in the CAS via very compact and common-sense interfaces.

3.2.1.2 Additional Data Types

In the case of PCL and OpenCV, automatic type converter generation is not as straightforward, since there is no notion of external type definitions as is the case in ROS. We therefore provide manually-created wrappers for the most important data types we encounter with these libraries.

- For PCL, the central data structure is the `PointCloud` which is in turn templated on the actual `Point` type that is stored in it. We provide RoboSherlock `PointCloud` and `PointIndices` types along with FeatureStructureProxy and FeatureStructureConversion that can be used analogously as described in the previous subsection. The only specialty is the reliance on the definition of type traits for all required Point types. The most common Point types are already provided, however if the user defines their own, a type trait must be provided if it is to be stored in the CAS.

- Furthermore, PCL features can be represented using `PclFeatureAnnotations`. The same pattern of using type traits is used to create a type-safe mapping between feature type string constants stored in the CAS and actual C++ classes used within Annotators by the use of a `FeatureDelegate` mechanism. All `FeatureDelegates` inherit an interface governing setters for the inputs and parameters and a *compute* method to invoke the feature estimator.

- Similarly, OpenCV's image types are supported in RoboSherlock, both for `cv::Mat` and the templated version, `cv::Mat_<T>`.
- For math data types, we support *tf*, Bullet and Eigen vectors and transformations.

3.2.1.3 Database Storage

In many cases, it is desirable to store information extracted over time in a persistent database. Example cases include:

- the recording of scene observations over time to offer reasoning over the temporal progression of the state of the environment;
- to identify object lifecycles; *E.g.* a plate can be usually found on a table, then later in the sink or the dishwasher, and finally in a designated cupboard;
- to group different partial views of a certain object for subsequent 3D model reconstruction;
- to act as a cache for expensive analysis operations;
- as a debugging tool due to the fact that a structured "log" of analysis results can be conveniently interpreted or visualized using a variety of existing data base interfaces;

For these purposes we use an object-oriented MongoDB database and created an automatic conversion mechanism modeled similar in spirit to the aforementioned conversion interface. Due to the fact that all CAS and FeatureStructure objects carry self-describing type information (or reference the type system to this effect), we created a generic, templated set of classes to offer fully automatic conversion between all RoboSherlock types, present and future, and MongoDB.

Note that all RoboSherlock types can inherit from an `Identifiable` class, which contains an object ID field to be used in MongoDB. This ID gets set when the object is stored in the data base; If the object is retrieved and changed at a later point, the ID

can be used to reference and update the appropriate, correct MongoDB object even when it has passed through a number of Annotators.

We will go into more details concerning the MongoDB backend in Section 6.6.

3.2.2 Object Hypotheses

Having covered the "raw" types that hold individual pieces of data, we will go into detail with some of the higher-level, more semantic types in this and the following subsections.

To support the notion of an object hypothesis generated from the various data sources, we created a set of types with specializations for the respective contexts. To be able to express the notion of an *object hypothesis*, we must provide several data representations that are directly rooted in the respective sensor or domain characteristics. We need to be able to express object locations in *point sets* (or subsets thereof), *images* (or masked subregions thereof), and of course *locations* in space. Specifically, we defined the base type `ObjectHypothesis` and the following subtypes to be available for these purposes:

- `ClusterPoints` represents a cluster of 3D points or a subset of a 3D point cloud. There are two specializations: `StandaloneClusterPoints` simply contains a (usually small) point cloud, stored in a field `cloud` of type `PointCloud`, where as `ReferenceClusterPoints` references an existing point cloud object, and stores an index vector specifying the subset of points that make up the cluster (containing the fields `cloud` and `indices`).

  We employ these types for storing the result of the segmentation of a point cloud into clusters. Note that `ReferenceClusterPoints` can be converted to `StandaloneClusterPoints` on-the-fly, which is helpful when it is necessary to store the cluster independently from the scene and its attached (full) point cloud, *e.g.* to serve as input for an object model reconstruction process or when using algorithms that are written to work on standalone point clouds.

- For image-based segmentation, we offer a region-of-interest type `ImageROI`, which – analogously to the point cluster types – are subclassed into `Standa-`

`loneImageROI` and `ReferenceImageROI`. Additionally, we added a perceptual hash tag representing a highly binarized and low-resolution version of the image that can be used for fast comparisons of images. The available fields are thus `image`, `mask`, and `phash`.

- As the notion of an object hypothesis is very general, there are occasions where the former two types are not readily applicable. As an example, consider a higher level decision making process that executes a put-down maneuver on the robot. Even if the put-down location is out of view, this process can assert the presence of an object in space. Another hypothesis might arise from a human tracking process which observes the human placing an object in the scene.

  For these cases, we provide two types which represent a spatial volume in a specific *tf* coordinate system: `TFLinkBBoxROI` and `TFLinkSphereROI` contain a field `tf_frame` which names the reference coordinate frame. Additionally, they store a bounding box (in a field called `bbox`) in the former, or a sphere defined by `center` and `radius` in the latter case.

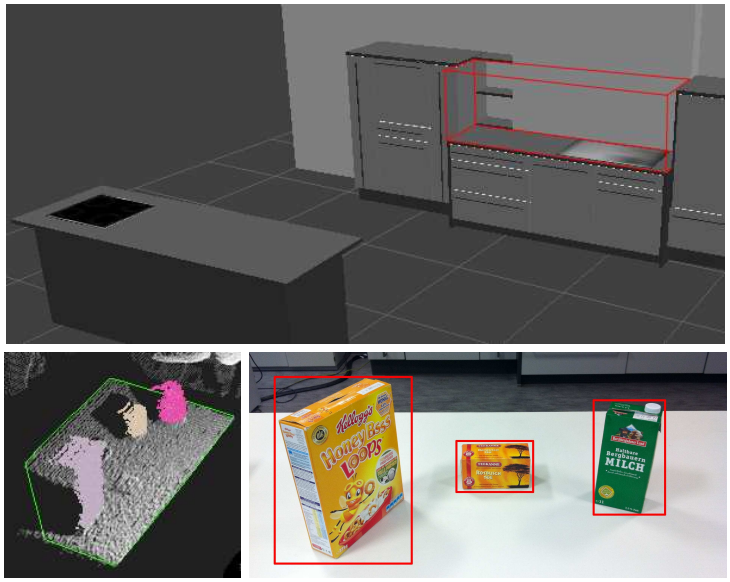


Figure 3.2.: *Object hypothesis annotations can hold various representations: a spatial volume relative to a* tf *frame (top), subsets of points in a point cloud (bottom left) or regions of interest in an image (bottom right).*

Figure 3.2 shows visual representations of these types.

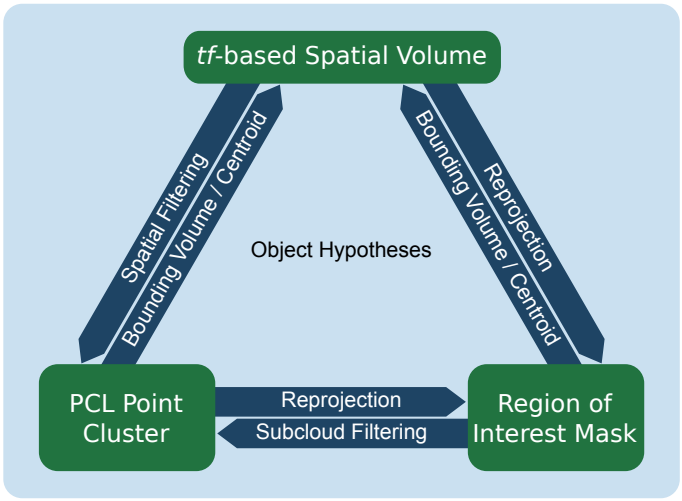


Figure 3.3.: *Object hypothesis annotations can be converted between each other given the robot's proprioceptive and calibration data.*

All of the `ObjectHypothesis` types are readily convertible between each other, where applicable. We provide methods that can reproject points into camera images given their relative calibration information and the kinematic chain between the sensors. Additionally, it is possible to create a `ClusterPoints` object that contains all the points within a spatial volume defined by one of the `TFLink` types, or vice versa compute the bounding volume of a point cluster in any given *tf* frame. This makes it possible to merge image-based segmentation techniques with 3D segmentation in order to compute multivariate object description taking both data sources into account, or to *e.g.* obtain a high-resolution image for a given point cluster (*cf.* Section 5.4 for a manifestation of this idea). An overview of the methods is shown in Figure 3.3.

In our experiments, we were able to express all object hypotheses that arose in our applications using these aforementioned types. However, it is not unthinkable that future scenarios have deviating requirements, in which case different subtypes will have to be designed.

3.2.3 Object Centric Annotations

Once an Annotator has identified an object hypotheses, there is an abundance of algorithms that can be used to infer more and higher-level information concerning

the identity, appearance or hidden parameters of the underlying object. While we attemp to cover a wide variety of possible use-cases, it is likely (or rather certain) that future applications will have to augment the set of available object-centric Annotation types. Currently, the following types are implemented.

3.2.3.1 Location Annotations

A very informative piece of information is a `LocationAnnotation` to express *where* the hypothesized object was detected. This can in fact already be annotated by the respective hypothesis generator in same cases. Segmentation processes that analyze spatial data can specify a centroid in the camera frame or even local coordinates in the supporting structure, *e.g.* a position on a table.

There are currently two subtypes: `SegmentationLocationAnnotation` can be used in conjunction with segmentation algorithms that define regions of interest. Consider a robot that has access to a semantically labeled 3D map of the environment (as created by a process as described in Chapter 7). There are a number of tables and counters that are specified as regions of interest, possibly containing objects of use. A segmentation algorithm can now create annotations that specify which parts of the input data contain points or image pixels from which region of interest. In our case, regions of interest (ROIs) are specified in terms of *tf* links, *e.g.* `counter_top_sink`. For every ROI encountered in the image data, we create a `SegmentationLocationAnnotation` that contains an image `mask` with all pixels that fall into the ROI's 3D bounding volume, and the name (*tf* frame stored in `frame_id`) of the ROI.

An alternative use case is for the case of 3D planar segmentation, where a model fitting approach is used to estimate a supporting structure plane model (*e.g.* table plane for objects or a door plane when detecting handles). This model is then used to locate objects protruding from the planar structure.

We created another, similar subtype, `TFLocationAnnotation` which we use for per-cluster annotations in this regard. Again, we store the respective ROI's `frame_id`. Additionally, we compute the cluster centroid and store it as a transformation matrix w.r.t. the *tf* frame, and a human-readable string that can be set for each ROI and that describes the geometrical relation to the supporting structure. This allows statements

such as “Cluster 3 was found on counter_top_sink” or “Handle was found attached to door_kitchen”.

For an example of such a semantic segmentation process, we refer to Section 4.2.2, where we use a 3D model and a shader-based algorithm to achieve semantically meaningful real-time segmentation of depth image and camera data.

In the future, these types or similar ones can be used to describe relative positions between pairs or sets of objects, *e.g.* for table arrangements like “The fork is left of the plate”.

3.2.3.2 PCL Feature Annotations

In terms of 3D spatial descriptors for point clouds, `PclFeatureAnnotation` is designed to hold any feature representation available in PCL (that is a subclass of `pcl::Feature`). The name of the type is stored in the field `name` (see the paragraph on `Point` type traits in Subsection 3.2.1.2), and the actual feature representation in `feature`. This can be used in conjunction with the `GenericFeatureFromNormals` Annotator described in Section 5, which can be parametrized to compute any PCL feature subclass that requires per-point normal information.

There is a wide and ever growing range of feature descriptors implemented in PCL. For an up-to-date list, we refer to PCL’s online reference[2]. Notable examples include feature descriptors from the Point Feature Histogram (PFH [127]) family, *e.g.* the Viewpoint Feature Histogram (VFH [126]), the Fast Point Feature Histogram (FPFH [128]), or the Clustered Viewpoint Feature Histogram (CVFH [1]). Other features offered within PCL include Ensemble of Shape Functions (ESF [173]), Signature of Histograms of OrienTations (SHOT [162, 163]) and Unique Shape Context (USC [162]).

The generic Annotator capable of computing PCL Features is described in Section 5.1.

[2] http://docs.pointclouds.org/

3.2.3.3 Google Goggles Annotations

As will be described in Section 5.4, we created an Annotator capable of querying the web service offered by Google Goggles that offers various image recognition tasks. RoboSherlock uploads an image of an object hypothesis to Goggles' server and receives a structured reply with a list of recognition results, *e.g.* similar images found on the web, text recognition, barcode recognition, logo/brand detection or matches with products on web stores.

The structure of responses from Google Goggles is modeled in `GogglesAnnotation` which is designed to store the extracted information within the CAS. The main fields are `category` which defines the *type* and `title` which holds the main *textual representation* of the response, *e.g.* the brand name for a logo result, the barcode number for a barcode result *etc.* Also, the image region where the respective match was detected is returned as well as a list of actions (*e.g.* a link for additional topical web search results, links to appropriate images, translation results for text recognition, *etc.*).

There are various additional fields that can change according to the context of the response category. A more detailed description of the Goggles response structure is given in the description of the Goggles Annotator in Section 5.4.

3.2.3.4 Classification Annotations

The PCL features computed for an object observation can be used by a classification node that creates `ClassificationAnnotations`. It is a type that contains the employed feature, classifier, and the resulting label. Additionally, it can store a list of class confidences for each class label & class accuracies given by the classifier. See Section 5.2 for details of the classification Annotator.

3.2.3.5 Tracking Annotations

For the purpose of tracking objects over time we offer a `TrackingAnnotation` (*cf.* Sec. 6.5), which for now simply holds a numerical tracking ID (`trackingID`) and an

identifier for the tracking algorithm (`annotatorID`) that was used. In the future, this can be expanded to represent more information, such as confidences or distributions over tracking IDs as appropriate.

3.2.3.6 Product Annotations

A `ProductAnnotation` is used by Annotators that can match object observations with a registry of products, *e.g*. from online stores or the Goggles response (*cf*. Sections 5.4 and 5.3). Since the kinds of information can vary greatly in this category, it is designed as a key-value store. As an example, keys could be *product category*, *weight*, *ingredients* and so forth.

3.2.3.7 Barcode Annotations

The `BarcodeAnnotation`, unsurprisingly, is used to store a barcode that was detected on an object observation. Again, one source can be the Goggles Annotator, but we have been using the zbar library[3] and the database from barcoo[4] in our lab to obtain more information about mainly food products. This is however outside the scope of this thesis and will not be discussed in detail.

As can be seen, RoboSherlock defines various object-centric annotations that cover a range of usage scenarios and are able to express various kinds of information, from low-level perceptual, spatial and appearance clues all the way to high-level semantic information such as names, ingredients, prices, manufacturer, text recognition and so forth.

All object annotations discussed here are supported by one or more Annotators provided in RoboSherlock and will be described over the following chapters. However, new annotations can easily be added or existing ones extended as needs arise in future development.

[3] http://zbar.sourceforge.net/
[4] http://barcoo.com

3.2.4 Scene Annotations

In the past sections, we have described several Annotation types that can represent individual pieces of information about perceived objects. However, we need to define a way of grouping these individual statements and attaching them to (the observation of) an individual object. Additionally, we need to be able to provide additional context, such as time of observation, exact locations as well as relative positioning with respect to other objects in the environment.

So in order to bridge the gap between statements *about* objects (above Annotations) to data structures that *stand for* the actual object, we created a `Cluster` type whose main purpose is to group and collect analysis results about a single object in the environment. It is "responsible" for a physical region in space (represented by one or more Object Hypotheses of their various kinds), and holds a list of annotations that where created during the analysis of these hypotheses. Furthermore, we store the transformation of the cluster w.r.t. to the original region of interest in which the object was detected (*cf*. Section. 4.2.2). For convenience (*e.g*. in visualization), we also directly store the centroid point of the point cloud cluster, if it is available.

Going up one level from object-based annotations, we created a type to represent whole scenes, which we defined as arrangements of objects (`Clusters`) at a certain time. This data structure is supposed to store *snapshots* of scenes that should be self-contained as far as possible to enable delayed interpretation of certain aspects, to allow for storage and later retrieval, or to be used in spin-off tasks that do not necessarily have access to up-to-date information on the robot's state.

The `Scene` type therefore contains: an array of `Clusters`, the robot's camera pose in world (in our case `"/map"`) coordinates, the timestamp of data acquisition, and an array of *tf* transformations.

Furthermore, we created a `Logging` annotation that can be treated like a text-based logfile. Annotators can use this to append their debug or analysis output to a scene's log, which makes it very convenient to quickly see which annotators were involved in the analysis of a scene or a certain object, what and how they performed, how much time was spent and whether anything went wrong. By attaching this information to the scene, this information can automatically be stored in the database along with

the rest of the data, so recreating an error scenario or quickly getting an overview of the path a piece of data followed through a processing engine can be done easily.

The reason to include an array of *tf* transformations in the scene representations follows a similar line of reasoning, but also has practical and pragmatic reasons: The way *tf* is designed, each *tf* listener internally maintains queues of transformations, which get updated as new transformation messages arrive via ROS topics. The client can then query these queues to retrieve a chain of transformations from source to target frame for a given point in time. However, storing all transforms in a robot indefinitely is not possible, so each listener maintains a certain queue length, *e.g.* 5 seconds. Elements of the queues which are older get deleted, which can lead to problems if succeeding Annotators – with their own *tf* queues – later on need to determine the state of the *tf* tree at the time of data acquisition.

We therefore added the ability for component processing engines (CPEs, *cf*. Section 2.3) to specify which transformation chains are of interest for subsequent Annotators. The ROS Kinect Bridge Annotator in turn performs the necessary lookups and stores the relative transformations in the scene. This of course requires that it is known for all later Annotators which transformations will be of interest to them. This is however perfectly possible in practice, since CPEs are generated by the task to be solved and are thus subject to change.

Additionally, the `Scene` is able to represent spatial (or other) object-object relations in a simple, probabilistic table, as can be seen in Table 3.1. Each line in the table contains the name of the property (*e.g.* left-of, on-top-of), the two objects for which the property holds, and a probability or degree of belief to which this property is true.

Table 3.1.: *Representation of spatial or other relations between objects as a table. Each row contains the name of a property, the objects for which the property holds, and a probability, confidence or degree of belief for this statement.*

relation	object 1	object 2	probability
left-of	cluster 17	cluster 3	82.4%
similar-appearance	cluster 12	cluster 3	64.1%
⋮	⋮	⋮	⋮

This representation is quite flexible, as it can be used to model all unary (by use of a special, empty symbol ∅) or binary predicates, and both probabilistic or hard facts ($p = 100\%$).

Note that for the scope of this thesis, we do not pursue the important topic of spatial arrangements for *e.g.* table setting scenarios, but we provide the representational capabilities to support them in the future. Furthermore, we use this predicate table representation for a different purpose, that of entity resolution, as detailed in Chapter 6.

Having such a representation available allows to separate the *creation* of such predicates (*e.g.* by reasoning on object locations) from the *interpretation* of them, *e.g.* for purposes of scene classification into breakfast scenario, meal preparation, family dinner *etc.* or for the inference of missing objects (*e.g.* "given a bowl, ceral, and spoon, which object is most likely missing?").

In conclusion, the scene annotation can represent a snapshot of the environment, important information about the state of the robot and the environment, hold references to all cluster observations found in the scene, along with their respective annotations. Furthermore, it can represent arbitrary object-object predicates and their distributions.

3.2.5 Belief State Annotations

In contrast to the `Scene` annotation, we also wanted to be able to express information that is not limited to a singular snapshot or observation, but to offer a data type that can express the complete current belief state of the world at a given time.

This exceeds the semantics of the `Scene` in several ways:

1. Objects that are out of sight given the current camera frustrum still need to be accounted for. This is also true for currently occluded objects. In layman's terms, the fact that the robot cannot *see* an object does not mean it's not there. It should still be able to provide information about objects that are *e.g.* behind the robot if he has perceived them earlier.

2. So far, we have considered each object hypothesis to be exactly that: an hypothesis that there could be an object. Synonyms would be an object observation or percept. There must be a clear distinction between this concept and that of an actual *object*. The former is always merely a function of the latter and as such not equivalent. However, for symbolic reasoning or grounding objects occuring in plans in perception data, we must at some point decide that a certain object observation along with all reconstructed meta-information was taken from a certain *object*. This process is often called perceptual anchoring [28], however many computer vision systems ignore this problem or equate the two concepts implicitly.

3. There are many cases, where a one-to-one mapping of observations to objects cannot be established without uncertainty. Examples include indistinguishable objects (*e.g.* a set of cups), or an ambiguity due to missing information. Consider the robot perceiving an object at location *A*. The location gets occluded and the robot sees a similar object at location *B*. It is at this point not clear whether the same object moved from *A* to *B*, or if there are now two objects, one at *A* and one at *B*. It is therefore necessary to express and maintain both hypotheses or the complete distribution over explanations.

We therefore provide a `BeliefStateInstance` type that represents the belief about the state of the world at a given time. In our context, it models several distributions: the mapping between object observations or clusters and the actual objects (*cf*. Sec. 6.5) and the mapping between currently observed clusters and cluster observations from the past (in a identity resolution tracking sense). This makes it possible to ask queries about objects (“where is X?”), or about clusters (“where was this object before?”) as well as about locations (“what’s currently on the kitchen table?”, regardless of whether it currently in view).

In this structure, we predominantly combine lists of object observations or objects:

1. The list of currently observed clusters.

2. The list of clusters observed at the last time step.

3. This list is augmented by the list of objects that we did *not* see at the last time step, but which we believed to still be present (*persist*) at an location observed previously.

4. The list of objects that the system knows of (the universe of known objects).

The elements in these lists are then connected in a graph by relations or predicates, in the same representation as spatial relations for a `Scene`. Once a new `Scene` of object observations arrives, it is compared with the lists of previous observations and of persisting objects to infer the equivalence between objects observed currently and previously. This can be thought of as a low-frequency tracking system, regardless of actual object *identity*, but by using as many of the reconstructed cluster annotations as possible to establish similarity or equivalence.

The second important distribution is that over equivalence of object observations and objects. For the scope of this work, we define an object in terms of object observations. That is, after tracking object observations over time, we can collect different views from different sensors with different reconstructions and features into a bag of observations, a *proto object*.

The notion of using proto objects is both convenient and powerful: It allows *bootstrapping* an object catalog from scratch, as the set of information an object represents is identical in structure to an object observation. Furthermore, it readily allows *comparisons* between objects and object observations, as they share a representation. Lastly, it is possible to replace or update this representation as new measurements are available.

It is important to note that one could still use a precomputed object catalog in the system by expressing the information stored per object as annotations. It would of course be necessary to treat certain corner cases specially, *e.g.* if high-resolution, ideal mesh models are available for a certain object, comparing a (noisy) point cloud of an object observation to the (ideal) mesh of the object should be performed by a distance function carefully designed to function appropriately.

Chapter 4

Annotators Generating Object Hypotheses

In many perception systems dealing with object recognition, reconstruction, classification or similar object-centric tasks, segmentation is an integral part, both in importance and in representation. Pixel-level post-processing usually requires image masks, 3D point cloud based feature descriptors require 3D regions or point cloud subsets (index vectors) to operate on, whereas higher-level knowledge reasoning on objects is mostly concerned on where something is in terms of some coordinate system.

4.1 Hypothesis Generation

We therefore propose to treat segmentation slightly more abstractly by dealing with *object hypotheses* on a higher level, which are subclassed for the different instantiations of a segmentation. This way, we can combine different algorithms in a consistent manner, *e.g.*:

- an attention mechanism that detects points of interest in pixel coordinates creates a `TFLinkSphereROI` in the camera's *tf* frame.
- image segmentation algorithms can generate masks or region maps (`ReferenceImageROI`), referencing the respective image.
- point cloud segmentation relying on supporting planar structures can generate index vectors (`ReferenceClusterPoints`).

- even uncommon object hypothesis generators such as a skeleton tracker which observes a "put down" event could just generate a point in 3D space to be analyzed as a potential region of interest.

Note that most of these types can be converted between each other relatively trivially, *e.g.* 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 (*cf.* Section 3.3). In a well-integrated platform such as the PR2, this can even be done at a later point in time from another vantage point to some extent (*e.g.* through the *tf* library). This allows us to retrieve the camera image region of interest corresponding to a 3D point cluster, enabling the combination of image analysis annotations and point cloud processing.

In RoboSherlock, we make use of an autonomously generated 3D semantic map of our kitchen (*cf.* Chapter 7). While processing a depth image taken by the robot, we can use the 3D model of what we *expect* in the scene to infer information contained in the acquired sensor data. This makes it possible to *e.g.* perform background subtraction, effectively removing points on surfaces that were already processed during the map acquisition. Furthermore, the 3D model contains named parts with a strict hierarchy of coordinate systems, which can be used to specify search regions of interest (*e.g.* all objects on top of links whose names start with "counter" or "table"). This reduces the number of points that need to be processed significantly, and it allows the segmentation algorithm to annotate an object cluster with (semantically meaningful) information concerning which surface the object was located on.

This way, it becomes possible for the designer of a component engine descriptor to create a basic flow controller in the form of a context-sensitive pipeline execution model. I.e. the designer can formulate that regions of interest in the environment matching a certain pattern be connected to different processing pipelines by stating *e.g.* "object detection should be performed on all tables, countertops and shelves" or "ensure empty space in front of the dishwasher" before opening it.

In this chapter, we will detail mechanisms in RoboSherlock to generate object hypotheses. We will describe pre-processing steps using GPU-accelerated methods for normal estimation and known structure segmentation in Section 4.2. Our multi-

variate approach for segmenting objects on supporting surface with high segmentation fidelity even for flat objects will be detailed in Section 4.3.

4.2 GPU-Accelerated Depth Image Processing

Since many object hypothesis generators, by their very nature, tend to be among the first annotators in larger collection processing engines, it makes sense to discuss some of the pre-processing steps required to ensure their successful performance at this point.

In this section, we will describe several low-level methods developed in the scope of this dissertation (and made available open-source within the PCL library) that rely on the vast computing power of modern graphics processors (GPUs).

The overall goal for these methods was to accelerate them to a level where they can be used without a major penalty in processing time. Managing data streams at a frame rate of *e.g.* 30 Hz leaves just about 33 ms (often 30 ms) of computation time for each data sample. If basic pre-processing tasks such as those addressed in this section use up a major fraction of this available time slice, it obviously severely limits higher level functions. Ideally, "primitive" data pre-processing should be available "for free" in terms of computing resources.

While free is next to impossible to achieve, we can trade computing power for processing time, and massively parallelized architectures such as GPUs are a prime candidate in this regard. However, general-purpose GPU (GPGPU) programming is not trivial, as many algorithms cannot be parallelized easily or at least not directly. We therefore opted for two algorithms that were central to most of our point cloud processing in order to maximize impact of these parallelization efforts.

These methods form the first steps in many of our processing engines:

- *Normal estimation* is a crucial step to obtain a per-point representation not only of surface *presence*, but also *orientation*.
- *Known structure segmentation*, *i.e.* the application of the agent's a-priori process and situation know-how to filter out data that the robot expected to be measured, *e.g.* points taken from the robot itself, or from static and known parts of the environment such as *e.g.* the floor. This can also be used to confine regions of interest, in essence reducing the point measurements that need to be processed by subsequent experts to those containing actually useful information.

These algorithms are explained in more detail in the following two subsections.

4.2.1 Normal Estimation

Point clouds by definition are a list of points, each capturing the presence of a surface at a certain point in space. However, the *underlying surface orientation* is not measured directly, and thus normal vectors are not available per se, making it necessary to estimate them using the available point data.

A survey and comparison of different normal estimation techniques is given in Klasing et al. [67], and a frequently used method is based on fitting a plane to a local point neighborhood by performing Principal Component Analysis (PCA) [109] on a demeaned point neighborhood, which can be reduced to an eigen value problem of the covariance matrix of the point neighborhood [144].

More formally, the covariance matrix $\mathcal{Q}$ for a point neighborhood of size k is computed as follows:

$$\mathcal{Q} = \frac{1}{k-1} \sum_{i=1}^{k} (x_i - \overline{x})(x_i - \overline{x})^{\mathrm{T}}, \tag{4.1}$$

where x_i denotes the i-th point in the neighborhood and $\overline{x}$ the centroid or sample mean of the point neighborhood:

$$\overline{x} = \frac{1}{k} \sum_{i=1}^{k} x. \tag{4.2}$$

The solution to the eigenvector problem defined by

$$\mathcal{Q}\vec{v}_i = \lambda_i \vec{v}_i, \tag{4.3}$$

with $i \in \{1,2,3\}, \lambda_1 < \lambda_2 < \lambda_3$ yields an estimate for the surface normal in $\vec{v}_1$.

A CPU implementation using kd-trees for neighborhood search [10] and singular value decomposition for PCA is available in PCL but proved too slow for true real-time applications on a full resolution Kinect cloud, as can be seen in Section 4.2.1.1. We therefore developed a GPU-based normal estimation method which is made available within PCL that is able to estimate normal vectors for a full Kinect frame in less than a millisecond, which puts it well within reach for actual real-time applications (at least for $30Hz$ sensor streams). This implementation only works on *organized* point clouds, *i.e*. 2.5D depth images, which is however perfectly suitable for our input data. We will refer to the technique defined by equations 4.1 through 4.3 as the reference implementation against which we compare our method.

The neighborhood search neccessary for the reference method is not trivially mapped to a GPU implementation since branching structures such as kd-trees do not perform well on current graphics architectures. This is due to the fact that on *e.g*. Nvidia GPUs, threads are organized in warps of 32 threads, where each thread must either execute the same instructions or wait for the duration of this instruction. Branch divergence between threads thus leads to serialized execution of seemingly parallel threads.

To replace this costly step, the GPU implementation estimates normals by taking into account only the 4 direct pixel neighbors of the query point. In order to be more robust to noise, a smoothing step is performed in advance. This is done by iteratively applying a simple and very fast box filter. The Central Limit Theorem states that given a sufficiently large number of independent random variables with finite mean and variance, their mean is approximately normally distributed[18, p. 173]. In this context, it means that the iterative convolution of a box filter approximates a regular Gaussian smoothing filter (see Figure 4.1).

In the case where neighboring pixels have a highly different depth value, a Gaussian blur filter will oversmooth by moving both points in depth far away from their original, measured depth value. The advantage of an iterative filtering approach is that between iterations, we can employ a clamping step that ensures that the smoother doesn't oversmooth the depth components of the point cloud.

Since the Kinect driver serves the computed disparity image as a shift image with a quantization of $1/8$ pixel, we can define an interval around each depth value to which the clamping step limits the smoothed depth values. Let the real and measured depth

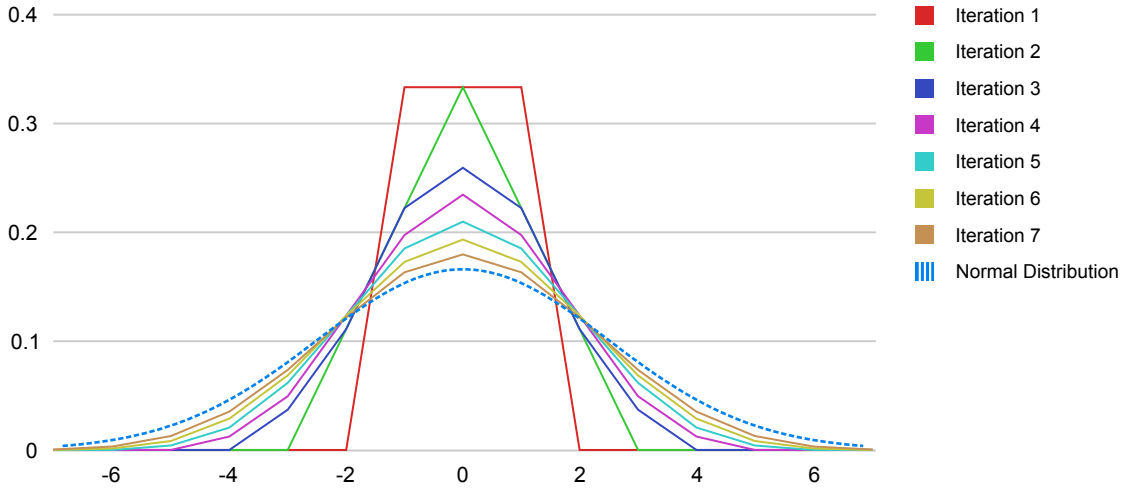


Figure 4.1.: *Due to the Central Limit Theorem, iterative application of a box filter approximates a normal (Gaussian) distribution. Here we show the effect of applying a 3-pixel wide box filter (shown in one dimension) and its convergence towards the Gaussian distribution (dotted blue line).*

values be denoted by d and d', respectively, the stereo baseline by b and focal length by f. Since disparity δ is related to depth as $\delta = bf/d$, the disparity interval defined by $\delta \pm a$ corresponds to

$$d - \left(\frac{ad^2}{ad + bf}\right) < d' < d + \left(\frac{ad^2}{ad - bf}\right). \tag{4.4}$$

In this form, $a = 1/8$ defines the expected discretization error around δ in disparity space (which can be increased to account for measurement noise).

Iterative applications of box filtering and clamping to this interval thus eliminates most noise effects in the depth components without distorting the point cloud more than our sensor (disparity) model allows.

Normals $\vec{n_i}$ are then computed per point p_i as the cross product between a horizontal vector $\vec{h}$ and vertical vector $\vec{v}$ which are computed from the east-west and north-south point neighbor pairs, respectively. If for a given axis, one pixel neighbor is not available, we resort to the point p_i, and if both are missing, we cannot compute a valid normal.

In the RoboSherlock context, we treat a map of normal vectors as another *View* of the point cloud data. Within the CAS, the normal map is therefore stored as a separate Subject of Analysis, and Annotators requiring normals for processing a scene can request them through our *SceneCAS* (*cf*. Section 3.1.1) wrapper.

As a side product, the normal estimation also provides us with the smoothed depth image, which is stored as a separate view of the original document.

4.2.1.1 Evaluation

In order to evaluate the described normal estimation technique, we performed tests with synthetic and empirical data. Synthetic scenes consisted of planar patches at different angles to the viewing direction and allowed comparisons of estimated normal vectors with ground truth. For empirical, real sensor data we cannot rely on ground truth and thus compared our method with the `NormalEstimation` class available in PCL. In both cases, we sampled 170 full-frame Kinect depth images and estimated normal vectors for every point. The angles between normal vector computed with our method and those from either ground truth or the reference implementation were then counted in a histogram, which is shown in Figure 4.2.

As can be seen, the normals from ours and the reference implementation differ quite significantly, with 75% of normals deviating between 0° and 8° and a (mild) peak at 3° (blue graph). However compared to ground truth values, our method fares much better. This is obviously due to the fact that the reference method itself uses merely an approximation to the true surface orientation, so we are essentially comparing two noisy estimates. Additionally, our smoothing process works solely along the depth dimension, which is consistent with the measurement principle and this noise model of the Kinect, where as the Principal Component Analysis of the reference method models the noise to be Gaussian isotropic in all three dimensions.

Therefore, the smoothing step makes our method relatively insusceptible to noise. To test this, we performed the same experiments with increasing levels of additive noise. To simulate Kinect-like properties, we injected Gaussian noise in the disparity space, in magnitudes from $\frac{1}{16}$ to $\frac{1}{4}$ pixels standard deviation. The results can be seen in Figure 4.3. The percentage of normals with errors exceeding 9.5° increase with

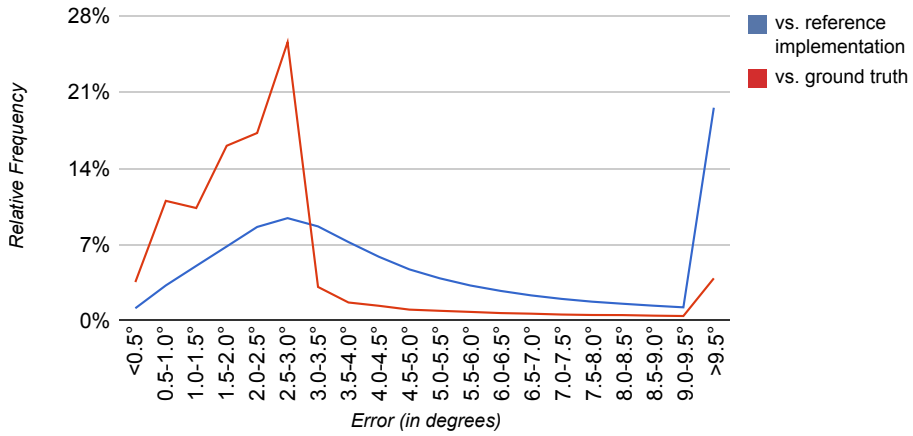


Figure 4.2.: *This histogram shows the error distribution of the GPU-accelerated normal estimation method. We compared the angular error of the normals generated with out method with those estimated by the reference implementation (blue graph) and with ground truth (red graph). Note that over 85% of all normals deviated no more than* 3.5° *from ground truth.*

higher noise levels, but the performance in general stays comparable to the synthetic case.

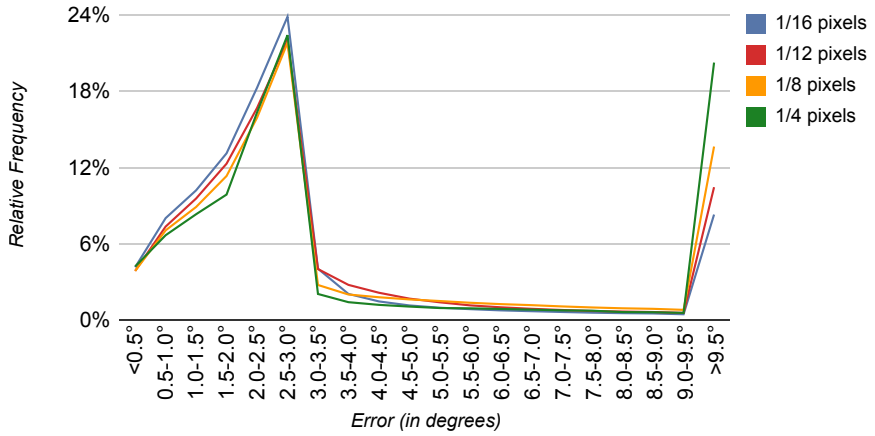


Figure 4.3.: *Due to the smoothing step, our method is relatively robust in the presence of disparity noise.*

On the other hand, smoothing in disparity also creates a bias and is thus a source of systematic errors. Since we average depth (actually disparity) values, any slope in neighboring points is prone to be distorted towards being more parallel to the sensor plane. In fact, plotting the histogram of normal deviations between our method

and ground truth over the orientation of the plane w.r.t. the sensor plane (*cf*. Figure 4.4), one can see that this effect is strongly correlated with the inclination of a given normal.

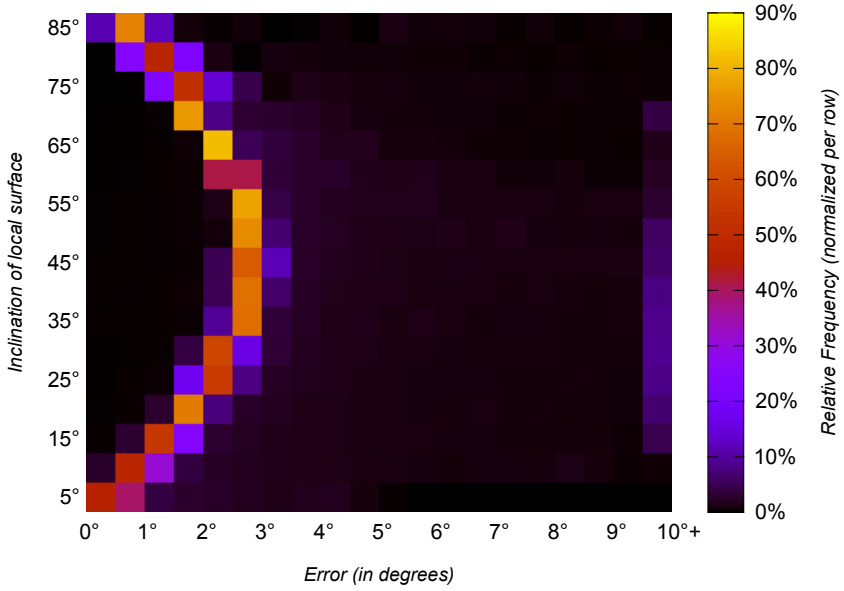


Figure 4.4.: *The deviation of normals from ground truth (horizontal axis) shown over different surface inclinations (vertical axis). Each horizontal line contains a single histogram for the specified surface inclination (angle between surface normal and the normal of the sensor plane). The inclination-dependent bias of our method creates a very sharp peak and its maximum follows a curve that approaches zero for orthogonal or parallel orientations, and reaches its maximum at* 45° *inclination with an error of* 3°.

We devided the range of angles between the sensor plane and estimated normals into bins of 5° and fitted a Gaussian to each histogram. We used the mean of this Gaussian (approximately the maximum per row in Figure 4.4) in a lookup table to rotate each normal away from the center view ray (normal to the sensor plane).

More formally, let $\vec{z}$ be the normal of the sensor plane, pointing towards the sensor origin, and $\vec{n}_i$ the normal for the i-th point. The angle ϕ between them is given by:

$$\phi = \cos^{-1}\left(\frac{\vec{z}\cdot\vec{n}_i}{|\vec{z}||\vec{n}_i|}\right) = \cos^{-1}\left(\vec{z}\cdot\vec{n}_i\right). \tag{4.5}$$

Let θ_ϕ denote the corresponding correction angle from our lookup table. The normal is then rotated by the transformation specified by the following axis-angle representation:

$$\langle \theta_\phi, \vec{z} \times \vec{n_i} \rangle. \tag{4.6}$$

We repeated the experiments from Figure 4.4 with the inclusion of this corrective rotation, and the results can be seen in Figure 4.5. Integration over all inclinations yields the corresponding graph in Figure 4.6 showing the overall accuracy of the method (compare to Figure 4.2).

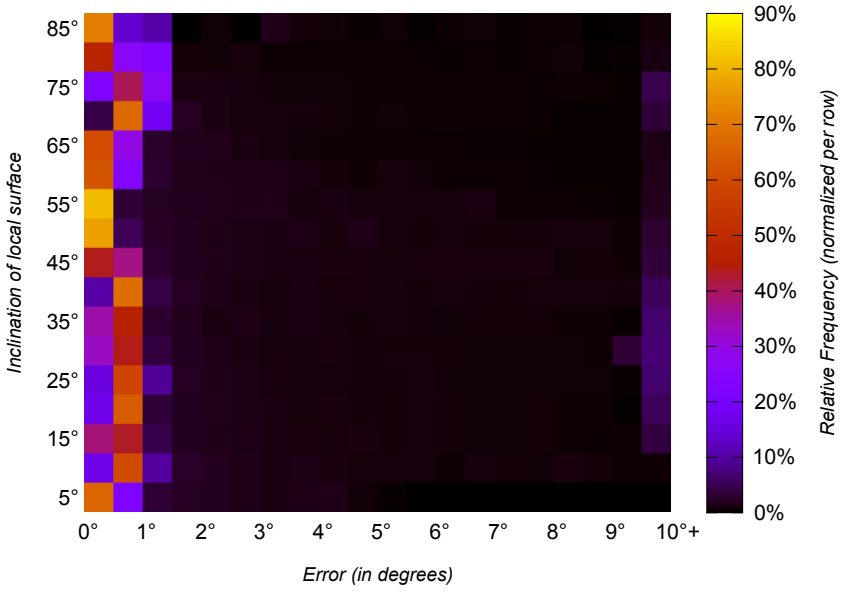


Figure 4.5.: *Applying a corrective rotation to estimated normals based on their inclination w.r.t. the sensor plane, we can eliminate most of the bias apparent in Figure 4.4.*

Another important, if not the most important aspect is the real-time performance of our method. We implemented the smoothing and normal estimation steps as kernels in CUDA[1] which are launched in a dedicated thread per pixel. The depth image from the Kinect is uploaded onto GPU memory, and we used the *thrust* library [2] to coordinate the launch and execution of processing units on the GPU.

[1] www.nvidia.com/cuda
[2] http://thrust.github.io/

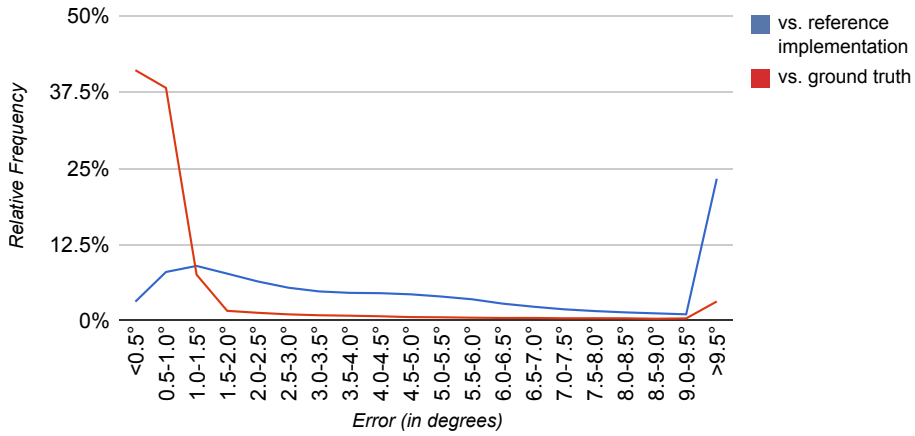


Figure 4.6.: *The overall error histogram of our normal estimation method, compared to the reference implementation (blue graph) and ground truth (red graph). Due to the corrective rotation, more than 85% of all normals have an error of* 1.5° *or less. Note that the blue graph is also shifted left compared to Figure 4.2.*

Experiments were performed on an Intel i5-2500K CPU and a NVIDIA Geforce GTX 570 GPU with 1280 MB of memory. We recorded a total of 740 Kinect frames and show a histogram over computation time in Figure 4.7. As can be seen, a majority of frames are computed in less than a tenth of a millisecond. In fact, only 6% of frames take longer than 0.2 ms to compute.

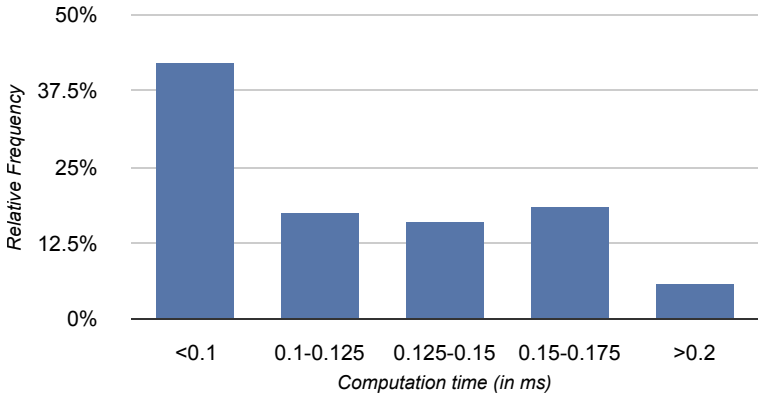


Figure 4.7.: *Computation time for our normal estimation method for full-frame Kinect scans. Displayed is a histogram over 740 frames.*

To compare performance between the available methods in PCL, we used 80 Kinect point clouds with $11k$ to $220k$ valid points. We estimated normals using four meth-

ods: *i*) our GPU-based method; *ii*) `pcl::NormalEstimation` (our reference method); *iii*) `pcl::LinearLeastSquaresNormalEstimation`; and *iv*) `pcl::IntegralImageNormalEstimation`. The results can be seen in Figure 4.8. Due to the fact that the reference implementation takes several orders of magnitudes longer than the other methods, the remaining three graphs in the top graph collapse onto one line close to zero. They are therefore displayed separately in the lower graph.

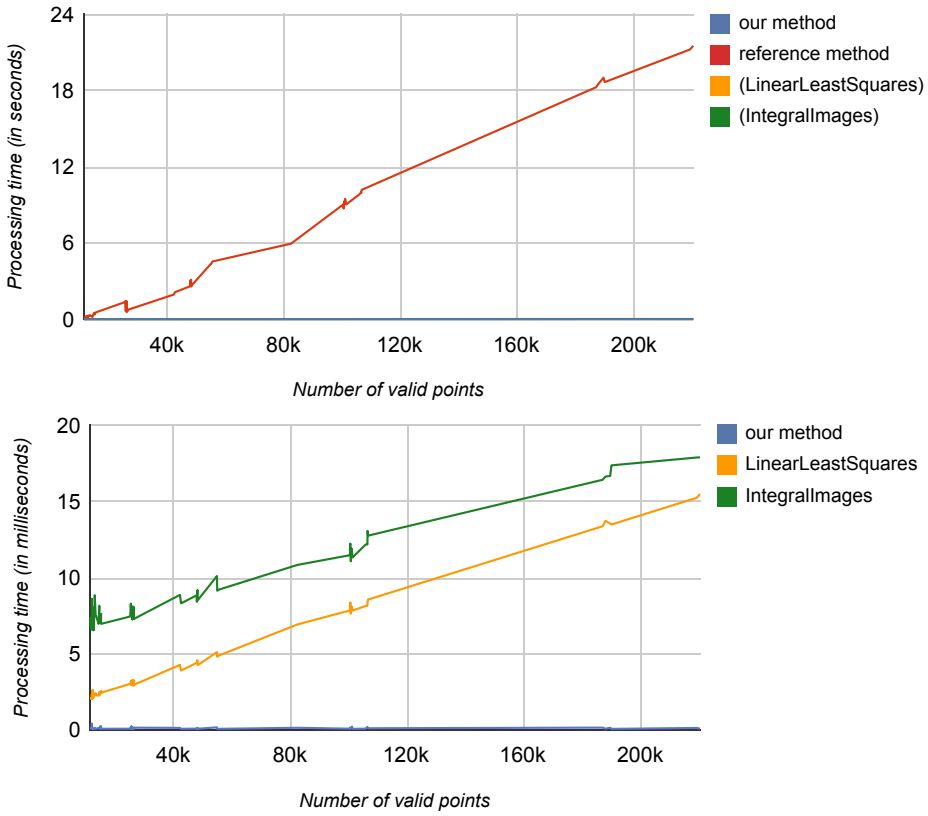


Figure 4.8.: *Computation time for four normal estimation methods with varying amounts of valid points. The reference implementation takes several orders of magnitudes more time (top), we therefore show the remaining three methods without it (bottom). Note that the top graph is displayed in seconds for the y-axis, whereas the bottom graph uses milliseconds.*

Since our method spawns one thread per pixel (not per valid point!), our method is constant for a given image size. The other three methods all have a linear dependence on the number of valid points in an image, however Linear Least Squares and Integral Image are both reasonably fast, with a maximum just over 15 ms. However, given a processing time window of 30 ms for real-time depth image processing, we

consider this still too slow, since we spend more than half the available time computing normal vectors. Compared to the next slower method, our technique performs between 15 and 200 times faster, and several thousand times faster than the reference implementation.

In summary, we presented a method that can take advantage of the massively parallel nature of modern graphics processors. At processing times several orders of magnitudes faster than other methods, we were able to achieve high accuracy with errors in general not exceeding 3° and robustness to noise. As a side product, we generate a smoothed depth image that can be reused by later processing units. We show several example images of smoothed data in Figure 4.9 and examples of normals estimated for various scenes in Figure 4.10.

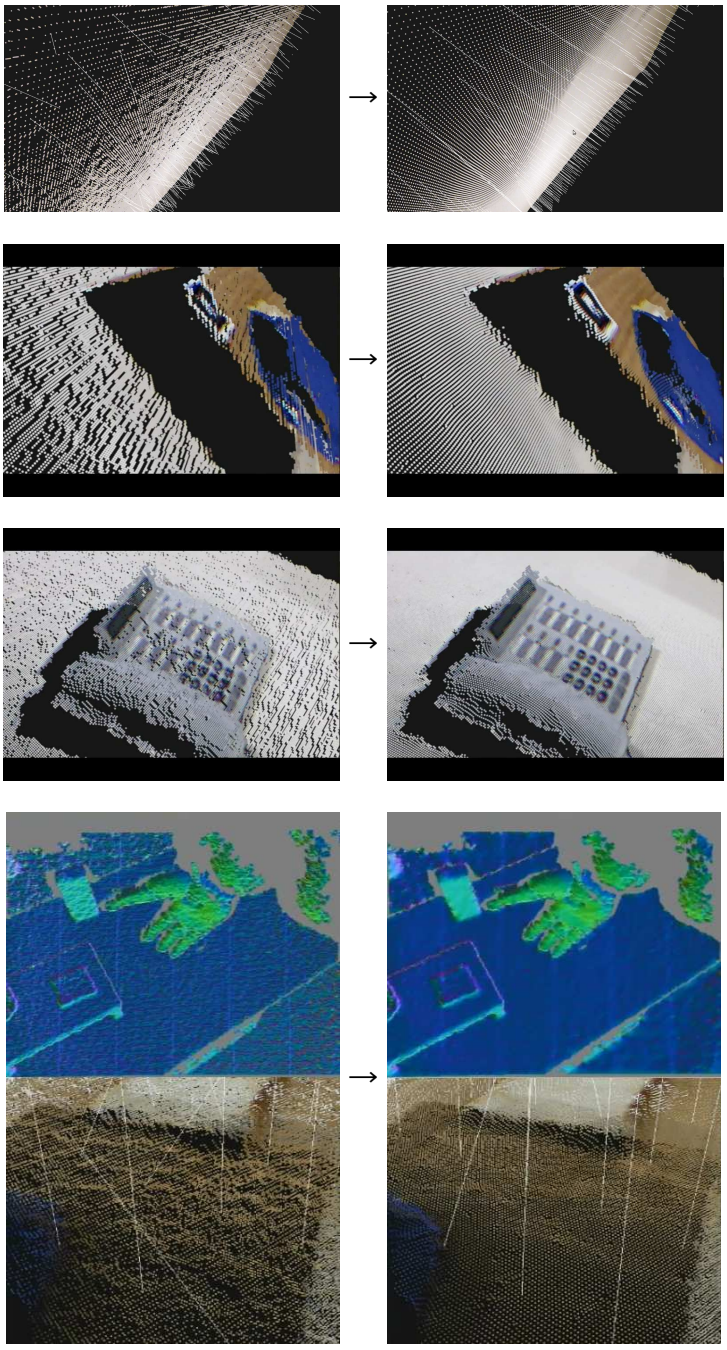


Figure 4.9.: *Several scenes showing the effects of disparity smoothing. Fine details in close ranges* (e.g. *cutting board, telephone) are preserved while far regions* (e.g. *wall in top image) have undergone more rigorous smoothing. This is due to the fact that we parameterize in disparity space. For the bottom image (cutting board close-up), we included the normal image, where the red, green and blue channels represent the three normal vector components per point. The effect of the Kinect-typical disparity discretization on estimated normals are clearly visible in the left image.*

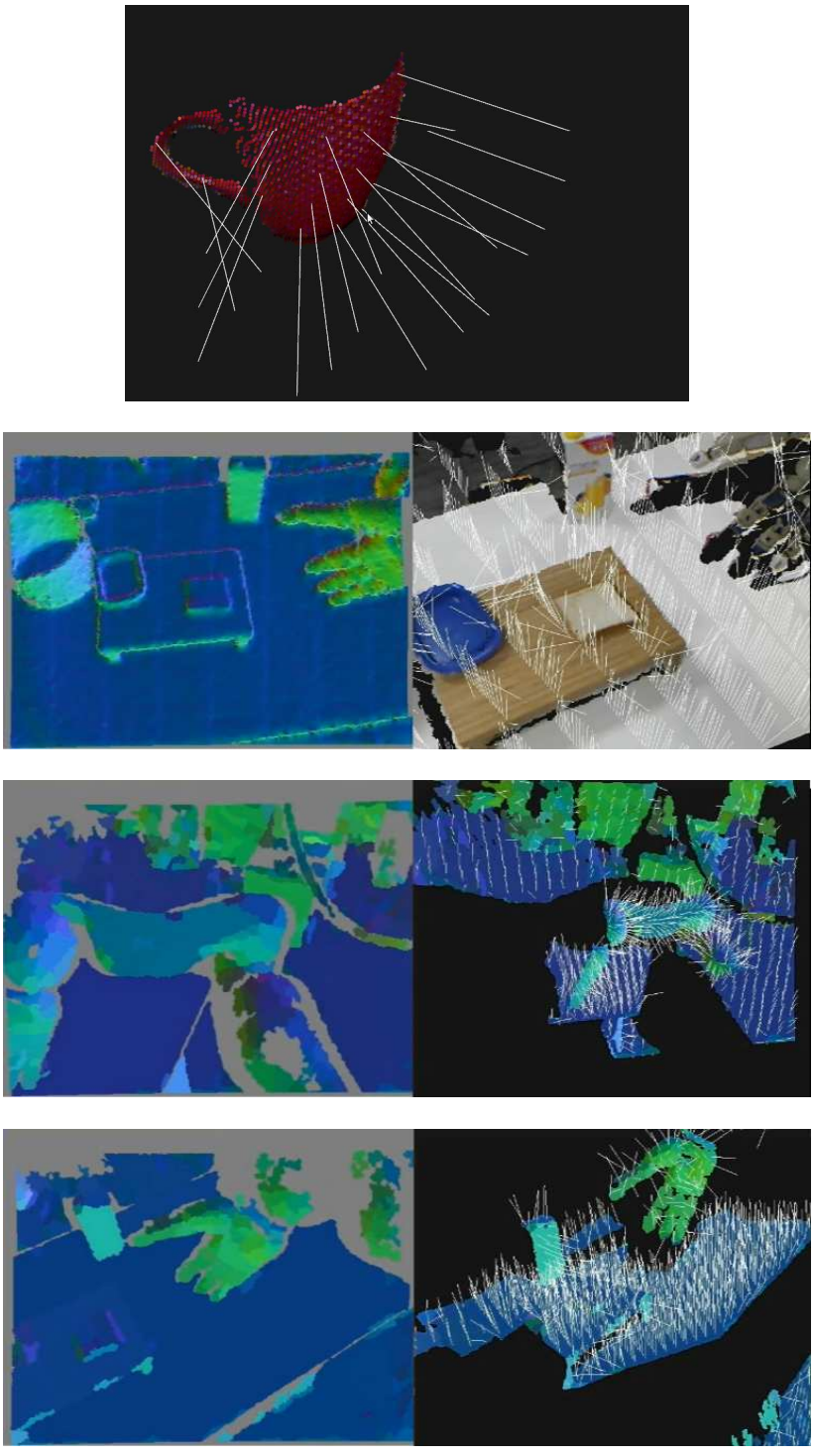


Figure 4.10.: *Several scenes showing normals estimated by our method. The bottom three images show the normal image on the left. For the bottom two, the colors are segmented using mean-shift, which leads to a fast surface orientation segmentation. Connected regions of similar surface orientation are clustered together.*

4.2.2 3D Known Structure Segmentation

As mentioned before, one kind of component of our unstructured input data is 3D point data taken as a function of static and known geometry in the environment. This includes the robot model itself as well as static furniture, both of which account for a considerable percentage of our data points. However, since the robot is localized within the map, and has access to proprioceptive information concerning *e.g.* joint angles, it is possible to filter out or label data points which lie on the surfaces of these known structures.

As described in Chapter 7, our robot can rely on a 3D semantic map of the environment, which contains – among other things – a hierarchical 3D model. This map is expressed as a URDF model (Unified Robot Description Format) within ROS. It represents a graph of *links* (*i.e.* coordinate systems and optional visual or collision geometry) connected by *joints*, which desribe the kinematic and dynamic connection between links.

Originally developed for robot models, a link typically represents a certain, *named* robot part, *e.g.* an arm segment, and the joint connected to it can be of various types, *e.g.* rotational or prismatic for articulated joints, or fixed for rigid link connections. Links are further annotated with inertias, visual features and collision geometry in the form of primitive shapes (cylinders, boxes) or freeform triangle meshes.

We opted to represent the semantic map (in addition to the robot model itself) in this format for several reasons, as described in Chapter 7, where we also show a 3D model (Figures 7.8,7.9,7.10), and a graph representation (Figures 7.11,7.12) of the whole kitchen map.

The availabitity of such a model gives rise to a number of ideas that prove beneficial to perception tasks:

1. Filtering of known structure that is irrelevant to the task. Most notably, removal of points on walls, floor and ceiling.

2. Specification of regions of interest. For example, specifying that objects of interest are located on certain, named parts in the environment, *e.g.* on counter tops or certain shelves. At the same time, objects segmented from these sur-

faces can be labeled with a `LocationAnnotation` that *names* the location with a semantically meaningful ID (*e.g.* `counter_top_sink`).

3. Association of processing steps to regions of interest. Examples include estimating the angle of doors or extrusion depth of drawers. Note that *e.g.* wall removal or object segmentation on countertops are special cases of this idea.

4. Filtering the robot model itself to avoid misinterpreting points, *e.g.* treating the gripper as part of an object when reaching for it. Also, human skeleton tracking can be easily confused in *hand-over* scenarios due to the close spatial proximity of the robot end effector to the human hand.

We implemented the RoboSherlock Annotator `URDFRegionFilter` for exactly these purposes. As inputs, it requests the URDF scene description – or multiple descriptions, *e.g.* a world model and two robot models – from the ROS infrastructure (*rosparam*) and parses it to create an internal tree of renderable models and meshes that is connected by transformations. At run time, for a given depth image or point cloud, we update the transformation tree through the *tf* library, which continuously reads sensors such as motor encoders to compute joint angles and transformations.

In the CPE descriptor, parameters concerning the URDF links which represent regions of interest as well as method assocation and other values such as bounding boxes describe the tasks that are to be performed.

As a typical example, we perform the following steps involving URDF-based point cloud filtering:

1. We perform a *self filtering* step, in which all parts of the robot (and the rest of the world model!) that are in the sensor frustrum are rendered in an offscreen buffer with the help of a GPU-accelerated *shader program*.

2. We proceed to *remove* points which are within a specified distance to all wall links, or sides of furniture which are of no interest to the task at hand using the same shader pass.

3. We then filter and label points that are within a specified *region of interest*, which is specified as a bounding box defined in a URDF link's coordinate system, *e.g.* a one meter high bounding box above a table.

The order of these steps is important. Even a slight localization error can lead to a wall behind the kitchen counter to fall into the bounding volume of interest, causing the ROI segmentation step to "hallucinate" a large, vertically planar "object" on top of the counter. It is therefore beneficial to account for all *known* scene geometry before attempting to interpret the remaining *unknown* measurements.

In the following subsections, we will provide details for these processing steps.

4.2.2.1 Robot self filtering

Robot self filtering is an important pre-processing step for subsequent application of algorithms that could potentially fail in cases where the robot parts, especially arms and end effectors, are reaching into regions of interest or are in close proximity or even contact to objects (*cf*. Figure 4.11).

Figure 4.11.: *Color image from the robot's camera.*

Another challenging aspect is that of gesture recognition for task control by a human operator or simple classification of humans in sensory data. Skeleton tracking systems can easily be affected by occlusions of the human by a robot manipulator. This is especially problematic when doing hand-over maneuvers. In this case, the robot end effector needs to move into close vicinity of the (estimated) operator hand. When they are physically close, points on the robot gripper can be too close to the

operator's hand, which can cause the tracker to misinterpret the skeleton, effectively assuming the robot hand to be the human hand (or an extension thereof). Since this in turn causes the system to pull the robot hand away from the human every time the human and robot hand approach each other, the hand-over must fail. We therefore preprocess the depth image in order to incorporate as much of the system's knowledge of the environment as possible.

In Figure 4.11, we show a typical scene in our kitchen, where the robot is looking at a counter top with several objects and is just grasping a milk carton.

The performance of this filtering step is critical, since it needs to work even if the robot arm is moving at high speeds, and cannot introduce large latencies for subsequent modules, especially human skeleton tracking. The tracker or other consumers generally have non-negligible processing times themselves, but the overall throughput should stay within frame rate.

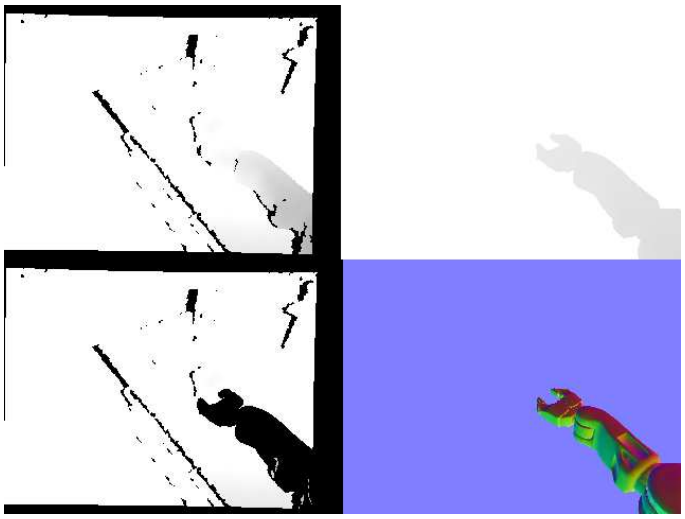


Figure 4.12.: *Visualization of the robot self filtering. The top left image shows the original depth channel from the Kinect data, top right shows the virtual depth image. On the bottom left, the robot arm has been filtered completely. For visualization, we show the normal space of the virtual scene in the bottom right.*

Since the RGBD camera is calibrated with respect to the robot system, we can render the robot model in a virtual view from the same vantage point as the depth image (*cf*. Figure 4.12). Given accurate calibration and a descriptive camera model, the

measured and virtual depth views correspond pixel-perfect, which allows filtering all points which we know to coincide with our robot model.

Let a camera calibration consist of the intrinsic matrix $\mathcal{P}$ (equivalent to $\mathcal{K}$ as commonly used by *e.g.* Hartley and Zisserman [53]):

$$\mathcal{P} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.7}$$

where f_x and f_y designate the horizontal and vertical focal lengths (in pixels), γ represents the skew between the horizontal and vertical axes, and c_x and c_y define the principal point, *i.e.* the ideal center of the image. For details on on how to obtain these by camera calibration, we again refer to Hartley and Zisserman [53].

Given the desired image width w and height h, as well as the far and near clipping planes at z_f,z_n, this can be converted to an OpenGL projection matrix $\mathcal{P}'$:

$$\mathcal{P}' = \mathcal{O} \begin{bmatrix} f_x & \gamma & -c_x & 0 \\ 0 & f_y & -c_y & 0 \\ 0 & 0 & z_f + z_n & z_f z_n \\ 0 & 0 & -1 & 0 \end{bmatrix}, \tag{4.8}$$

where $\mathcal{O}$ is an orthographic matrix responsible for scaling the coordinates to normalized device coordinates [157] within OpenGL. The negative entries are caused by the fact that the OpenGL camera looks along the negative z-axis (in OpenGL, object and world space is right handed, but normalized device coordinate and window space is left-handed).

$\mathcal{O}$ is defined as:

$$\mathcal{O} = \begin{bmatrix} \frac{2}{w} & 0 & 0 & (0) \\ 0 & \frac{2}{h} & 0 & (0) \\ 0 & 0 & -\frac{2}{z_f - z_n} & -\frac{z_f + z_n}{z_f - z_n} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.9}$$

The extrinsic camera calibration data can then be used in conjunction with the robot's *tf* transformation tree to position the virtual camera w.r.t. the world and robot models.

Using $\mathcal{P}'$ as the projection matrix, the Kinect buffer and the OpenGL depth buffer can be compared pixel for pixel. This is achieved by loading the measured depth image into a texture on the GPU, which the virtual renderer has access to. A fragment shader is used to compare each depth value of the virtual view with the corresponding measured depth value to perform filtering.

As the depth d stored in the OpenGL depth buffer is expressed in a range between 0 and 1, which corresponds to cartesian coordinates of z_f and z_n, respectively, we need to convert d to cartesian coordinates d_z before we can compare it to the depth value z stored in the Kinect depth image (see also The Khronos Group [157, Section 12.050]):

$$d_z = \frac{z_f z_n}{d(z_n - z_f) + z_f}. \tag{4.10}$$

Using Equation 4.10, the fragment shader can convert the depth values stored in the depth buffer to cartesian coordinates. For pixels where $d_z - z > \epsilon$, we can deduce that the Kinect "saw" an obstacle in front of the virtual model. We set $\epsilon = 2\,\text{cm}$ and filter all points where $d_z - z \leq \epsilon$.

The corresponding images from the robot self filter are shown in Figure 4.12, where you can see the original depth image, the virtual depth image and the processed output image with the robot arm removed.

The whole process takes few milliseconds on a standard GPU, and can filter the known environment and the robot arm even when moving at maximum speed. Figure 4.14 shows some examples where the human hand is very close to the robot hand. The remaining (white) pixels after filtering follow the contour of the human very closely.

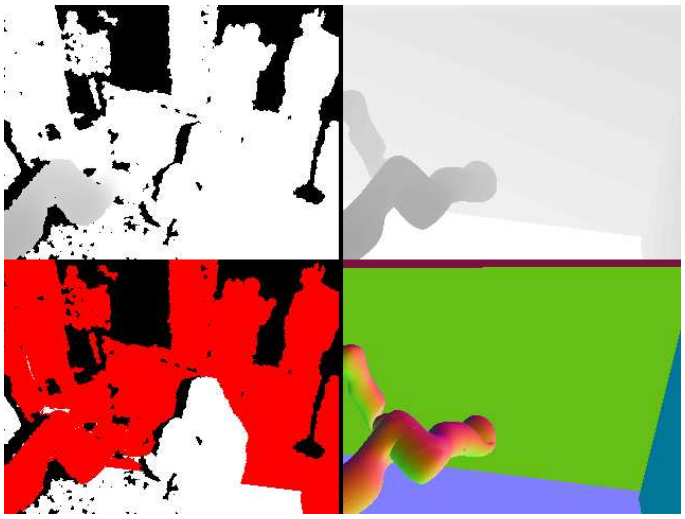


Figure 4.13.: *Filtering known structure* (e.g. *robot arms) from the depth image to ensure that the skeleton tracker only considers actual operator points. The original depth image (top left) is compared with virtual depth image (top right) to filter out irrelevant regions (red parts, bottom left). For visualization purposes, the bottom right image shows the virtual scene, including two virtual walls to limit the operator workspace.*

4.2.2.2 Evaluation

For a more thorough performance evaluation, we performed an experiment where we filled the virtual environment with an increasing number of PR2 models. The PR2 consists of 45 individual meshes, for a total of 155490 vertices. We measured the time for *i*) setting up the models and *ii*) rendering them separately. The first phase is dominated by iterating over all joints and querying the *tf* library for up-to-date transforms for each submesh. The second phase is concerned with rendering the meshes that were loaded into Vertex Buffer Objects on the GPU, and includes the shader-based filtering. We performed each step 30 times and averaged the results.

The results of the experiment can be seen in Figure 4.15. It can be seen that time requirements grows linearly with the number of robots present in the scene, for both phases. We can render up to 35 articulated robots before starting to violate real-time requirements. Within the ROS community, there are several implementations [3] [4] of the same general idea, but they take at least an order of magnitude more time than

[3]http://ros.org/wiki/robot_self_filter
[4]http://ros.org/wiki/camera_self_filter

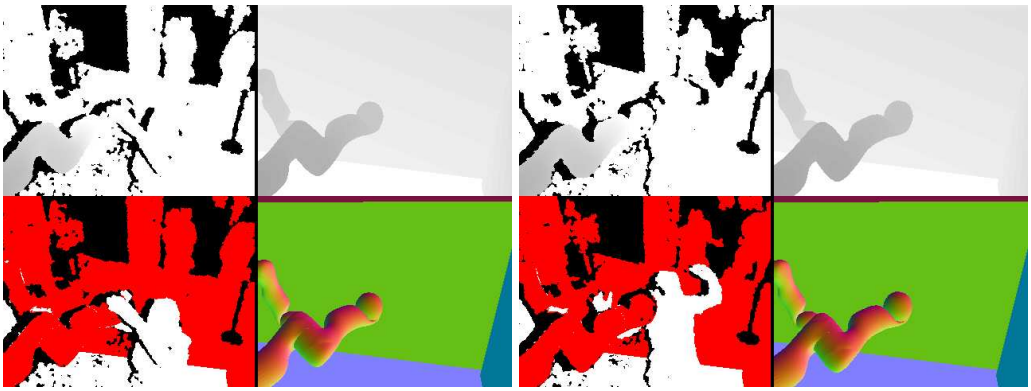


Figure 4.14.: *Two examples of situations where the human and robot hand are in close proximity (left images: human occludes robot, right images: robot occludes human). In both cases, filtering assures robust performance of the skeleton tracker. (Refer to Figure 4.13 for the meaning of respective sub-images.)*

our method. This is due to the fact that we use the GPU both for rendering the scene *and* comparing it with the real depth data.

However, in general, *tf* lookups are the more expensive operation, which leads to the conclusion that static environments can be rendered with much more detail, since the number of transformation lookups is much lower. We therefore disabled *tf*, and rendered each submesh at a random position in front of the camera, measuring the required time to render an increasing number of individual but static meshes. Each mesh has on average approximately 3500 points. The results can be seen in Figure 4.16. It can be seen that we can in fact render highly complex scene geometry very efficiently, crossing the 30 ms threshold at just over 10000 meshes. This corresponds to 35 million vertices.

We are not sure of the reasons for the distinct plateau between 10000 and 15000 meshes, but repeated executions of the experiments yielded the same behaviour consistently. We assume this is due to the behaviour of the NVIDIA driver, but this should be analyzed in more depth in the future.

4.2.2.3 Region of interest filtering

Also, a rather low percentage of the whole image in Figure 4.11 contains actually interesting information. The counter top and objects make up for approximately 25%

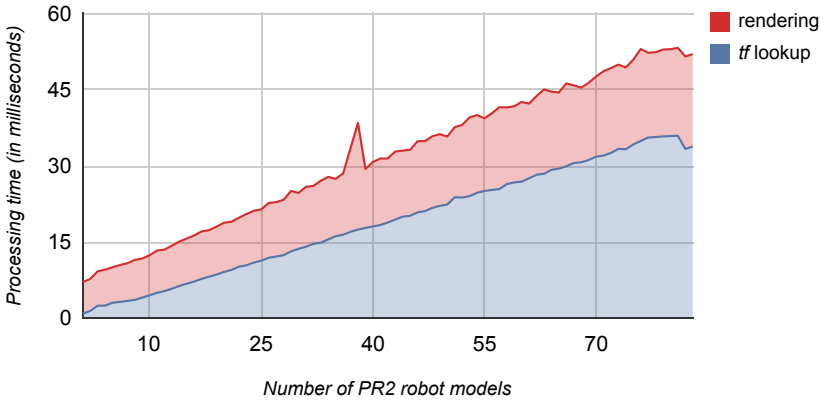


Figure 4.15.: *Required processing times to update and render an increasing number of PR2 models. It is possible to render about 35 complete robot models while still staying in the* 30 ms *time frame for real-time performance. Processing time is divided into* tf*-lookup (blue) and actual rendering including filtering (red).*

of the image area, which means that by prefiltering the robot arm and selecting only those points in the bounding box above the counter top, we can reduce the number of points that subsecquent processing modules have to consider by 75%. This is clearly visible when comparing the original image with the point cloud visualized in Figure 4.17. Here, the final point cloud contains approximately $75k$ points, which is quite a reduction from the full frame, which at full VGA resolution contains up to 307200 pixels.

Note how the robot arm is almost completely removed from the point cloud, while the counter top, the objects on it and the visible parts of the milk carton in the robot gripper still remain present in the filtered cloud. We do not filter out the points on the supporting surface since flat objects might be located there, which can be of interest for later analysis.

In the experiments described in Chapter 8, we collected 103 scenes of typical household scenarios in order to evaluate RoboSherlock. During these experiments, `URDFRegionFilter` was used to presegment the point data to consider only relevant regions in space. In total, the point clouds contained over 25 million points, of which only 9.9 million remained after the filtering step. This amounts to an average reduction of over 61%.

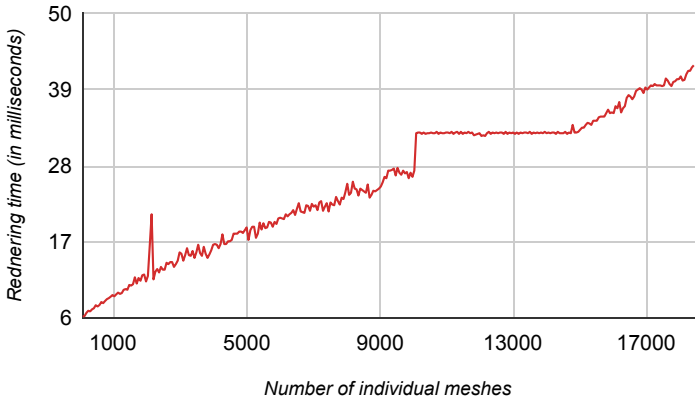


Figure 4.16.: *Processing time requirements for filtering increasingly more complex static geometry using our method. Due to the omission of the transformation lookup step, we can create much more complex scenes for filtering. Each mesh contains on average 3500 vertices, and we can render approximately 10000 meshes or 35 million points in* 30 ms.

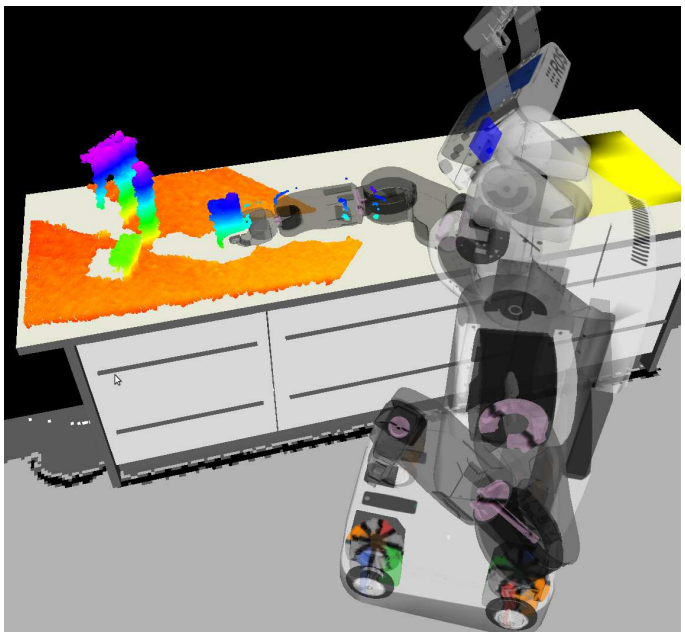


Figure 4.17.: *After removal of points on the robot model, and removal of points outside the regions of interest, approximately* 75% *of the original points have been removed, leaving ony the supporting surface and objects on top (and in the robot gripper!).* All *points left in the cloud are therefore meaningful in our context.*

4.3 High-Fidelity Supporting Surfaces Segmentation

Given the inherently noisy depth data of range sensors and depth cameras, there is always a certain amount of ambiguities when it comes to model fitting. This is to say that a plane fitting approach, as described in previous publications [131], must work with a relatively high inlier threshold when taken by itself. Additionally, not all supporting surfaces are perfectly planar, *e.g.* when placing two tables side-by-side.

Consider a scene where the objects on a table surface are relatively high, *e.g.* bottles, cereal boxes, TetraPaks *etc*. Finding the table plane and removing it can be done rather reliably using *e.g.* Random Sample Consensus (RANSAC) methods [40] even in the presence of noise since we can employ a high inlier threshold when removing the table plane. In our work with PCL, we have often chosen this value in the range of 2cm to 5cm, depening on the data acquisition device and its noise and error properties. Note that this is no problem as long as the condition is enforced that inliers also must have normal vectors that coincide with the normal of the plane in question.

However, as soon as the objects we want to be able to segment become lower in height, this approach becomes unreliable. Examples include plates and cutlery, but also DVD cases or books.

To deal with such flat objects, we decided to improve the sensitivity and accuracy of our segmentation methods to be able to deal with objects of 1cm height.

As test objects, we used wooden track pieces from a toy train set which were available in abundance at the lab due to another project. At 10mm, these objects are within the same order of magnitude as error and noise levels in our sensor data, especially at higher distances, and thus posed a harder problem than most other objects in typical kitchen environments.

Figure 4.18 shows an example of multiple track pieces on a surface formed by two tables. We overlayed the plane computed with a Sample Consensus [40] approach as a reprojected regular grid.

This plane is used to transform the depth image into a rectified image as can be seen in Figure 4.19, which creates a virtual top-down-view, regardless of the actual sensor pose. This is a basic projection step that simplifies further computation since

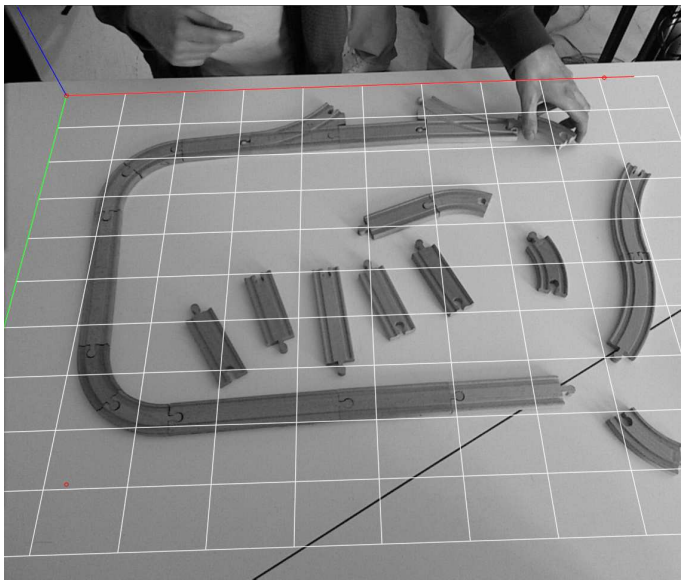


Figure 4.18.: *Camera image of the table area with the surface plane superimposed.*

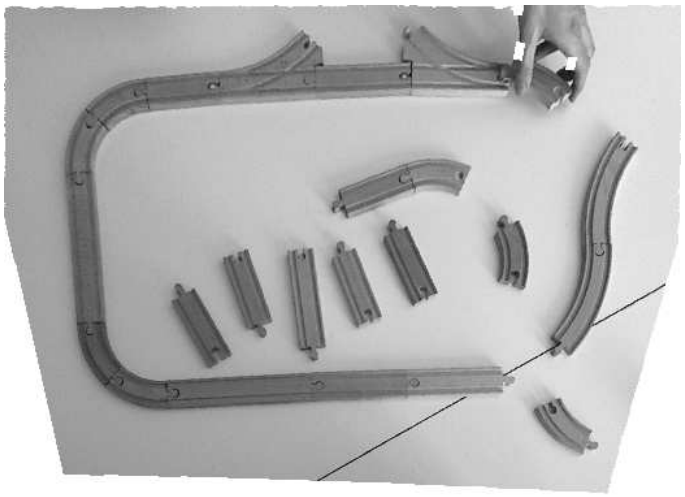


Figure 4.19.: *Virtual top-down view of the table area. The image area corresponds to the grid in Figure 4.18.*

we can treat the scene as a 2D problem directly: note that there is a known, fixed relationship between actual world units and image coordinate units, which can be set as a parameter in centimeters per pixel. It is therefore trivial to compute actual measurements from pixel offsets and vice-versa as pixel coordinates (x, y) and pixel values (z) are directly meaningful in *e.g.* table coordinates.

When simply lowering the inlier threshold to *e.g.* 5 mm, traditional approaches such as plane fitting to the underlying surface lead to false positive and false negative segmentation errors for the track pieces. In the case of the Kinect sensor, lens imperfections can create distortions over the whole imaging field with a magnitude of approximately 1 cm at close distances and up to 8 cm at 5 m [64]. Locally, depth errors are in the range of few millimeters, but the deviation from an ideal plane over the whole field of view are considerably larger. This can be seen in Figure 4.20, which shows a rectified top-down projection of the point data onto the extracted plane shown in Figure 4.18 as a height map.

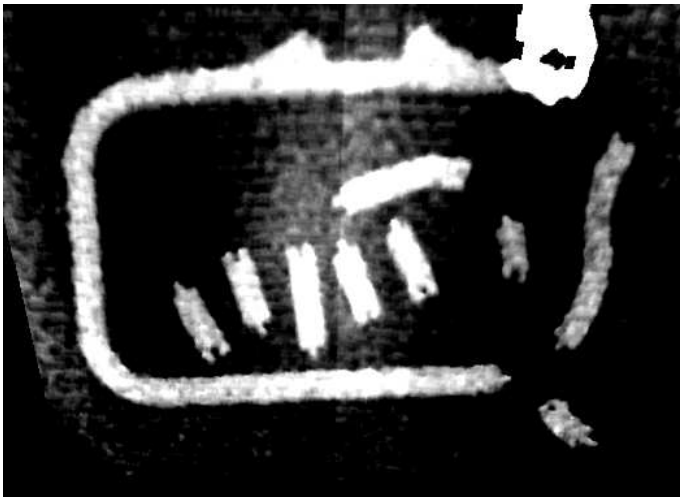


Figure 4.20.: *An uncalibrated height map of the table area shows the cumulative distortion of the sensor and the underlying surface.*

One can clearly see in Figure 4.20 that parts of the table deviate from the ideal plane in a similar amount to track pieces in other areas, *e.g.* the table surface in the image center has a similar height value as the right most track piece (black is zero deviation, and white is 1cm or higher). A detailed analysis of accuracy and resolution for Kinect sensors is given by Khoshelham and Elberink [64].

To overcome these problems, we took two measures: *i*) For segmentation, we perform a combined color and depth segmentation method that works in two steps, as described below. *ii*) In cases where we can assume a fixed camera position for the duration of an experiment, *e.g.* a wall-mounted Kinect or an experiment that does not involve robot locomotion, we precede this with a calibration step of the distortion in the plane of the supporting structure to reduce the non-planarity of the surface.

One can think of the surface plane calibration as a background subtraction approach in the projected height-map representation of the table top. This remedies both problems at once, camera distortion remaining after intrinsic and extrinsic calibration (see [53]) and unevenness of the building surface (*e.g.* two joined tables, or even carpet). The depth image is cropped to the table area through the use of URDF filtering (see Section 4.2.2).

At startup the system takes 100 depth images of the empty workspace, and computes an average surface deviation height map. Subsequently, all incoming RGBD images are rectified to the top down view using the ideal table plane equation, and the surface calibration map is subtracted from the height map, giving a height map of higher fidelity. This surface offset map is stored for later use, such that recalibration is only neccessary when changing the physical system setup.

In cases where this is not possible due to a more dynamic scenario setup, the calibration step is omitted, since it would require taking calibration images of an empty table top for every new viewpoint.

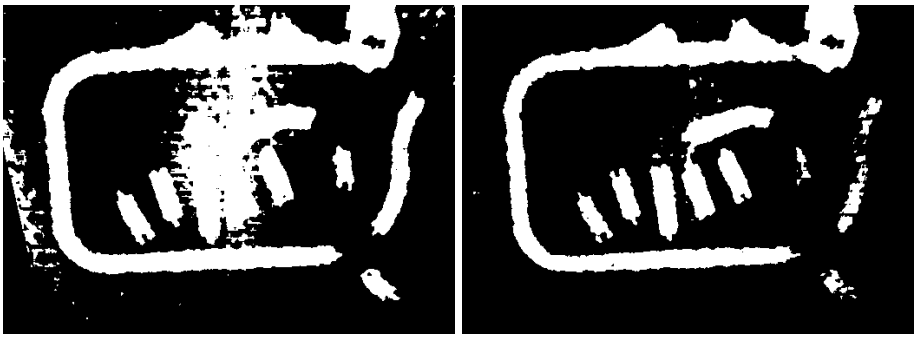


Figure 4.21.: *A simple thresholding operation that separates background from objects based on height can lead to gross mislabeling. Low threshold values (left) misclassifies higher areas of the table surface as object,* e.g. *at the image center, where as high threshold values (right) fail to accurately segment some objects,* e.g. *the right most track piece.*

A trivial segmentation step could now be performed by thresholding at *e.g.* 5 mm in this height map, leading to "ground" and "track" labels. However, the height of a pixel cannot be used by itself to decide between object and background association, as can be seen in Figure 4.21, where neither a low nor a high threshold value can successfully eliminate misclassification. This is due to the aforementioned distortions

in the sensor optics and the surface unevenness. Additionally, the relatively low resolution and the speckle pattern of the depth sensor leads to artifacts along the track boundaries, especially at the track piece end points.

Figure 4.22.: *Conservatively estimating table and object classes by setting very tight thresholds reduces our false positive rates for the respective classes. Blue shows table points, and green points on objects. We train a color classifier on these regions that can then be used to label all points in the scene.*

Therefore, a two-step segmentation process is employed: We first perform a very conservative segmentation by underestimating "table" and "object" points. points with height lower than 2mm are considered to be table, and points above 8mm height are labeled as object. This can be seen in Figure 4.22, where the conservatively estimated table regions are colored blue, and conservatively estimated track or object regions are shown in green. These regions are used to compute a color histogram in hue and saturation which is used to classify every point by means of histogram back projection. Histogram back projection computes the probability of a pixel belonging to either of the two classes and is commonly used for purposes such as blob tracking, face or hand detection, and also forms the basis of the CAMShift tracking algorithm [20].

This classifier is then used to grow these underestimated image regions much closer to the actual segmentation boundaries. This has the additional benefit that the RGB image of the Kinect can be obtained at twice the resolution of the depth image, leading to a segmentation as can be seen in Figure 4.23. Pixels outside our region of

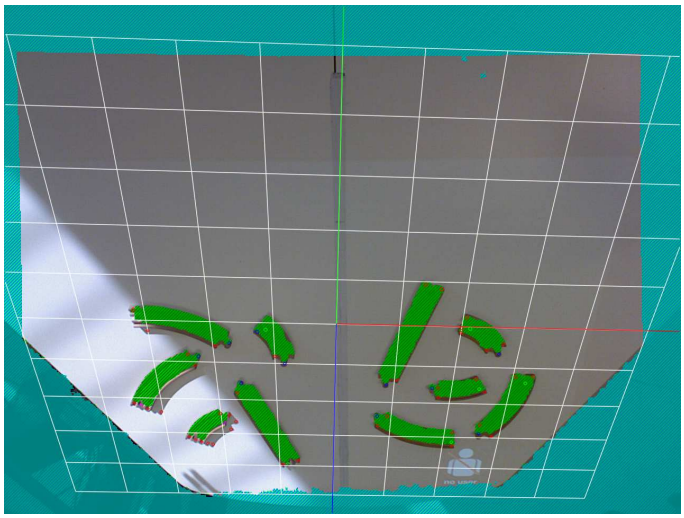


Figure 4.23.: *Example segmentation of track pieces using the surface calibration and two-step segmentation. Track pieces are overlayed with a green pattern, cyan depicts discarded regions and table regions are unaltered.*

interest bounding box are colored cyan, segmented objects are colored in green, and table points are unaltered.

Since the color classifiers are trained anew in each frame, illumination changes do not affect the segmentation step as can be seen in the sequence of images shown in Figure 4.24, where a strip of bright light from a window moves accross the scene. Automatic exposure adaptation on the Kinect results in changes in individual pixels' brightness within a class without affecting the segmentation.

As a result, the segmentation masks per object are reprojected into the original image frame and stored as `ReferenceImageROI`, referencing the high-resolution camera image present in the `CAS`.

This two-stage approach has been used in the aforementioned train track project, where two robot arms were used to pick up track pieces to build complete tracks in collaboration with humans. The setup has been used extensively, *e.g.* at the Automatica trade fair 2012, where it was on display for 4 days, continuously and reliably detecting all pieces scattered on the building area. While we regrettably did not perform an in-depth analysis in terms of under-/oversegmentation and accuracy due to time constraints, the segmentation quality was sufficient to allow the robot to reliably pick up each piece and join it with the track, which requires accuracy in the

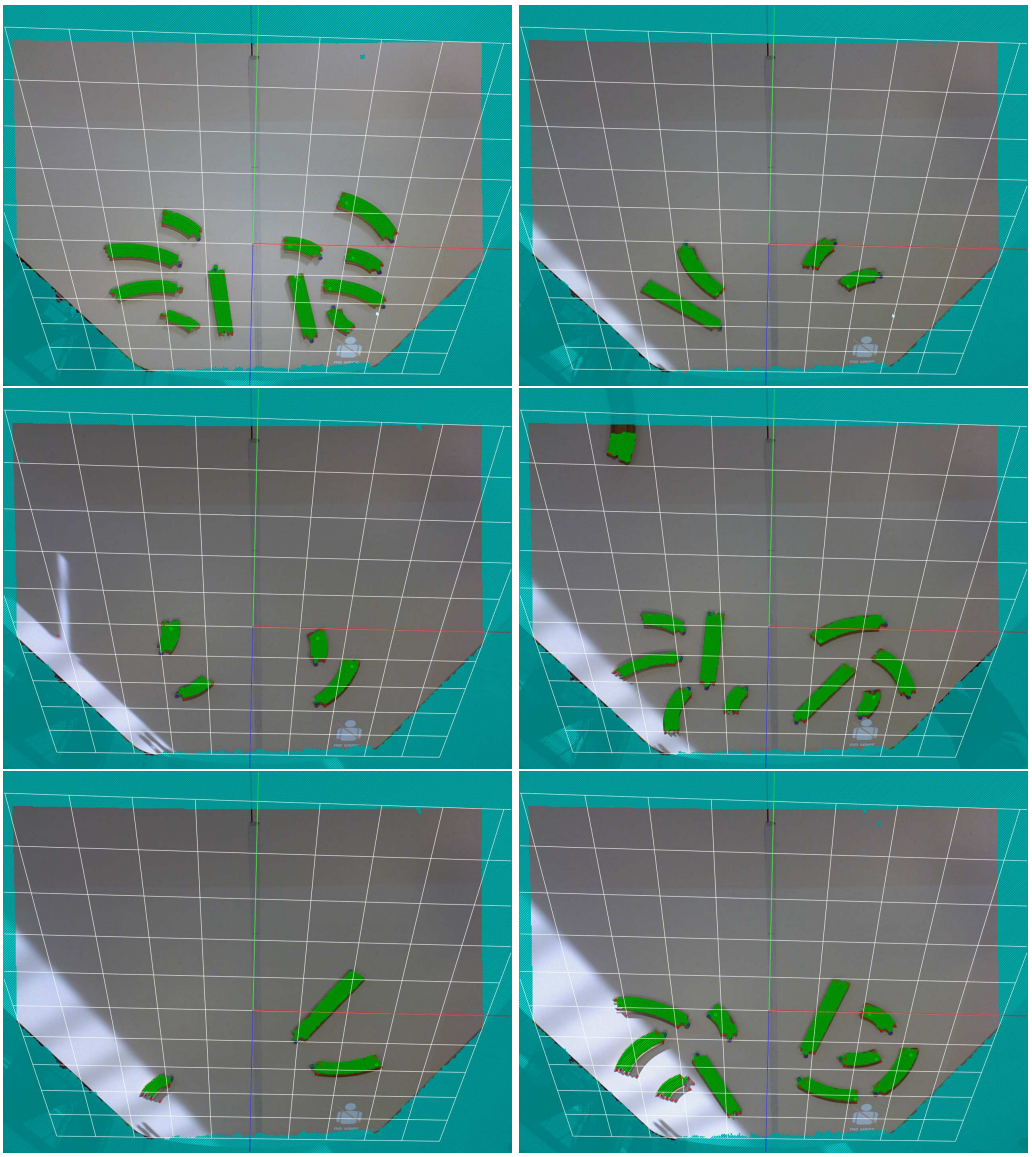


Figure 4.24.: *Illumination-independent segmentation: several scenes with different piece arrangements. The images show the same augmented camera view as Figure 4.23. Note the reboustness against illumination changes due to the fact that the color classifiers are retrained for each incoming sensor frame.*

range of a few millimeters. We believe this success in a dynamic environment with no “do-overs” to still be a very strong argument for the quality and robustness of our method.

4.4 Summary

In this section, we detailed some of the novel ideas that went into the creation of object hypotheses methods, specifically:

- the combination of a sensor error model and smoothing as a means to create an immensely efficient normal estimation algorithm;
- the use of a 3D semantic map in a filtering and segmentation algorithm that is flexibly parameterizable to provide filtering of unwanted spatial regions, specification of search regions for objects or custom-tailored analysis methods (*e.g.* door angle estimation) and robot-self-filtering, all in real-time;
- a two-step segmentation algorithm that creates an initial, conservative (under-) segmentation which can be used to train classifiers that are used in a higher-fidelity second stage. These classifiers are adapted per-frame and thus able to deal with changing illumination conditions and very flat objects with an elevation above support surface in the same order of magnitude as sensor noise.

RoboSherlock however offers more methods for object hypotheses generation which are of lesser novelty and thus not detailed in the scope of this work. These methods include mainly table top segmentation, *i.e.* segmenting objects protruding from a planar model fitted to data points, and euclidean segmentation, where a breadth-first region-growing approach joins points if their distance does not exceed a given threshold. First steps in integrating Depth-Adaptive Superpixel segmentation [171] and TextonBoost [147] have been already undertaken and show promising results in adding completely separate object hypothesis generators.

Chapter 5

Annotators Analyzing Object Hypotheses

The previous chapter described the generation of object hypotheses through mostly segmentation-based mechanisms. The result of these Annotators are a list of `ObjectHypothesis` Annotations that can be subclassed to fill specific needs of the object hypothesis representation, *e.g.* `ImageROI` types for regions in camera images or `ClusterPoints` for subsets of point cloud data.

An evident next step in an object perception framework is the analysis of these regions of interest. In the case of RoboSherlock, we provide several instances of this important class of object hypothesis Annotators, which will be described in this chapter.

In Section 5.1, we describe a very powerful and versatile Annotator, `GenericFeatureFromNormalsEstimator`, which can process a point cluster (that has estimated normal vectors) and compute any feature defined in PCL (specifically any subtype of `pcl::Feature`), depending on parametrization. This utilizes the largest collection of open source 3D feature computation methods, including Viewpoint Feature Histograms (VFH) [129], CVFH, FPFH, Spin images, RIFT, SHOT *etc*.

Additionally, we integrated our existing classification framework [89, 86] into a `FeatureClassificationAnnotator` capable of examining and classifying all object hypotheses that have a PCL feature annotation. As described in Section 5.2, it can be parametrized (*e.g.* to process all clusters with VFH features) and creates a `ClassificationAnnotation`.

As classification in general requires a database of trained objects, which is not feasible for all objects, RoboSherlock offers possibilities to reconstruct various additional

kinds information that do not rely on such an object catalog. One important idea was to make use of the power and versatility of existing web service technologies by offering the robot access to *Google Goggles*, as explained in Section 5.4.

Google Goggles is a web service allowing the analysis of an image with various different annotations. It returns a highly structured list of matches including product descriptions, barcodes, logo/brand recognition, OCR text recognition or a list of similar images. Note that for each Goggles reply, there is an abundance of additional information, such as URLs to web stores, price ranges, text translations *etc*.

Another Annotator capable of harnessing online data sources is explained briefly in Section 5.3. The approach uses a database of product images scraped from an online store that is then used to match products (along with their descriptions) to observations of objects. This product information contains categories, product names and descriptions, as well as information such as prices, weight, ingredient lists and perishability.

5.1 3D Feature Estimation

In computer vision, the term *feature* is used to denote a value or set of values describing an image area or area in a point cloud. In the following, the terms *pixel*, and *point* are used interchangably and can be interpreted as applicable.

On a high level, there are several dimensions along which one can differentiate them.

Locality
: There are features that operate on a per-pixel level, *e.g*. to designate the presence or absence of an edge or corner [52], and others describe whole image segments or regions, *e.g*. color histograms. In the context of 3D features one usually refers to *local* – computed individually for many or all elements in the input space – and *global* features, which are computed once for a given input, *e.g*. a complete object model.

Dimensionality
: The value of a feature is often called the *feature descriptor* and as such rep-

resents a point in a vector space, the *feature space*. A simple feature such as an edge detector can be represented with a single binary or continuous value, whereas a histogram of gradients [31] can require a very high-dimensional feature space. This directly influences the choice for data structures for storage and retrieval, ranging from flat lists to complex tree-based constructs.

Detection

While many features can be extracted for every pixel, it is often desirable to only compute them for promising, *i.e*. interesting pixels. In these cases, a *feature detection* step is used that finds *keypoints* that are distinct or interesting in the input space (*e.g*. corners). An expensive feature descriptor (in terms of computational or representational complexity) must then only be invoked on a subset of the original input space.

Invariance

Depending on the application domain, features are often designed to be invariant under certain transformation, *e.g*. rotation or scale, such that input data yield the same descriptor before and after the transformation. This way, a matcher can detect a known feature even when it is upside down or much closer to the camera than the training exemplar. *E.g*. SIFT [83] is invariant to scale and rotation, and also able to cope with noise and changes in illumination and viewpoint. VFH [126] on the other hand is purposely sensitive to the viewpoint to allow rough pose estimation from the matching process directly.

Distance metric

Often, factors such as noise, scaling or illumination changes influence a feature value, so perfect equivalence cannot be used when comparing values. In these cases, feature *distance* in some distance metric (*e.g*. Euclidean distance, histogram intersection or Mahalanobis Distance) is used to measure *similarity* of feature instances. In classification of features, these distance metrics play an integral part, as will be shortly discussed in Section 5.2.

Other noteworthy properties of features include stability for rededection or tracking purposes, averageability and additivity of features (*e.g*. GRSD by Marton et al. [86]), confidence, robustness under noise or other influences, expressiveness (*e.g*. do they encode visual or geometrical properties), and obviously computational complexity.

While we will only give a very brief overview here, there is a review of a large variety of common feature descriptors for computer vision by Li and Allinson [79], covering corner detectors, region detectors, feature descriptors based on filters and on distributions, textons and so on. Please refer to Chapter 9 for more details on the related work in this area.

For 3D shape matching, efficient point cloud registration or object recognition, more recent 3D feature descriptors have been introduced that commonly capture characteristics of a local point neighborhood to encode the surface properties around a query point. Notable examples include spin images [62], curvature maps [47], conformal factors [9], RIFT [76], SHOT [162, 163] as well as the family of point feature histograms: point feature histograms (PFH) [127], fast point feature histograms (FPFH) [128], viewpoint feature histograms (VFH) [126] and clustered viewpoint feature histograms (CFVH) [1], many of which were co-developed during the course of this thesis. The numerous variants of this descriptor family each strive to achieve different goals, *e.g.* pose invariance (PFH, FPFH), reduced computational complexity (FPFH), pose reconstruction while matching (VFH, CVFH) or matching partial object views (CVFH). Again we refer to Chapter 9 for more details.

Within the Point Cloud Library, many and more of these 3D features are incorporated into the largest collection of open source 3D feature computation methods. As they are all implemented as subclasses of a general `pcl::Feature` class, this allowed us to implement an aptly named *GenericFeatureFromNormalsEstimator*, which is an annotator with the possiblilty to process any point cluster (which has normals estimated) and compute any subtype of `pcl::Feature`, depending on parametrization.

As described in Section 3.2.1.2, the annotator produces a `PclFeatureAnnotation` for each computed feature, and can *e.g.* be invoked on every `Cluster` present in a `Scene`.

Programmatically, the Annotator can be configured by the parameter `feature_type`, which is internally mapped to a lookup table of `FeatureDelegate` objects, which wrap the behaviour of the specific feature in question.

5.2 Classification Framework

The process of digesting the information encoded in feature descriptors to reach a decision concerning a higher-level statement about a region or object is called classification. It can be thought of as a projection from the feature space into a classification space, mapping groups of "similar" feature values into classes that could describe *e.g.* object instances, object categories, surface types, color classes *etc*.

The employed classifiers are usually differentiated into those that require a training set of input data (feature vectors) labeled with their target output classes (supervised learning) and those that do not require pre-labeled ground truth data (unsupervised learning). A third category is that of reinforcement learning, which defines models that can be adapted over time based on actions taken in a target environment (or within a given problem) that seek to maximize a reward function.

Supervised methods include the popular support vector machines [17, 29] (SVMs), a kernel-based methods that bases its decisions on a separating hyperplane in a high-dimensional space into which training points are mapped through a kernel function. The resulting classifier, while slow to train, is very fast to use when classifying previously unseen features.

K-nearest neighbor is another supervised method that classifies features based on a majority vote of its k closest neighbors. This means that training k-NN effectively amounts to storing all training vectors and class labels, while during testing, the class label most prevalent in the k nearest neighbors' labels is assigned to the tested feature.

For RoboSherlock, we opted to integrate our existing classification framework [89, 86] into a `FeatureClassificationAnnotator` capable of examining all object hypotheses that have a certain `PclFeatureAnnotation`. It can be parametrized to select the feature to be used, the classifier and a distance metric (*e.g.* to process all clusters with VFH features using k-NN and the Euclidean (L2) distance) and creates a `ClassificationAnnotation`. As of now, this annotator provides object-level classification using global point cloud features, but the framework supports any type of feature (and bag-of-features), so it can potentially be used for classifying local features as well.

The `ClassificationAnnotation` type contains the class label, a list of class confidences & class accuracies ($\langle label, probability \rangle$ pairs), and information on which feature space and classifier combination was used to reach this decisision.

The class confidences and accuracies are reported by the classifiers and are useful for ensemble learning [89, 161].

Note that this annotator can be used to process past data, which is interesting for time consuming operations like post-processing large amounts of data collected over extended periods to complete models for the universe of objects present, or to create and test classifiers and ensemble of experts methods.

For the scope of this thesis, we treat this classification framework as a black box, a tool that can be used out-of-the-box and that has been analyzed and evaluated in previous topical publications. Just as an example, we show two results of experiments using 4 class labels (sphere, box, flat and cylinder) and PFH [127] features, using two different classifiers, k-NN (Table 5.1) and SVM (Table 5.2).

The experiment contained 25 percepts of spheres, 95 boxes, 76 flat objects, and 110 cylinders, and as can be seen, boxes were very often mistaken for flat objects using both classifiers, while other classes had a much higher success rate.

Table 5.1.: *Classification with PFH, using k-NN, where $k = 10$, and with Chi-Squared metric. The top row contains the classification result, and the leftmost column the ground truth class label.*

	sphere	box	flat	cylinder	success rate
sphere	25	0	0	0	100%
box	3	41	47	4	43%
flat	0	4	71	1	93%
cylinder	0	3	12	95	86%

A `ClassificationAnnotation` for a single classification result in this experiment can then contain the information shown in Listing 5.1.

As can be seen, the label “cylinder” was given since 9 of the k-NN votes (with $k = 10$) were for this label, and only 1 of the nearest neighbors was disagreeing. Information on the classifier, parameter values, employed model and individual class confidences is available for possible future Annotators which might be interested.

Table 5.2.: *PFH descriptor and SVM classification. The top row contains the classification result, and the leftmost column the ground truth class label.*

	sphere	box	flat	cylinder	success rate
sphere	24	0	1	0	96%
box	4	36	55	0	38%
flat	0	5	71	0	93%
cylinder	1	0	10	99	90%

Listing 5.1 JSON representation of the ClassificationAnnotation as stored in the database.

```
{
  "_type" :"ias.uima.classification.ClassificationAnnotation",
  "_id" :{  "$oid" :"5109964f47ae251e851d0440"},
  "type" :"shape",
  "classname" :"cylinder",
  "featurename" :"PFHSignature125",
  "classifier" :"knn",
  "parameters" :"-m chisquared -k 10",
  "model" :"pfh_knn",
  "confidences" :
    [
      {  "_type" :"ias.uima.classification.ClassConfidence",
         "name" :"sphere",
         "score" :0.9},
      {  "_type" :"ias.uima.classification.ClassConfidence",
         "name" :"box",
         "score" :0.0},
      {  "_type" :"ias.uima.classification.ClassConfidence",
         "name" :"flat",
         "score" :0.1},
      {  "_type" :"ias.uima.classification.ClassConfidence",
         "name" :"cylinder",
         "score" :0.0}
    ]
}
```

In future work, we plan on using not only the classification result per se, but also this auxiliary information in order to assess the quality or significance of a label given information about the reliability of the chosen combination of feature, classifier, model and parameters.

5.3 WWW Knowledge Sources — Scraping Online Stores for Product Annotations

Rich information obtained from online sources is a promising source for individual or categorical semantic knowledgethat can help in acquiring the neccessary common sense knowledge to perform complex task, as shown by Waibel et al. [170]. An example is the Google 3D Warehouse[1] (now Trimble 3D Warehouse), a large collection of 3D CAD models that has been used in the past to train object recognition systems for household objects like furniture or dishes [66, 182, 97]. Lai and Fox [73] have reported on using domain adaptation to facilitate the transfer of classifiers trained with synthetic data to real sensor data.

As will be shown in the following section, object descriptions gathered from sensor data can be greatly enriched by links to online data which can contain 3D models, product images, structured object descriptions in web stores or other knowledge bases. It also allows to reason on more than what is visible, *e.g*. backsides, required storage temperature, weight *etc*. [155]. This combination of perception and knowledge reasoning widens the scope of a "vision system" and allows to treat perception as an integrated capability of a robot, with access to much richer information and capabilities to answer complex queries about the environment [107].

In previous work [155], we have reported on the acquisition of highly structured product data bases including images, textual descriptions and information pertaining to *e.g*. perishability and ingredients by preprocessing web sites of online retailers such as *e.g*. GermanDeli.com. This information is scraped and stored offline in a database for ready access by any annotator. An interesting possibility is the training of appearance features from the product images for image based recognition of objects, or textual searches for product descriptions mentioning *e.g*. "Nestlé" or "Kellog's", after corresponding logo/brand results from the Goggles annotator (*cf*. Section 5.4).

We incorporated the available "ODUFinder"[2] ROS package by our colleagues Vladimir Haltakov and Dejan Pangercic into RoboSherlock. It extracts SIFT features [83] and vocabulary trees from the product images of about 3500 objects from the GermanDeli

[1] http://sketchup.google.com/3dwarehouse
[2] http://www.ros.org/wiki/objects_of_daily_use_finder

web store to offer a visual search interface of objects hypotheses. This Annotator creates a `ClassificationAnnotation` and a `ProductAnnotation`.

5.4 Using Web Services as Annotators — Google Goggles

So far, we mainly focused on *"offline"* Annotators that contained single algorithms with a clearly defined purpose such as segmentation of table-top surfaces or feature-based object classification. However, as smartphones, tablets and other mobile devices with ever increasing capabilities continue to infuse our daily lives, we witness the creation and evolution of powerful web services dealing apparently effortlessly with problems traditionally considered hard or even impossible. Examples include Apple's Siri voice interaction system[3], which is a commercialization of the CALO project[98], or the visual search assistant Google Goggles [4], which allows smartphone users to take photos that get analyzed by Google's servers.

Note that the idea of using Google Goggles in robotics has been picked up by *e.g.* Kehoe et al. [63], however they fail to realize the true potential of the approach: In their implementation, Goggles is trained in an offline phase using data sets of manually labeled images. Semantic data, *e.g.* CAD models or object properties such as weight or center of mass are also manually linked to this object catalog, while grasp points are estimated by GraspIt Miller and Allen [92]. This reduces Goggles to a lookup system to index the object database. In this chapter however, we attempt to gain *a priori unknown* knowledge about objects that the robot has never seen before. We believe that the true power of using such online services does not lie in merely out-sourcing computing resources, but in adapting and making use of massive, potentially user-contributed systems that are geared towards a much larger audience in a different context. We hope that eventually, manual labeling and training of custom-tailored databases will not be neccessary anymore and make way for more flexible, powerful and general approaches.

As a side note, there are other potentially useful services that might not seem relevant to robotics at first, such as online maps, business listings, or social networking platforms, however these also represent interesting opportunities depending on the scenario. Examples include urban mobile robots, which navigate using GPS and *e.g.* Google Maps, or robots that interact with and assist the elderly or people with mo-

[3] http://www.apple.com/ios/siri
[4] http://www.google.com/mobile/goggles

tion impairments in all aspects of their daily lives using new outlets, social media or online entertainment services.

Considering this possible wealth of available systems, it seems logical to attempt to tap into this development and benefit from the efforts that went into the design and implementation of these services.

In our scenario of perceiving everyday objects, Google Goggles is possibly the service promising the most direct applicability to the problem at hand. We therefore implemented an Analysis Engine which can annotate any image region stored in a CAS using the responses obtained from querying Goggles.

In the following sections, we will describe the Goggles service and its capabilities (*cf*. Sec. 5.4.1), and give detailed information about the message protocol definition (*cf*. Sec. 5.4.2). We will analyze the merit and mechanics of acquiring these annotations in our scenario and demonstrate the rich object descriptions we can obtain (*cf*. Sec. 5.4.5).

5.4.1 About Google Goggles

First released in 2009, Google Goggles is a smartphone app available on Android and iOS devices for *visual search*. It is meant to add a third paradigm for searching on mobile devices, besides text search and voice search, by letting the user acquire a photo using the device's camera which is then treated as a search query by the application. The actual image is processed by an online web service which processes the query image and responds with a list of possible search results.

One can merely speculate on the architecture behind Google Goggles, since there are no scientific publications or in-depth articles. However, Google announced [5] that there are plans to open it as an "app platform" with an open API for developers to create solutions on top of the Goggles infrastructure. This suggests that the underlying architecture might not be much different from the one proposed in this thesis, in terms of ensemble-of-experts which span a set of hypothesis or annotations which

[5] http://phandroid.com/2010/04/14/google-goggles-to-become-app-platform/

get weighted and filtered before being presented to the user. We will therefore use similar terminology when referring to Goggles.

The annotations that are returned in a response for a search image can be classified into the following categories:

Product
: denotes that an object was detected that resembles a product found on one or more online shopping sites.

Text
: refers to OCR (optical character recognition) results.

Book
: can be thought of as a Product result as well, but contains additional information (*e.g.* ISBN numbers) that make sense only in the context of books.

Logo
: results are among the most interesting, since they are generally well recognized and add a strong indicator when comparing two objects observations.

Landmark
: hits rarely occurred in our test images, but would in theory describe well-known landmarks, such as the Eiffel tower. We did once get a Landmark result of a famous Coca-Cola billboard when searching using an image of a coke can.

Similar Image
: simply refers to images Goggles found on the internet that are visually similar. We found these to be quite unreliable, as it seemed that the overall image color was the strongest discriminator for the selection of Similar Image results.

Barcode
: responses contain a barcode detected in the image, with additional information concerning the Product marked with it.

User Submitted Results
: are the mechanism with which Goggles allows common users to improve the results in case Goggles performed poorly. In this case, a user can specify a name

for the misdetected image, which will be incorporated for future search results. This is especially interesting with *e.g.* regional products which are not commonly found in web stores (where Goggles would otherwise find it as a Product match).

The most important data fields that are associated with the various categories are shown in Figure 5.1. Let's first however investigate how communication is implemented for the Goggles service.

5.4.2 Protobuf and Message Definitions

Since the Goggles service is composed of two parts, the on-board app and the off-board processing backend, communication is a central topic when discussing Goggles' inner workings. There are two message types that are used:

- The *Request* is sent from the app to the backend server and encodes the camera image taken by the user, as well as some additional details, such as language and localization information or device identification.
- The *Response* is returned after analysis and contains one of two things: if interesting artifacts were detected, they are returned as a list of category-description pairs with additional information. An example could be a result of category *Logo* with description *"Kellog's"*, as well as details on *e.g.* where in the image the logo was detected and links to a higher-resolution version of that logo. In essence, these additional fields contain all information that the app requires to display to the user, or to provide actions to the user (*e.g.* search online, link to product pages, translate text).

The actual messages are encoded using a serialization protocol called *Protocol Buffers* (also protobuf). It relies on the definition of an interface description (in this case, the request and response message formats), which is used to serialize structured data in these formats into a byte stream and recreate the structured data using the protobuf byte stream and the message definitions. Protocol buffers are inherently less verbose than other common message interchange languages such as XML or JSON, and are both forwards-compatible and backwards-compatible, which makes

it possible to update these message definitions over time as necessary (to a certain degree).

Protocol buffers are however not self-describing, so a priori knowledge about the protocol definition is required to parse a serialized message correctly. Since there is no openly available API for the Goggles platform as of this writing, we were forced to reverse-engineer these protocol definitions by sniffing the network traffic between the app and the backend servers. Most important fields could be determined quite easily, while the significance of various short numerical values and hexadecimal strings could not be discovered. However, the forward-compatibility property enables the parser to skip over message parts that are unknown.

To help with detection of new message fields, our implementation double checks the parsed information in the following way: After receiving and decoding a message, we use the current response message definition to re-encode the decoded message. This is then compared with the original message, and if the two messages differ, their structural differences are stored to disk so that they can be analyzed manually to identify the addition of new message fields.

5.4.3 Request Message Definition

The message definition for a minimal request that can be sent for processing that still results in a complete result is given in Listing 5.2. Note that these message definitions are likely best read bottom-to-top, since the most encapsulating type can only be defined after the types it depends on.

We will refrain from showing more detailed information in the protobuf message definition format and refer the reader to Appendix A.

The most important part of the Request is obviously the uploaded image, which is encoded in a JPEG bytestream, and serialized into the field `image_bytes`. Other than that, it is important to specify a language in which to expect results in the struct `LanguageID`, where one must also set a flag to enable extended information in the Response.

Listing 5.2 Minimal request message definition that warrants a fully defined response.

```
package goggles;

// Image is represented as a JPEG byte stream,
// wrapped into a 2-level deep struct hierarchy.
message Image {
  optional bytes image_bytes = 1;
}
message WrappedImage {
  optional Image image = 1;
}

// Omitting this struct disables detailed information in the response.
message EnablesUIResponse {
  optional int32 val1 = 1 [default=1];
  optional int32 val2 = 7 [default=1];
}

// LanguageID specifies the device's language settings.
// This can affect e.g. OCR Text translation results
message LanguageID {
  optional string string1 = 4; // e.g. "en"
  optional string string2 = 6; // e.g. "US"
  optional EnablesUIResponse enable_ui = 15697207;
}

// the minimal goggles request contains the image
// and language settings
message GogglesRequest {
  optional WrappedImage wrapped_img = 1;
  optional LanguageID lang = 10;
}
```

The minimal working example to call Goggles is shown in Listing 5.3 as a Python code snippet. In line 1, we import the message definition (or rather, a python module compiled from the message definition). From line 3 on, we define a function which, given the filename of an JPEG image, returns a Request protobuf object ready to be sent.

The first step is to read in the JPEG file (lines 4 & 5), and subsequently create a Request object (line 7) and filling it with the image bytes (line 8). The remainder is concerned with setting the values required for specifying the language (lines 10 & 11) and requesting extended information in the Response (lines 12 & 13).

Listing 5.3 Exemplary Python code showing the usage of the given protocol buffer definition.

```
1  from proto import request_pb2
2
3  def create_goggles_request (image_filename):
4      with open (image_filename, "rb") as f:
5        jpeg_image = f.read ()
6
7      request = request_pb2.GogglesRequest ()
8      request.wrapped_img.image.image_bytes = jpeg_image
9
10     request.lang.string1 = "en"
11     request.lang.string2 = "US"
12     request.lang.enable_ui.val1 = 1
13     request.lang.enable_ui.val2 = 1
14
15     return request
```

5.4.4 Response Message Definition

The `GogglesResponse` message can contain multiple `Info` fields, which contain the actual responses from different expert algorithms, if any. It can also contain multiple `AlternativeInfo` structures, which in turn encapsulates a `Info` message and some additional, as of this writing unknown fields.

Figure 5.1 shows the information represented in a Response message and the various Response categories. We again refer to Appendix A for a detailed message definition.

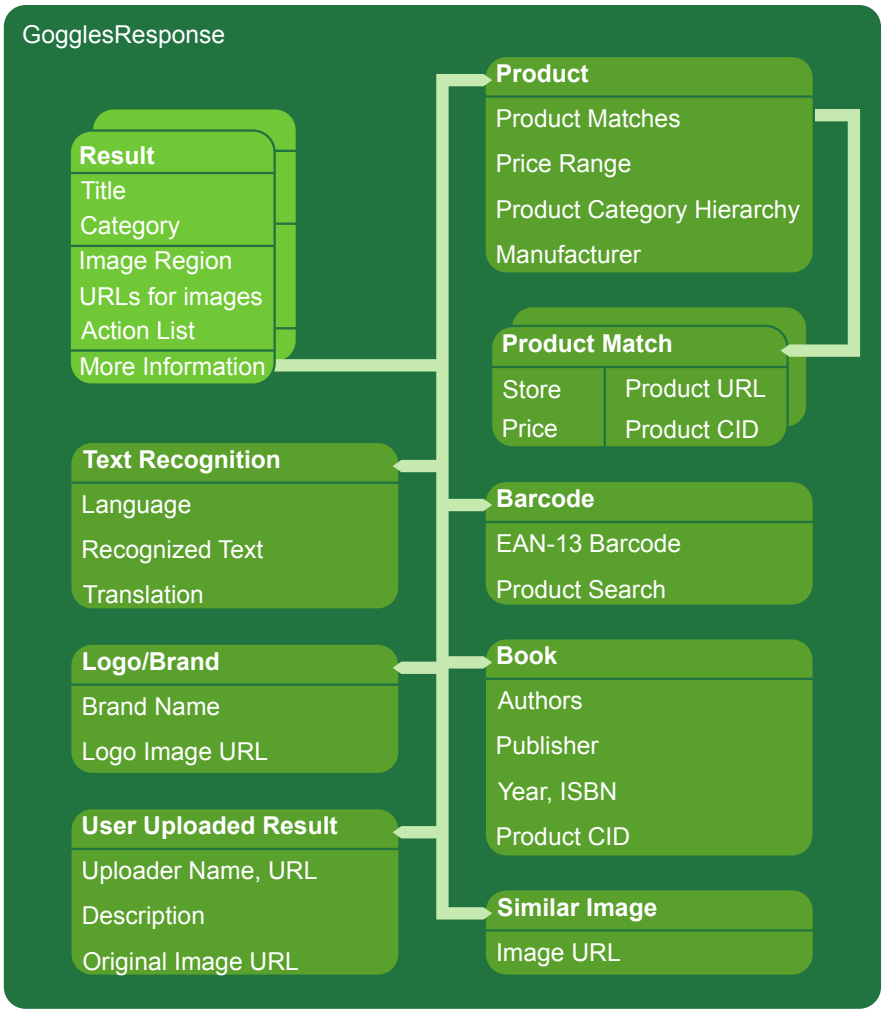


Figure 5.1.: *Schematic overview of Goggles responses: A return message from Goggles contains multiple Result fields. Each has a category and a title, as well as additional information concerning* e.g. *which image region contained the result, links to preview or high-resolution images, and a list of actions that can be performed* (e.g. *Search more, Visit Store* etc.*). Depending on the result category, more information is represented with specific fields. The most relevant result categories for our application scenario are shown along with their most important fields.*

The `Info` message contains the two most important sources of information, which is the `category` and `title` fields. Category initially was used for specifying the type

of result that was returned, *e.g.* “Product” or “Logo”. Recently, Goggle has started to treat this field as a subtitle to the actual title field, *e.g.* showing the web site where a result was found.

There is another important struct which encapsulates all additional information that the Goggles App would require to present to the user, such as the bounding box within the search image where a result was found.

During the development of the RoboSherlock Goggles client, we also created a graphical user interface to help in identifying the various fields and visualize the transmitted information. An example can be seen in Figure 5.2, where we show a graphical and a structural representation. The product contained in this image yielded two responses, one detailing the product (highlighted in red), and one containing all text recognized within the photo, highlighted in blue.

5.4.5 Experiments

To test the validity of the Goggles Annotator for application to our target scenario, we performed an experiment involving regular food and household product images taken at a supermarket. A total of 147 products were photographed (a subset of which is depicted in Figure 5.3).

The images were downsampled to a resolution of 584 by 388 pixels and analyzed by the Goggles Annotator. For the 147 images, we got a total of 371 Goggles responses, so 2.5 responses per image on average. As can be seen, Goggles successfully detects text written on one third of the objects, and a brand logo on 22.1% of the objects. Similar Images (which have a textual name!) are found for 18.9%, and Product results for 11.3% of all objects. User submitted results are similar to similar images in that they refer to an online photo taken by a fellow Goggles user, and they also carry a textual description. They account for 9.4%, and barcode results for 3.8% of all results.

All of these results carry actually interesting information concerning the objects. In fact, 97.8% of all results contain informative features, and all of them have a textual description, and potentially more, such as links to images or product pages.

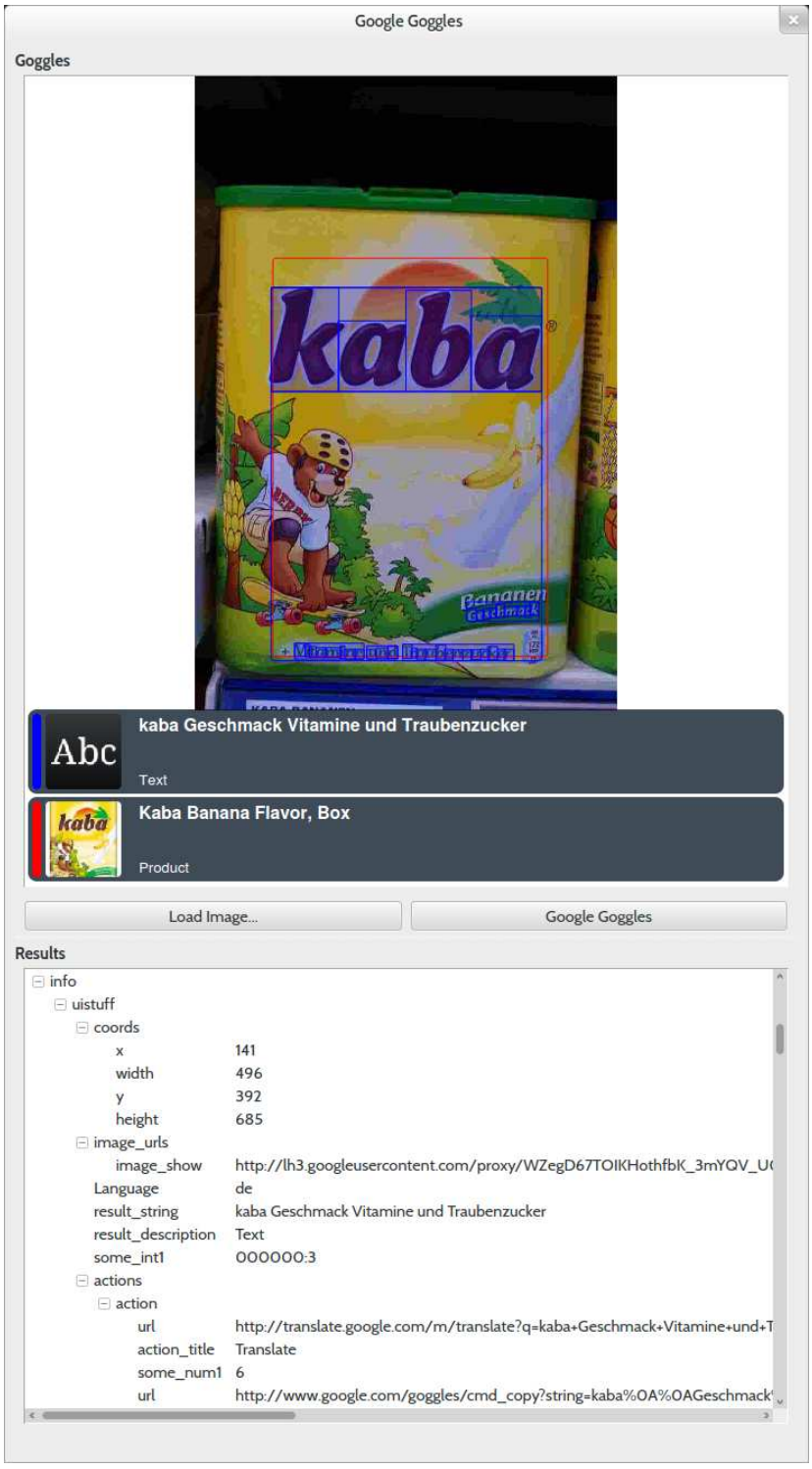


Figure 5.2.: *Example of a Goggles response for a powdered banana drink, shown in our reverse-engineered goggles client GUI. On top, we visualize the image and bounding boxes for results, and on the bottom a tree-view can be used to inspect results in more detail. A Text and a Product result were obtained, along with links to an online store and translations.*

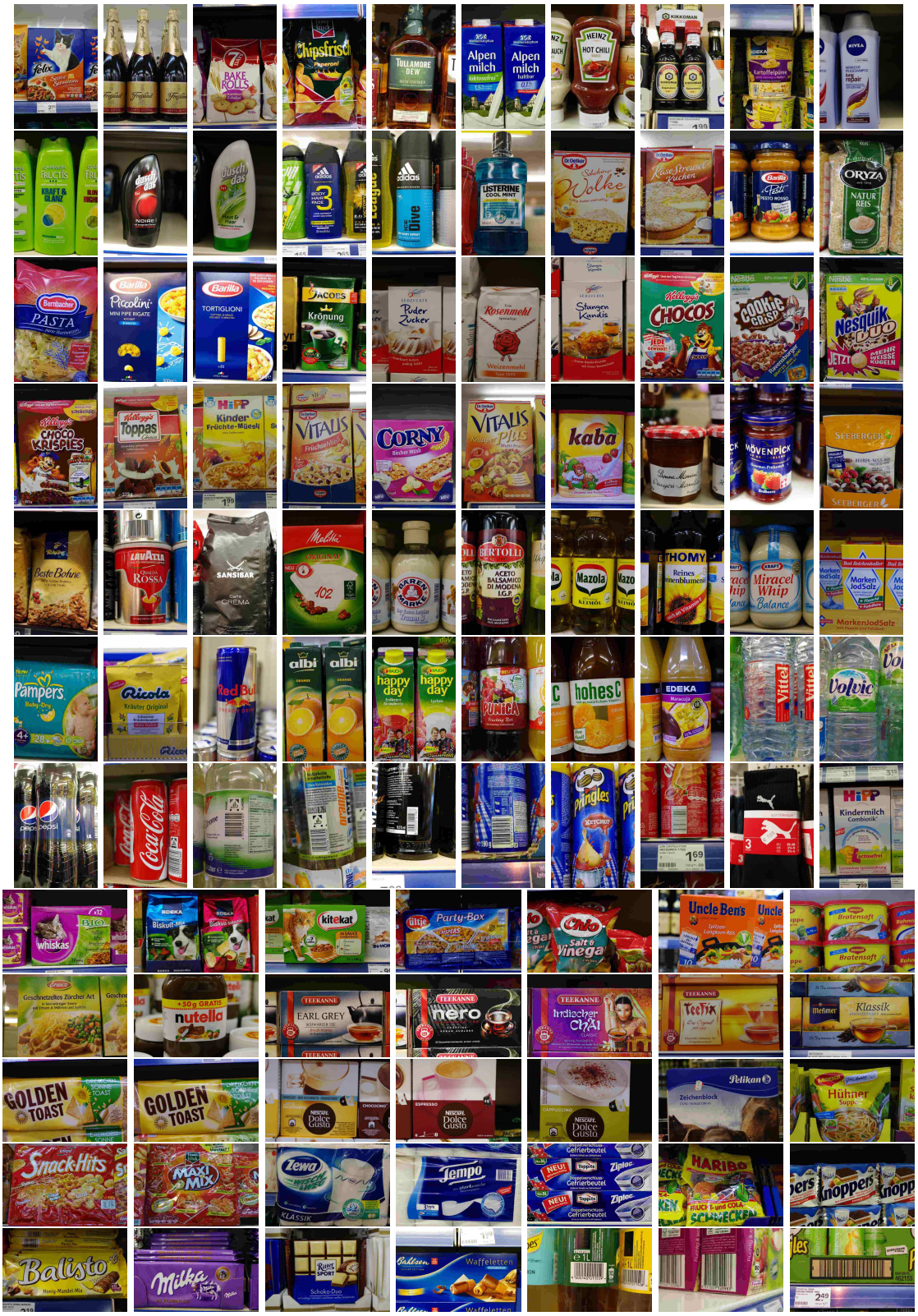
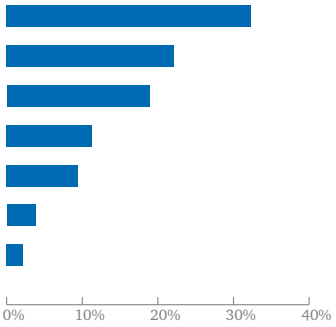


Figure 5.3.: *Subset of test images for evaluating Google Goggles.*

Table 5.3.: *Frequencies of different categories for Goggles response*

category	# occurence	relative frequency
Text	120	32.3%
Logo	82	22.1%
Similar Image	70	18.9%
Product	42	11.3%
User Submitted Result	35	9.4%
EAN-13 Barcode	14	3.8%
Other	8	2.2%
(total)	371	



The abundance of textual information in each Goggles response gives rise to the idea to use simple word-based statistics to determine relevant descriptive terms for a given object observation.

We therefore proceeded to extract words from the responses (defined as strings of letters with at least 3 characters). A list of certain words were filtered out as they were common to all responses (*e.g.* “www”, “google” *etc.*). The remaining words were counted by occurence (as most important words occurred multiple times per response) and the top 5 words by frequency selected to be terms that describe the object. Examples can be seen in Table 5.4. Note that all images have been analyzed only once, that is to say the word occurance frequencies are referring to occurances within a single Goggles response. Erroneous text recognition sometimes leads to misspelled words, but due to the frequency analysis, they tend to be pushed back in favor of more stable and frequent terms.

We manually counted how many of these 5 words were *relevant terms* for the respective product. We treated words to be relevant if they contained the name of the product, the brand or manufacturer name, were retrieved via the barcode, or text that is written and detected on the product, since these words are likely to be redetected in future encounters of the object. As can be seen in Table 5.5, there were some few objects where all 5 terms were relevant (7 instances or 4.8%), and even less where none of the words were relevant (5 instances or 3.4%). The rest contained between 1 and 4 relevant terms, with 3 and 4 relevant terms appearing for 64% of objects.

Table 5.4.: *Some examples of the 5 most frequent terms appearing in Goggles responses for selected objects. Note that each object was analyzed only once, so the displayed numbers count the occurence of a word within a single Goggles response.*

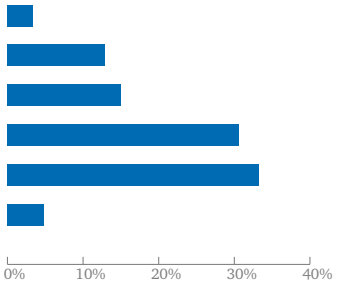
	3	Chio		1	tullamore
	3	Chips		1	vhttp
	4	Salt		1	wds
	5	chips		1	whiskey
	5	egal		2	translate
	3	laktosefrei		2	Similar
	3	Weihenstephan		2	translate
	4	Fett		3	http
	4	haltbar		3	Kikkoman
	8	milch		8	Sojasauce
	2	images		4	Traubenzucker
	2	translate		4	und
	4	Aceto		4	Vitamine
	4	Balsamico		5	kaba
	7	Barilla		9	Kaba
	4	Gati		4	and
	4	Hate		4	Chips
	4	Storck		4	Chipsletten
	5	http		4	Hot
	7	Knoppers		4	Lorenz

Furthermore, in 105 of the 147 cases, the single most frequent word was a relevant term as by our definition.

As said before, the list of filtering words that we ignored contained those words that were common to *all* goggles responses. However, there were a number of additional words that should be treated as "noise", such as *e.g.* "translate", "Similar", "images" and especially "http". For future iterations of this Annotator, we plan on combining this idea of filter or stop words with dictionary lookups in freely available word lists to also filter out nonsensical words such as *e.g.* "wds". This way, a robot evaluating a scene can find relevant and descriptive words for observed objects automatically and display them in a visualization or even name them using text-to-speech software.

Table 5.5.: *Frequencies of relevant terms within the 5 most frequent words per Goggles response*

# relev. terms	# occurence	relative frequency
0	5	3.4%
1	19	12.9%
2	22	15.0%
3	45	30.6%
4	49	33.3%
5	7	4.8%
(total)	147	



We also measured the time needed for each Goggles request, and the result can be seen in Table 5.6. The mean for lookup times was 867 ms, and the minimal and maximal times were 504 ms and 2314 ms, respectively. For multiple clusters in a single scene, the Goggles requests can be performed in parallel, however that still does not suffice for real-time performance.

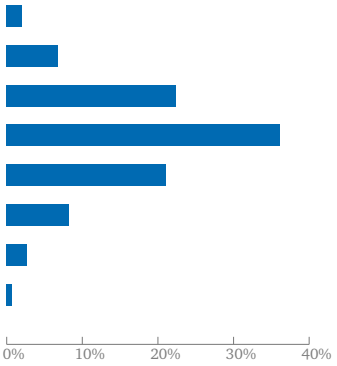
For this reason, it makes sense to perform all Goggles requests asynchronously, in a pipelined manner. In this case, the Goggles Annotator must be called after an ID has been given to the cluster (*e.g.* by a TrackingAnnotation, *cf*. Section 6.5). The Annotator can then spawn a Goggles request thread per cluster in the scene, and return control to the flow controller, which can in turn call subsequent Annotators. Once a thread is finished with the Goggles request, it can periodically test whether the Cluster object is written to the data base and append its results there.

This is however not ideal either, since subsequent Annotators do not have access to the Goggles annotations, so real-time performance has to be weighed against ready availability of the extracted information. This is usually not much of an issue, since during the round-trip time of around $0.5 - 2$ seconds, most scenes remain mostly unchanged. To alleviate this situation and in order to reduce network bandwidth, the Goggles annotator also contains a cache that uses perceptual hashing [180] on the cluster image to prevent continuously resending very similar image regions. This principle of composition of change detection with any algorihm could be applied to a large number of annotators and should in future work be factored out as a general capability of RoboSherlock.

In the future, we plan to overcome this class of problems by treating asynchronous Annotators specially with a custom flow controller that can join threads spawned in this way. Annotators that do not rely on Goggles annotations can then be run in the mean time, and those that do need them are queued until the asynchronous worker threads have finished.

Table 5.6.: *Lookup times per Goggles request, with a mean of* 867 ms.

lookup time [s]	# occurrence	relative frequency
$t \leq 0.6$	3	2.0%
$0.6 < t \leq 0.7$	10	6.8%
$0.7 < t \leq 0.8$	33	22.4%
$0.8 < t \leq 0.9$	53	36.1%
$0.9 < t \leq 1.0$	31	21.1%
$1.0 < t \leq 1.1$	12	8.2%
$1.1 < t \leq 1.2$	4	2.7%
$1.2 < t$	1	0.7%
(total)	147	



In summary, Google Goggles has proven to be a very informative and diverse Annotator that offers a powerful way of analyzing a priori unknown objects. In fact, since the backend data base of Google Goggles must be very vast and potentially has access to the whole web and various indexes Google creates for their manifold search products, we believe this to be an Annotator that will grow in functionality and importance in the future. Over time during development of this thesis, the message formats have been extended with additional information regularly.

Chapter 6

Annotators for Object Identity Resolution, Information Fusion and Storage

In the previous chapters, we have outlined several annotators that can generate object hypotheses using camera or 3D sensor data (*cf*. Chapter 4) and annotate them using various recognition, reconstruction, classification and analysis tools (*cf*. Chapter 5). The cumulative per-cluster Annotations created by these modules are very rich in information content and multivariate by design.

Another important aspect however is not answered yet by these modules, which is that of reasoning over *objects* and *actions* performed on them given *observations* of objects. Consider the robot show in Figure 6.1, which has in the past grasped a mug form the cupboard in the back and is currently placing it down on the table.

Shown in the overlayed thought bubble, the robot should have both a *temporal* understanding of each location, *e.g*. the progression of object arrangements on the table and the actions performed on them, and *spatial* information such as where the same or a similar object has been seen previously, or what is currently in the cabinet.

Similarly, if the robot drives past a certain location several times, and is consequently asked to list the objects at that location, the robot should not have to look there to answer the question. Instead, it makes sense to *passively* and *preemptively* maintain a belief state about the environment to be able to directly answer such queries, and to assist in other tasks where such a belief state is potentially beneficial, *e.g*. situation assessment or generating an object model catalog while performing other tasks.

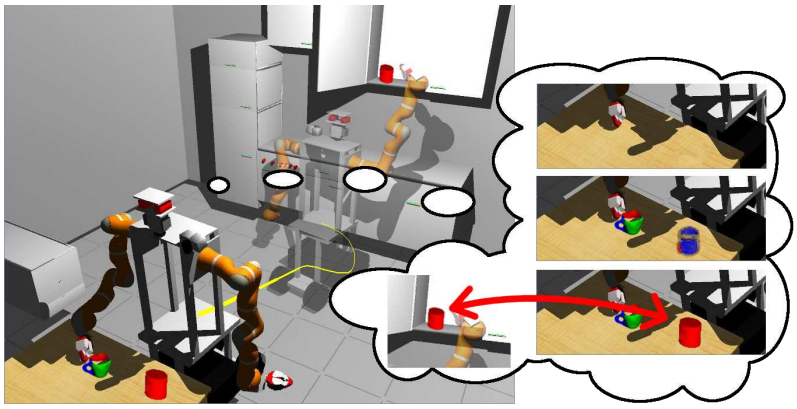


Figure 6.1.: *A service robot in a household environment must be able to reason about the temporal and spatial aspects of objects found in the environment.*

6.1 Object Identity Resolution

As argued throughout this thesis, autonomous service robot systems acting in a human environment in a goal directed manner need rich semantic information about their surroundings. The requirements include knowledge about objects, places, furniture, affordances and spatial relations between these. Trying to *combine* all this is in a working system performing in human environments is in general a very hard task.

To make matters more complicated, our proposed system is capable of processing data from a variety of visual sensors, such as regular cameras, laser range finders, 3D time-of-flight or stereo setups, but also other sources of information, such as web services, online ontologies and knowledge bases. This leads to a set of very different representations in terms of annotations produced within RoboSherlock. In order to be able to make actual use of all the bits of information gathered through multi-annotator pipelines, but also gathered over time, it is essential to develop a system capable of abstracting away from these heterogeneous, low-level object descriptions and formulate statements concerning the identity of objects, object histories and object-centric events.

This means the perception system on the lowest level needs to deal with very different representations. Furthermore, the algorithms that work on these raw sensor data produce different representations too, *e.g.* a histogram over color distribution from an image analysis system or the classification results based on a 3D feature extracted from one of the 3D sensors.

Sometimes, the robot might be able to collect information about a certain object using all sensors available to him, however in general, this might not be possible due to different task (*e.g.* the camera might be needed by another subsytem) or sensor properties (*e.g.* different frame rates or view frustra). Hence, partial observability in terms of available information for a given object and therefore partial object descriptions are an important problem the system has to be able to cope with.

Another problem that needs to be addressed in autonomous agents acting in human living environments is the problem of finding the right tool for the perception task at hand. Some objects are best perceived by their shape in 3D, some have a very distinct color or texture. Of course, the decision about what object a given percept most probably belongs to should be made based on the most distinct features of the object. For example, if a mustard bottle is the only yellow object in a scenario, the color classification should be given a higher weight than *e.g.* the height of the object, which might be too ambiguous. If the sensor data leading to the object description is not complete, *e.g.* lacking the camera image and thus the color information, and the higher level system components still need some statement, one needs to use other percepts and data representations in order to reach an object classification with enough confidence. However, in these cases temporal and spatial aspects can be exploited to great effect.

It is quite common for robotic systems to have sensors that take percepts of the world in a regular, repetitive, manner. Most camera systems are being clocked at 15 to 30 frames per second while a sweeping 3D laser scanner might be set to take scans at a frequency of 0.1 to 1 Hz. Depending on the dynamic aspects of the scene, there is a certain probability that nothing has changed since the time of the last percept. There is also a chance that an object stays at the same position over the course of taking several, maybe tens or hundreds of percepts from different sensors. This means that it should be possible for a perception system to combine and accumulate all these percepts into a better object description.

To this effect, we need to establish and remembers the temporal trajectory of world and object states in a way that allows powerful and efficient queries to be answered in a meaningful way.

6.1.1 System Overview

We want RoboSherlock to be able to know where things *are*, where they *were*, and on top of that, we need rich object descriptions with information about what data we collected about them in the past by spatial and temporal association.

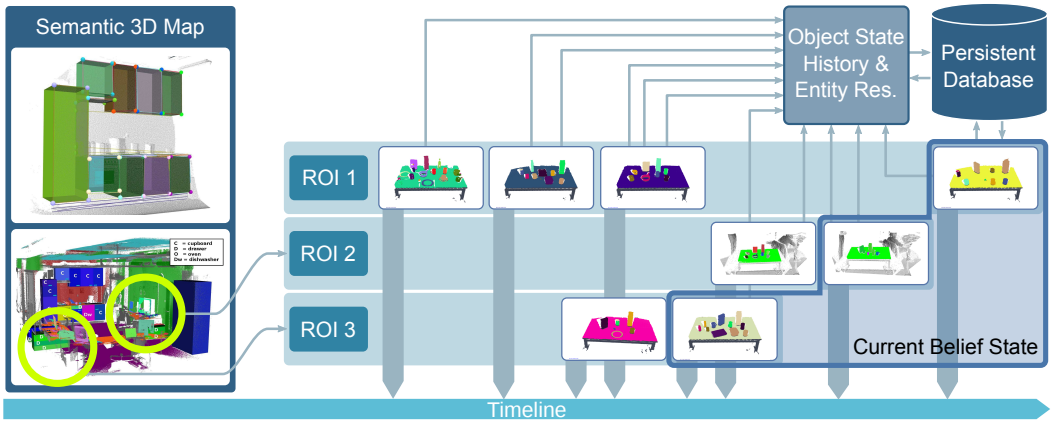


Figure 6.2.: *World state and history*

This knowledge is being stored in a database that holds information that is *object* centric as well as *world* centric. By *world* centric information, we mean information about object arrangements, distributions on support surfaces, or other object-location relationships. In a way, the object information potentially stays relevant over long periods of time, over multiple environments, whereas the world state information is rather episodic and usually of less relevance when *e.g.* moving to a different environment.

The central idea is depicted in Figure 6.2. The belief state about the static environment that we are operating in contains walls, tables, cupboards, furniture handles, doors and door handles, drawers and so on in our semantic map (Chapter 7). Any observations of objects detected in our regions of interest are segmented (Chapter 4) and processed by a set of reconstrucion Annotators as described in Chapter 5. For each ROI, a chain of scene objects is available, with the most recent cross-section of scenes forming the current belief state. Object history and accumulated reconstruction information is determined through our entity resolution method and stored in a persistent database (Section 6.6). The database also contains the “trajectory” of world belief states.

The Annotators described in this chapter are concerned with a very important perceptual subtask, namely object identity resolution. In our previous work [14], we identified two important predicates that need to be computed for every observed object hypothesis: the key predicate was the association between observations and the objects they were taken of. This allowed us to evaluate how likely it was that a given observation has been taken from a certain object.

We treated the problem of determinining which object observations are equivalent, *i.e*. whether they were taken from the same object, as a secondary one, even though it was to some degree independent of the knowledge about object identity. This way, we were able to track objects with high confidence even though identity was ambiguous. The task of comparing two (possibly partial) object description was being performed by the same reconstruction or refinement algorithms that provided the representation.

In Blodow et al. [14], we embedded the entity resolution in our passive perception pipeline, which can be regarded as a predecessor to RoboSherlock, however much smaller in scale and ambition.

Similar to the tracking Annotators mentioned earlier, we attempt to reach a conclusion about which objects observations from the current scene were taken from the same object as other, earlier observations in the past.

The central problem we faced with this approach however was the reliance on full knowledge of the universe of objects, or at least object classes. We found this to be a limitation that prevented the application to a bootstrapping system scenario.

Additionally, the evidence that was available to the inference step were custom-tailored to certain kinds of per-cluster information, *e.g*. the three-dimensional extents of a cluster, and the shape.

For these reasons, we modified the Markov Logic Network (MLN) model and by extension the way we use it for inference to overcome some of these limitations.

These modifications include:

1. We changed the query predicates for the inference step to answer two questions: *i*) "which object observation corresponds to which previous object obser-

vation?" and *ii*) "which actions were performed on each (previous and current) object?"

2. We also replaced the previous evidence predicates concerning shape, extents and pose similarity between clusters and replaced them with two very general predicates: *i*) how similar do two objects observations look and *ii*) whether a previous and a current cluster are located at the same position. This shifts the burden of adapting the model for different available annotations from the inference module to the caller.

3. We discarded the time domain as an explicit domain of discrete symbols and treat time implicitly.

4. We also discarded the shape domain as the corresponding predicates are no longer part of the model.

In summary, the whole model was replaced by a very different model with similar motivations and core ideas. In the process, the model got simpler, and apart from the way we model similarity, requires less bookkeeping by the caller.

The RoboSherlock Annotator built around this MLN model is organized in the following way. As an input, we retrieve a Scene of object observations from the CAS, along with the various descriptions and reconstructions stored by previous object Annotators.

We proceed by transforming feature descriptions of objects into a *similarity measure* between pairs of objects that are independent of the underlying feature used and scaled to the interval $[0 \dots 1]$. This way, we propose to treat the resulting similarity measure as a probability of the objects being equal, based on some aspect of their appearance.

The second piece of evidence we compute is a measure of *proximity* between objects. This is also scaled to the interval $[0 \dots 1]$, and can be regarded as the probability that two object observations occupy the same location (at different times).

The inference result are a distribution over *equivalence* of object observations taken at different times, and a distribution over explanations concerning the *action* that was performed to each object. We will give concrete details in the following sections.

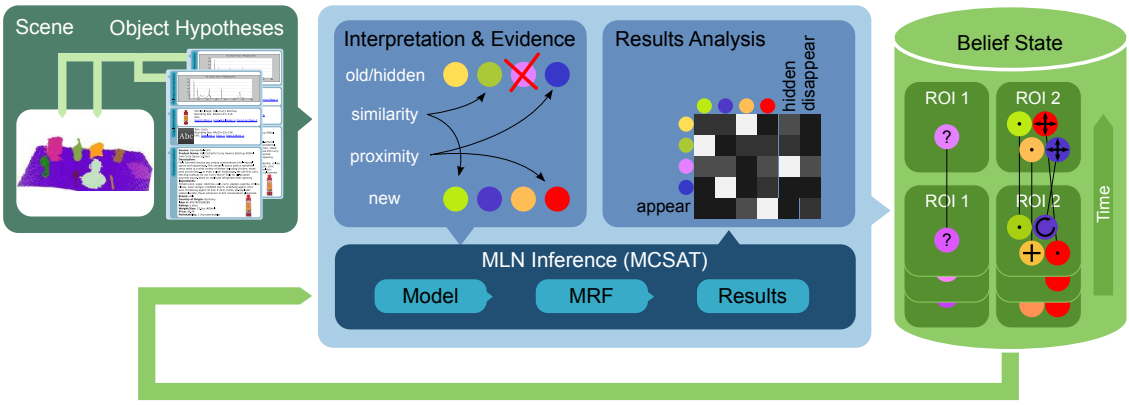


Figure 6.3.: *Overview of the proposed entity resolution architecture. A Scene containing annotated object hypotheses serves as input. Together with information from the previous belief state, it is interpreted and a data base of evidence predicates are established. This in turn is used in our Markov Logic Network inference step to produce several distributions: statements concerning tracking information (object association), as well as actions performed per object observation: appear for new objects, disappear or hidden for previous objects, and binary predicates connecting old and new measurements: move, new view and stay for objects that have not changed. The resulting interpretation is then stored as our temporally indexed belief state of objects and their history.*

The system overview is pictured in Figure 6.3. As can be seen, the input to our entity resolution is a Scene with a list of object hypothesis descriptions (as well as the currently valid belief state). Internally, we compute the respective evidence predicates, which serve as input to the MLN inference step. The results are interpreted and stored in a belief state annotation that contains the equivalence relation (in a tracking sense) as well as the distributions over action explanations.

6.1.2 From Object Annotations to Similarities

For each incoming object hypothesis in the CAS, we have access to a variety of different features that estimate and describe geometric, semantic and appearance-based properties of the object.

The Annotators extracting these cluster features also need to provide a method that lets the entity resolution component compare two features of the same type. This

is necessary in order to determine a belief about the measure of similarity that the specific feature suggests. In our case, due to the concept of FeatureStructureProxies (*cf*. Section3.1), it is trivial to incorporate a similarity function directly in the annotation's proxy wrapper. This way, there is a central code location where the feature and its distance function are defined.

It is of course in general not sufficient to rely on just one feature, because *e.g*. a purely geometry-based object description like a shape classification fails to incorporate color or texture properties and might therefore misjudge the actual similarity. For the scope of this thesis, we incorporated a relatively elementary approach based on combining multiple feature dimensions using the geometric mean. This is a non-trivial problem that deserves a more thorough investigation in the future.

The similarity computations are performed within the `SimilarityAnnotator`, which computes these similarity measures for the cartesian product of the current and last scenes' clusters. Additional metrics in the future can for example be integrated there or, due to the modular nature of RoboSherlock, in a separate "sibling" Annotator.

Let a Scene $\mathcal{S}$ be a set of object observations: $\mathcal{S} = \{o_i | i \in [0 \dots n]\}$. Each object observation o_i in turn is a tuple of a set of object hypothesis descriptions (*e.g*. image regions, point cloud clusters) and a set of Annotations: $o_i = \langle \mathcal{H}, \mathcal{A} \rangle$, where $\mathcal{H} = \{h_i\}$ and $\mathcal{A} = \{a_i\}$.

Every $a_i \in \mathcal{A}$ has a certain datatype $T(A)$. For Annotations a_j, a_k of same type $T = T(a_j) = T(a_k)$, we define a similarity function $sim_T : T \times T \rightarrow [0..1]$. The result of such a function *sim* represents the degree to which the two given clusters appear similar, given the respective data representation.

Since the different Annotation types can encode arbitrary information about the cluster, the similarity function needs to act as a bridge between raw or processed sensor data, which usually contain geometrically, visually or semantically meaningful values, and the probabilistic degree of belief to which we consider two clusters to appear similar. For many Annotations, one can define an actual distance metric, *e.g*. color hue distance, cylinder length difference, *etc*. For all types T where such a distance metric d_T is possible, we offer two convenience functions g and l that are designed to transform the codomain of d_T to be in the range $[0..1]$:

$$g(x) = e^{-\left(\frac{x}{2\sigma}\right)^2}, \tag{6.1}$$

$$l(x) = 2/(2-(1+e^{-x})), \tag{6.2}$$

where g is a zero-mean gaussian function normalized to have a maximum of 1 at $x = \mu = 0$, and l is a function that decays approximately linearly at first and approaches 0 asymptotically. Annotation similarity functions have access to these functions in their implementation.

We understand $g(d_T(a_j, a_k))$ and $l(d_T(a_j, a_k))$ to be a "percentage" of how close the feature distance is to the optimum at $d_T(a_j, a_k) = 0$. The variance σ^2 needs to be estimated for each Annotation type.

For some, this variance can be easily justified, *e.g.* the variance of the differences in position of two cluster observations taken from the same objects, but from different points of view, is expected to be in the same order of magnitude as the localization covariance of the robot itself and the sensor noise. This is to say that any error in localization directly affects a cluster position. However, for other feature spaces, *e.g.* color hue, this variance is not strictly meaningful, and must be adjusted as a user parameter or learned in order to account for the special environment properties, such as lighting conditions, or desired color sensitivity.

6.2 Object Identity Resolution

Object identity resolution constitutes a link between the higher level components such as the knowledge base and the task executive, both of which need a symbolic and unambiguous description of the environment, and the lower level perception routines outlined in the previous sections. The approach we propose in chapter is able to probabilistically model object trajectories while coping with hard conditions such as partial observability of the world, objects of similar appearance, and noisy and partial object descriptions.

A small example problem is given in Figure 6.4, where we schematically depict a number of measurements over time taken from one table. There are instances of

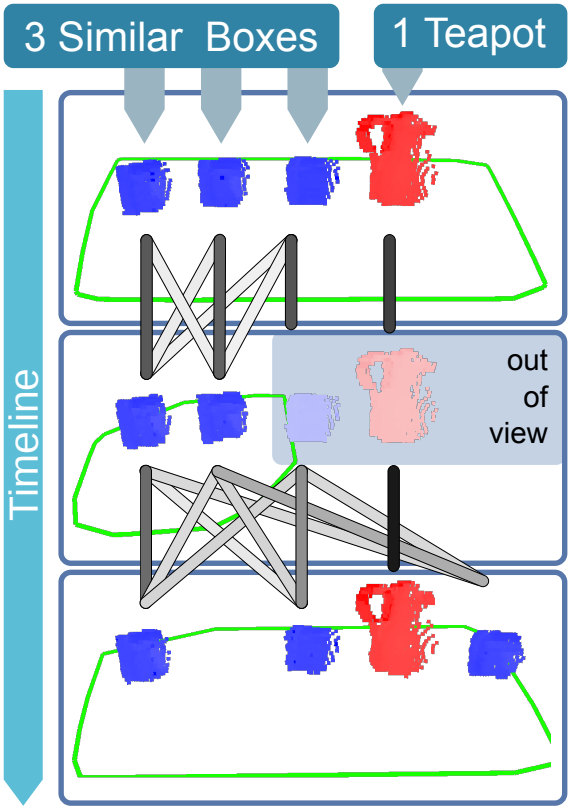


Figure 6.4.: *Belief distributions of cluster associations. Darkness of the connecting lines is proportional to the degree to which we believe two cluster observations belonging to the same object. It can be seen that the method considers the square in question to most likely be the one that disappeared from its position in timestep* t_{k-1}.

two different object categories (boxes containing teabags and a teapot). These may get replaced, moved, taken away or added by an agent not directly perceived by our system. The questions arising in this scenario we would like to answer include: At any point in time, which observation refers to the same object as another one, which objects have been moved where, and which objects have remained at the same position? Also, which objects observed in the past have been removed, which new objects have appeared, and which objects can not be seen in the current view, but are still likely to be at their last location?

At the second depicted time step, a human has stepped in front of the table and the robot is therefore unable to observe the right hand side of the table. In order to be able to answer queries about objects that are temporarily out of view (shown faded

in Figure 6.4, since these were not observed, but carried over by the inference logic), we need to maintain a sensible belief state which assumes that objects have not been moved unless we perceive evidence to the contrary (*e.g.* the object has appeared elsewhere or the location is observed empty). This obviously gets more difficult if there are multiple objects with same or similar appearance.

Figure 6.4) also shows the result of the association reasoning, overlayed as grayscale lines on top of the segmented scans (black corresponds to 100% probability, white to 0%). Note that the mode of the belief distributions of cluster associations conforms to the explanation offered by common sense, which would assume that the middle tea box was moved over to the right.

We apply state-of-the-art statistical relational learning methods in order to tackle this problem. In statistical relational models [49], we can capture the interactions between objects that were observed, their attributes and relations – and represent these interactions generally, abstracting away from concrete entities. Most importantly, statistical relational models are capable of considering all the observations and facts that are relevant simultaneously, achieving a posterior belief on the associations between objects and observations that is guaranteed to be *globally consistent*.

6.2.1 A Markov Logic Network for Object Identity Resolution

In particular, we use Markov logic networks (MLNs) [121] as our representation formalism, for they combine the full power of first-order logic with the probabilistic semantics of graphical models. Formally, an MLN L is given by a set of pairs $\langle F_i, w_i \rangle$, where F_i is a formula in first-order logic and w_i is a real number, the weight of formula F_i. For each finite domain of discourse D (set of constants), an MLN L defines a *ground Markov random field* $M_{L,D}$, whose set of variables X we obtain by grounding all the predicates in L with the entities in D, and whose set of weighted features is given by groundings of L's formulas. $M_{L,D}$ specifies a probability distribution over the set of possible worlds, *i.e.* the set of possible assignments of truth values to each of the ground atoms in X, as follows,

$$P_w(X = x) = \frac{1}{Z} e^{\sum_i w_i n_i(x)} \tag{6.3}$$

where $n_i(x)$ denotes the number of true groundings of F_i in x and Z is a normalization constant.

In Blodow et al. [14], we presented an MLN model we used to solve the problem of object identity resolution. The model was based on a notion of *objects* and *clusters*, and modeled most predicates directly or indirectly involving a predicate *is*. This predicate encodes the assignments of clusters to objects, and requires a database of objects, or put differently, an a priori known universe of objects.

For the scenario presented in this work, we remodeled the MLN in order to eliminate this limitation and be able to tackle the problem of object identity resolution even without a known universe of objects. This means that predicates that were modeled around *is* needed to be changed to rely on *similarity measures* only in order to estimate a probability of two clusters being the same object.

Table 6.1 shows the concrete MLN model we use to solve the problem of object identity resolution. As entity types, the model considers object hypotheses pertaining to objects placed in a scene, as well as abstract entities representing *explanations* for pairs of object observations taken at different points in time. As such, we implicitly model the time domain in a heavily discretized way, as we differentiate only between the most recent point in time (at wich all entities $n \in$ *newCluster* were observed) and points in time that preceeded it (these past observations are all entities $o \in$ *oldCluster*).

As a scene containing new object observations arrives in our Annotator, we interpret the scene to populate the domain *newCluster* and create the evidence predicates (marked "e" in Table 6.1): *similar*, *proximity* and *outOfView*. We then apply the model to update our beliefs which we extract by analyzing the query predicates (marked "q"): *is*, *appear*, *disappear* and *hidden*.

Markov logic networks are particularly well-suited for the representation of such a model, as they allow us to specify the various hard constraints that any valid entity-observation association must satisfy in first-order logic. At the same time, probabilistic rules (which either increase or decrease the likelihood of associations) can be expressed as soft constraints using the weights w_i for each formula F_i. Any beliefs computed by the model are guaranteed to be probabilistically sound and globally consistent with respect to the constraints that were specified.

e/q	*predicate declarations*	*domain declarations*
e	similar(oldCluster, newCluster)	oldCluster = {O1, O2,...}
e	proximity(oldCluster, newCluster)	newCluster = {N1, N2,...}
e	outOfView(oldCluster)	
q	is(oldCluster, newCluster, explanations!)	explanations = {*move*, *stay*, *newview*, *not* }
q	disappear(oldCluster)	
q	appear(newCluster)	
q	hidden(oldCluster)	

i	F_i
1	(is(a, c, e1) ∧ is(b, c, e2) ∧ ¬ (e1=*not*) ∧ ¬ (e2=*not*)) → a=b.
2	(is(c, a, e1) ∧ is(c, b, e2) ∧ ¬ (e1=*not*) ∧ ¬ (e2=*not*)) → a=b.
3	is(o, n, *stay*) ↔ (similar(o, n) ∧ proximity(o, n)).
4	is(o, n, *move*) ↔ (similar(o, n) ∧ ¬ proximity(o, n)).
5	is(o, n, *newview*) ↔ (¬ similar(o, n) ∧ proximity(o, n)).
6	outOfView(o) → ¬ (∃ n (is(o, n, *stay*) ∨ is(o, n, *newview*))) ∧ (is(o, n, *not*) ∨ is(o, n, *move*)).
7	appear(n) ↔ ¬ (∃ o (is(o, n, *stay*) ∨ is(o, n, *move*) ∨ is(o, n, *newview*))).
8	disappear(o) ↔ ¬ (∃ n (is(o, n, *stay*) ∨ is(o, n, *move*) ∨ is(o, n, *newview*))) ∧ ¬ outOfView(o).
9	hidden(o) ↔ ¬ (∃ n (is(o, n, *stay*) ∨ is(o, n, *move*) ∨ is(o, n, *newview*))) ∧ outOfView(o).

Table 6.1.: *Markov logic network for object identity resolution. Free variables in formulas are implicitly assumed to be universally quantified. Formulas 1 and 2 are hard formulas and essentially have an infinitely large weight, which, in practice, is substituted by a sufficiently large real number. The domain* explanations *is fixed across all instantiations of the model, which is why we declare it explicitly. The domains* oldCluster *and* newCluster *change for every instantiation, depending on the number of objects observed in a scene. The predicate declarations indicate the domains to which the predicates are applicable. The predicate declaration for* is(oldCluster,newCluster,explanations!) *contains an argument suffixed by an exclamation mark, which means this predicate is declared as functional,* i.e. *for each pair of old and new clusters, there must be exactly one explanation for which the predicate is to hold in any possible world.*

In the following paragraphs, we will iterate over the various domains and predicates, detail their significance and application, and explain the various hard and soft rules of our model.

newCluster

The domain *newCluster* contains one entity for each object hypothesis in the scene. This is simply an atomic label, in our case we use the notation N1, N2 *etc*.

oldCluster

Similarly, *oldCluster* contains the atomic labels for each distinct previously perceived object. It contains *i*) all objects that were element of *newCluster* in the previous iteration; and *ii*) all objects that were inferred to be still present, but hidden from view in the previous iteration (see *hidden* below in this list). Together, these objects constitute the previous belief state of objects in the environment.

similar

Similar is an evidence predicate, defined over pairs of entities from *newCluster* and *oldCluster*. As has been described above, it expresses the similarity in appearance of two object observations.

proximity

Proximity is the second binary evidence predicate, also defined over pairs of entities from *newCluster* and *oldCluster*. It specifies the probability that the referenced clusters occupied the same location during their respective acquisition times.

outOfView

The third and final evidence predicate *outOfView* is defined over *oldCluster* and encodes whether the area that was occupied by a certain cluster in the past has been inside the field of view for the currently observed scene. For each object hypothesis, the centroid or point cluster is reprojected into the camera frustrum to this effect.

In the model, formula 6 specifies the implication by this predicate: if a cluster *o* is out of view, it cannot possibly be equivalent to a new object observation with

an explanation of *stay* or *newview*. The only explanation for each n is either that o is *not* the same object as n or o was *moved* to a new location, where it was perceived as n.

This allows us to cope with partial observability: The model is able to handle the fact that areas in which we previously made observations can currently be unobserved and maintain an appropriate belief about these areas. By default, we assume that the observations in unobserved areas are likely to carry over into the present (and are in fact *hidden*, see below), unless there is evidence to the contrary (which is the case if another cluster n whose area is currently observed refers to the same object as o). This would lead to a stronger belief in *is(o,n,move)*, which in turn contradicts *hidden*. With respect to the consistency of cluster-object associations, we treat persisting clusters as if they were currently observed.

is For object identity resolution in a tracking sense, the key information we strive to estimate is the association between object observations that the robot perceives currently and objects that it has perceived earlier. Additionally, we are interested in the distribution over explanations concerning actions performed on them. This is modeled by the predicate *is*, which is a functional predicate, since in our model, exactly one of the explanations must be valid for a given pair of old and new observations o, n.

Formally, *is*, with the exception of the explanation *not*, defines an injective relation over *oldCluster* and *newCluster*, which is stated in formula 1, and is^{-1} is also injective, which is stated in formula 2. In essence, a single object can not split into two, nor two objects merge into one.

The different explanations for a pair $\langle o, n \rangle$ are directly influenced by the evidence predicates through formula 3 through 5 and can take these values:

stay expresses the static case and signifies that o and n refer to the same object, which has not moved between the two observations (formula 3).

move
means that o and n refer to the same object which has been moved by external forces between the two observations, but the overall appearance

has not changed significantly. formula 4 gives the formal definition for this rule.

newview
: is an explanation for the case that *o* and *n* were taken from the same object, at the same location, but the object has undergone a change in appearance, as made explicit in formula 5. This is likely due to the fact that the two observations were taken from different vantage points (*e.g.* front and back sides), or that *o* and *n* provide different sets of object annotations. In both cases, this explanation expresses the belief that the system should merge the two partial object descriptions into a richer and more complete representation.

not
: is the explanation opposing the previous three and expresses the belief that *o* and *n* refer in fact to different objects.

appear
: The predicate *appear* is defined over the domain of new clusters, and as such models the belief that the corresponding object observation *n* has no equivalent object in previous percepts, which means the robot perceives it for the first time. This can be due to two reasons: the location of the object referred to by *n* was previously out of view or it has been placed there between now and the last time we perceived that area.

  It is obvious that for a cluster *n* to *appear*, it cannot be associated with any old object observations. This is stated in formula 7.

disappear, *hidden*
: The equivalent explanations for old clusters *o* that have no equivalent cluster *n* are *disappear* and *hidden*. They are almost identical except for the fact that clusters for which *outOfView* does not hold are considered to have disappeared (and as such do not need to be tracked anymore), and those for which *outOfView* is true are treated as being *hidden*. This is modeled in formulas 8 and 9.

  Hidden clusters will have to be remembered for redetection, and to populate *oldCluster* for the next iteration. As described above, the former can happen if

the area is perceived again or if the hidden object is moved into the field of view of the robot. In both cases, the MLN inference step will assign the appropriate new cluster.

6.3 Experiments and Discussion

In order to assess the validity, performance and robustness of our approach, we performed a series of experiments. To illuminate various aspects with statistical significance, we implemented a synthetic test case that simulated a number of tables where actions were performed to put, remove, and move objects around, as well as change their appearance to simulate different partial views of an object. A virtual sensor with limited view frustrum was used to generate noisy measurements of a subset of the objects, which was used to infer the actions that were performed as well as the the belief state of all objects in the environment.

We used 4 tables of 1 square meter area each and a sensor that is placed at a randomly changing position in front of any one table that can see up to 1 meter in width. We used a set of 15 objects of random color.

For each object in the sensor's view, we generated an observation by adding gaussian noise with a standard deviation of $\sigma_1 \in \{0.025, 0.05, 0.075, 0.1\}$ per color channel to the color vector, and we used the same principle (and the same additive gaussian noise) in two dimensions to simulate a noisy measurement of the true position.

Let the true position on the table of an object s be $\vec{p}_0(s)$ and its color $\vec{c}_0(s)$, and further let their measurements be denoted $\vec{p}(s) = \vec{p}_0(s) + \vec{\epsilon}^2_{\sigma_1}$ and $\vec{c}(s) = \vec{c}_0(s) + \vec{\epsilon}^3_{\sigma_1}$, respectively, where $\vec{\epsilon}^n_\sigma$ is a shorthand for a n-dimensional vector of random numbers sampled independently from a zero mean gaussian distribution with standard deviation σ.

The computation of *similar* and *proximity* is then performed per pair of old and new measurements by computing the euclidean distances between them and applying g (*cf*. Equation 6.1) with a standard deviation of $\sigma = 0.25$:

$$similar(o,n) = g(\vec{c}(o) - \vec{c}(n)) \tag{6.4}$$

$$proximity(o,n) = g(\vec{p}(o) - \vec{p}(n)). \tag{6.5}$$

This leads to errors e_p, e_c for the proximity and similarity predicates that can be seen in Table 6.2. It can be seen that the errors are very high in the case where $o = n$, and much lower for the opposite case. This is due to the fact that the two different random points in a multi-dimensional vector space have a relatively high distance on average, and as such lie in the tails of $g(x)$, where the influence of noise is much lower. Additionally, note that for objects that are placed on different tables, *proximity* is set to 0, which further reduces this error.

σ_1	$\mu(e_p) \pm \sigma(e_p)$	$\mu(e_c) \pm \sigma(e_c)$
	$o = n$	
0.025	0.175 ± 0.321	0.205 ± 0.338
0.05	0.250 ± 0.301	0.324 ± 0.311
0.075	0.370 ± 0.304	0.468 ± 0.296
0.1	0.470 ± 0.311	0.598 ± 0.285
	$o \neq n$	
0.025	0.142 ± 0.238	0.055 ± 0.141
0.05	0.139 ± 0.235	0.053 ± 0.136
0.075	0.138 ± 0.233	0.052 ± 0.136
0.1	0.131 ± 0.231	0.048 ± 0.131

Table 6.2.: *The errors as deviation from the true values of the predicates* similar *($\mu(e_c) \pm \sigma(e_c)$) and* proximity *($\mu(e_p) \pm \sigma(e_c)$) depending on σ_1. This is computed separately for the cases that the observations o and n refer to the same object (top half) and the opposite case (bottom half). It can be seen that we consider relatively high amounts of noise, approaching* e.g. *almost* 50% *for the* proximity *of observations from the same object at $\sigma_1 = 0.04$, and up to* 60% *for* similar.

For each noise level, we performed 1 to 5 actions between observations, with an upper limit of 3 to 10 objects in the scene. For each combination of these parameters, we performed 20 experiment steps, leading to a total of 800 scenes per noise level,

action	frequency
appear	3189
move	1752
newview	1393
disappear	3013
total	9347

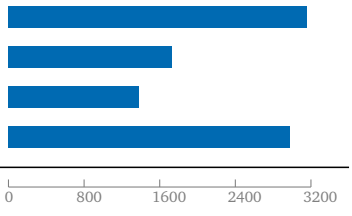


Table 6.3.: *Frequencies of actions selected during the experiments.*

3200 in total. The frequencies of actions are given in Table 6.3, where you can see that we performed a total of 9347 actions over the duration of the experiment.

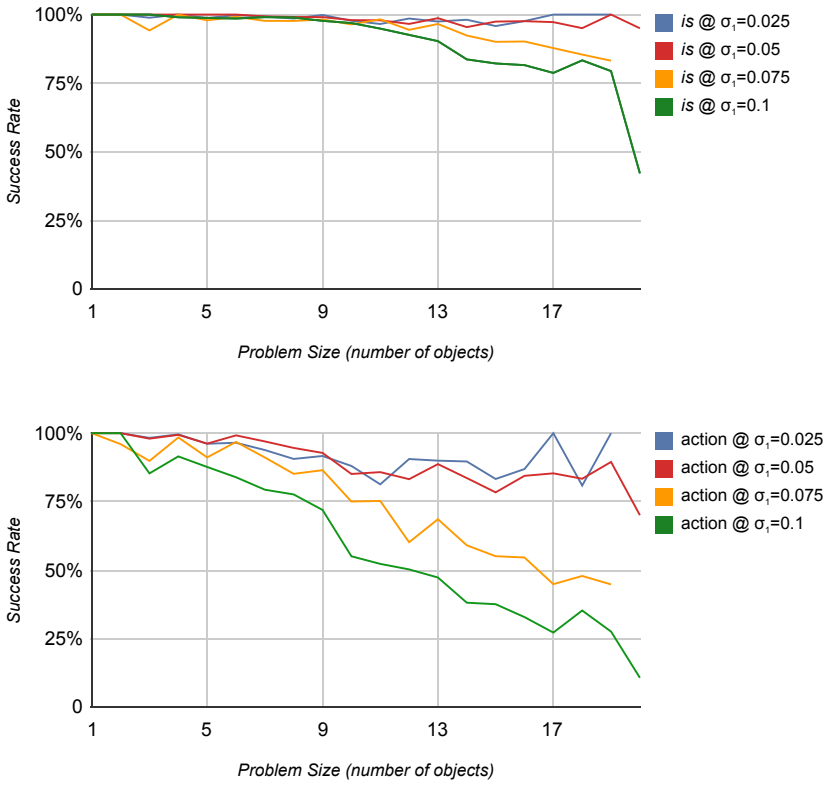


Figure 6.5.: *Success rates of inference results over problem size (number of objects involved in inference), shown for increasing levels of noise (σ_1). (Top) correct cluster association (mode of* is*), (bottom) correct action explanation (*stay, move, newview, appear, hidden, disappear*).*

We also evaluated the success rate of the object identity resolution method depending on noise level and the problem size (number of objects involved) in the inference step. This is shown in Figure 6.5, where you can see that the cluster association (sub-

figure a) has a very high success rate throughout the entire domain of problem sizes, up until the highest level of noise we evaluated. It is clear that as the number of objects increase, the chance of mistaking objects for one another rises. The correct explanation for what *happened* to a certain object (subfigure b) is more susceptible to noise, and fails faster for larger problem sizes. This is due to the fact that the various explanations are directly modeled after their respective combination of evidence predicate configurations.

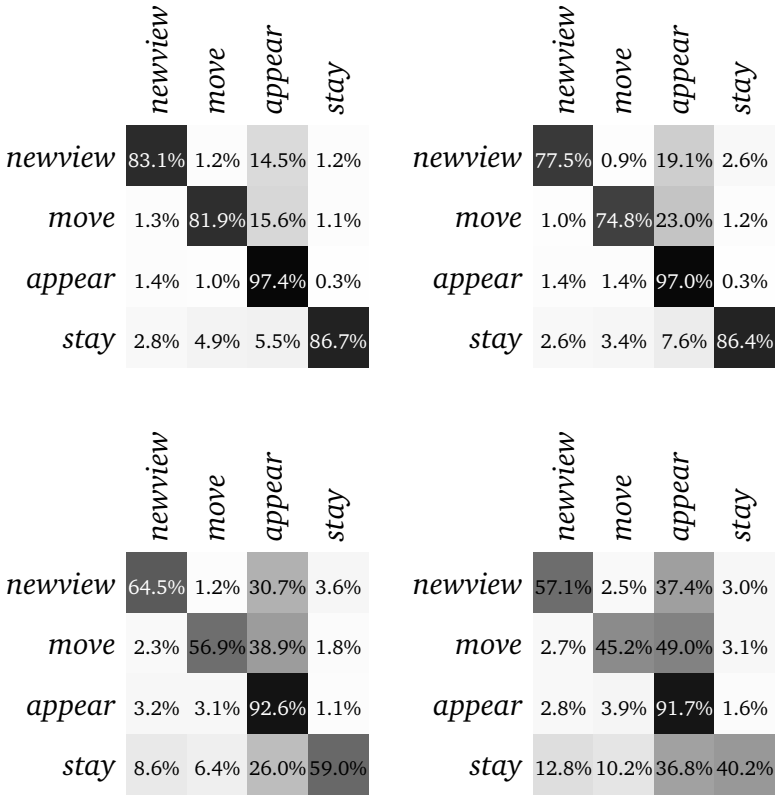


Figure 6.6.: *Confusion matrix for different action explanations for new clusters, combining* is *and* appear *results.*

The confusion matrix over action explanations for new clusters are shown in Figure 6.6, where we combined the two relevant predicates for new clusters, *is* and *appear*. Displayed are multiple noise levels to show the effect on the quality of the inference results: $\sigma_1 = \{0.1, 0.2, 0.3, 0.4\}$ (top left to bottom right). It can be seen that with increasing noise, object observations from the same object are increasingly considered different from each other, and are therefore explained as new objects "appearing". Nonetheless, the modes of these distributions are along the diagonals,

very distinctly at lower noise levels and less so, but still noticable, for higher noise levels.

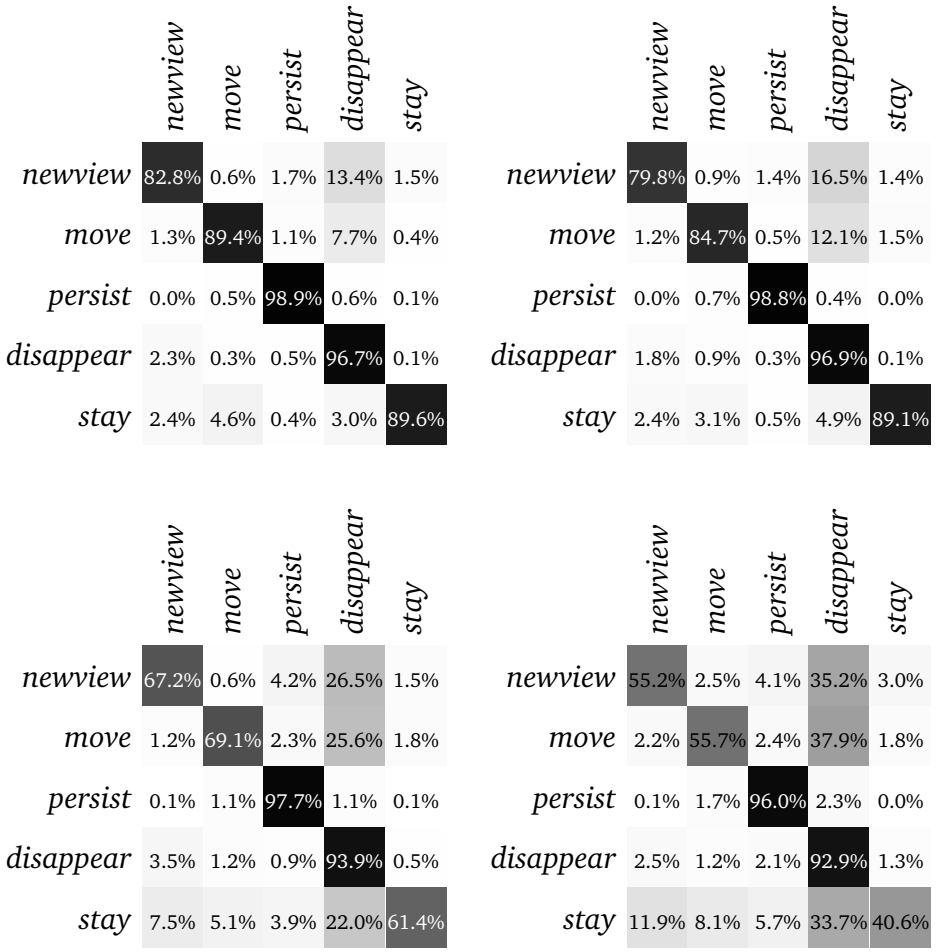


Figure 6.7.: *Confusion matrix for different action explanations for old clusters, combining* is*,* hidden *and* disappear *results.*

The same evaluation for *old* clusters is shown in Figure 6.7, where we display the confusion matrix for the predicates *is*, *hidden* and *disappear*. Again, we separated the results by noise level to show its effect on the quality of the inference results: $\sigma_1 = \{0.1, 0.2, 0.3, 0.4\}$ (top left to bottom right). Analogously to Figure 6.6, increasing noise leads to favoring disappearance as an explanation, since similarities decrease. However for reasonable amounts of noise, the method is very robust in determining the correct unbserved actions.

Surprisingly, our method is not that sensitive to the number of actions that were performed before observing the scene and inferring what happened. In Figure 6.8,

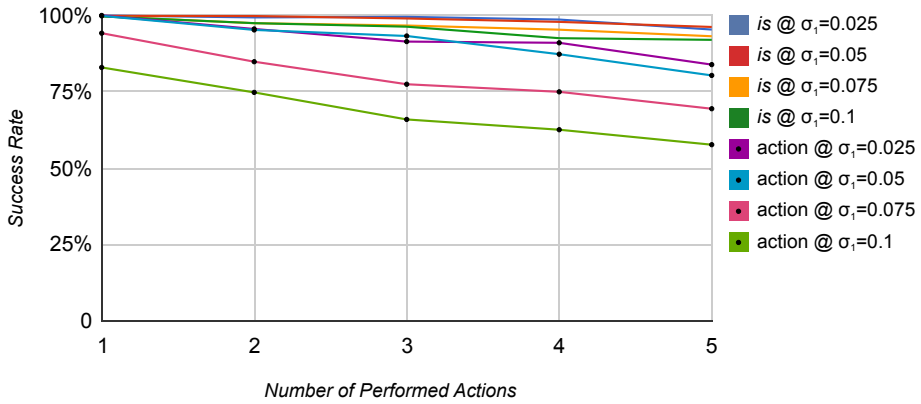
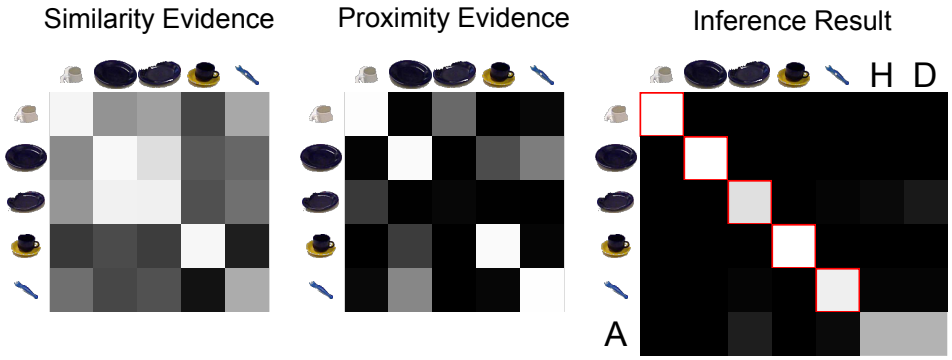


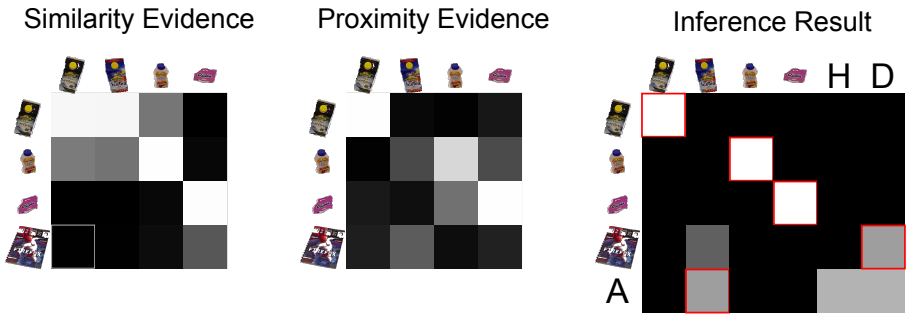
Figure 6.8.: *Success rates of inference results of cluster association and action explanation as a function of the number of actions performed. Again, errors are plotted separately for different noise levels.*

we show the same success rates, graphed over the number of actions, and it can be seen that the cluster association problem is not much affected by performing 5 actions between observations. For moderate noise levels, even the action explanation can mostly deal with this issue. At high noise levels, performance begins to drop to the point where only 3 or 4 out of 5 performed actions were inferred correctly.

Finally, we show the inference results for two different example scenes from our real-world experiments (*cf*. Chapter 8) in Figure 6.9. We can observe various effects of the object identity resolution for a single time step each: Shown in each sub figure is a visualization of *similar* and *proximity* input predicates (left, center) and the inference results showing the mode (red) of the distribution over *is* as well as the predicates *appear* (A), *hidden* (H) *disappear* (D). In the first scene, there are two identical, textureless plates that yield very high similarity values, but can be distinguished due to pose information. In fact, the second plate has been moved, but was assigned to the correct old observation. The second scene contains more dynamism, since a magazine (bottom left image) has disappeared and a new ice tea TetraPak has been added to the scene. The most likely explanation for the magazine is the correct one, which is *disappear*. The ice tea, while visually very similar to the second ice tea present, is correctly identified as having *appeared*, and the *is* explanation is favored for the remaining objects.



(a) There are two identical blue plates, one of which has been moved. The association is still inferred correctly.



(b) The magazine disappeared, the ice tea bottle appeared, the remaining object stayed.

Figure 6.9.: *Exemplary results showing the distributions of input and output predicates for two scenes. New clusters are shown on the top, old clusters along the left side.*

6.4 Conclusion

In this Chapter, we have shown our approach to the problem of entity resolution, *i.e.* the association of current to past observations to enable spatial and temporal associations leading to accumulated models of objects. 3D, color, texture and other information can be collected into a consistent description, even if these pieces of information were reconstructed from different observation instances or to a degree even from different sensing modalities.

While some of these goals still require more work, such as loosening the clustering restrictions in the pre-processing to allow re-identification of objects through learned SIFT features in more difficult contexts or clutter, we have successfully demonstrated that a system that can perceive only occasional snapshots of a scene, consisting of

objects that are not stored in a database, can be able to probabilistically reason over the data association problem, and even infer actions that have been performed on individual items.

The resulting belief state, integrating the inference results over time, contains information pertaining to world state history and a rich set of object descriptions along with their history that can be used for a variety of purposes. These include relatively simple tasks like finding a certain object, which can be answerwed with a probability distribution over places where the objects has been perceived before, or providing a history of where a given object has been in the past.

On the other hand, this can also be used to provide training information for other systems, such as learning probabilistic models of object involvement in different situations, such as learning that a certain food container, *e.g*. holding cereal, is usually seen in the context of breakfast. These models can then in turn be used to evaluate the current world states such as to classify the situation or to infer the missing objects [107].

The usage of MLN inference guarantees the results to be probabilistically sound and globally consistent with respect to the constraints specified in the model. This can be seen in the experiments, where the degree of belief about an object being added to the scene is influenced by all other objects and their interactions in terms of similarity and which action explanations emerge as probable for them as well. This makes it possible to deal with the hard issue of indistinguishable objects, as long as not all of them have been moved and manipulated.

The proposed approach has been in use for several years in our robot programs and here experimentally validated in synthetical experiments comprising 3200 scene arrangements, with over 9300 actions performed on them, spanning the dimensions of noise level and problem size (both in numbers of actions and number of objects involved). Action explanations were influenced by very high amounts of noise, such that when 5 actions were performed between two observations, only 3 or 4 could be recovered correctly. Apart from that, the association and action explanation distributions demonstrated robust and sometimes impressive behaviour, almost giving the impression of common sense.

6.5 Tracking and Entity Resolution

The entity resolution system proposed in the previous section uses hard and probabilistic rules about the problem domain to infer probability distributions over *cluster observation identities* over time as well as the *actions performed on objects* which models the degree of belief the system maintains over the state, trajectory and history of objects.

While this proves very powerful in dealing even with harder cases, such as partial visibility (since the robot cannot perceive the whole environment at any given time) or multiple indifferentiable objects (such as the set of equal plates), computation speeds for a fully populated scene did not allow real time application.

We therefore propose to preprocess the easier cases of the entity resolution problem using less powerful, greedy but therefore much faster methods. In simple terms, a cluster that looks similar and is at the same position as one from half a second ago should be treated directly and not exponentiate the MLN inference problem.

Currently, the following, independent tracking systems are in place: In its simplest form, a *i*) position based tracker attempts to relate current observations of objects to past observations, ignoring appearance. *ii*) 3D appearance is used in a shape based tracker in the form of PCL features, and *iii*) image appearance in a tracker based on color histograms. These components assign tracker-local IDs to clusters and share a common Annotation type, `TrackingAnnotation`, which expresses the tracker's belief about the cluster trajectory and identity. A separate tracking arbiter uses these annotations in a voting scheme to attempt to find an explanation consistent to all the evidence.

These can deal with most frame-to-frame changes of static and moderately dynamic scenes. However, there are situations in which these tracking methods are too simple in nature and thus contradict each other. In these cases, we propose to employ the more challenging inference step detailed in Section 6.2 as an arbiter. As demonstrated, this method can make use of arbitrary similarity metrics computed between current and previously observed clusters to compute a probability distribution of which current clusters are referring to the same object as which previous cluster, while being able to deal with hard cases such as indiscernible objects or partial visi-

bility of the scene. It also maintains a belief of which previous observations of objects could still be true, even if they are currently out of sight, and reuses these persisting, hidden objects in the next inference step. To ensure the continuity of maintaining possibly hidden clusters, the MLN entity resolution arbiter is executed for every scene, but filters out all object observations in cases where the simpler trackers agree.

This way, the tracking arbiter simply votes and if all trackers agree, decides the association problem. The entity resolution Annotator is consequently executed and infers the Association and action explanations (appear, disappear, move) for the remaining, unexplained & hidden objects.

6.6 Information fusion, storage and reuse

An important aspect in a system such as RoboSherlock, where various different experts create, analyze, share and reuse information to structure and enrich perceptual data, is a memory mechanism capable of aggregating and filtering the information contained in these annotations. This leads to, among other things, better decisions about object identity, pose and class and can also provide consensus based confidence estimates. The benefit of combining different cues for the object perception task was already shown by Marton et al. [89], and the system presented there is integrated into the architecture described here.

Memory as defined in neurosciences and cognitive systems is a very complex issue and to this day is being argued over by different schools of thought. Many efforts have been made at *understanding* and *recreating* human memory behaviour.

On a coarse level, memory can be categorized into two types [137, pp. 225-229], *declarative* (explicit) memory, and *procedural* (implicit) memory. While the latter deals with subconscious memories, *e.g.* knowing how to ride a bike or tie shoelaces, the former can be used to consciously recall facts or personal experiences. This is reflected in the main subdivisions of declarative memory, namely *semantic* memory for facts or encyclopædic knowledge and *episodic* memory for consciously recallable past experiences and impressions, a distinction introducted by Tulving [166].

Of special importance in the field of autonomous robotics is *episodic* memory as it is the mechanism by which humans can relive their experiences and build a model of the world, learn from new experiences [5] and make predictions about the future [54]. For example, in our scenario, the robot must remember which objects exist in the environment, how they look, where they usually stand, when and where they were seen the last time, which objects were in use together, who used them and so forth.

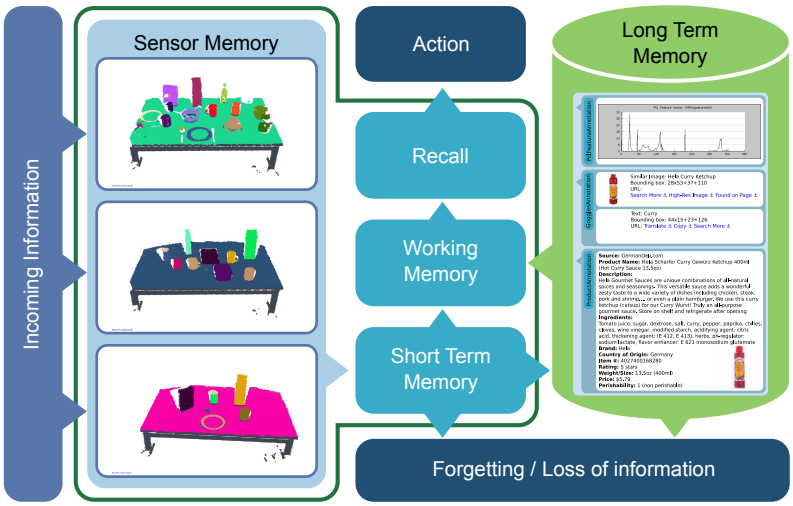


Figure 6.10.: *Model of information flow in short and long term memories (adapted from Schmidt et al. [137, p. 228]). Sensory information is fed into the sensory memory, from which it flows into short-term memory. Consolidation leads to storage in the long-term memory, from which it can be brought back to working memory, and can be used after recall. Forgetting happens both from short- and long-term memories. Note that in* RoboSherlock*, the roles of the modules enclosed by the green line are being provided by the CAS, and the database acts as a long-term memory.*

Furthermore, another commonly cited differentiation of memory is into *sensory* (SM), *short-term* (STM) and *long-term memory* (LTM, *cf*. Figure 6.10). Percepts arrive in sensory memory for a very short period of time before they get discarded or move on to short-term memory. STM has a much lower capacity than SM and serves as temporary storage (for few seconds or maximally minutes) for verbally encoded material, and through repetition information can flow into long-term memory, where it can be stored permanently. Information that needs to be kept “ready” for more than just a

few seconds can be stored and manipulated in *working memory*, which is sometimes considered a part of STM.

While research on understanding human memory, especially with the goal of creating intelligent machines, is highly fascinating and widely discussed, we cannot give a full overview in this thesis. For more information on this topic, we refer to Chapter 9 and to the intriguing if sometimes controversial discussions in the books by *e.g.* Kurzweil [71] and Hawkins and Blakeslee [54].

In Figure 6.10, these kinds of memory are shown in their interaction and in their context within RoboSherlock: incoming information travels through SM and STM to LTM. Working Memory is fed from either STM or LTM, and is used for performing an action (after recall). Information which is either not consolidated into LTM or not repeated or recalled over a period of time can be forgotten. The mapping of these memories to our system is also shown in the figure. The Common Analysys Structure (CAS) in RoboSherlock is depicted by the dark green outline, and comprises all memory that is directly accessible by annotators during artifact analysis, including all but long-term memories.

The long-term memory within RoboSherlock is implemented as a MongoDB data base, an object-oriented data base which follows the same type system as the rest of RoboSherlock. This is achieved by the fact that the main mechanism of filling the data base is a *CAS Consumer* that serializes subsets of the UIMA object graph contained in the CAS into an equivalent graph in MongoDB. Specifically, we store every `Scene`, its `Clusters` and their annotations and inserts it as a hierarchically structured observation. Each of these objects is automatically timestamped and tagged with a unique ID that can be used later on to reference existing, prior percepts. An example has been given in the Section 5.4, where the result of a Goggles query is attached to the MongoDB Cluster object upon arrival (which can potentially be relatively late), allowing processing the Cluster through further Annotators while the query is still waiting.

As was described in Section 3.2.1.3, the CAS Consumer makes use of the generic RoboSherlock $\longleftrightarrow$ MongoDB conversion interface that is based on the self-describing type information within the type system, making the concept generic and automatically future-proof.

The long-term memory as modeled serves multiple purposes:

1. storage of percepts as a staging area for later evaluation, including problem formulations such as modeling object life cycles or common storage locations, tasks which both require statistical evaluations of object-location co-occurrence over a large number of examples. This represents the main function of a memory, *i.e*. *remembering* past experiences and processing them into a denser, more optimized or generalized representation;

2. a knowledge source for feedback into the next iteration of scene processing, providing *e.g*. priors for tracking-based methods or models for object re-detection. This constitutes the basic underpinning for the *prediction* of future events based on a recollection of the past; and

3. pragmatically a *developer tool*, a backend for a convenient web page which allows for online supervision of the processing engines, *visualizations* of intermediate results and individual annotations, *logging* functionalities, and maybe most importantly, a mechanism for querying the acquired data base in various ways.

The last point is a prime example why we think the decision to choose a regular (if object-oriented) data base makes sense. The presence of client bindings for many popular programming languages, convenient tools for analysis, management and maintenance, inherently distributed deployment (*e.g*. not neccessarily on the robot) and direct linkage to web technology make it the perfect backend for long-term storage in our eyes.

We created a set of web pages as a frontend to this data base to cover the previously mentioned use cases of inspecting logs, visualization, and performing queries. Several example screenshots of the webpages are shown in Figure 6.11. Unfortunately, they are rather long and narrow, but they give an idea of both the convenience and information richness we mentioned earlier.

The two left most columns show a single scene, split for the aforementioned reason, complete with logging annotations at the top. The logging information contains output from the various Annotators that helped in reconstructing this particular set of information. Some information is only available here, *e.g*. the similarity matrices for

the objects of a scene are logged using this mechanism until we create a dedicated visualization for it. For each cluster, we can see various information and annotations: *i*) a color image, *ii*) a tracking ID for the respective object track, *iii*) a perceptual hash, *iv*) a list of available annotations, if possible with an additional visual or textual representation for *e.g.* histograms, images, extracted text and so forth.

The third and forth columns show the result of textual search queries, in this case we visualize all clusters in a scene that matched the terms “ricola” and “cereal”, respectively. In all the web pages, the left most column contains a list of matching scenes. It can be seen that for example the term “cereal” appeared in 14 scenes (clusters from one is shown). For an evaluation of the relevance of search terms to the individual objects, we refer to the analysis in Section 5.4.5.

In conclusion, we employ an object-oriented data base to serve as a long term memory store, with mechanisms to automatically convert information from short-term and sensor memory – our CAS – and back. This data base can be queried either programmatically using client libraries, or using the web page that we developed as a front end to facilitate visualizing what the perception system is doing in detail, both for past percepts and for live visualization.

We plan to extend the functionality of this web page in the future with mechanisms to provide manual ground truth data, data labeling and annotation corrections. This can be invaluable for training machine learning systems – semi-automatic through the help of the tracking mechanisms – or as a general debugging tool. Another dimension we would like to add to the web page is the search using spatial or temporal queries. Currently, it is possible to query for scenes that contain objects from a certain region of interest using the logging information or the per-object `SegmentationLocationAnnotation`, and obviously for scenes or objects taken at a certain time or within a given interval using the time stamp information. However, we currently cannot directly make use of the per-cluster cartesian positions stored in the centroid field of the `TFLocationAnnotation` which contains position information as vectors containing “hard” numbers. To this effect, one can either create query objects within the database – using JavaScript to perform the comparison operations – or retrieve a subset of objects and filter them on the client side.

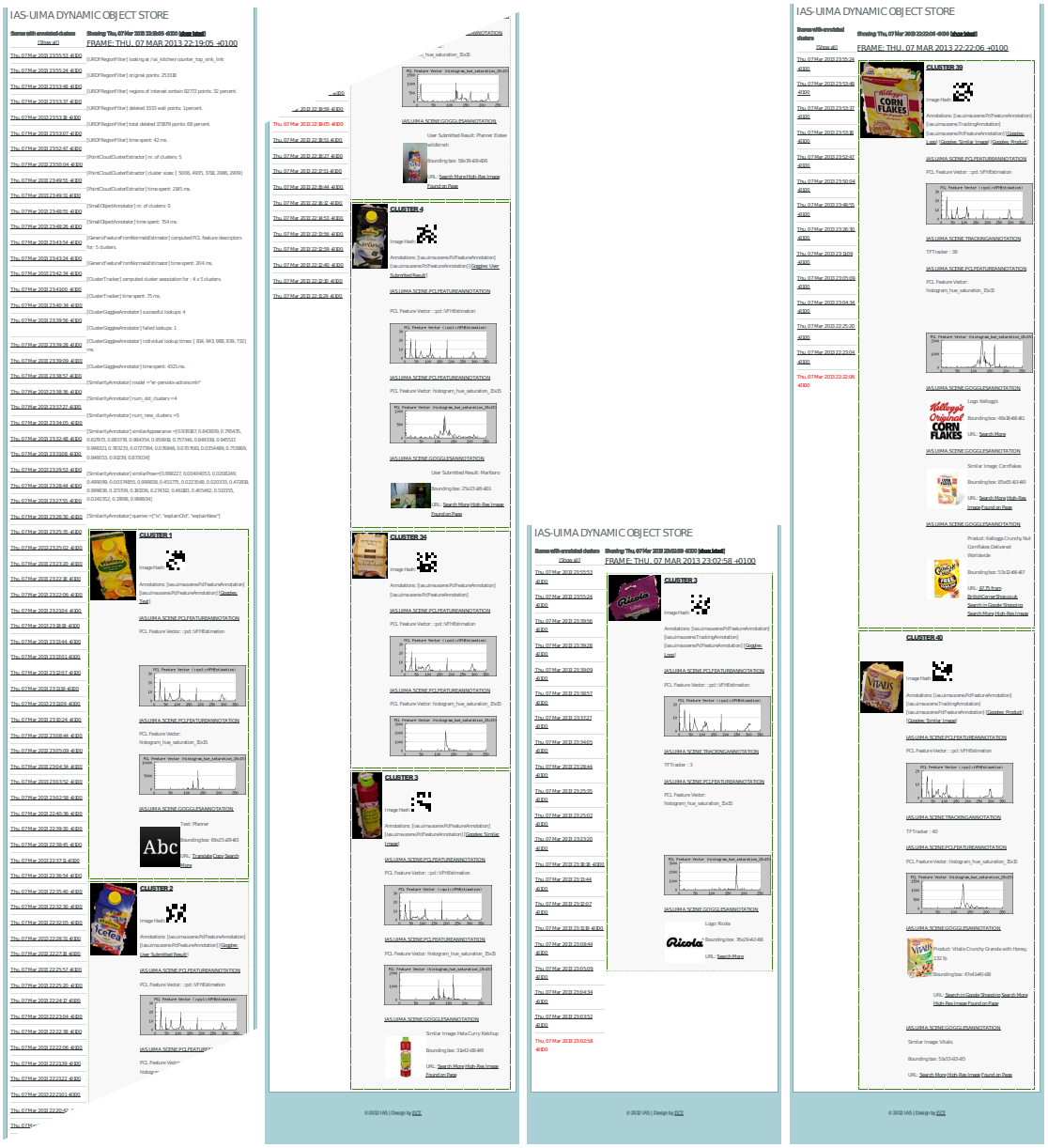


Figure 6.11.: *The web page interface for the database enables convenient visualization and debugging.*

Chapter 7

Autonomous Indoor Exploration and 3D Semantic Mapping

This chapter presents our contributions in the fields of autonomous exploration of indoor environments and the creation of Semantic 3D Maps, containing static and semi-static components such as walls, furniture, tables, drawers, doors and other articulated objects.

Consider the following scenario: A service robot is deployed and unboxed in a new environment and will be required to acquire the necessary skills and information to perform a range of future tasks. One of the most central and basic pieces of information for the successful performance of such a robot is that of a detailed environment map, in which the robot represents its knowledge about navigable areas, potential obstacles, horizontal support surfaces for object manipulation, and task-relevant appliances such as the dish washer and the oven. Additionally, it is necessary for the robot to be able to express certain *affordances* of these components, such as how to open or close appliance doors, where handles are located, or what surfaces can hold objects of daily use.

The acquisition of these 3D object maps from 3D sensors includes methods to scan and register the environment, to segment it based on priors about typical layouts and properties of indoor environments, and to categorize the resulting hypotheses. It is also necessary to *express* and store this information in a structured and reusable manner so that other system components can perform their tasks in an informed way.

We call environment maps that can provide these kinds of information *Semantic Object Maps* (SOMs).

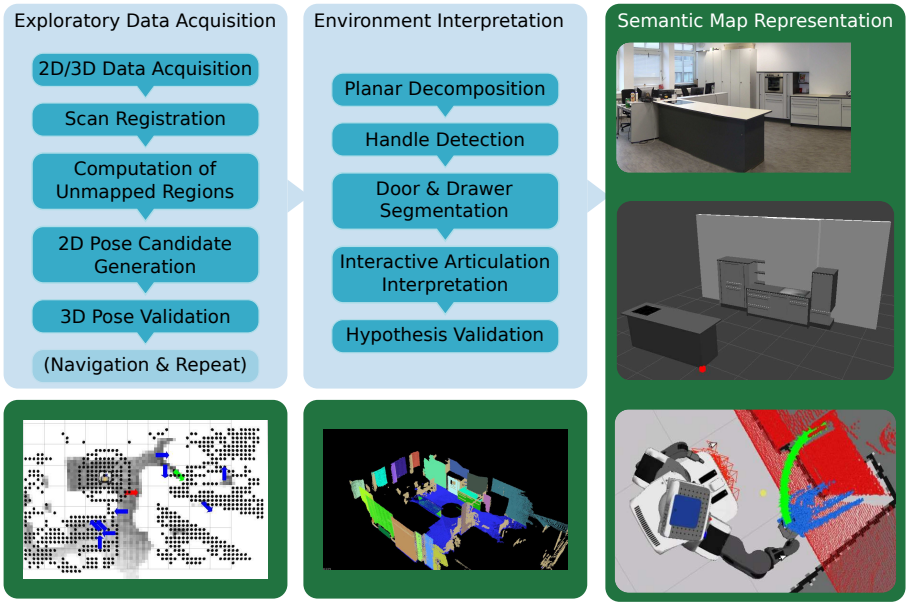


Figure 7.1.: *Conceptual overview of our autonomous mapping system: The three main components are the exploration and data acquisition phase (left column, Section 7.3), environment interpretation using point cloud and color image data (center column, Section 7.4), and the creation and representation of the resulting Semantic 3D Map (rightmost column, Section 7.5). Note that blue boxes represent processing algorithms, whereas green areas denote data representations. The striped parts are only explained briefly for completeness.*

More specifically, we attempt to achieve the following skills in our SOM acquisition system:

1. The ability to acquire a full 3D scan of a room from scratch through the use of an autonomous exploration strategy that can decide *where* to take the next scan, and how to *merge* multiple partial scans.

2. The generation of a *functional* model of the furniture that can be used in a physical simulator where cupboard doors and drawers can be opened and closed, and objects put and grasped inside them.

3. The association of *names* to different parts of the environments that share a common structure (*e.g.* all drawers should have unique names prefixed with "drawer_"). This is to allow the association of data processing modules with

different kinds of objects (*e.g.* delete all wall points, cluster all objects on tables *etc.*).

The work investigated here rests on previous research described in Rusu et al. [132], Blodow et al. [13]. Improvements have been made with respect to the autonomy of map acquisition, the multi-modality of sensing technology and the use of interaction with the environment to resolve ambiguous segmentation results. Additionally, we propose to represent the resulting map as an hierarchical, articulated model that allows visualization, simulation and semantic and fast data segmentation. Note that the work on interactive articulation model estimation has been performed by our colleagues Rühr et al. [124] and does not constitute original research within this dissertation. It is however described very briefly for completeness.

The complete autonomous mapping system as described in this chapter incorporates the following contributions with respect to related work:

1. Autonomous exploration, which includes the selection of the next best view pose to actively explore unknown space using a novel visibility kernel approach and associated costmaps.

2. Integrated registration, segmentation, and interpretation routines of color point cloud data, supported by physical interaction using the robot end effector to validate segmentation and estimate articulation models.

3. Representation of the resulting SOM in a hierarchical, semantic model that is easily integratable into other components to use for various purposes: Physical simulation, visualization, segmentation of new sensor data by specifying semantically meaningful search regions, or the association of specific data operations to parts of the model.

The remainder of this chapter is organized as follows. We will review related work in the next section, and describe the system overview in Section 7.2. We detail the data acquisition and next best view submodules in Section 7.3, and sketch the final point cloud interpretation system in Section 7.4. The semantic map generation and representation is given in Section 7.5. In the end we give insights into map usage and conclude with future work (Section 7.6).

7.1 Related Work

In the field of semantic environment mapping for robot manipulation, there is a body of interesting work over the past decades, as reviewed in [13].

Yamauchi [178] presented an approach based on a 2D map of grid cells which are labeled as *open*, *unknown* or *occupied*, and morphological operations and image processing techniques are used to find "frontiers" between unknown and open space. The frontier closest to the robot is then set as the goal for navigation, effectively creating a greedy exploration strategy. Surmann et al. [151] also propose an autonomous exploration strategy based on *seen* and *unseen* lines in a two dimensional projection of the environment. An extension of SLAM (simultaneous localization and mapping) to 6 dimensions (position and orientation) has been proposed by Nüchter and Hertzberg [101], where the sensor trajectory is estimated from frame to frame by registering consecutive scans. The authors proceed to segment the registered point cloud into regions like walls, floors and doors, and detect objects in the data.

Scott et al. [141] gives a review of methods for solving the next-best-view planning problem especially in the context of object modeling. Our approach to this problem draws inspiration mainly from two sources: *i*) The work-space exploration approach based on a labeled voxelized spatial decomposition as presented by Renton et al. [120] which is neccessary for next-best-view estimation and navigation planning with collision avoidance, as well as model verification [15]. We use OctoMap [176] to this effect. *ii*) The flexible object modeling method by Impoco et al. [59] which we adapted to our somewhat simplified problem that our scanner effectively has 3 degrees of freedom, position and orientation within the ground plane.

Point cloud registration, *i.e.* merging data from different view points to create a single and consistent representation of the environment, is often performed using the Iterative Closest Point (ICP) algorithm [11] or one of its many variants and improvements. ICP detects correspondences in two point sets using a simple closest point assignment and computes the affine transformation that minimizes the sum of squared distances between the correspondences. Iterative application converges the scans by applying transformations and re-evaluating correspondences until the two point sets match. Again, since our robot and ergo the mounted 3D sensor moves in the ground plane only, we can neglect errors in roll, pitch and height, which reduces the space of pos-

sible transformations between scans from 6D to 3D. The planar structure of their environment is exploited by Pathak et al. [108], which propose a consensus based algorithm that estimates the transformation between two scans using fitted planes and their uncertainties.

While the previous work used laser scanners or multiple view geometry using cameras, Henry et al. [55] used data from inexpensive RGB-D sensors for registration, detection of loop closure and bundle adjustment to achieve a globally consistent alignment.

The gap problem in AI, *i.e*. the transfer from the spatial to the semantic domain, has been addressed by Eich and Dabrowska [34]. Using a tilting laser scanner, spatial entities are extracted and, using a shape reconstruction step based on alpha shapes and feature descriptors, mapped to entities in the semantic domain such as desk, table or door. It is however unclear how the approach would scale to more fine-grained entities such as drawers, furniture doors and handles. The authors do promise to build a descriptive ontology that would allow for spatial reasoning in the semantic space, however we have addressed this issue in our previous work by Tenorth et al. [156], which we successfully integrated with our mapping approach in [13].

The work presented in this chapter is based on previous work in Rusu et al. [131, 132], where we registered multiple point clouds of a kitchen environment to perform a model-based interpretation and segmentation step to detect vertical and horizontal planar regions. We proceeded to split these into individual furniture doors and handles using region growing in the curvature space. The work presented here and by Blodow et al. [13] extends on this by projecting colors from a high-resolution camera onto the point cloud and rendering the mapping process autonomous.

Note that the whole system described here was a joint effort of multiple colleagues [13], and we will concentrate on the components and algorithms that fall within the scope of this dissertation. More specifically, this includes the Next Best View strategy (*cf*. Section 7.3.1) for autonomous exploration, and the semantic map generation and representation (*cf*. Section 7.5). The components involving fine-grained segmentation and interactive articulation model estimation (*cf*. Section 7.4) are the respective works of other colleagues and are described shortly purely for completeness.

7.2 System Overview

An overview of the system is shown in Figure 7.1. Initially, the process is started with the robot standing at an arbitrary position in the room. The process consists of two phases, depicted in the light blue boxes: Exploration and data acquisition, followed by environment and articulation model interpretation.

In the first phase, the robots iteratively takes scans and performs a Next Best View planning step in order to determine positions from which the robot can increase the coverage of the room. This position is chosen to maximise the amount of holes (due to occlusions and limited viewing frustrum) in the point cloud that could potentially be filled by taking a new scan. The robot navigates to the new position and the process repeats itself.

In the second phase, the full room scan acquired thus far is interpreted to extract the structure of the environment, which in our scenario consists most importantly of detecting furniture front faces, *i.e.* doors and drawers. These are further analyzed to detect handles and knobs, which are used as starting points for the interactive articulation model estimation and a fine-grained door boundary correction.

As a result of this acquisition and interpretation process, we finally compute a 3D Semantic Object Model of the environment [156], as can be seen in the third column of Figure 7.1. This model can then be used by other modules. Several possible use cases are given in Section 7.5.

7.3 Sensor Data Acquisition

For the system described in this chapter, we used our PR2 robot [177], which is equipped with a tilting 2D laser scanner in the neck (Hokuyo UTM-30LX) and a registered, high-resolution color camera by Prosilica (GC2450C with a resolution of 2448×2050 pixels). We opted not to use the Kinect RGB-D camera for reasons of resolution, both in depth and in color: The point clouds generated by the Kinect are spatially too coarse to detect the smaller surface variations *e.g.* at the seams between cupboard doors, and the color images are generally more blurry than the Prosilica

camera. Additionally, we can make full use of the wide opening angle of the Hokuyo scanner, which can be set to take full 180° scans.

The fact that both sensors are calibrated and registered to one another [115] allows us to perform segmentation along two feature dimensions, using both visual and geometric cues.

Starting with the second scan, we perform a registration step of the current scan with the previous scans based on the well-known ICP algorithm[11]. We use a point-to-point distance metric, and some minor modifications have been made to ensure successful convergence. ICP is performed on the subsets of all points which fall into the overlapping regions between the clouds to be registered. This is done with a fixed-radius nearest neighborhood search, and has been previously described by Rusu et al. [131] and Blodow et al. [13]. The search radius for this overlap estimation is modeled after the expected localization error between scan poses, since this displacement is the main source of misalignment between scans.

As an indicator for registration accuracy, we point to the handle validation step in [13], where out of a total of 18 handles present in our kitchen, 17 were successfully detected and manipulated.

7.3.1 Next Best View

In this section, we will detail the approach taken to enable the robot to actively and autonomously expore the indoor environment it is deployed in. This in essence amounts to the ability of the robot to determine a promising position that it can navigate to and take a new 3D scan. The central idea is that based on the point data gathered thus far, we determine robot poses from which as much new (unknown) scene geometry as possible is visible. However, there must be sufficient overlap to the existing data to allow for registration.

The proposed solution performs the following steps:

1. We compute a set of “interesting” voxels in space that promise high information gain. These points are defined as those that lie on a boundary between *mapped* and *unknown* volumes. We refer to these voxels as *fringe* voxels.

2. We generate a distribution over robot poses that encodes an estimate of how many of these fringe voxels can be seen from each robot pose. For performance reasons, we reduce this to a two-dimensional problem, *i.e*. we disregard the height components of these voxels.

3. Before determining the best pose from this distribution, we perform a computationally more expensive but more accurate validation step in three dimensions that considers a subset of the best poses sampled from this distribution. This step uses a simulated sensor (modeled after the actual sensor) to render a virtual view of all voxels from each pose to be validated.

Step 2 and 3 are done this way based on the intuition that the 2D problem formulation can generate promising poses much quicker, and the more costly 3D validation step only has to consider a considerably smaller subspace of the problem.

7.3.1.1 Determining unmapped regions

The selection of interesting voxels that promise acquisition of unmapped areas is done with the help of an octree [90] representation of the accumulated point cloud. We opted to use the OctoMap library [176] due to the fact that on top of octree creation and search, it contains methods for ray casting which we use in the 3D validation step (see below).

The octree is constructed from all point clouds accumulated up to this step, and all octree voxels containing points are marked as *occupied*. All voxels that lie between the scanner position and an occupied voxel are marked as *free*. The laser rays have traveled throught these without detecting an obstacle and are thus considered known and empty. All other voxels are labeled as *unknown*, *i.e*. unmapped.

Since intuitively, we would like for the robot to stand in *free* space and look into unmapped (*unknown*) space, we compute the voxels that lie on the boundary between these two kinds of volumes. As they constitute the fringe of our knowledge horizon, we call these *fringe* voxels, and they can be thought of as “windows” into unexplored space. Algorithmically, we iterate over *free* voxels and relabel them as *fringe* if they have any neighbors in unknown space.

An example is shown in Figure 7.2.

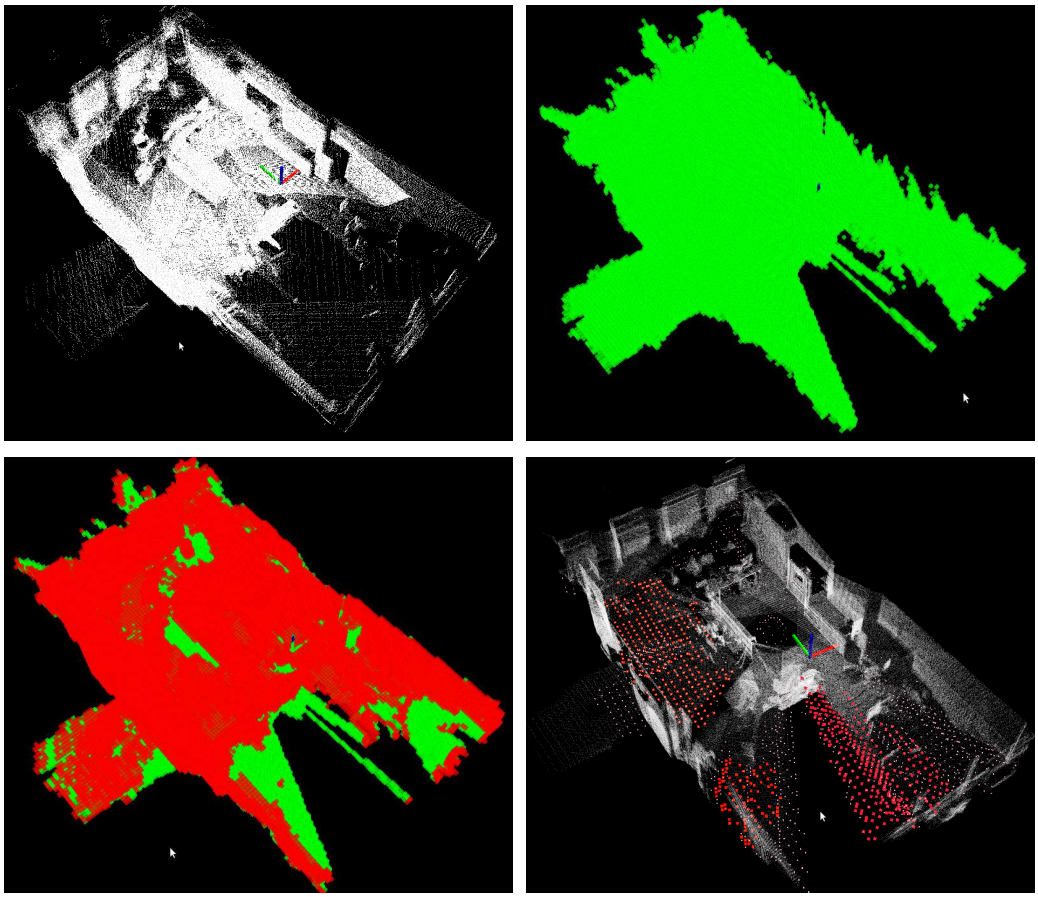


Figure 7.2.: *Visualization of input data and voxels in the Next Best View system: (Top left:) accumulated point cloud after registering several scans of the room. (Top right:) green voxels represent* free *space, (bottom right:) red voxels represent* occupied *space. (Bottom right:)* fringe *voxels are represented by red points. These represent "windows" from free into unknown space.*

7.3.1.2 Computing candidate robot poses

Given a set of *fringe* voxels, the next step is to estimate poses from which the robot can "see" as many of them as possible. As argued above, we reduce the dimensionality of this problem by considering only the x and y components of each voxel, ignoring z (height). This is effectively a projection of each voxel onto the ground plane.

Consider the left image in Figure 7.3. The robot sensor is located at point s, oriented to look along ϑ. Given sensor properties ϕ (opening angle) and d_{min}, d_{max} (minimum and maximum range), the light blue area contains all voxels v_i that are visible in this pose. To ensure a certain scanning resolution, one can shape that visibility region to the dark blue area by tightening the sensor parameters accordingly.

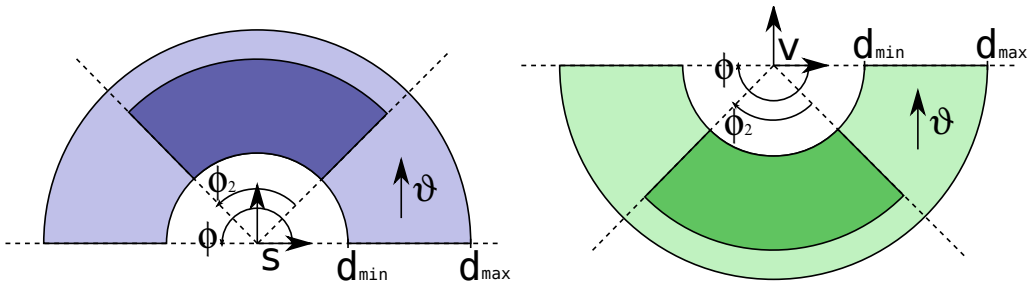


Figure 7.3.: *Basic idea of the Visibility kernel: The blue template represents all voxel positions that are visible from sensor position s with orientation ϑ, given sensor parameters such as minimum and maximum range (d_{min}, d_{max}) and opening angle (ϕ). The inverse template, shown in green, represents the "Visibility Kernel" for a given voxel v: the shaded areas contain all possible robot positions from which v is visible.*

Since we are interested in the reverse problem (find possible sensor poses given *fringe* voxels), one can simply "flip" that stencil to define a *visibility kernel* relative to a voxel v, as depicted in the right image in Figure 7.3.

More formally, we define the *visibility kernel* $K(\phi, d_{min}, d_{max})$, which encodes the set of poses (relative to a voxel v) from which that voxel is visible. This kernel is again a function of the sensor's horizontal opening angle ϕ, and its minimum and maximum range d_{min} and d_{max} and represents a volume in the robot's 3D planar pose space $\langle x, y, \vartheta \rangle$. In Figure 7.3 (right), we show one slice of the visibility kernel for a given robot orientation (view direction) ϑ. The shaded areas represent poses from which the voxel v can be seen given a sensor opening angle of ϕ (light green) or ϕ_2 (dark green).

Note that in our case, we set d_{min} to the distance of the robot base center to the closest point that can be scanned on the ground ($\sim$ 1.0 m), and d_{max} to 4.0 m, since we consider the point density of areas which are farther away too low.

To use this kernel, we define a discretized representation of the robot's pose space within our environment using a stack of *costmaps* (essentially a 3D voxel grid). The dimensions of this grid in x and y are taken from the maximal extents of the accumulated point cloud, dilated by d_{max}, and we discretize the robot's rotation into n bins. For our experiments, we chose a coarse discretization into $n = 8$ bins because of the large opening angle of our sensor. This amounts to 8 2D costmaps. We set the spatial resolution to 10 cm in x and y.

We loop over the fringe voxels and apply the visibility kernel as an additive stencil on each costmap, rotating the kernel to match the respective costmap layer. The result of this process is a stack of costmaps in which cells with a high value correspond to poses from where many fringe points can be seen.

Note that one can also apply a weight w_i for different areas of the visibilty kernel. In Figure 7.3, costmap cells within the dark areas could be assigned a higher weight than the in light areas. However, we did not see significant differences using this approach and did not pursue it further.

A nice effect of using a costmap-based algorithm is the fact that one can trivially combine other kinds of costmaps to steer the exploration behaviour by overlaying several quality measures for each robot pose represented by a voxel in our costmap stack. A straightforward example is intersecting our costmaps with a navigation-based *reachability* map. This way, robot poses that are not reachable from the current position get assigned a weight of zero. We also ensure that the robot can stand at a given pose by overlaying an occupancy map representing if we scanned points on the floor at each pose. If there are stairs or other areas where the ground level drops below zero (to be more precise: where we did not perceive a floor surface at $z \approx 0$), we can prevent the robot from considering those poses.

Additionally, as stated above, we want to steer the exploration strategy in such a way as to ensure sufficient overlap with existing scan points for successful registration. So far we have considered pose reachability, presence of floor to stand on, and maximized information gain in number of *fringe* voxels.

To achieve an estimated overlap of 50%, we compute a new stack of costmaps for all occupied voxels in the octree, and combine the two costmap stacks using a minimum intersection approach: Let $C_F = \{f_{x,y,\theta}\}$ be the fringe and $C_O = \{o_{x,y,\theta}\}$ the occupied

costmap. The resulting costmap is thus defined as follows: $C = \{c_{x,y,\vartheta}\}$ with $c_{x,y,\vartheta} = \min(c_F, c_O)$ for all x, y, ϑ. Maxima in C represent poses from which many *fringe* **and** *occupied* voxels are visible according to the reduced 2D problem formulation.

Note that we intersect C with a dilated version of the occupancy grid used for navigation to eliminate impossible poses such as within walls, and we set all cells $c_{x,y,\vartheta}$ to zero if the octree does not have an occupied voxel on the floor at $\langle x, y \rangle$. Other penalty or reward functions can easily be added, such as the distance that the robot would need to travel to the next scan pose.

7.3.1.3 Pose validation

The previous subsection has illuminated our approach to generate promising poses that warrant successful navigation, discovery of new scene geometry and successful registration. However, since we reduced the dimensionality of our voxel representation of the accumulated point cloud, we cannot be sure of those properties due to the possibility of occlusions.

As an example, consider the occlusion "shadow" cast behind a table. Our approach will consider scan poses on the wrong side of the table to be valid candidates, but scanning from them would not result in new information. Desirable poses would lie on the opposing side of the table, so our validation step weighs promising poses while taking into account the three dimensional nature of our problem.

We therefore sample a set of poses from C and validate them using a raycasting approach that considers the whole 3D geometry. The probability of selecting a cell (= pose) $c_{x,y,\vartheta}$ is proportional to the pose quality associated with it. We then use the OctoMap to cast rays from the 3D sensor position $\langle x, y, z_s \rangle$, where z_s is the height above ground of the optical center of the laser scanner. These rays are cast according to the sensor model and we count the number of rays that intersect an occupied or fringe voxel as n_o and n_f, respectively. We compute a score for each pose using a function $w_{3D}(n_o, n_f)$ that is designed with the following properties in mind: at $n_o = 0$ or $n_f = 0$, w_{3D} should return zero, and at $n_o = n_f (n_0, n_f ! = 0)$, we want w_{3D} to take a maximal value. Letting

$$\begin{aligned} p_1 &= n_o/(n_o + n_f) \\ p_2 &= n_f/(n_o + n_f) = 1 - p_1, \end{aligned} \tag{7.1}$$

this behaviour can be achieved by the well-known function for the binary entropy of a random variable, H:

$$w_{3D}(n_o, n_f) := H(p) = -\sum_{i=1}^{2} p_i log(p_i). \tag{7.2}$$

H is maximal ($H(0.5) = 1$) if the numbers of visible fringe and occupied voxels are the same.

For every pose sample, we multiply the reward $c_{x,y,\vartheta}$ from the costmap with w_{3d} and sort the list of poses by this score. The robot traverses this list and selects the first for which the navigation planner can actually generate a path. We make use of the PR2's built-in *navigation stack*, which was has demonstrated its ability to complete a "marathon" (26.2 miles) of autonomous navigation in a real office environment [85].

In Figure 7.4, we visualize the results of the Next Best View algorithm for the scene shown in Figure 7.2. The stack of costmaps has been flattened into a single costmap, disregarding sensor orientation, by per-voxel accumulation. Black cells have high weight, white means low weight. Fringe voxels are shown as black dots, and arrows represent poses sampled from the costmap stack. 11 Poses have been sampled from C (9 shown in blue), the green pose has the highest number of fring points visible in its sensor frustrum, but the red pose is the best one after weighting with w_{3D}. Due to the sampling step, the algorithm takes into account poses from the whole pose space as opposed to all poses being clustered densely around the most promising pose according to the heuristics presented above.

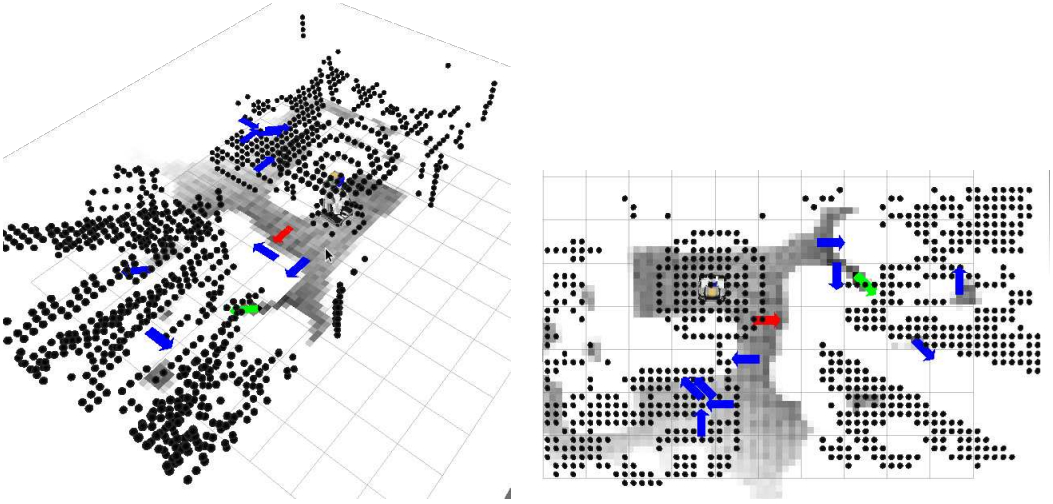


Figure 7.4.: *Computation of next best view poses in a perspective (left) and topographical (right) view. Black dots show fringe voxels, arrows represent samples drawn from the costmap stack C: pose with highest number of fringe poses (green) and final, weighted best-registration pose (red).*

7.4 Point Cloud Data Interpretation

In order to extract the structures mentioned above, the system extracts relevant planes from the registered point cloud, categorizes them as doors or drawers, walls, floor, ceiling, and tables or other horizontal structures [13]. This is achieved by locating the relevant planar structures using Sample Consensus based methods, checking for the existence of fixtures, and segmenting the different doors.

In most indoor environments, many surface normals and therefore planar orientations coincide with one of the three main axes of the room, these main axes can be used to limit and accelerate the plane extraction.

The primary planes are classified into floor and ceiling based on their orientation and height, and walls based on the observation that they are adjacent to the ceiling. Note that while furniture does occlude much of the wall surface in lower regions, walls usually are unobstructed where they connect to the ceiling.

Remaining planar connected components of sufficient area constitute candidates for tables and furniture faces. Fixtures are located in front of – and within the bounding polygon of – their respective furniture front. A Sample Consensus step with linear

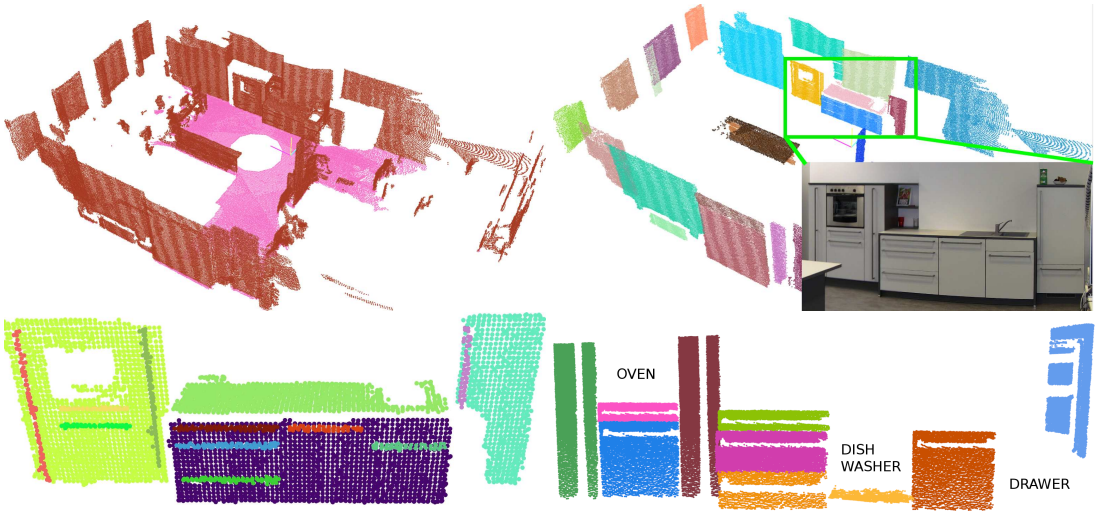


Figure 7.5.: *Resulting segmentated point cloud (shown for an incomplete room for occlusion reasons): highlighted floor shows regions where robot can navigate to and scan (top left), horizontal and vertical planes are segmented using RANSAC based methods (top right). Furniture fronts and handles are coarsely segmented (bottom left) and refined using the assumption that one drawer has at most one handle (bottom right). Note that the last image shows a scan taken after opening the drawers, dishwasher and fridge. Segmentation of the oven and the drawer under the fridge failed due to the lack of points on the handle fixture, whereas the dishwasher's inner metal surface is highly reflective. The final map is shown in Figure 7.8.*

and circular models performs the model fitting and classification into handles and knobs.

The result of this process can be seen in Figure 7.5.

A region growing process in curvature and color spaces is performed starting at each detected fixture to subsegment furniture faces, if necessary (see Figure 7.6 for detected handles and doors).

We make use of the fact that our robot can interact with the environment to improve our knowledge about the articulation mechanics of our environment. Doors and drawers can be segmented by opening them and evaluating the temporal differences. This makes it possible to determine the type of joint (rotational or translational) by analyzing the trajectory generated by a compliant motion control strategy. This is inspired by previous work to overcome uncertainties [33] or verify object

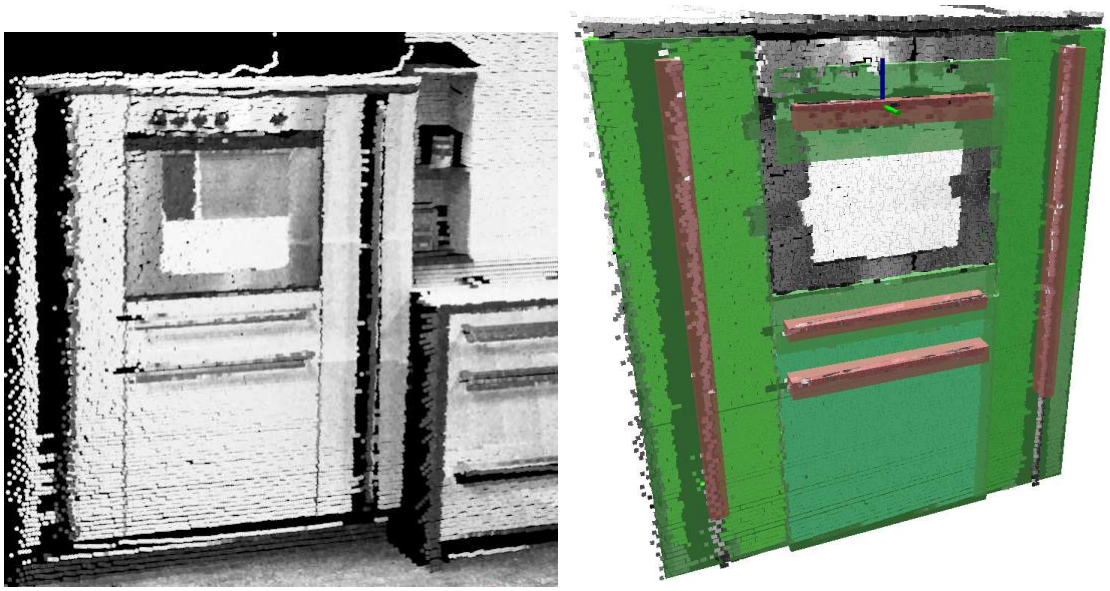


Figure 7.6.: *To subsegment coincident and neighboring drawers and doors, a region growing step is performed starting at each located handle. This is based on the reasoning that each door or drawer will have at most one handle. Points are colored with intensity information and detected handles and cabinet fronts are shown in brown and green, respectively. Note that a rectangle fitting step is performed to force the resulting models to be locally axis-aligned. Again, the specular reflectivity of the metal oven front introduces large errors.*

grasp models [23], and has been extended to operate furniture pieces without a priori knowledge about the validity of the handle hypothesis or the underlying articulation model.

A final temporal registration step is used to verify the door or drawer outline [131, 13]. The interactive articulation model generation and temporal difference segmentation is depicted in Figure 7.7 for several different handle hypotheses, where gripper trajectories are shown in green, and temporal differences in blue.

The methods discussed in this section have a certain amount of overlap with previous work and work done by colleagues and are therefore sketched briefly here. For further details, we refer to our earlier publications [131, 88, 13].

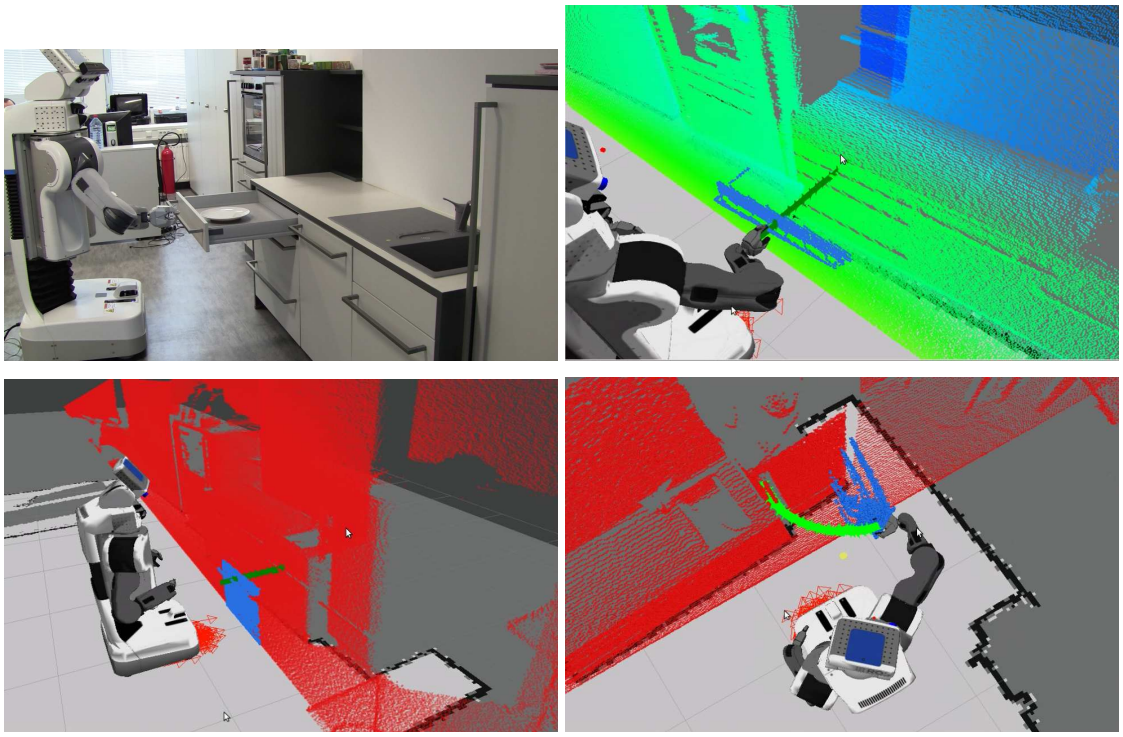


Figure 7.7.: *Examples of interactive segmentation.*

7.5 Semantic Map Generation

The structured combined models of the preceding algorithms are accumulated in a Semantic Object Model, or 3D Semantic Map. This is an extension of our work presented by Rusu et al. [131], Rusu [125].

The map is represented as an OWL-DL ontolgy and can thus be shared between different modules or even different robots operating in the environment. It is furthermore being used in the KnowRob knowledge processing system [156, 153]. The OWL-DL representation and its use in knowledge reasoning falls outside the scope of this thesis. However, extending the mentioned previous work, the semantic map can additionally be exported to a URDF (Unified Robot Description Format) model. This model represents an acyclic graph of *links* connected by *joints*, which desribe the kinematic and dynamic connections between links.

URDF was developed with robots in mind, and a link thus typically represents *named* robot part, *e.g.* an arm segment, and is connected to its parent link by a joint. Joints

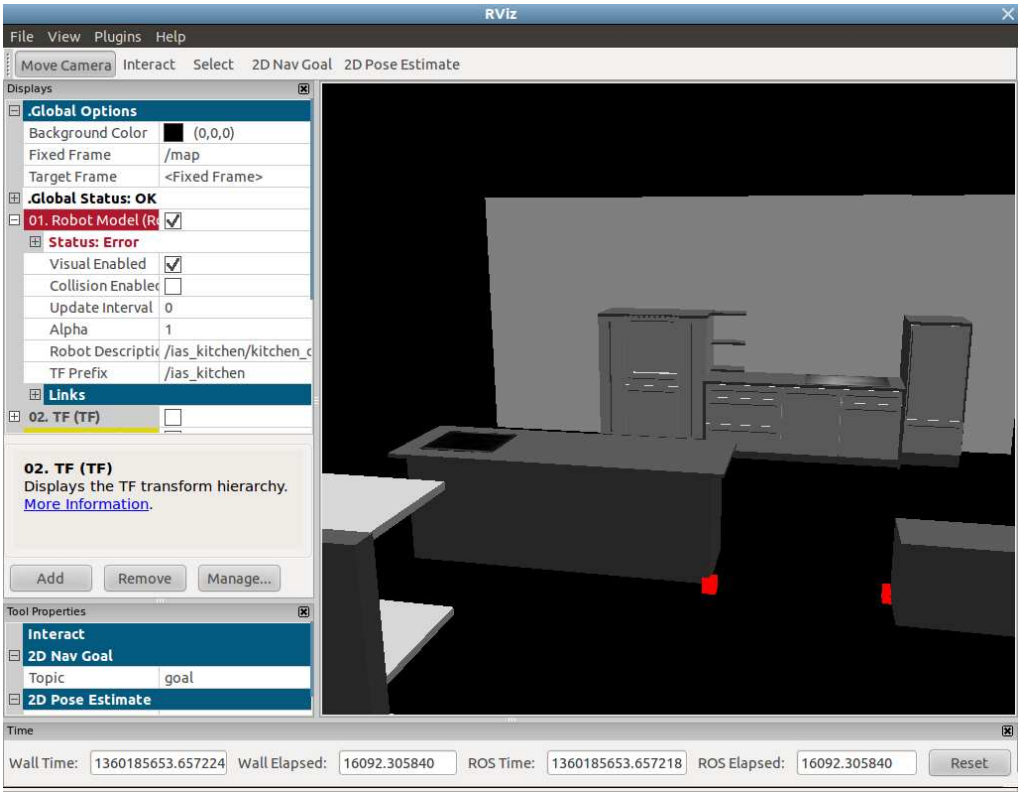


Figure 7.8.: *Resulting Semantic Map created by the process described here (visualized in RViz, the ROS visualization software).*

can be of types, *e.g.* rotational or prismatic for articulated joints such as an elbow or telescoping torso, or fixed for rigid link connections. Links are further annotated with inertias, visual features and collision geometry in the form of primitive shapes (cylinders, boxes) or freeform triangle meshes. This serves multiple purposes, *e.g.* the possibility to simulate the robot in Gazebo or visualization in RViz, the ROS visualization software. Both application scenarios are demonstrated in Figure 7.10 for the simulation (left) and actual execution (right) of the pancake making demonstration of our lab [7].

However, the format lends itself perfectly well to our semantic map requirements, where every detected furniture object – or part thereof – is represented as a link, connected by mostly fixed joints (for non-articulated parts). For every articulated part, the joint encodes the respective articulation model. Additionally, geometric extents

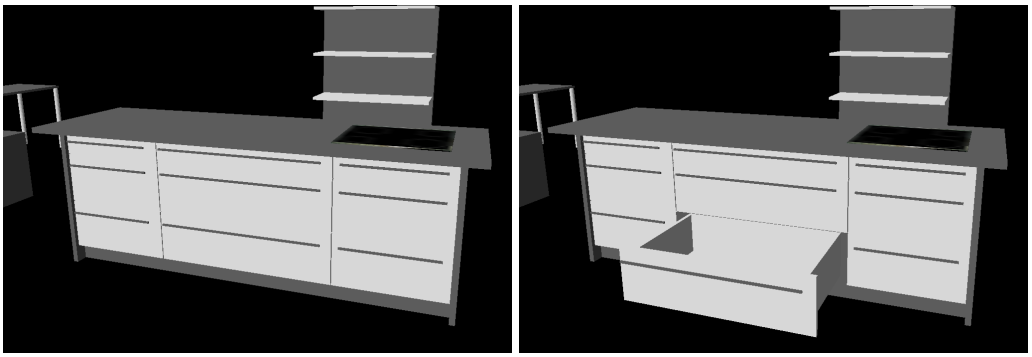


Figure 7.9.: *Articulated parts of the URDF model can be parameterized by setting a joint angle (*e.g. 50*cm in this example).*

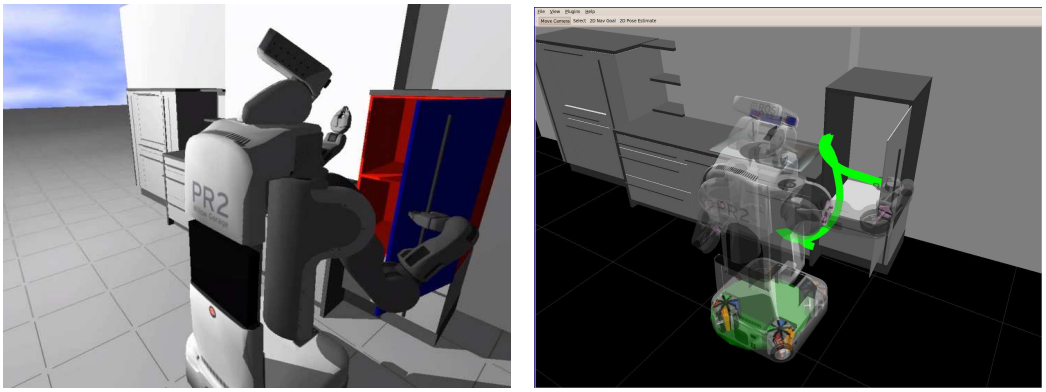


Figure 7.10.: *Usage example of the generated URDF model of the Semantic Object Map: the robot opens the fridge door in the Gazebo simulator for the Pancake Demonstration (left). The corresponding visualization in RViz shows the trajectory of the end effector during the real-world (non-simulated) execution (right).*

and the visual and collision geometry are stored in the link nodes as well. The final map, represented in URDF is show in Figure 7.8.

In the robot case, sensor readings from *e.g.* motor encoders representing a joint angle can automatically be converted into a full transformation matrix by the *Robot State Publisher* node within ROS. This node takes these scalar values, and with the use of the robot URDF model publishes *tf* transforms. In the same way, it is possible to manipulate the articulated parts of the semantic map by setting a joint value (angles for rotational joints, length units for prismatic ones). This is depicted in Figure 7.9, where

the joint corresponding to the central bottom drawer in the free-standing kitchen island has been set to $50cm$.

All nodes in the robot system requesting the current location and orientation of the drawer handle from the *tf* library are automatically served with the corresponding transform.

The URDF graph is shown at the end of this section in Figure 7.12, with links and joints represented as boxes and ellipses, respectively. As can be seen, the graph is rather large, which is partly due to the conversion process: When generating the URDF model, detected drawer cuboids get replaced by a more detailed drawer macro, parameterized by the cuboid dimensions. This makes it possible to define a *prototype* drawer with a higher level of detail, *i.e*. left and right sides, the drawer bottom where objects rest on, and a front part with handle. Without this replacement macro, the drawer visible in Figure 7.9 would simply be an opaque box, and physics reasoning techniques would be blind to the possibility of placing objects within these drawers.

For clarity, we extracted the subgraph of the kitchen island block in Figure 7.11. For readability, the graph has been simplified: identity transforms as well as chains of fixed joints or purely structural links have been removed or shortened. We segmented the resulting tree in several gray boxes, corresponding to larger components of the island, and emphasized several links and joints in different colors to illustrate different usage scenarios.

The aforementioned articulated joints have been highlighted in orange, showing the 9 degrees of freedom contained in this grid of 3×3 drawers. The links representing drawer fronts are shown in light blue. A model fitting perception module could use all of these links to *estimate* each drawer state, publishing the resulting joint angle, as described above. While we performed preliminary experiments in this direction, we have to refer to future work for a more thorough investigation of this interesting topic. What *is* perfectly possible though is testing the presence of obstacles in front of drawers, *e.g*. if queried by a planning module *before* opening a cabinet.

Decision making processes performing tasks involving drawer or cabinet manipulation can directly use the handle positions in the model, shown in green. This is currently only possible in the closed state, due to the lack of articulation state estima-

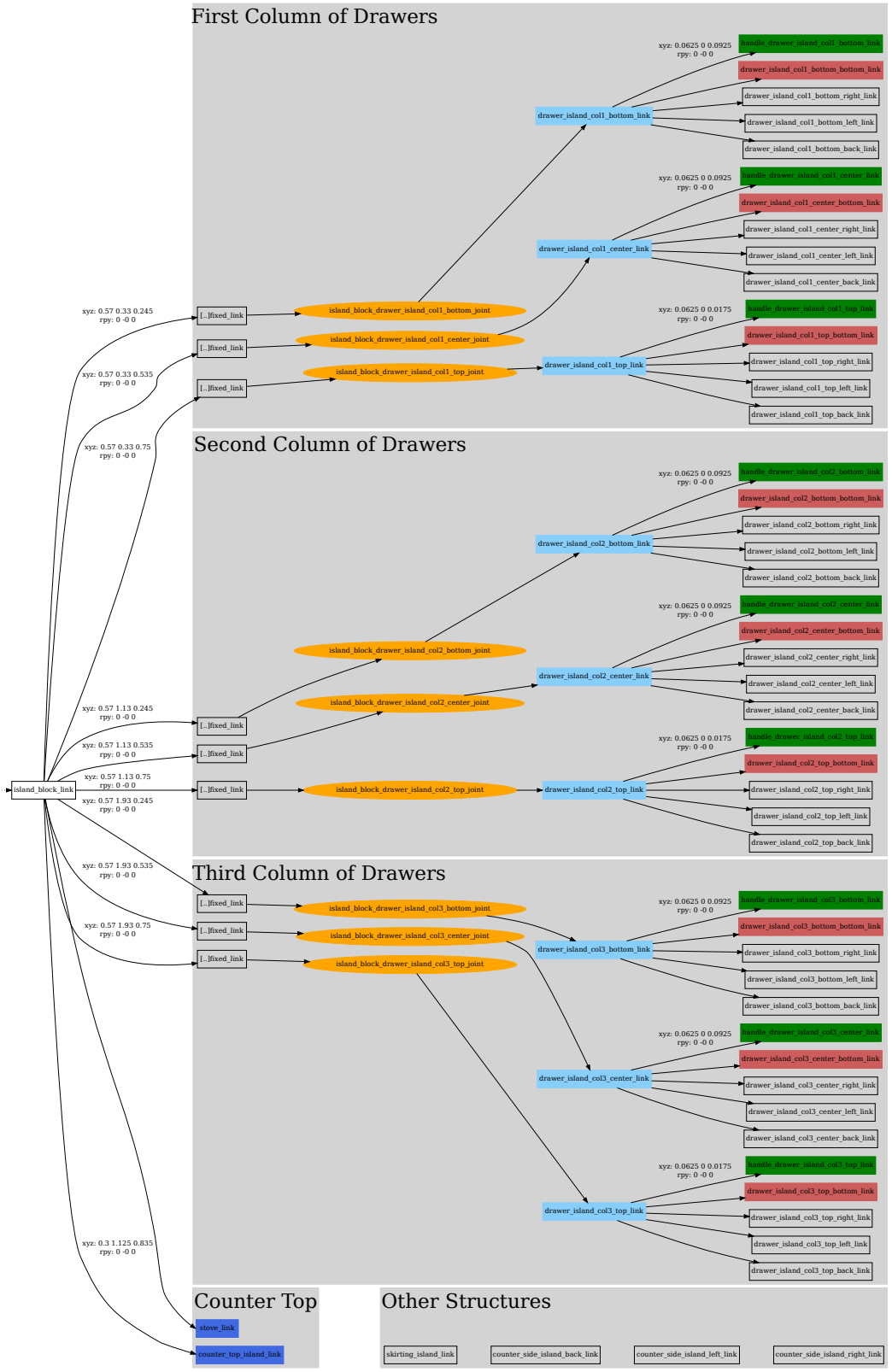


Figure 7.11.: *Simplified diagram of the part of the URDF model encoding the free-standing kitchen island, consisting of 3 rows of drawers and a counter top with a glass-ceramic cooktop. For the complete model, please refer to Figure 7.12*

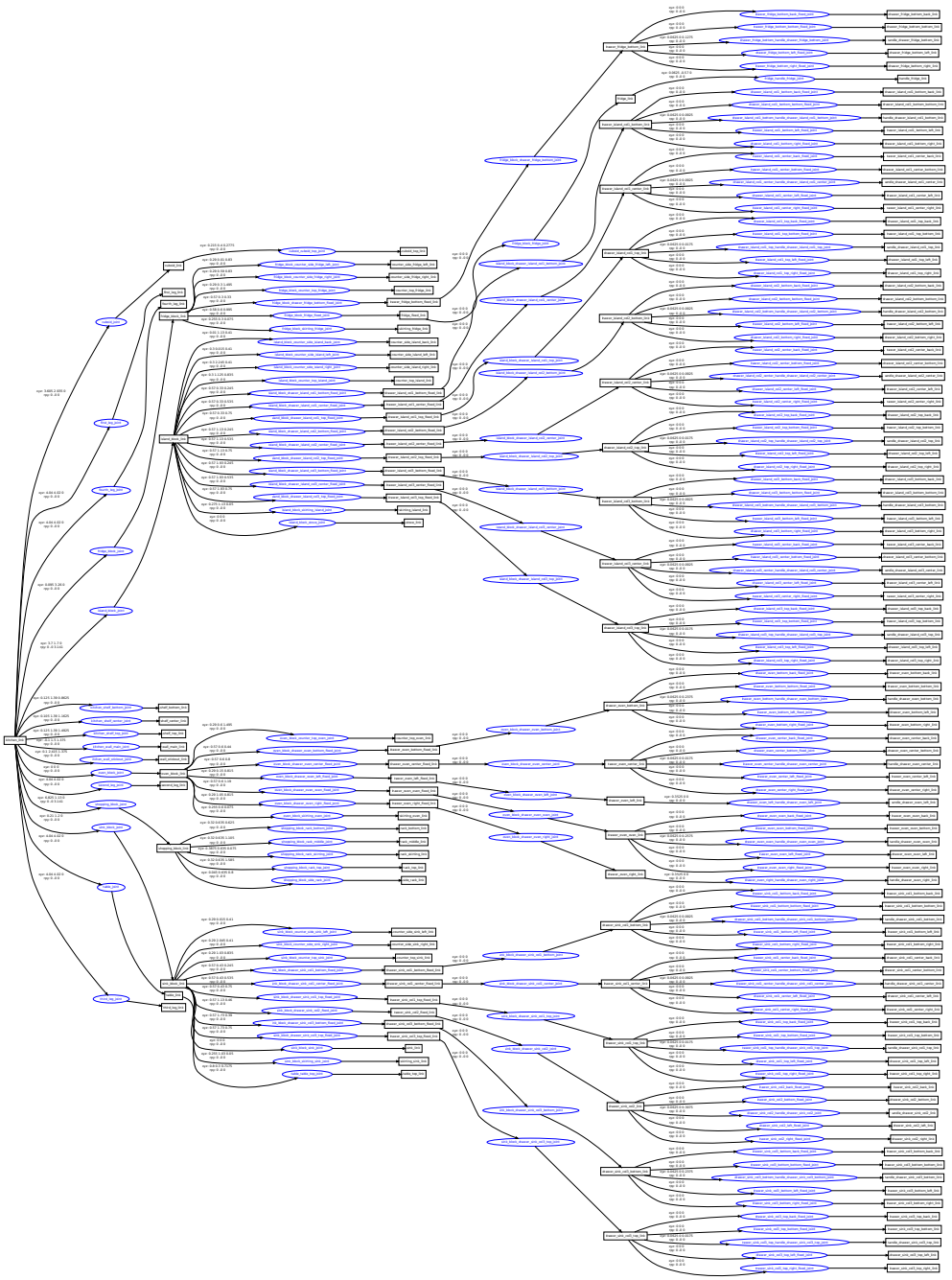


Figure 7.12.: *Graph diagram representing the whole URDF model of the 3D Semantic Map: Links and joints are shown as boxes and ellipses, respectively, and their connecting edges are annotated with the respective transform where appropriate.*

tion, as explained above. However, if a higher level decision making process performs an action on an articulated door, it can simply assert the result of its action.

Shown in red are the bottom parts of drawers, where objects can be placed upon. This in essence is not much different from the links shown in blue, which represent the counter top and stove top. Both are possible horizontal support surfaces for objects of daily use. The only difference is that drawer bottoms are not fixed in real-world coordinates. Yet, objects detected in the local coordinate system would in general stay fixed in this reference frame, even when opening or closing a drawer.

For the purposes of our perception system, Section 4.2.2 has reported on several ways on how to best make use of the information available in this representation.

7.6 Conclusions and Future Work

In this chapter, we presented our work published in Blodow et al. [13] on an integrated system for autonomous exploration and semantic mapping enabling the robot to automatically explore an indoor environment while simultaneously building its semantic map. The semantic aspects of this map lie in the fact that we extract functional models for manipulable parts of the environment, such as drawers or furniture doors. We furthermore can estimate an articulation model for these parts using an interactive step in which the robot attempts to open any furniture front for which a handle has been detected, using force sensors in the arms or end effectors to manipulate the door compliantly.

In previous work [131, 156], we demonstrated the link between the spatial domain and a symbolic, semantic representation using OWL-DL to allow problem formulations such as furniture categorization or place classification based on our semantic map.

Our exploration and mapping solution does have some deficiencies when dealing with highly specular or translucent surfaces, such as the glass in windows or oven doors, as these parts do not get captured using current 3D sensing technology. To address this problem, we plan to incorporate computer-vision-based approaches such as the one presented by Fritz et al. [43]. Another possibility to pursue in the future

is to improve the registration performance to allow scan acquisition with less overlap [55, 108], which can be steered by adjusting the use of the entropy-like weighting function mentioned earlier (Equation 7.2).

Chapter 8

Evaluation of RoboSherlock

In this thesis, we have described RoboSherlock, a software framework for implementing perception systems that can scale towards real-world perception task complexity. Evaluation of the performance and capabilities of the perception system within our framework in terms of comparisons with existing perception systems has proven difficult to do. This is due to the fact that most related work in this area has been focusing on single algorithms, and very little research into combining ensembles of various heterogeneous methods can be found in the literature. In this thesis, the performance of individual methods usually has been discussed in the corresponding scientific publications or the respective topical chapters earlier.

However, comparing a system as powerful and encompassing as RoboSherlock with individual, specialized algorithms would in most cases be unfair to both sides. It is evident that a specialized algorithm that is tuned to its application domain (and usually evaluated on a dataset with a certain prior) will achieve *e.g.* high precision and recall on its target data, and simply fail to run on data which it is not designed for. As an example, consider a SIFT-based object detector that is being applied to non-textured or even transparent objects. Or consider evaluating its abilities to generate product information such as ingredients or price for these objects.

On the other hand, the performance in terms of run-time of such an algorithm would obviously be better than a larger system that utilizes a multitude of such specialized algorithms.

Additionally, since most of our Annotators can be thought of boot-strapping an object catalog, our current implementation relies very little on pre-trained datasets. Actual

object identification is not being performed at the moment, which makes it hard to assess ground-truth fidelity.

We will therefore not evaluate RoboSherlock in terms of performance, recall, accuracy, and quality *in comparison to other methods*. In fact, we take the view that perhaps the best indication for the strength of the approach is the richness of the object annotations that the robot can generate autonomously by our application system, and which we believe to be unrivaled in our field.

8.1 Exemplary RoboSherlock Analysis for a Cluster

Figure 8.1, depicts the results of different Annotators that were used for analysis of a ketchup bottle: A `PclFeatureAnnotation` of type VFH, a `GogglesAnnotation` with two results, and a `ProductAnnotation` with a positive database match from GermanDeli.com. Some annotations, *e.g.* `ClusterPoints`, `LocationAnnotation`, `ColorHistogram` and `TrackingAnnotation` have been omitted for simplicity.

The Goggles results are of two types: *Similar Image*, along with the name "Hela Curry Ketchup", and three URLs, pointing to a Google search page, a high resolution image, and the page on which the image has been found. The second hit is OCR *Text* detection, "Curry", with links offering *e.g.* a translation. It is evident that these two strings by themselves are very informative and could be used in a variety of contexts, *e.g.* matching a user voice command to "pass the ketchup" or to narrow down a list of matches of similar products based on flavor.

Based on the `ImageROI` of the cluster, a match has been found in the database of GermanDeli products, and the resulting `ProductAnnotation` contains a list of key-value pairs, some generic (*e.g.* source and product name), some specific to the source (*e.g.* ingredients and perishability). From this, it can be deduced that *e.g.* the bottle needs not be stored in the fridge, and it gives rise to the idea of the robot ordering a new bottle if so instructed. A robot deployed in a clinic or other clinical care context might point out the sugar content before handing it to a diabetic patient.

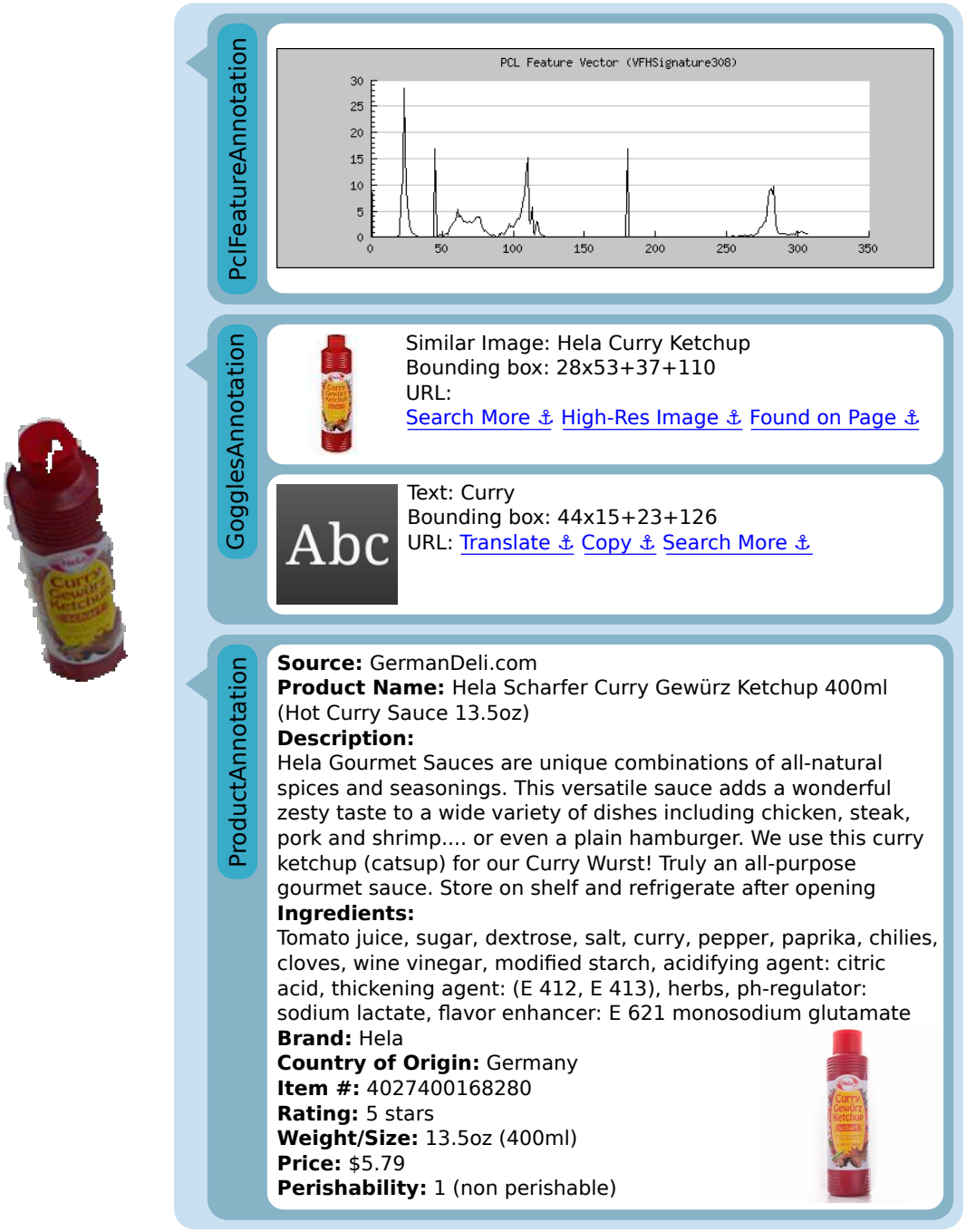


Figure 8.1.: *Results of* RoboSherlock *(right) for an observation of a ketchup bottle (left). The most common annotations, such as the point cluster, high-resolution image, and location annotations have been omitted for brevity.*

8.2 Experimental Evalutaion

In order to demonstrate the rich information gathering capabilities of RoboSherlock, we set up 103 scenes in our lab kitchen environment, on top of two counters. The range of objects varied from small items such as a bottle cap and cutlery to relatively large items, *e.g.* a pancake maker or cereal containers. Several objects were rather flat (*e.g.* a book and a magazine, cutlery), others were three-dimensional (*e.g.* bottles or boxes), and there were textured (*e.g.* food products, drink containers) as well as non-textured objects (*e.g.* plates and cups). All in all, except for transparent objects, we attempted to create a comprehensive set of objects that occur in regular household environments and can be manipulated by the robot.

Table 8.1.: *Histogram over number of flat objects per scene.*

# flat objects	# occurrence	relative frequency
0	27	26.2%
1	30	29.1%
2	16	15.5%
3	14	13.6%
4	7	6.8%
5	7	6.8%
6	2	1.9%
(total)	103	

Table 8.2.: *Histogram over number of three-dimensional objects per scene.*

# 3D objects	# occurrence	relative frequency
0	1	1.0%
1	6	5.8%
2	9	8.7%
3	24	23.3%
4	22	21.4%
5	23	22.3%
6	17	16.5%
7	1	1.0%
(total)	103	

Table 8.3.: *Histogram over total number of objects per scene.*

total # objects	# occurrence	relative frequency
3	9	8.7%
4	19	18.5%
5	20	19.4%
6	28	27.2%
7	10	9.7%
8	11	10.7%
9	46	3.9%
10	1	1.0%
11	0	0.0%
12	1	1.0%
(total)	103	0% 7.5% 15% 22.5% 30%

Table 8.4.: *Histogram over object sizes (measured in number of points) for all 587 object observations.*

object size	# occurrence
$t \leq 1000$	145
$1000 < t \leq 2000$	98
$2000 < t \leq 3000$	127
$3000 < t \leq 4000$	71
$4000 < t \leq 5000$	42
$5000 < t \leq 6000$	29
$6000 < t \leq 7000$	15
$7000 < t \leq 8000$	13
$8000 < t \leq 9000$	15
$9000 < t \leq 10000$	9
$10000 < t \leq 11000$	8
$11000 < t \leq 12000$	7
$12000 < t$	8
(total)	587 0 40 80 120 160

Each scene contained 0 to 6 small (flat objects) and 0 to 7 three-dimensional objects, for a total of 3 to 12 objects per scene. Detailed histograms are given in Tables 8.1 to 8.3, where it can be seen that most scenes (around 75%) contained 4 to 7 objects.

The experiment comprised a total of 587 object observations, with an average point cluster size of 3083 points. For a more detailed view, Table 8.4 shows a histogram of the object sizes. It can be seen that most objects (75% of all clusters) contain up to 4000 points.

A subset of the set of objects that have been used for the experiment is shown in Figure 8.2, with multiple views of each object. It can be seen that some objects are segmented suboptimally. There are several kinds of errors in this regard: Some objects were placed in such a way that the view frustrum of the camera clipped parts of the object. Surface reflectivity can lead to holes in the data, as can some kinds of occlusions, *e.g.* in the orange cup. The speckle pattern of the Kinect often leads to jagged edges, and of course there are focus and blur problems that can arise.

The experiment was performed as follows: objects were selected from the pool of available objects and placed on one of two counters. The robot was being driven manually, looking at either one counter, and RoboSherlock was triggered manually when the operators were done setting up the scene. The regions of interest for the semantic-map-based segmentation were set up such that points from walls and sides of furniture (*e.g.* the fridge) were removed.

Object Hypotheses were then found using 3 Annotators detailed in Chapter 4:

1. The URDF-based Region Filter (which also deleted points on walls *etc.*) was responsible for filtering out all points which were not contained in the bounding volumes specified to be regions of interest (the spatial volume above the two kitchen counters).

2. Point cloud cluster extraction: a plane model was fit to the coplanar points within each region, and euclidean clustering was performed on all remaining points to group all points into object clusters based on local connectivity.

3. Small object segmentation: A color model was learned from the table and object pixels and used to segment objects of different appearance than the table background but that were too small or flat to be detected by point cloud cluster extraction.

Figure 8.2.: *Example clusters of some of the objects used in the experiments.*

These hypotheses were then analyzed by the Object Annotators put forth in this thesis, including *i*) normal estimation, *ii*) the Generic PCL Feature Annotator for 3D features, in this case VFH features computed per cluster, *iii*) the Goggles Annotator for image analysis yielding diverse Annotations, *iv*) the ODUAnnotator using the product data from GermanDeli, *v*) shape classification, *vi*) color histograms, *vii*) converters for ROS input or Object Hypothesis types, *viii*) Annotators for similarity measures, tracking and entity resolution, *ix*) as well as the CAS consumer for database storage.

The statistics over resulting annotations is shown in Table 8.5. All objects were successfully segmented and in turn processed. While some annotation types are present for every scene, *e.g*. the `Scene` structure itself, and others for every cluster, *e.g*. the `PointCloud` annotation, most of the interesting results only get added on an as-reconstructed basis, *i.e*. only where relevant.

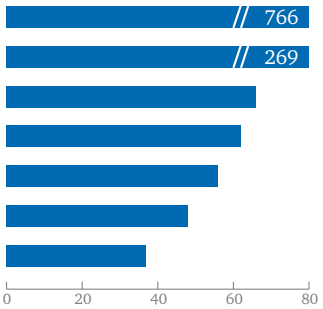
As can be seen, every scene contains potentially interesting information pertaining to CPE execution, as well as all the input data. Each Object Hypothesis furthermore contains all per-object input data, and additionally some reconstructions that can be retrieved independent of which object is represented, such as feature descriptors, classification annotations or color histograms. Other meta data, *e.g*. logo and text recognition, product information or similar images found online, is more sparse, since there are many untextured objects such as cutlery or dishes. However, for the other objects there is quite a lot of additional information.

One of the richest annotations, that of WWW hyperlinks pointing to additional search engine queries, result pages where images were found, *e.g*. online stores or private blogs, or high-resolution images, is very abundant and unfortunately not yet analyzed in detail. This requires a whole new class of algorithms, that can make sense of a set of web pages that were found relevant by a third party.

A little more detail on Goggles Annotations is given in Table 8.6, where we simply list the most commonly occurring categories and titles for individual results. The most consistent bit of information is the logo recognition for “Kellogg’s” and “Ricola”, then similar images found for “Cornflakes” and “Vitalis”, both cereal brands, then Product and Text results, but also images with titles such as “Pack of orange juice” or “Hela Curry Ketchup”. As can be seen, there is an abundance of relevant and meaningful

Table 8.5.: *Histogram over annotations.*

Annotation Type	# occurrence
Scene Annotations	
Scene	103
Logging Information	103
Location Annotation	103
Cluster Similarities Matrix	103
Cluster Distances Matrix	103
High-resolution Camera Image	103
Full Point Cloud	103
Normal Map	103
Annotations for all Clusters	
Cluster	587
Reference Cluster Points	587
Tracking Annotation	587
Standalone Image ROI	587
Image Mask	587
Perceptual Hash	587
Point Cloud	587
Point Indices	587
PCL Features	587
Shape Classification	587
Hue Saturation Histogram	587
Annotations for selected Clusters	
Named WWW Link	766
Goggles Annotation	269
Logo	66
Similar Image + Desc.	62
User Submitted Result + Desc.	56
Text Recognition	48
Product	37



information, and many results appear consistently over multiple observations of an object with varying view point.

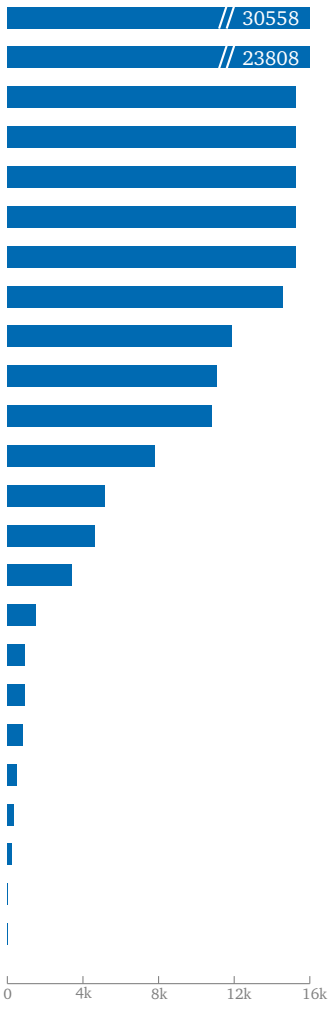
Finally, to whet your appetite for future machine learning applications, we show a frequency table of annotations collected over several months during the development of

Table 8.6.: *Most frequent results from Google Goggles, showing Category and Title fields only.*

Count	Category	Title
27	Logo	Kellogg's
22	Logo	Ricola
21	Similar Image	Cornflakes
14	Product	Vitalis Crunchy Choco with Milk Chocolate, 1.32 lb
12	Similar Image	Vitalis
10	User Submitted Result	Pack of orange juice
9	Product	Kelloggs Crunchy Nut Cornflakes Delivered Worldwide
8	Text	Pfanner
8	Similar Image	Hela Curry Ketchup
8	Logo	Dr Oetker
6	Logo	Corn Flakes
5	User Submitted Result	Pringles Sour Cream & Onion
5	User Submitted Result	Pringles macaroni cheese
5	User Submitted Result	Pfanner Eistee wildkirsch
5	User Submitted Result	Nutri Grain raspberry granola bars
5	Text	Milch
5	Similar Image	Cornflakes Marken
4	User Submitted Result	Pfanner Eistee Pfirsich
4	Text	Voll Milch
4	Text	planner
3	Text	FLAKES
3	Logo	Eggo Waffles
2	User Submitted Result	Reinigungsmittel
2	User Submitted Result	Ranch Pringles
2	User Submitted Result	Prinles
2	User Submitted Result	Pringles sour cream & onion
2	User Submitted Result	ja! Sonnenblumen Margarine
2	User Submitted Result	Ja natürliches Mineralwasser
2	Text	VITALIS
2	Text	iced Tea
2	Similar Image	Ricola
2	Similar Image	Pfanner
2	Similar Image	Müsli
2	Similar Image	Ja Cornflakes
2	Product	Vitalis Crunchy Granola with Honey, 1.32 lb
2	Product	Ricola Sugar Free Spearmint Breath Mints - 12 x 0.87 oz. Box
2	Product	Chocolate cereal classic

Table 8.7.: *Histogram over annotations over several months of* RoboSherlock *development time.*

Annotation Type	# occurrence
Header	30558
Mat	23808
PointCloud	15279
PointIndices	15279
ReferenceClusterPoints	15279
Cluster	15279
StandaloneImageROI	15277
NamedLink	14575
TrackingAnnotation	11888
PclFeatureAnnotation	11097
ClassConfidence	10820
Transform	7808
Scene	5140
GogglesAnnotation	4643
ClassificationAnnotation	3430
Similar Image	1520
User Submitted Result	921
Text	904
Product	816
Logo	478
Logging	329
ODUAnnotation	228
TFLocationAnnotation	33
QR Code	4
(total)	205393



RoboSherlock in Table 8.7. After all, the database offers a convenient persistent storage facility that can collect percepts and annotations potentially over the lifetime of a robot. As can be seen, simply logging everything the robot sees during any execution of a perception process yields hundreds of thousands of annotations that can serve as an interesting dataset for all kinds of machine learning or statistics purposes.

8.3 Queries that can be answered by RoboSherlock

We will now give examples of how to answer certain types of queries by outlining possible processing steps. We have throughout the thesis used verbal representations of these queries, and will in the following paragraphs explain how they map to computational problems in our context.

Consider the seemingly simple question "where is object *X*?". If *X* refers to an object within a database for which machine learning models have been trained, this amounts to a simple retrieval step to find the most recent Object with a `ClassificationAnnotation` containing the respective label. If on the other hand, the object is described using a general name, *e.g.* "Vitalis cereal", one can use the textual retrieval using Optical Character Recognition from *e.g.* Google Goggles detailed in Section 6.6, or by filtering `ProductAnnotations`.

"Where has *X* been before?" can be answered using the same mechanisms as in the last paragraph, followed by examining `TrackingAnnotations` to go back in the object detection history. Alternatively, instead of finding the *most recent* matching object and iterating over tracking results, one could simply find *all* occurrences in the database, likely filtered by the timestamps within a relevant interval to prevent an unmanageable flood of results.

Spatial object retrieval can be either treated semantically or geometrically: One could look for "objects on table `table_kitchen`", using *SegmentationLocationAnnotions* which contain exectly the ROI information. Alternatively, one could retrieve all objects whose centroid in the *TFLocationAnnotation* is within a certain distance or spatial volume.

Other interesting queries possibly need to be more directly connected with the actual Annotator or data source. A search for perishable items can use *ProductAnnotations* containing values for the key "perishability", however we do not have a central taxonomy concept that would assure consistent key names across possible future Annotators which might use different naming conventions than the GermanDeli methods. Alternatively, the use of an ontological reasoning system such as KnowRob [156, 153] could allow mapping of other product information to the concept of perishability or other properties (*e.g.* all milk is perishable). We currently consider this to be a some-

what separate problem as the required information to start these inference processes is currently contained within our database.

Another benefit of such an approach is the possibility of mapping general concepts, *e.g.* milk, to instances of specific brands of milk. It would also allow scaling the processing of this class of queries to include *e.g.* which products contain nuts (if a human is allergic to them), sugar (for diabetics) or alcohol.

We know of no other robot perception system aiming at providing perceptual information about objects of daily use that are as rich as the ones computed by the RoboSherlock application.

Yet, these capabilities only seem to be the tip of the iceberg. The UIMA framework offers sophisticated methods for combining results with confidence. A number of open source tools exist in the UIMA software libraries that enable robots to extract and use knowledge from the world-wide web. Also, UIMA provides a Prolog interface to the CAS data structure that allows for a seemless integration of perception and knowledge processing. Along another dimension the learning infrastructure within the UIMA framework is particularly sophisticated and promising. The potential of this combination of ideas has been impressively demonstrated in the context of the Watson system as it has won the *jeopardy!* challenge. The question whether the impact of these methods, or rather the application of the system level principles put forth in this work, on robot perception is equally strong is uncertain, but we believe that this is not only possible but probable given the positive effects of exactly those strategies in the field of natural language processing and open-domain question answering.

Chapter 9

Related Work

In this chapter, we will attempt to characterize the state of the art as far as it is relevant to the RoboSherlock project and explain how our proposed framework solves some of the issues that arise naturally given the ambitious task at hand.

As argued throughout this thesis, one of the biggest challenges for robot perception is the scaling towards real-world scene and task complexity. We propose that scaling towards real-world complexity should be tackled on a system level where we can combine strengths of experts to complement each other and thus adapt to different problem domains and task contexts, and not at the level of improvement and new development of individual algorithms. Tasks such as object detection, recognition, categorization, localization and reconstruction will have to be performed in an integrated manner. At the same time, these systems will have to be able to deal with textured, form-characterized or translucent objects, and eventually with fluids, deformable objects such as pillows or clothing, cables, or even sliced cucumbers and cracked eggs.

Robot perception research has so far tackled some of these problems in individual algorithms that typically are limited to certain classes of objects, *e.g.* geometric point features for 3D opaque objects [126], appearance based features [83, 6] for textured or other solutions for translucent objects [65, 84].

As has been shown in Horswill [58], higher accuracy, robustness and efficiency are attainable with specialized, targeted algorithms, as long as the assumptions and constraints of this reduced problem domain are met. On the other hand, it has been shown that methods with lower individual accuracy, when combined, can outperform single algorithms in cases where the shortcomings of individual methods are

independent of each other [161, 143]. This leads to the conclusion that combinining methods with respective strengths and weaknesses seems beneficial.

Some systems combine several techniques to increase their generality or robustness, *e.g.* the combination of edge based detection of shapes, tracking and SIFT features for learning and recognizing appearance models in the blocks world robotics vision toolbox [96] leads to reliable detection of 3D objects from single camera images, as long as they are of cuboid or cylindrical shape.

Point based appearance features such as SIFT [83] and SURF [6] are considered in their relative spatial relationships in MOPED [26] to form a state-of-the-art object recognition system that is highly tuned for performance on all levels, from algorithmic improvements to technical or architectural decisions exploiting modern GPU and CPU features. The system is thus able to very robustly and scalably perform object detection and pose estimation for textured and three-dimensional objects.

LINEMOD by Hinterstoisser et al. [56] is a template-based approach for object detection, which in recent work [57] has been combined with pose estimation to allow automatic modeling, detection and tracking of non-textured, three-dimensional objects.

Very interesting research is presented by Hager and Wegbreit [51], which employs a probabilistic transition model to estimate changes between different snapshots of a scene. It is the only system we are aware of that is able to segment and reconstruct objects while maintaining a belief state of hidden objects (“object persistence”) and physical reasoning. The scene model considers support relations and non-intersection of objects to explore a space of explanations for a scene transition. The inferred actions, similar to RoboSherlock, include addition, removal and moving of objects, as well as an additional concept of "perturbing" objects (*i.e.* undergoing small motions). The scalability of the system is unclear however, as experiments only consider three-dimensional objects on a single planar surface with a fixed view point. It is furthermore not clear how to combine the proposed system with more analysis methods reconstructing additional object information.

We believe these systems to be closest in mindset to the ideas we propose in that they attempt to make a real-world robust and functional system. They draw on various

methods and combine them intelligently to solve difficult problems. However they fall short on generality, *i.e.* the end result is in itself a specialized expert.

With the increased availability of middleware frameworks such as ROS [119], and algorithm repositories such as PCL [130], OpenCV [19] (see also Section 9.5), we believe the time ripe to address these issues of method combination and cooperative systems which have received surprisingly little attention in our field.

The basic idea of method composition to improve a solution has been employed in the machine learning community for some time [104, 112, 123]. This might be rooted in the fact that classifiers are in essence mathematical constructs that share many structural similarities, which makes them easier to integrate & combine than more diverse classes of algorithms.

A taxonomy of ensemble methods has been given by Jain et al. [61], in which he differentiates schemes based on the architecture: *i*) parallel (possibly gated), *ii*) cascading (or serial), *iii*) hierarchical, or *iv*) a combination of these. It is straightforward to see that RoboSherlock, by the use of Annotator aggregation, can support any of these principles, and using more complex flow controllers, can do so even dynamically. Jain et al. [61] also look at whether the method combiner is trainable, and whether it is adaptive, *i.e.* whether contributions of individual experts are weighted depending on input pattern. This is both in theory possible in RoboSherlock, and future investigations along these lines will prove intriguing.

Rokach [123] also mentions four central aspects for successful crowd decisions: diversity of opinion, independence, decentralization and aggregation. It is clear that in our context, this requires a diversity of experts that can draw on local, independent knowledge, but also a way to combine their outputs in case of disagreements.

On a rough level, three popular classifier combination meta-algorithms are bagging [21], boosting [41] and stacking [174]. Bagging is based on the idea of combining multiple classifiers trained on subsets of the input data, while boosting focuses on creating strong learners using a set of weak learners, *i.e.* learners that are marginally better than random decisions. Stacking trains levels of classifiers, where later levels are trained on the outputs of earlier ones. In our current implementation, RoboSherlock is most similar to a bagging approach in that we have individual experts that operate on sometimes different kinds of input data, and in the future we will have

to explore these other methods, especially stacking, that make it possible to learn on experts' opinions independently of their respective input data.

9.1 Semantic Mapping

Many, if not all robots need to maintain some representation of the world to carry out their tasks, and a good overview of different approaches mainly related to mapping and navigation is presented in Burgard and Hebert [22]. Maps are used for estimating the agent's position in the environment, to test the reachability of destinations and to compute navigation plans [160]. Among the most common methods are simultaneous localization and mapping (SLAM), usually based on two-dimensional grid occupancy maps [158] or sparser feature maps [77, 94].

Due to the considerable memory requirements, these approaches reach a limit when modeling large areas at high resolutions. To remedy this problem, topological maps have been proposed to deal with working environments that extend over a large scale where the aforementioned metrical methods often accumulate errors or would be too coarse-grained. These models might contain landmarks as proposed by Montiel et al. [95] or represent the topological structure of the environment as reported by Kortenkamp and Weymouth [68].

Since these methods encode space in a 2D map, which limits the kind of environments you can model with sufficient detail, there have been efforts to extend the principle to allow mapping in 3D [102]. These methods need different map representations, such as the octree based map format described in Wurm et al. [176].

There are mapping systems that model indoor (see Rusu et al. [132]), outdoor (see Rusu et al. [133]), underwater (see Smith et al. [150]), and aerial environments as in Shen et al. [145]. These maps are usually learned for an initially unknown environment using *e.g.* particle filters, and can be used directly to localize the robot accurately within the emerging map.

Some approaches strive to combine the strengths of geometrical and topological mapping methods in a hybrid fashion [70] or by adding certain point features [142], texture or other landmarks [22].

A *semantic map* enriches these navigation maps by annotating places of interest [101], storing objects that the agent can manipulate, and reconstructing these objects in order to get better models that can be used for grasping or redetection. In previous work, we have presented the geometrical and functional reasoning steps that allow us to interpret a 3D scene in near-real time, leading to a very rich description of household environments.

Some authors approach semantic mapping by marking positions of detected objects overlayed on their 2D navigation map [169], however this only serves for rather limited application scenarios, as there is no notion of time or object identity, only object class associations.

In [131], we presented how an autonomous mobile robot can acquire an environment object model of a kitchen. Starting from a set of point clouds acquired with a tilting 3D laser scanner, we presented how to automatically compute a mesh representation of the environment, where furniture such as cupboards and drawers in a kitchen are modeled as cuboids with front doors and handles, and tables and shelves as horizontal surfaces, possibly supporting objects of interest, such as dishes or food products.

In Vasudevan et al. [169], the authors present a system that is designed around similar ideas as put forth in this work. They represent the robot's surroundings in a hierarchical map encoding places and connections between them (such as doors) with local probabilistic object graphs. However, the authors do not tackle the problem of doing this for an unknown universe of objects, nor do they explicitly model time in order to be able to answer queries about a former world state from memory. The first limitation effectively eliminates the problem of object identity resolution, while the second one reduces the scope of the world belief state to model the current state only.

A research area that has received surprisingly little attention is the automatic acquisition of environment maps that enable robots to perform human-scale manipulation tasks, such as setting a table, preparing a meal, and cleaning up. In the most related work to RoboSherlock, Pronobis [116] proposes a system for building an internal representations of space for robots to act in human living environments by memoriz-

ing a sparse set of objects and their spatio-temporal relations, which allows powerful task queries.

Wuenstel and Moratz [175] propose a graph representation to detect chairs, but the relation descriptions are manually estimated, and thus it is unclear whether the proposed method scales well. Posner et al. [114] use probabilistic graphical models such as Markov Random Fields (MRF) to label planar patches in outdoor urban datasets. They use appearance-based classification locally and an MRF to model scene-wide relationships. Their work is based on that of Anguelov et al. [3] and Triebel et al. [165], which define point-based 3D descriptors and classify them with respect to object classes (such as chairs, tables, screens, fans, and trash cans in the former, and wires, poles, ground, and scatter in the latter), however on a low abstraction level.

An EM-based algorithm for learning 3D models of indoor environments is presented by Liu et al. [82]. The maps are created using mobile robots equipped with laser range finders, but they do not include any semantic information. Grau [50] uses a stereoscopic camera system and a knowledge base in the form of a semantic net to create 3D models of outdoor environments. A hybrid representation including spatial and semantic aspects is proposed by Galindo et al. [44] for an indoor environment, but their approach needs further investigation.

In Óscar Martínez Mozos et al. [105, 106], places are semantically labeled into doorways, kitchens, corridors and rooms. The advantages of these representations are straightforward: they keep the computational cost sufficiently low and base their localization and pose estimation on the well-known 2D Simultaneous Localization and Mapping (SLAM) problem, while the problem of place labeling is solved through the usage of feature descriptors and machine learning. However, by reducing the dimensionality of the mapping to 2D, most of the world geometry neccessary for robot manipulation is lost.

With few exceptions, in particular in the area of cognitive mapping as in Vasudevan et al. [169] and the work of Modayil and Kuipers [93], but also including the more recent work by Triebel et al. [164], maps do not represent objects relevant to any robot task apart from navigation.

For modeling the static part of the environment, Nüchter and Hertzberg [101] classify 3D sensor data from a laser sensor into walls, floor, ceiling, and doors based on an-

gular thresholds. Static objects including standing humans are detected in two steps, where a hypothesis is created from depth images using appearance-based methods, and 3D matching to a model is then performed to evaluate the classification. However, this requires 2D models for all poses of each object to be detected, as well as the complete 3D models. The same holds true for postures in the case of humans. [111] proposes a textured surface reconstruction of static indoor environments using a Poisson formulation to generate the signed distance function and Marching cubes to generate the final iso-surface. Reconstruction of the texture is done through blending. The approach however does not deal with everyday objects or the kind of semantic information that would directly enable mobile manipulation tasks.

9.2 Segmentation and Object Reconstruction

We have introduced our definition of objects of daily use, which are to be manipulated by the robot, in Section 2.2. We will now present recent work in segmentation, object detection, categorization, localization, and reconstruction for robots, with a focus on work intended for grasping, as this is one of the ultimate goals for robotics vision systems. Since in many approaches, segmentation and object detection go hand in hand, we will look at them in context over the following paragraphs.

We have by now mentioned many problems where the segmentation of objects is a crucial step in the process. In our context, we defined segmentation as a process that identifies a region in space, in a point cloud or camera image, that could potentially represent an object. Many might argue that it is neccessary to *e.g.* perform under-segmentation and part-based recognition to be able to generalize to unknown objects and clutter.

This seems to contradict our approach, but we believe it is possible to unite the two ideas. At some point, a decision to group a set of superpixels is made by these segmentation algorithms to form a new object hypothesis, and it is at this point that the `ObjectHypothesis` generator becomes visible. If that segmentation process has generated object information at this point, it can be added to the object hypothesis before it continues to be analyzed by the remaining object Annotators.

An excellent survey of existing methods is given in [69], and more related work can also be found in our previous literature, *e.g.* Marton et al. [87] for object perception and Mozos et al. [97] for environment modeling.

A central aspect of object detection is separation of foreground and background. Some have approached the problem by collecting very large training sets containing labeled scenes, and using data mining techniques using simple or complex features to generate feature classifiers for fore- and background [118, 179, 134]. These methods generally allow low variance in scale and texture.

Another segmentation approach is using interactive techniques to actively explore the environment and segment objects by moving them to generate 3D shape information, as done by *e.g.* Welke et al. [172] and Feldman and Weinshall [36].

The scale variance can be reduced significantly by prior segmentation. [135] try to reconstruct the 3D world from a single view in order to improve the segmentation of objects. Another approach is to actively explore the environment and segment objects using motion to generate 3D shape information, as done by *e.g.* [172] and [36].

To improve the segmentation, Lai and Fox [72] take a randomized approach by classifying a "soup of segments" generated by different combinations of over-segmented patches. They also explore different domain adaptation techniques in order to incorporate synthetic data in training their classifier.

Richtsfeld and Vincze [122] present a computer vision system which segments objects on a table and constructs triangular meshes for grasp analysis. Other approaches have employed model fitting of geometric primitives like boxes, spheres, cylinders or cones as in Miller and Allen [92], or superquadrics as in Biegelbauer and Vincze [12] and Zhang et al. [181]. These approaches require a-priori object databases with decomposed models to work, which we do not consider scalable or practical.

A method that skips the segmentation problem and directly attempts to identify grasping points in stereo images is given by Saxena et al. [136]. Their approach works reliably only to the extent provided by the training data, and as mentioned ignores object segmentation and the correspondences between objects and the identified grasp points.

Another interesting primitive-based approach using Random Sample Consensus (RANSAC) is presented by Schnabel et al. [139]. Several optimizations such as localized sampling and the use of octrees have been used to allow efficient model fitting in noisy point clouds.

In purely computer-vision-based approaches, features like the ones described by [83] or [78] are used to find matches between parts of a scene and a database of object images. One problem with these kinds of approaches is the requirement for a comprehensive object database, and the lack of 3D information, which can easily cause mistakes and false positives (*e.g.*, a picture of a milk bottle on a cereal box can be taken for a milk bottle).

Some of the solutions adopted involve the off-line creation of complete 3D models for the targeted objects and identifying feature spaces to match the partial views with models in the database, as presented by [24]. Another approach to obtain 3D information directly from camera images is to project CAD models from a database onto the image and search for matches in the edge domain, as in *e.g.* [168]. While this is a more direct method, it still requires a database of CAD models.

Symmetries have been exploited successfully in object segmentation by Thrun and Wegbreit [159] which attempts to identify and estimate various symmetry relations in point clouds. They show interesting results from single scans, and demonstrate the possibility to complete back sides and occluded areas by projecting points via the locally detected symmetry axis or plane. In Blodow et al. [15], we reported our method for detecting and modeling rotationally symmetric objects. The core idea is to estimate rotation axes in pointclouds using a RANSAC approach and surface normal information. Polynomial contours are then fitted for points around these axes. To improve the fitting results, we also hypothesized missing or occluded surfaces and reasoned over model plausibility to reject models that contradicted our observations of free space.

9.3 Symbolic Knowledge

For performing high-level tasks, robots need access to large amounts of semantic information from the 3D map and the objects contained in the environment, like the

locations, identities and properties of objects. Very few systems exist that offer this kind of deep semantic environment information.

At the highest level of abstraction, one considers the recognized objects, whose semantic meaning is only implicitly represented: Humans immediately associate various properties with something called a “cupboard”, while robots usually do not have this kind of common sense knowledge. Without an explicit knowledge representation, different robots or even different parts of the same robot may have a very different notion of an object.

Okada et al. [103] present a system that supports various semantic object properties, such as grasping points, container handles or pouring openings of bottles. These unfortunately are not based on a hierarchical abstract knowledge representation, but on hard coded information.

Bordering on mapping topics, Galindo et al. [45] combine a spatical hierarchy of maps that is globally topological and locally metric in nature with a conceptual semantic hierarchy of object and room properties. While similar in concept to our approach, the spatial descriptions are much coarser, and the object information is far simpler than in RoboSherlock.

For highly complex domains, in which relations between a variable number of entities and uncertainty need to be modeled declaratively, a representation formalism is required that can combine statistical with relational components. Concrete entities are treated abstractly to allow a compact representation of general principles about the relevant aspects of the real world. Getoor and Taskar [49] and [32] present a number of statistical learning relational formalisms, which have a wide range of possible applications, such as collective classification ([100]), link prediction ([152]) and object identification ([149]).

In particular, statistical relational learning methods can represent full-joint distributions over logical propositions about a changing set of entities in a concise, declarative manner, and they provide a fully integrated framework for learning and inference. Therefore, they are ideally suited to the representation of the probabilistic components of robot belief states as we envision them, since we need to consider spatial relations between objects in the environment, their attributes changing over time, their relevance to activities and the effect these activities may have upon them. In our

previous work [14], we used statistical relational learning to model the problem of object identity resolution, *i.e*. the data association problem given partial descriptions of objects. The extensions of this work have been presented in Chapter 6.

Andrienko et al. [2] use GPS data from a car to learn concepts like the working place, the living place, typical navigation routes and other spatio-temporal behavior patterns, similar to the functionality now offered by Google Now. While in their case, the data mining tasks are performed by human experts using visual analytics methods, other researchers, such as [81, 80], perform some of these learning tasks using probabilistic learning methods, *e.g*. hierarchical Conditional Random Fields. Acquiring such models is also investigated in the pervasive systems community, where *e.g*. [110, 75] and [60] learn models of daily activities from ubiquitous sensor networks.

9.4 Entity Resolution

The problems of object identity resolution and the maintenance of an object memory system have garnered surprisingly little attention in the research community.

The entity resolution problem has been addressed in several unrelated fields. In information integration, Fellegi and Sunter [37] have formulated a classification problem to decide whether a pair of data records describe the same entity. Singla and Domingos [149] have formulated this as a link predicition problem in statistical relational models.

In the field of computer vision, Nelson and Green [99] have proposed a memory assistance tool that employs camera surveillance to track objects to help people with slight to moderate memory loss. Their system however requires unique objects and can thus not cope with the more complex domain we're faced with, and assumes full observability of locations and activities. Entity resolution is not addressed.

Multi-hypothesis object tracking deals with the spatio-temporal nature of the problem, as described by Cox and Hingorani [30], Schmitt et al. [138]. Elfring et al. [35] also addressed this problem using a technique they called multiple hypothesis anchoring. In these cases, multiple hypotheses concerning object trajectories are con-

sidered in parallel, and a motion model allows prediction of future states. This is in theory more general than our approach, *e.g.* in more dynamic scenes, but their object information is rather limited and the approach is verified with a very low number of objects.

Schulz et al. [140] use statistical data association in the context of tracking multiple people using 2D laser scanners, but the information they collect about the tracked objects (*i.e.* humans) is limited to positional data.

The approach presented in this thesis is different from these methods (and other *e.g.* feature-based tracking systems in computer vision) since we do not consider scene dynamics. Our goal, establishing object identity, is not neccessarily concerned with frame-to-frame changes, but with long-term data collection. This prohibits the use of motion models and requires stronger reliance on object properties to establish equivalence. The inclusion of logical and probabilistic constraints is hence the decisive factor for the successful operation of our Annotator.

9.5 Related Open Source Software

The Intelligent Autonomous Systems group already has noteworthy contributions in the areas that are significant to the realization of RoboSherlock. We will shortly present these and other 3rd-party projects as follows:

- UIMA[1] and its C++ interface, UIMACPP, have been the core of the IBM Watson system [39]. First developed by IBM, it is now maintained by the Apache foundation and has seen great success in the fields of natural language processing.
- The Point Cloud Library[2] (PCL) is a large-scale international open-source 3D data processing framework. Initially founded by Radu Rusu, Nico Blodow and Zoltan-Csaba Marton, it has grown to several hundreds contributors from institutes around the world and is now supported by the Open Perception Foundation[3] and numerous industrial partners.

[1] http://uima.apache.org/
[2] http://www.pointclouds.org
[3] http://www.openperception.org

- As part of our ongoing collaboration with Willow Garage Inc. [4] and our involvement in the Robot Operating System[5] (ROS) community, we published and maintain multiple open-source packages in the areas of *e.g.* knowledge representation (KNOWROB [154], RoboEarth[6]) and 3D mapping (*e.g.* PCL), which have been adopted by various international robotics groups in the community.
- OpenCV[7] is an open-source computer vision library originally developed by Intel, and now supported by Willow Garage and Itseez. With a community of tens of thousands of users, it includes a large array of computer vision and machine learning algorithms.
- MongoDB is an open-source document-oriented NoSQL data base that has been adopted as a backend solution by various online service.
- protobuf[8], is an open-source library for using protocol buffers, Google's data interchange format.
- phash[9][180], is an open-source perceptual hashing library.
- We indirectly use the open-source Bullet physics engine[10] through ROS's *tf* transformation library, and the linear algebra library Eigen [11] via PCL.

[4]http://www.willowgarage.com
[5]http://www.ros.org
[6]http://www.roboearth.com
[7]http://opencv.org
[8]http://code.google.com/p/protobuf
[9]http://www.phash.org
[10]http://www.bulletphysics.org
[11]http://eigen.tuxfamily.org

Chapter 10

Conclusions

In this thesis, we have argued for a paradigm change in computer perception for robotic household assistants. Where currently the main focus of the computer vision community lies on increasing the performance, robustness, efficacy and efficiency of singular algorithms, we outlined a system that draws on the collective cognitive and perceptual processing power of an ensemble-of-experts approach in order to achieve the scalability towards real-life scenarios.

We are convinced that the next big leap for powerful computer vision applications happens on a system level using ensemble-of-experts approaches, which allow us to tackle hard problems more successfully, such as:

- Dealing with *multimodal data*, not just from 2D cameras and 3D range scanners, but including diverse sensors such as tactile or force-sensing transducers, imaging sensors in non-visible spectra, temperature sensors and so forth;
- Incorporating *context* into the process to shape algorithm selection and parameterization, as well as defining priors over what to expect in a given scenario. While the experts as a basic building blocks can stay the same, parametrization and reconfiguration allows the adaptation of such systems to various applications;
- An important part of a perception system is a mapping of visual stimuli to *semantic concepts*. While this can to a degree be achieved by large, potentially manually annotated databases, we are confident that semantics must be incorporated into data processing by a strong interconnection of knowledge reason-

ing and perception methods to perform at the required scale. The data model and type system in RoboSherlock offers these possibilities;

- This leads to the relationship between *decision-making* and perception, as they are interlocked in both ways: higher-level reasoning methods define the task context in which perception has to perform, while at the same time relying on perception to provide the information and capabilities upon which to base decisions. An ensemble-of-experts system such as outlined in this thesis can offer the required functionality, adaptability and interconnection with other data-processing-intensive processes such as knowledge reasoning or task executives.
- On top of offering all the required functionality, systems like RoboSherlock need to be able to provide real-time *performance* for many tasks.

It seems obvious that for systems on this scale, one of the biggest problems will be *managing complexity*. The use of modular, reusable components together with execution models such as flow controllers that can form anything from trivial to highly complex processing engines offers a way of achieving this goal. This has been proven in the natural language processing field with IBM Watson, demonstrated by winning the *jeopardy!* challenge against prior human champions of the *jeopardy!* TV quiz show. Watson provided the capabilities to answer very open-domain questions with high accuracy using the software framework which is also the foundation of RoboSherlock.

It is important to note that while we developed RoboSherlock with a robotic problem domain in mind, all of the above points can be made for various application scenarios. As an example, consider an autonomous car that needs to be able to segment and annotate the environment around it, detect obstacles, terrain, lane markings, running children and so on. Night driving might require a completely different combination of experts than day mode, similarly for highway driving compared to inner city traffic or even parking. Obviously, real-time performance is of paramount important for many tasks in this context, such as person avoidance, whereas other tasks can be assigned lower priorities, such as *e.g.* reconstructing a holistic 3D model of all streets that are frequented by a certain manufacturer's cars.

While we could only illuminate a subset of the problems that we will need to solve in order to create complete perception systems as envisioned, this thesis hopefully

provided sufficient insight into the matter to make it very plausible that the chosen approach can be successful.

To recapitulate, we stated three principles in Chapter 2 that we followed during the development of RoboSherlock and this thesis, namely *i*) the treatment of perception as an unstructured information processing problem; *ii*) ensembles of experts create and annotate object hypotheses; and *iii*) information interpretation through fusion and selection of hypotheses and annotations to reach a consistent and semantically meaningful interpretation of perceived scenes.

In the same chapter, we also gave a system overview of the following chapters. We introduced a novel type system for object detection and analysis in Chapter 3, where we also explained the interoperability between RoboSherlock and ROS, PCL and the MongoDB database in terms of data formats, and our additions to UIMA to provide more convenient, compact and error-safe interfaces for accessing and manipulating data.

In Chapter 4, we presented our contributions to the problem of detecting objects, or more specifically generating hypotheses where objects could be. We employ GPU-accelerated methods as well as a static, semantic 3D map of the environment to quickly and semantically segment 3D and camera input data into irrelevant regions, regions of known structure (according to the static map) and objects of interest. These object hypotheses can reference or contain the corresponding point clusters, image regions or other spatial volumes, and can be translated automatically between the various representations, where applicable. For example, to obtain a high-resolution camera image for a 3D point cloud cluster, we can determine the corresponding image region through reprojection of the 3D points, using the calibration and proprioceptive kinematics information of the robot.

For segmenting flat objects on planar supporting surfaces, we introduced a high-fidelity segmentation method that uses conservative background-foreground estimation from noisy depth data to train a per-frame classifier that can make use of a high-resolution camera to successfully segment objects with a small height. The results of these algorithms are descriptions of object hypotheses that are rooted in multimodal sensor data, which allows subsequent Annotators to rely on a multitude of evidence for analysis.

We proceeded in Chapter 5 with the description of various object Annotators that strive to create more informative interpretations of the object observations. Seeing as feature descriptors are a widely used and very powerful foundation for classification, we created an Annotator that is able to compute any 3D or 2D feature that is available within PCL. This amounts to a very wide collection of 3D feature descriptors that can be used in RoboSherlock. We shortly described our wrapper around the classification framework by Marton et al. [89, 86], which can use different features, classifiers, training methods and distance functions to offer very flexible and powerful machine learning capabilities.

We also explored the usage of online resources, such as scraping of online stores to acquire hierarchically organized, structured product information with a wide array of metadata that can be used with image search to annotate camera images with this extracted metadata. Additionally, we reverse engineered Google Goggles to create a compatible client for RoboSherlock. This means the robot can use the Goggles service to visually search the world wide web by image, from images to online shops to private blogs or other user content, offering various kinds of rich and meaningful information.

Chapter 6 detailed our proposed method to solve the problem of object identity resolution. We defined this to include the problems of determining similarities between object observations and assumptions about possible actions that can be performed on objects by humans, on which we can build using state-of-the-art statistical relational models to infer conclusions pertaining to the following questions: *i*) Which current observations is referring to the same object as which past observation? *ii*) Which of these objects have remained unchanged, which have been moved, newly added, or removed? Our solution to this problem of "low-observability-tracking" showed very good results even for larger problems and in the presence of noise. This lies the corner stone for applications such as maintaining a current belief state about which objects are present, when did they get there, where had they been before, tracking objects over long periods of time, establishing object lifecycles or typical usage patterns and so forth.

In the same chapter, we outlined our memory strategy, using the Common Analysis Structure for short-term and a MongoDB data base as a long-term storage solution, with automatic conversions and interfaces, supported by a versatile web page able to

visualize present and past percepts and analysis results, offering search capabilities and Annotator performance inspection.

Another piece of the puzzle, the creation of semantic 3D environment models, is detailed in Chapter 7. We presented an autonomous exploration strategy that is able to determine unmapped areas in indoor environments and generate robot scan poses to acquire the missing data and register it with previous scans. These 3D point cloud models are then analyzed to detect geometric features such as floor and walls, horizontal support structures, furniture fronts, door and drawer handles and so forth.

The gap between geometric and functional models is bridged using an interactive estimation step developed by colleagues whereby the robot autonomously verifies the validity of handle hypotheses by trying to compliantly open them, and creating articulation models for hinged doors and drawer slides.

While the previously mentioned algorithms have been partly evaluated in their respective topical chapters, an overall system evaluation has been performed in Chapter 8. We could show that RoboSherlock as a framework and our instantiated system created within RoboSherlock can create rich object information using the various sources and Annotators, both on-robot and off-robot (*e.g.* using world wide web resources). We showed exemplary results for the analysis of a single cluster to give a feel for what kinds of information can be generated from as little as a small, Kinect-acquired RGB-D image region, and then gave a statistical analysis of frequencies and content of annotation results for a large experiment using several hundred object observations in a typical household environment.

10.1 Future Work

The future possibilities of RoboSherlock are virtually endless. As is natural for a system of such ambitious scope, extensions and improvements can be made along several dimensions.

Obvious additions to an ensemble-of-experts approach are new experts. More specifically, and staying within the application scenario of this thesis, we could envision the inclusion of different *segmentation* or object detection methods, *e.g.* segmenta-

tion methods models that attempt to capture the "object-ness" of local point cloud or image regions, or attention methods using some form of saliency detection. Other promising approaches include approaches utilizing conditional random fields (CRF) such as *e.g.* TextonBoost [147] or Depth-Adaptive Superpixel segmentation [171], both of which have already been incorporated into RoboSherlock in a very early alpha state but with promosing potential.

In the area of *object annotations*, many singular algorithms exist in the literature that could be adapted or included as RoboSherlock Annotators. We believe that especially 2D and 2.5D image-based methods are currently underrepresented, and methods such as LINEMOD [56] or Histograms of Oriented Gradients (HOG, [31]) would be a welcome addition. Note that some of these Annotators would work on the whole image, which would not fit into our separation of hypothesis generation and object annotation. This is no problem, as they would incorporate both of these functionalities in a single Annotator, creating an object hypothesis pre-annotated with their respective meta-information. Of course, care must be taken in merging multiple overlapping but separate object hypotheses and their annotations.

The third kind of Annotators as characterized in this work, *information fusion* experts, can and must be improved as well as there are several topics that need to be addressed in the future in this regard. Most importantly, we would like to see model completion approaches incorporated, where partial scans from different points of view and in time are merged into a complete 3D model of objects that can be used *e.g.* in physics simulations, grasp analysis *etc*. While we did address the association problem of observations in terms of detecting if multiple object observations pertain to the same object, we did not specifically address how to *merge* and *condense* multiple descriptions. In a sense, the problem of "remembering" the right information or statements is a *selection* process that could just as well be formulated as a "forgetting" problem, where we attempt to identify wrong or irrelevant information.

A whole set of interesting Annotators dealing with human segmentation (*e.g.* using HOG), face detection [167, 8, 27] and tracking, human pose detection [146] and tracking [4] and gesture or action recognition [113] could be explored in our setting. This would alleviate current problems such as a hand reaching into the spatial volumes of interest getting misdetected as an object and allow the association of in-

dividual people, their actions and involved objects to increase the inference quality and allow inferring relations such as “Michael’s cup”.

However, as the pool of available Annotators increases, there is bound to be contradicting information due to various sources of error due to illuminations, noise, or imperfect experts, *e.g*. misclassifications of a machine learning classifier, which will have to be dealt with intelligently.

This leads directly to extensions to RoboSherlock that relate to control flow (as opposed to individual experts). It would be beneficial to express completely new problems using our framework, such as *e.g*. optimal algorithm selection for a given object retrieval task, applications of machine learning to optimize data flow and Analysis Engine execution, or continuous, life-long 3D mapping that can deal with *e.g*. furniture being moved.

Some of these problems could be solved currently by creating action templates, *i.e*. Component Processing Engine descriptors that can be instantiated and parameterized by a higher-level reasoning process to create one-off tasks that need to be solved to be able to execute a given plan. Examples include looking for a specific object, or triggering re-evaluation of a certain object hypothesis.

In fact, we believe that as one realizes the information depth and density that can be generated using a system as ambitious and cross-connected as ours, one can think of more and more application scenarios and problem formulations to tackle. In fact, as more people can devote time to extending and improving it, new paths will unfold, new ways of thinking can evolve, and whole families of problems will present themselves to be explored and hopefully solved, using new experts, new pipelines, new flow controllers, more spin-off tasks, reflection and introspection, machine learning applications and richer information extraction.

However, to achieve this level of functionality, widespread evaluation or adoption of RoboSherlock is required to encourage the formation of a vibrant research community centered around the framework, to get enough skilled researchers and programmers to use, modify and improve the system, and to contribute their unique view on new problem formulations. We believe that an essential requirement for allowing such a community to form and thrive is the clear definition of a set of difficult percep-

tion problems and intriguing challenges that can not be solved without employing a systems approach.

List of Prior Publications

The work presented in this document is partly based on prior publications. Sections of this work that drew upon content from prior publications cited the respective publications where appropriate. A complete list of publications that were (co-)authored during my research as a doctoral candidate is provided below.

Journal Articles

Michael Beetz, Nico Blodow, Ferenc Balint-Benczedi, Zoltan-Csaba Marton, Daniel Nyga, Florian Seidel, and Christian Kerl. RoboSherlock: Unstructured Information Processing for Robot Perception. *International Journal of Robotics Research*, 2014. (submitted for review).

Michael Beetz, Dominik Jain, Lorenz Mösenlechner, Moritz Tenorth, Lars Kunze, Nico Blodow, and Dejan Pangercic. Cognition-Enabled Autonomous Robot Control for the Realization of Home Chore Task Intelligence. *Proceedings of the IEEE, Special Issue on Quality of Life Technology*, 100(8):2454–2471, 2012.

Zoltan-Csaba Marton, Ferenc Balint-Benczedi, Oscar Martinez, Nico Blodow, Asako Kanezaki, Lucian Cosmin Goron, Dejan Pangercic, and Michael Beetz. Part-based Geometric Categorization and Object Reconstruction in Cluttered Table-Top Scenes. *Journal of Intelligent and Robotic Systems*, 2013.

Zoltan-Csaba Marton, Dejan Pangercic, Nico Blodow, and Michael Beetz. Combined 2D-3D Categorization and Classification for Multimodal Perception Systems. *International Journal of Robotics Research*, 30(11):1378–1402, 2011.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge in Robotics)*, 56(11): 927–941, 2008.

Conference and Workshop Articles

Michael Beetz, Freek Stulp, Bernd Radig, Jan Bandouch, Nico Blodow, Mihai Dolha, Andreas Fedrizzi, Dominik Jain, Uli Klank, Ingo Kresse, Alexis Maldonado, Zoltan Marton, Lorenz Mösenlechner, Federico Ruiz, Radu Bogdan Rusu, and Moritz Tenorth. The Assistive Kitchen – A Demonstration Scenario for Cognitive Technical Systems. In *IEEE 17th International Symposium on Robot and Human Interactive Communication (RO-MAN), Muenchen, Germany*, pages 1–8, 2008. Invited paper.

Nico Blodow, Lucian Cosmin Goron, Zoltan-Csaba Marton, Dejan Pangercic, Thomas Rühr, Moritz Tenorth, and Michael Beetz. Autonomous Semantic Mapping for Robots Performing Everyday Manipulation Tasks in Kitchen Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

Nico Blodow, Dominik Jain, Zoltan-Csaba Marton, and Michael Beetz. Perception and Probabilistic Anchoring for Dynamic World State Logging. In *10th IEEE-RAS International Conference on Humanoid Robots*, pages 160–166, 2010.

Nico Blodow, Radu Bogdan Rusu, Zoltan Csaba Marton, and Michael Beetz. Partial View Modeling and Validation in 3D Laser Scans for Grasping. In *9th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2009.

Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach. Real-time Compression of Point Cloud Streams. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

Zoltan-Csaba Marton, Ferenc Balint-Benczedi, Nico Blodow, Lucian Cosmin Goron, and Michael Beetz. Object Categorization in Clutter using Additive Features and Hashing of Part-graph Descriptors. In *Proceedings of Spatial Cognition (SC)*, 2012.

Zoltan-Csaba Marton, Dejan Pangercic, Nico Blodow, Jonathan Kleinehellefort, and Michael Beetz. General 3D Modelling of Novel Objects from a Single View. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

Radu Bogdan Rusu, Jan Bandouch, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Action Recognition in Intelligent Environments using Point Cloud Features Extracted from Silhouette Sequences. In *IEEE 17th International Symposium on Robot and Human Interactive Communication (RO-MAN), Muenchen, Germany*, 2008.

Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, May 12-17*, 2009.

Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning Point Cloud Views using Persistent Feature Histograms. In *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26*, 2008.

Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Close-range Scene Segmentation and Reconstruction of 3D Point Cloud Maps for Mobile Manipulation in Human Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

Radu Bogdan Rusu, Nico Blodow, Zoltan-Csaba Marton, Alina Soos, and Michael Beetz. Towards 3D Object Maps for Autonomous Household Robots. In *Proceedings of the 20th IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2007.

Radu Bogdan Rusu, Andreas Holzbach, Nico Blodow, and Michael Beetz. Fast Geometric Point Labeling using Conditional Random Fields. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning Informative Point Classes for the Acquisition of Object Model Maps. In *Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV), Hanoi, Vietnam, December 17-20*, 2008.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent Point Feature Histograms for 3D Point Clouds. In *Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS-10), Baden-Baden, Germany*, 2008.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Emanuel Dolha, and Michael Beetz. Functional Object Mapping of Kitchen Environments. In *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26*, 2008.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Andreas Holzbach, and Michael Beetz. Model-based and Learned Semantic Object Labeling in 3D Point Cloud Maps of Kitchen Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

Workshop Articles

Aitor Aldoma, Markus Vincze, Nico Blodow, David Gossow, Suat Gedikli, Radu Bogdan Rusu, and Gary R. Bradski. CAD-model recognition and 6DOF pose estimation using 3D cues. In *IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011*, pages 585–592, 2011.

Michael Beetz, Nico Blodow, Ulrich Klank, Zoltan Csaba Marton, Dejan Pangercic, and Radu Bogdan 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*, 2009. Invited paper.

Nico Blodow, Zoltan-Csaba Marton, Dejan Pangercic, and Michael Beetz. Making Sense of 3D Data. In *Robotics: Science and Systems Conference (RSS), Workshop on Strategies and Evaluation for Mobile Manipulation in Household Environments*, 2010.

Nico Blodow, Zoltan-Csaba Marton, Dejan Pangercic, Thomas Rühr, Moritz Tenorth, and Michael Beetz. Inferring Generalized Pick-and-Place Tasks from Pointing Gestures. In *IEEE International Conference on Robotics and Automation (ICRA), Workshop on Semantic Perception, Mapping and Exploration*, 2011.

Zoltan-Csaba Marton, Nico Blodow, and Michael Beetz. Advantages of Spatial-temporal Object Maps for Service Robotics. In *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2011.

Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, Moritz Tenorth, Radu Bogdan Rusu, and Michael Beetz. Autonomous Mapping of Kitchen Environments and Applications. In *Proceedings of the 1st International Workshop on Cognition for Technical Systems, Munich, Germany, 6-8 October*, 2008.

Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Interpretation of Urban Scenes based on Geometric Features. In *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on 3D Mapping, Nice, France, September 26*, 2008. Invited paper.

Appendix A

Goggles Protocol Buffer Message Definitions

This Appendix contains the actual Protocol Buffer message definitions that were obtained by reverse-engineering the Google Goggles service and underlying communications protocol.

We will begin with the definitions for the `Request` message, and continue with the `Response` on page 210.

Listing A.1 Goggles Request message format definition

```
package goggles;

// Image is represented as a JPEG byte stream,
// wrapped into a 2-level deep struct hierarchy.
message Image {
  optional bytes image_bytes = 1;
  optional int32 unknown_num_1 = 2;
  optional int32 unknown_num_2 = 3;
}
message UnknownStruct1 {
  optional int32 unknown_val = 1;
}
message WrappedImage {
  optional Image image = 1;
  optional UnknownStruct1 unknown_struct = 20054927;
}
```

```
17
18 // Unknown structures, can generally be omitted
19 message UnkonwnStruct2a {
20   optional fixed32 val1 = 2;
21   optional int64 val2 = 3;
22   optional fixed32 val3 = 4;
23 }
24 message UnkownStruct2 {
25   optional UnkonwnStruct2a struct1 = 2;
26 }
27
28 // Omitting this struct disables detailed information in the response.
29 message EnablesUIResponse {
30   optional int32 val1 = 1 [default=1];
31   optional int32 val2 = 7 [default=1];
32 }
33
34 // LanguageID specifies the device's language settings.
35 // This can affect e.g. OCR Text translation results
36 message LanguageID {
37   optional int32 unknown_val_1 = 1;
38   optional string string1 = 4; // e.g. "en"
39   optional int32 unknown_val_2 = 5;
40   optional string string2 = 6; // e.g. "US"
41   optional EnablesUIResponse enable_ui = 15697207;
42 }
43
44 // Information concerning device (OS, vendor etc.)
45 message DeviceInfo {
46   optional string os = 1;
47   optional string os_version = 2;
48   optional string device_model = 4;
49 }
50 message UnknownStruct3 {
```

```
  optional int32 val1 = 1;
  optional int32 val2 = 3;
}
message DeviceInformation {
  optional bytes unknown_ID_string = 1;
  optional int64 unknown_num = 7;
  optional DeviceInfo dev_info = 9;
  optional UnknownStruct3 unknown_struct = 10;
}

// the minimal goggles request contains the image
// and language settings
message GogglesRequest {
  optional WrappedImage wrapped_img = 1;
  optional UnkownStruct2 some_struct = 2;
  optional int32 unknown_num = 6;
  optional LanguageID lang = 10;
  optional DeviceInformation dev_info = 15705794;
}
```

Listing A.2 Goggles Response message format definition

```
package goggles;

message Coords {
  optional int32 x = 1;
  optional int32 width = 2;
  optional int32 y = 3;
  optional int32 height = 4;
}

message ImageURLs {
  optional string image_highres = 1;
  optional string image_show = 2;
  optional string image_site = 3;
  optional string image_show_highres = 4;
}

message UnknownStruct1 {
  optional fixed64 unknown_int_1 = 1;
  optional fixed64 unknown_int_2 = 2;
}

message Uploader {
  optional string name = 2;
  optional string user_url = 3;
  optional string image_url = 4;
}
message UploadedInfo {
  optional Uploader uploader = 1;
  optional int64 unknown_id = 3;
  optional string text = 4;
  optional string abuse_link = 5;
  optional int32 unknown_num = 6;
  optional string unknown_random_string = 7;
```

```
}

message Action {
  optional string url = 1;
  optional string action_title = 2;
  optional int32 unknown_num_1 = 3;
  optional int32 unknown_num_2 = 4;
}
message ActionList {
  repeated Action action = 1;
}

message WordPos {
  optional Coords word_coords = 1;
  optional string chars = 2;
  repeated Coords char_coords = 3;
}
message TranslationResult {
  repeated string translation = 1;
  repeated WordPos word_position = 2;
}

// mostly unknown structures
message UnknownProductStruct1 {
  optional string unknown_string_1 = 1; // link
  optional string unknown_string_2 = 2; // link
  optional string unknown_string_3 = 3;
}
message UnknownStruct2a {
  optional int32 unknown_num_1 = 1;
}
message UnknownStruct2 {
  optional UnknownStruct2a unknown_struct = 1;
}
```

```
message UnknownStruct3 {
  optional int64 unknown_id = 1;
  optional int64 unknown_num = 2;
}
message UnknownStruct4 {
  optional fixed32 unknown_num_1 = 1;
  optional fixed32 unknown_num_2 = 2;
  optional fixed32 unknown_num_4 = 4;
}
message UnknownStringStruct {
  optional string unknown_string_1 = 1;
  optional string unknown_string_2 = 2;
  optional string unknown_string_3 = 3;
  optional string unknown_string_4 = 4;
}
message UnknownProductStruct2 {
  optional fixed32 unknown_num_1 = 1;
  optional string unknown_url = 3;
  optional string html_title = 4;
  optional int64 unknown_num_2 = 5;
  optional fixed32 unknown_num_3 = 6;
}
message UnknownStruct5 {
  repeated UnknownProductStruct2 product_struct = 1;
}
message UnknownStruct6 {
  optional string unknown_string_1 = 1;
  optional string unknown_string_2 = 2;
}

message LocalizationInfo {
  optional int32 unknown_num_1 = 1;
  optional string unknown_string_1 = 2;
  optional string language = 6; // e.g. "en"
```

```
  optional string country = 7; // e.g. "US"
}
message WrappedProduct {
  optional LocalizationInfo localization_info = 1;
  optional string product_string = 2; // "product "
  optional Product product = 3;
}
message UnknownStruct7 {
  optional string title = 1;
  optional string empty_string_1 = 2;
  optional string category = 3;
  optional UnknownProductStruct1 unknown_product_struct = 6;
  optional UnknownStruct2 stupid_1 = 9;
  optional string empty_string_2 = 10;
  optional UnknownStruct3 stupid_2 = 11;
  optional UnknownStruct4 stupid_3 = 12;
  optional UnknownStringStruct bunch_of_strings = 14;
  optional UnknownStruct5 bunch_of_structs = 15;
  optional int32 unknown_int_1 = 23;
  optional string empty_string_3 = 25;
  optional int32 unknown_int_3 = 27;
  optional UnknownStruct6 stupid_4 = 32;
  optional WrappedProduct named_result = 17711604;
}

// product related structs
message PriceRange {
  optional string range_string = 1;
  optional fixed64 unknown_num_1 = 2;
  optional fixed64 unknown_num_2 = 3;
}
message ProductMatch {
  optional string unknown_string_id = 1;
  optional string link = 2;
```

```
136   optional string store = 3;
137   optional PriceRange price_range = 5;
138 }
139 message ProductHierarchy {
140   optional int32 hierarchy_level = 1;
141   optional string group_name_1 = 2;
142   optional int32 group_id = 3;
143   optional string group_name_2 = 4;
144 }
145 message Product {
146   optional fixed32 unknown_num_1 = 1;
147   optional int32 unknown_num_2 = 2;
148   optional PriceRange price_range = 3;
149   // book result
150   optional string authors = 4;
151   optional string publisher = 5;
152   optional string year = 6;
153   optional string ISBN = 7;
154   optional string product_cid = 8; // to be used in:
155   // http://www.google.com/products/catalog?cid=${product_cid}&source=
          goggles&client=goggles
156   repeated ProductMatch match = 9;
157   optional string manufacturer = 10;
158   optional int32 unknown_num_3 = 11;
159   optional string sequence_id_str = 12;
160   repeated ProductHierarchy prod_hierarchy = 13;
161 }
162 message HierarchyPathList {
163   repeated ProductHierarchy hierarchy_level = 3;
164 }
165 message HierarchyPath {
166   optional HierarchyPathList hierarchy_path_list = 1;
167   optional string unknown_num_string = 2;
168 }
```

```
169
170 message CommentStruct {
171   optional string comment = 1;
172 }
173
174 message StringPair {
175   optional string key = 1;
176   optional string value = 2;
177   optional int32 unknown_value = 3;
178 }
179
180 message NewProductInfo {
181   repeated StringPair geo_info = 1;
182   repeated StringPair struct = 3;
183 }
184
185 // central structure referencing almost all relevant information
186 message UIInfo {
187   optional Coords coords = 1;
188   optional int32 unknown_int = 2;
189   optional ImageURLs image_urls = 3;
190   optional UnknownStruct1 unknown_struct = 5;
191   optional string Language = 6;
192   optional string result_string = 7;
193   optional string additional_info = 8;
194   optional string result_description = 9;
195   optional string unknown_int_1 = 12;
196   optional string unknown_int_2 = 13;
197   repeated ActionList actions = 14;
198   optional NewProductInfo new_product_info = 15;
199   optional string search_more = 19;
200   optional string buy_link = 20;
201   optional Product product = 21;
202   optional TranslationResult translation = 22;
```

```
203   optional UploadedInfo uploaded_info = 23;
204   optional string location = 24;
205   optional HierarchyPath hierarchy_path = 29;
206   optional CommentStruct unknown_struct_2 = 30;
207 }
208
209 message SearchQuery {
210   optional string url = 2;
211 }
212
213 message DirectResult {
214   optional string result = 1;
215   optional string desciription = 3;
216 }
217
218 message Info {
219   optional string title = 1;
220   optional string category = 2;
221   optional UIInfo ui_info = 15690847;
222   optional UnknownStruct7 unknown_struct = 16766290;
223   optional SearchQuery query = 15693652;
224   optional DirectResult direct_result = 16045192;
225 }
226
227 message AlternativeInfo {
228   repeated Info alternative_info = 1;
229   optional fixed64 unknown_int64 = 2;
230   optional string unknown_hex_string = 6;
231   repeated Info info = 7;
232 }
233
234 // most encapsulating, 'root' structure
235 message GogglesResponse {
236   repeated Info info = 1;
```

```
  optional string unknown_string = 5;
  optional AlternativeInfo alt_info = 15705729;
}
```

Bibliography

[1] Aitor Aldoma, Markus Vincze, Nico Blodow, David Gossow, Suat Gedikli, Radu Bogdan Rusu, and Gary R. Bradski. CAD-model recognition and 6DOF pose estimation using 3D cues. In *IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011*, pages 585–592, 2011.

[2] Gennady Andrienko, Natalia Andrienko, and Stefan Wrobel. Visual Analytics Tools for Analysis of Movement Data. *ACM SIGKDD Explorations*, 9(2):38–46, 2007.

[3] Dragomir Anguelov, Ben Taskar, Vassil Chatalbashev, Daphne Koller, Dinkar Gupta, Geremy Heitz, and Andrew Ng. Discriminative learning of Markov random fields for segmentation of 3d scan data. In *In Proc. of the Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 169–176, 2005.

[4] Jan Bandouch. *Observing and Interpreting Complex Human Activities in Everyday Environments*. PhD thesis, Technische Universität München, 2011.

[5] Emilia Barakova and Tino Lourens. Spatial navigation based on novelty mediated autobiographical memory. In *Mechanisms, Symbols, and Models Underlying Cognition*, pages 356–365. Springer, 2005.

[6] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In CGIV*, pages 404–417, 2008.

[7] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. Robotic Roommates Making Pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots*, 2011.

[8] Peter N. Belhumeur, João P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

[9] Mirela Ben-Chen and Craig Gotsman. Characterizing Shape Using Conformal Factors. In *3DOR*, pages 1–8, 2008.

[10] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[11] Paul J. Besl and Neil D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[12] Georg Biegelbauer and Markus Vincze. Efficient 3D Object Detection by Fitting Superquadrics to Range Image Data for Robot's Object Manipulation. In *IEEE International Conference on Robotics and Automation (ICRA), Rome, Italy*, 2007.

[13] Nico Blodow, Lucian Cosmin Goron, Zoltan-Csaba Marton, Dejan Pangercic, Thomas Rühr, Moritz Tenorth, and Michael Beetz. Autonomous Semantic Mapping for Robots Performing Everyday Manipulation Tasks in Kitchen Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[14] Nico Blodow, Dominik Jain, Zoltan-Csaba Marton, and Michael Beetz. Perception and Probabilistic Anchoring for Dynamic World State Logging. In *10th IEEE-RAS International Conference on Humanoid Robots*, pages 160–166, 2010.

[15] Nico Blodow, Radu Bogdan Rusu, Zoltan Csaba Marton, and Michael Beetz. Partial View Modeling and Validation in 3D Laser Scans for Grasping. In *9th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2009.

[16] Manuel Blum and Sampath Kannan. Designing Programs That Check Their Work. In *STOC*, pages 86–97, 1989.

[17] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[18] R.N. Bracewell. *The Fourier transform and its applications*. McGraw-Hill electrical and electronic engineering series. McGraw-Hill, 1978.

[19] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[20] Gary R Bradski. Computer Vision Face Tracking For Use in a Perceptual User Interface. *Interface*, 2(2):12–21, 1998.

[21] Leo Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996.

[22] Wolfram Burgard and Martial Hebert. World Modeling. In *Springer Handbook of Robotics*, pages 853–869. 2008.

[23] Matei Ciocarlie, Kaijen Hsiao, E. Gil Jones, Sachin Chitta, Radu Bogdan Rusu, and Ioan A. Sucan. Towards Reliable Grasping and Manipulation in Household Environments. In *Proceedings of RSS 2010 Workshop on Strategies and Evaluation for Mobile Manipulation in Household Environments*, 2010.

[24] Alvaro Collet, Dmitry Berenson, Siddhartha S. Srinivasa, and Dave Ferguson. Object Recognition and Full Pose Registration from a Single Image for Robotic Manipulation. In *IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan*, 2009.

[25] Alvaro Collet Romea. *Lifelong Robotic Object Perception*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2012.

[26] Alvaro Collet Romea, Manuel Martinez Torres, and Siddhartha Srinivasa. The MOPED framework: Object recognition and pose estimation for manipulation. *International Journal of Robotics Research*, 30(10):1284 – 1306, 2011.

[27] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(6):681–685, 2001.

[28] Silvia Coradeschi and Alessandro Saffiotti. Perceptual Anchoring of Symbols for Action. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 407–416, 2001.

[29] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20 (3):273–297, 1995.

[30] Ingemar J. Cox and Sunita L. Hingorani. An Efficient Implementation of Reid's Multiple Hypothesis Tracking Algorithm and Its Evaluation for the Purpose of Visual Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.

[31] Navneet Dalal and William Triggs. Histograms of Oriented Gradients for Human Detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05*, 1:886–893, 2004.

[32] Luc De Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer, 2008.

[33] Mehmet Dogar and Siddhartha Srinivasa. Push-Grasping with Dexterous Hands: Mechanics and a Method. In *Proceedings of 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, 2010.

[34] Markus Eich and Malgorzata Dabrowska. Semantic Labeling: Classification of 3D Entities Based on Spatial Feature Descriptors. In *Best Practice Algorithms in 3D Perception and Modeling for Mobile Manipualtion. IEEE International Conference on Robotics and Automation (ICRA-10), May 3, Anchorage, United States*, 2010.

[35] J Elfring, S Van Den Dries, MJG Van De Molengraft, and Maarten Steinbuch. Semantic world modeling using probabilistic multiple hypothesis anchoring. *Robotics and Autonomous Systems*, 2012.

[36] D. Feldman and D. Weinshall. Motion Segmentation and Depth Ordering Using an Occlusion Detector. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(7):1171–1185, 2008.

[37] Ivan P. Fellegi and Alan B. Sunter. Ad//dx.doi.org/10.2307/2286061. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[38] David Ferrucci, Eric Brown, Jennifer Chu-carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, and John Prager. Building Watson : An Overview of the DeepQA Project. *AI Magazine*, pages 59–79, 2010.

[39] David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. Building Watson: An Overview of the DeepQA Project. *AI Magazine*, 31(3):59–79, 2010.

[40] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In *Comm. of the ACM, Vol 24, 1981*, 1981.

[41] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[42] John P. Frisby and James V. Stone. *Seeing: The Computational Approach to Biological Vision*, chapter 8: Seeing Objects, pages 178–179. MIT Press, 2010.

[43] Mario Fritz, Trevor Darrell, Michael Black, Gary Bradski, and Sergey Karayev. An Additive Latent Feature Model for Transparent Object Recognition. In *NIPS*, 2009.

[44] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, and et al. Multi-Hierarchical Semantic Maps for Mobile Robotics, 2005.

[45] Cipriano Galindo, Juan-Antonio Fernández-Madrigal, Javier González, and Alessandro Saffiotti. Robot task planning using semantic maps. *Robotics and Autonomous Systems*, 56(11):955–966, 2008.

[46] T. Gao and D. Koller. Active Classification based on Value of Classifier. In *Advances in Neural Information Processing Systems (NIPS 2011)*, 2011.

[47] Timothy Gatzke, Cindy Grimm, Michael Garland, and Steve Zelinka. Curvature maps for local shape comparison. In *Shape Modeling and Applications, 2005 International Conference*, pages 244–253. IEEE, 2005.

[48] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *ICAR*, pages 317–323, 2003.

[49] Lise Getoor and Ben Taskar, editors. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. MIT Press, 2007.

[50] O. Grau. A scene analysis system for the generation of 3-D models. In *NRC '97: Proceedings of the International Conference on Recent Advances in 3-D Digital Imaging and Modeling*, page 221, 1997.

[51] Gregory D. Hager and Ben Wegbreit. Scene Parsing Using a Prior World Model. *International Journal of Robotics Research (IJRR)*, 30(12):1477–1507, 2011.

[52] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[53] Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[54] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, adapted edition, 2004.

[55] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Using Depth Cameras for Dense 3D Modeling of Indoor Environments. In *International Symposium*

on Experimental Robotics (ISER), 2010.

[56] 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. *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[57] 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. 2012.

[58] Ian D. Horswill. *Specialization of perceptual processes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1993.

[59] G. Impoco, P. Cignoni, and R. Scopigno. Closing Gaps by Clustering Unseen Directions. In *SMI '04: Proceedings of the Shape Modeling International 2004*, pages 307–316. IEEE Computer Society, 2004.

[60] S. Intille, K. Larson, E. Munguia Tapia, J. Beaudin, P. Kaushik, J. Nawyn, and R. Rockinson. Using a live-in laboratory for ubiquitous computing research. In *Proceedings of PERVASIVE 2006*, volume LNCS 3968. Springer-Verlag, 2006.

[61] Anil K Jain, Robert PW Duin, and Jianchang Mao. Statistical Pattern Recognition: A Review. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 22(1), 2000.

[62] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(5):433–449, 1999.

[63] Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. Cloud-Based Robot Grasping with the Google Object Recognition Engine. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[64] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[65] Ulrich Klank, Daniel Carton, and Michael Beetz. Transparent Object Detection and Reconstruction on a Mobile Platform. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[66] Ulrich Klank, Muhammad Zeeshan Zia, and Michael Beetz. 3D Model Selection from an Internet Database for Robotic Vision. In *International Conference on Robotics and Automation (ICRA)*, pages 2406–2411, 2009.

[67] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss. Comparison of surface normal estimation methods for range sensing applications. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3206–3211. IEEE, 2009.

[68] David Kortenkamp and Terry Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, AAAI'94, pages 979–984, 1994.

[69] Danica Kragic and Markus Vincze. Vision for Robotics. *Found. Trends Robot*, 1(1): 1–78, 2009.

[70] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119(1):191–233, 2000.

[71] Ray Kurzweil. *How to create a mind : the secret of human thought revealed*. Viking, New York, 2012.

[72] Kevin Lai and Dieter Fox. 3D laser scan classification using web data and domain adaptation. In *Proceedings of Robotics: Science and Systems*, 2009.

[73] Kevin Lai and Dieter Fox. Object Recognition in 3D Point Clouds Using Web Data and Domain Adaptation. *The International Journal of Robotics Research*, 29(8):1019–1037, 2010.

[74] Louisa Lam and Ching Y. Suen. Optimal combinations of pattern classifiers. *Pattern Recognition Letters*, 16(9):945–954, 1995.

[75] N. Landwehr, B. Gutmann, I. Thon, M. Philipose, and L. De Raedt. Relational Transformation-Based Tagging for Human Activity Recognition. In *Proceedings of the 6th Workshop on Multi-Relational Data Mining (MRDM)*, 2007.

[76] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A sparse texture representation using affine-invariant regions. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–319. IEEE, 2003.

[77] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee, 1991.

[78] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(9):1465–1479, 2006.

[79] Jing Li and Nigel M Allinson. A comprehensive review of current local features for computer vision. *Neurocomputing*, 71(10):1771–1787, 2008.

[80] L. Liao, D. Fox, and H. Kautz. Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields. *International Journal of Robotics Research*, 2007.

[81] L. Liao, D. Patterson, D. Fox, and H. Kautz. Learning and Inferring Transportation Routines. *Artificial Intelligence*, 2007.

[82] Yufeng Liu, Rosemary Emery, Deepayan Chakrabarti, Wolfram Burgard, and Sebastian Thrun. Using EM to Learn 3D Models of Indoor Environments with Mobile Robots. In *ICML*, pages 329–336, 2001.

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

[84] Ilya Lysenkov, Victor Eruhimov, and Gary Bradski. Recognition and Pose Estimation of Rigid Transparent Objects with a Kinect Sensor. In *Proceedings of Robotics: Science and Systems*, 2012.

[85] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The Office Marathon: Robust Navigation in an Indoor Office Environment. In *International Conference on Robotics and Automation*, 2010.

[86] Zoltan-Csaba Marton, Ferenc Balint-Benczedi, Nico Blodow, Lucian Cosmin Goron, and Michael Beetz. Object Categorization in Clutter using Additive Features and Hashing of Part-graph Descriptors. In *Proceedings of Spatial Cognition (SC)*, 2012.

[87] Zoltan Csaba Marton, Dejan Pangercic, Nico Blodow, and Michael Beetz. Combined 2D-3D Categorization and Classification for Multimodal Perception Systems. *The International Journal of Robotics Research*, 30(11):1378–1402, 2011.

[88] Zoltan-Csaba Marton, Dejan Pangercic, Nico Blodow, Jonathan Kleinehellefort, and Michael Beetz. General 3D Modelling of Novel Objects from a Single View. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[89] Zoltan-Csaba Marton, Florian Seidel, Ferenc Balint-Benczedi, and Michael Beetz. Ensembles of Strong Learners for Multi-cue Classification. *Pattern Recognition Letters (PRL), Special Issue on Scene Understandings and Behaviours Analysis*, 2012. In press.

[90] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

[91] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, 3(1):43–48, 2006.

[92] Andrew Miller and Peter K. Allen. Graspit!: A Versatile Simulator for Robotic Grasping. *IEEE Robotics and Automation Magazine*, 11(4):110–122, 2004.

[93] Joseph Modayil and Benjamin Kuipers. Bootstrap learning for object discovery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-04)*, pages 742–747, 2004.

[94] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, pages 593–598, 2002.

[95] J. Montiel, J. Civera, and A. Davison. Unified Inverse Depth Parametrization for Monocular SLAM. In *Proceedings of Robotics: Science and Systems*, 2006.

[96] T Mörwald, J Prankl, A Richtsfeld, M Zillich, and M Vincze. BLORT - The Blocks World Robotic Vision Toolbox. In *Best Practice in 3D Perception and Modeling for Mobile Manipulation (ICRA Workshop)*, 2010.

[97] Oscar Martinez Mozos, Zoltan Csaba Marton, and Michael 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, 2011.

[98] Karen L. Myers, Pauline Berry, Jim Blythe, Ken Conley, Melinda T. Gervasio, Deborah L. McGuinness, David N. Morley, Avi Pfeffer, Martha E. Pollack, and Milind Tambe. An Intelligent Personal Assistant for Task and Time Management. *AI Magazine*, 28(2):

47–61, 2007.

[99] Randal C. Nelson and Isaac A. Green. Tracking Objects Using Recognition. *Pattern Recognition, International Conference on*, 2:21025, 2002.

[100] Jennifer Neville and David Jensen. Collective Classification with Relational Dependency Networks. *Journal of Machine Learning Research*, 8:2007, 2003.

[101] Andreas Nüchter and Joachim Hertzberg. Towards Semantic Maps for Mobile Robots. *Journal of Robotics and Autonomous Systems (JRAS), Special Issue on Semantic Knowledge in Robotics*, 56(11):915–926, 2008.

[102] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6D SLAM – 3D mapping outdoor environments. *Journal of Field Robotics*, 24(8-9):699–722, 2007.

[103] 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.

[104] David Opitz and Richard Maclin. Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[105] Óscar Martínez Mozos, Axel Rottmann, Rudolph Triebel, Patric Jensfelt, and Wolfram Burgard. Semantic Labeling of Places using Information Extracted from Laser and Vision Sensor Data. In *In Proceedings of the IEEE/RSJ IROS Workshop: From sensors to human spatial concepts*, 2006.

[106] Óscar Martínez Mozos, Rudolph Triebel, Patric Jensfelt, Axel Rottmann, and Wolfram Burgard. Supervised Semantic Labeling of Places using Information Extracted from Laser and Vision Sensor Data. *Robotics and Autonomous Systems Journal*, 55(5):391–402, 2007.

[107] Dejan Pangercic, Moritz Tenorth, Dominik Jain, and Michael Beetz. Combining Perception and Knowledge Processing for Everyday Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1065–1071, 2010.

[108] Kaustubh Pathak, Andreas Birk, Narūnas Vaškevičius, and Jann Poppinga. Fast registration based on noisy planes with unknown correspondences for 3-D mapping. *Trans. Rob.*, 26:424–441, 2010.

[109] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[110] W. Pentney, M. Philipose, J. Bilmes, and H. Kautz. Learning Large Scale Common Sense Models of Everyday Life. In *Proceedings of AAAI 2007*, 2007.

[111] B. Pitzer, S. Kammel, C. DuHadway, and J. Becker. Automatic reconstruction of textured 3D models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3486–3493, 2010.

[112] Robi Polikar. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45, 2006.

[113] Ronald Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[114] Ingmar Posner, Mark Cummins, and Paul Newman. Fast Probabilistic Labeling of City Maps. In *Proceedings of Robotics: Science and Systems*, 2008.

[115] Vijay Pradeep, Kurt Konolige, and Eric Berger. Calibrating a multi-arm multi-sensor robot: A Bundle Adjustment Approach. In *International Symposium on Experimental Robotics (ISER)*, 2010.

[116] Andrzej Pronobis. *Semantic Mapping with Mobile Robots*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2011.

[117] Andrzej Pronobis, Óscar Martínez Mozos, Barbara Caputo, and Patric Jensfelt. Multimodal semantic place classification. *The International Journal of Robotics Research*, 29 (2-3):298–320, 2010.

[118] Till Quack, Vittorio Ferrari, Bastian Leibe, and Luc J. Van Gool. Efficient Mining of Frequent and Distinctive Feature Configurations. In *IEEE 11th International Conference on Computer Vision (ICCV)*. IEEE Computer Society, 2007.

[119] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System. In *In IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[120] Pamela Renton, Michael Greenspan, Hoda A. Elmaraghy, and Hassen Zghal. Plan-N-Scan: A Robotic System for Collision-Free Autonomous Exploration and Workspace Mapping. *Journal of Intelligent and Robotic Systems*, 24(3):207–234, 1999.

[121] Matthew Richardson and Pedro Domingos. Markov Logic Networks. *Machine Learning*, 62(1-2):107–136, 2006.

[122] Mario Richtsfeld and Markus Vincze. Grasping of Unknown Objects from a Table Top. In *Workshop on Vision in Action: Efficient strategies for cognitive agents in complex environments*, 2008.

[123] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[124] Thomas Rühr, Jürgen Sturm, Dejan Pangercic, Daniel Cremers, and Michael Beetz. A Generalized Framework for Opening Doors and Drawers in Kitchen Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[125] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Technische Universität München, 2009.

[126] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, May 12-17*, 2009.

[127] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning Point Cloud Views using Persistent Feature Histograms. In *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, September 22-26*, 2008.

[128] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2155–2162. IEEE, 2010.

[129] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John 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)*, 2010.

[130] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4, 2011.

[131] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge in Robotics)*, 56(11):927–941, 2008.

[132] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Andreas Holzbach, and Michael Beetz. Model-based and Learned Semantic Object Labeling in 3D Point Cloud Maps of Kitchen Environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[133] Radu Bogdan Rusu, Aravind Sundaresan, Benoit Morisset, Kris Hauser, Motilal Agrawal, Jean-Claude Latombe, and Michael Beetz. Leaving Flatland: Efficient Real-Time 3D Navigation. *Journal of Field Robotics (JFR)*, 2009.

[134] S. Savarese and L. Fei-Fei. 3D generic object categorization, localization and pose estimation. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.

[135] A. Saxena, M. Sun, and A.Y. Ng. Learning 3-D Scene Structure from a Single Still Image. *IEEE 11th International Conference on Computer Vision (ICCV), 2007.*, pages 1–8, 2007.

[136] Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. Robotic Grasping of Novel Objects using Vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[137] Robert F Schmidt, Gerhard Thews, and Florian Lang. *Physiologie des Menschen*, volume 29. Springer Berlin, 2005.

[138] Thorsten Schmitt, Robert Hanek, Michael Beetz, Sebastian Buck, and Bernd Radig. Cooperative Probabilistic State Estimation for Vision-based Autonomous Mobile Robots. *IEEE Transactions on Robotics and Automation*, 18(5), 2002.

[139] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, 2007.

[140] Dirk Schulz, Wolfram Burgard, Dieter Fox, and Armin B. Cremers. Tracking Multiple Moving Targets with a Mobile Robot using Particle Filters and Statistical Data Association. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1665–1670, 2001.

[141] William R. Scott, Gerhard Roth, and Jean-Francois Rivest. View Planning for Automated Three-Dimensional Object Reconstruction and Inspection. *ACM Computing Surveys*, 35(1):64–96, 2003.

[142] Stephen Se, David Lowe, and Jim Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 2051–2058. IEEE, 2001.

[143] Martin Sewell. Ensemble learning. Published online: http://machine-learning.martinsewell.com/ensembles/ensemble-learning.pdf, 2007.

[144] Craig M Shakarji et al. Least-squares fitting algorithms of the NIST algorithm testing system. *JOURNAL OF RESEARCH-NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY*, 103:633–641, 1998.

[145] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous multi-floor indoor navigation with a computationally constrained MAV. In *ICRA*, pages 20–25, 2011.

[146] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[147] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Computer Vision–ECCV 2006*, pages 1–15. Springer, 2006.

[148] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-Weighted Linear Stacking. *CoRR*, abs/0911.0460, 2009.

[149] Parag Singla and Pedro Domingos. Entity Resolution with Markov Logic. In *In ICDM*, pages 572–582. IEEE Computer Society Press, 2006.

[150] C. M. Smith, J. J. Leonard, A. A. Bennett, and C. Shaw. Feature-based concurrent mapping and localization for autonomous underwater vehicles. In *Proceedings of IEEE Oceans*, 1997.

[151] Hartmut Surmann, Andreas Nüchter, and Joachim Hertzberg. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3-4):181–198, 2003.

[152] Ben Taskar, Ming F. Wong, Pieter Abbeel, and Daphne Koller. Link prediction in relational data. In *in Neural Information Processing Systems*, 2003.

[153] Moritz Tenorth. *Knowledge Processing for Autonomous Robots*. PhD thesis, Technische Universität München, 2011.

[154] Moritz Tenorth and Michael Beetz. KnowRob – Knowledge Processing for Autonomous Personal Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4261–4266, 2009.

[155] Moritz Tenorth, Ulrich Klank, Dejan Pangercic, and Michael Beetz. Web-enabled Robots – Robots that Use the Web as an Information Resource. *Robotics & Automation Magazine*, 18(2):58–68, 2011.

[156] Moritz Tenorth, Lars Kunze, Dominik Jain, and Michael Beetz. KNOWROB-MAP – Knowledge-Linked Semantic Object Maps. In *10th IEEE-RAS International Conference on Humanoid Robots*, pages 430–435, 2010.

[157] The Khronos Group. OpenGL FAQ / 12 The Depth Buffer. Website: http://www.opengl.org/archives/resources/faq/technical/depthbuffer.htm, retrieved: 05.05.2013.

[158] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map Learning and High-Speed Navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998.

[159] S. Thrun and B. Wegbreit. Shape From Symmetry. In *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE, 2005.

[160] Sebastian Thrun. Robotic Mapping: A Survey. In *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[161] Kai Ming Ting and Ian H. Witten. Stacked Generalization: when does it work? In *in Procs. International Joint Conference on Artificial Intelligence*, pages 866–871. Morgan Kaufmann, 1997.

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

[163] Federico Tombari, Samuele Salti, and Luigi Di Stefano. A combined texture-shape descriptor for enhanced 3d feature matching. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 809–812. IEEE, 2011.

[164] Rudolph Triebel, Óscar Martínez Mozos, and Wolfram Burgard. Relational Learning in Mobile Robotics: An Application to Semantic Labeling of Objects in 2D and 3D

Environment Maps. In *Annual Conference of the German Classification Society on Data Analysis, Machine Learning, and Applications (GfKl)*, 2007.

[165] Rudolph Triebel, Richard Schmidt, Óscar Martínez Mozos, and Wolfram Burgard. Instance-based AMN classification for improved object recognition in 2d and 3d laser range data. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), (Hyderabad, India)*, 2007.

[166] Endel Tulving. Episodic and Semantic Memory1. *Organization of memory*, pages 381–402, 1972.

[167] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[168] Markus Ulrich, Christian Wiedemann, and Carsten Steger. CAD-Based Recognition of 3D Objects in Monocular Images. In *IEEE International Conference on Robotics and Automation*, pages 1191–1198, 2009.

[169] Shrihari Vasudevan, Stefan Gächter, Viet Nguyen, and Roland Siegwart. Cognitive maps for mobile robots-an object based approach. *Robot. Auton. Syst.*, 55(5):359–371, 2007.

[170] Markus Waibel, Michael Beetz, Raffaello D'Andrea, Rob Janssen, Moritz Tenorth, Javier Civera, Jos Elfring, Dorian Gálvez-López, Kai Häussermann, J.M.M. Montiel, Alexander Perzylo, Björn Schießle, Oliver Zweigle, and René van de Molengraft. RoboEarth - A World Wide Web for Robots. *Robotics & Automation Magazine*, 18 (2):69–82, 2011.

[171] David Weikersdorfer, David Gossow, and Michael Beetz. Depth-Adaptive Superpixels. In *21st International Conference on Pattern Recognition*, 2012. Accepted for publication.

[172] K. Welke, T. Asfour, and R. Dillmann. Object separation using active methods and multi-view representations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 949–955, 2008.

[173] Walter Wohlkinger and Markus Vincze. Ensemble of shape functions for 3D object classification. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 2987–2992. IEEE, 2011.

[174] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[175] Michael Wuenstel and Reinhard Moratz. Automatic Object Recognition within an Office Environment. In *CRV '04: Proceedings of the 1st Canadian Conference on Computer and Robot Vision*, pages 104–109, 2004.

[176] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010. Software available at http://octomap.sf.net/.

[177] Keenan Wyrobek, Eric Berger, H.F. Machiel Van der Loos, and Ken Salisbury. Towards a Personal Robotics Development Platform: Rationale and Design of an Intrinsically Safe Personal Robot. In *Proc. International Conference on Robotics and Automation (ICRA)*, 2008.

[178] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146 –151, 1997.

[179] P. Yan, S.M. Khan, and M. Shah. 3D Model based Object Class Detection in An Arbitrary View. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–6, 2007.

[180] Christoph Zauner. Implementation and benchmarking of perceptual image hash functions. *Master's thesis, Upper Austria University of Applied Sciences, Hagenberg Campus*, 43, 2010.

[181] Y. Zhang, A. Koschan, and M.A. Abidi. Superquadric Representation of Automotive Parts Applying Part Decomposition. *Journal of Electronic Imaging, Special Issue on Quality Control by Artificial Vision, Vol. 13, No. 3*, pages 411–417, 2004.

[182] Muhammad Zeeshan Zia, Ulrich Klank, and Michael Beetz. Acquisition of a Dense 3D Model Database for Robotic Vision. In *International Conference on Advanced Robotics (ICAR)*, 2009.