UNIVERSITÄT OSNABRÜCK

Institute for Cognitive Science
Knowledge-Based Systems

Master's thesis

Autonomous Tabletop Object Learning

Michael Görner

November 2015

First supervisor:	Prof. Dr. Joachim Hertzberg
Second supervisor:	Dipl.-Systemwiss. Jochen Sprickerhof

Abstract

[Gö15] In everyday life, humans are regularly confronted with new situations and unknown objects. Usually, they quickly conceptualize such objects and adapt to their changed environment. By contrast, most current robotic frameworks require very detailed descriptions of their working environment to operate successfully. This includes adequate maps for navigation as well as detailed models and trained classifiers for all objects the robot might encounter. As aquiring these is costly, researchers started to focus on methods which enable a robot to acquire some of these descriptions on its own.

This thesis contributes to this line of research. It presents the idea of *curious table exploration* together with a framework which enables mobile robots to discover unknown objects on top of tables in their environment and learn 3D models of those objects. To do so, the robot autonomously navigates around the table and acquires multiple views of the objects on top. Once an object representation has been acquired, similar looking objects can be recognized in the environment.

Zusammenfassung

Im alltäglichen Leben finden Menschen sich oft in neuen Umgebungen wieder und begegnen Objekten, die ihnen bis dato unbekannt sind. Im Allgemeinen konzeptualisieren sie solche Objekte sehr schnell und passen sich an die neue Umgebung an. Im Gegensatz dazu benötigen aktuelle Robotik-Frameworks noch immer sehr detailierte Beschreibungen der Umgebung in der ein autonomer Roboter agieren soll. Diese Beschreibungen beinhalten angemessene Karten der Umgebung sowie Modelle und Klassifikatoren für alle Objekte mit denen der Roboter unter Umständen interagieren soll. Aufgrund dieser sehr aufwändigen Anforderungen beschäftigen sich immer mehr wissenschaftliche Arbeiten damit, wie möglichst viele dieser Beschreibungen autonom erfasst werden können.

Diese Arbeit leistet einen Beitrag zu dieser Art von autonomer Umgebungsexploration. Sie beschreibt das Konzept *curious table exploration* und eine Implementierung dieses Konzepts in einem Software Framework. Hierdurch wird ein autonomer Roboter in die Lage versetzt unbekannte Objekte auf Tischen in seiner Umgebung zu erkunden und 3D Modelle dieser Objekte zu generieren. Der Roboter fährt autonom verschiedene Posen um einen Tisch herum an, um die Objekte auf ihm aus verschiedenen Perspektiven zu betrachten. Sobald ein Modell eines Objekts einmal generiert wurde, können ähnlich aussehende Objekte in der Umgebung wiedererkannt werden.

You don't always have to stand on the shoulders of giants.
Sometimes it's enough to stand on a pile of dwarfs.
– Joscha Bach

.ibaku lei minji cu cilre
.i la'edi'u ba galfi roda

Contents

Chapter 1

Introduction

These days, autonomous robots are able to perform a variety of tasks in human-made environments. In recent years, researchers demonstrated robots that, among other things, fetch a sandwich from the refrigerator or the cafeteria, prepare coffee, make pancakes, fold clothes, and catch balls you throw to them. These demonstrations are tremendous achievements. Even so, there is a catch to all of them: While the demonstrations took place in the laboratories of the respective researchers, none of these tasks could be performed in a different mundane environment without a significant amount of work by the involved researchers. This is mostly for two reasons:

- Many algorithms include assumptions (that are often unmentioned) about the world that appear reasonable at first glance, but do not necessarily hold true in new environments. Examples of such assumptions could be “A refrigerator can be opened with a handle on the right side”, “All clothes appear on surfaces of a different color”, or “a bottle of pancake dough always contains enough dough for a pancake”. Although examples can be given, the most problematic assumptions are actually those that are not formulated, but silently assumed by an algorithm.
- In new environments, the robot will encounter different types of coffee makers, T-shirts, pans, balls, etc. Because the respective new objects were not observed before, they will usually not be recognized and cannot be used for the requested task.

To resolve problems with the demonstrated capabilities in new environments, either the environment has to be adjusted to adhere to the assumptions of the algorithms or the algorithms will have to be modified to account for specific variations of the environment that were not considered before. To make use of objects which were not observed before, these objects are usually manually scanned and categorized. After adding the resulting models to the robot’s knowledge base, they can be recognized and utilized by the robot.

These shortcomings, especially the high involvement of a researcher in the adaptation process, motivate the research field of *autonomous exploration*. Instead of providing static knowledge of the robot’s environment, methods in this field attempt to extract models of the environment in an unsupervised fashion by analyzing the data perceived by the robot. With some tasks the extracted models can be used directly by respective software modules and the robot can perform its task autonomously in the new environment. This is the case with autonomously acquired maps for navigation and 3D models of objects that are required to grasp those objects safely.

With abstract tasks such as “preparing coffee”, the acquired models of the environment usually cannot be used directly and a human will have to categorize them and add additional semantic information. Even so, such a step does not necessarily require the involvement of a researcher anymore and it poses a challenge to the field of Human-Robot-Interaction to allow an informed user of the robot to add the required information.

This thesis contributes to the field of autonomous exploration. Focussing on tabletop scenarios, it presents a framework of software modules for the open-source robotics framework ROS [QCG+09] that implement a concept I call *curious table exploration.* This concept comprises

- The autonomous navigation to as well as around interesting tables in the environment
- The autonomous perception, discovery and exploration of objects on top of these tables
- The acquisition of models of the explored objects
- The ability to recognize objects explored before in new scenes

Hence, a robot that is capable of all of these aspects will be able to actively acquire a lot of information on tabletop scenes in its environment autonomously over time. Especially, this information includes the appearance of objects that lie in the environment, the places in which specific objects appeared over time, and the context in which an object appears in terms of its relation to other objects in the environment. Clearly, this information is characteristic of the robot’s environment and can be used right away to answer queries such as “Where did you see the object in front of you in the last week?” or “Where is the most probable location of this object?”. Also, it can provide a basis for various methods which aim at the unsupervised construction of semantically enhances models of the environment, i.e. *semantic maps.*

The complete framework was integrated and tested with the robot “Calvin”, depicted in Figure 1.1. The three main characteristics of this platform relevant for this work are as follows. First of all, it constitute a mobile robotic agent that can move around in indoor environments. Secondly, two horizontal laser scanners are attached to the mobile base and can be utilized to localize the robot in a map. Thirdly, the robot is equipped with an RGB-D camera that observes its environment from a human-like perspective and allows to perceive objects on top of a table. These characteristics are not specific to this robot setup, but can be found with various autonomous robots used in current research. Hence, the framework should be readily applicable with many other autonomous robots as well, although minor adjustments might be required.

The remainder of this chapter introduces some background terminology required to understand the successive chapters. The next three chapters describe the different modules of the software framework in detail. Afterward, Chapter 6 presents the results of experiments performed with the whole framework and discusses the performance of the individual modules. The last chapter gives an overview of possible directions of future work and concludes the thesis.

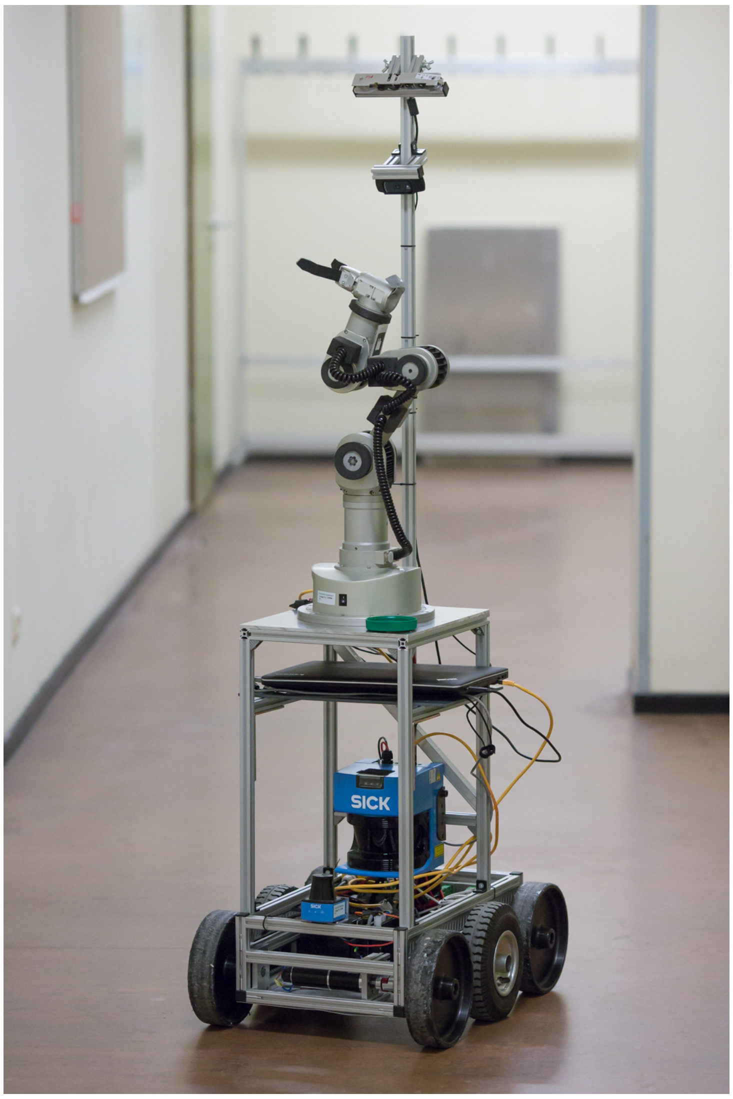


Figure 1.1: The autonomous robotic platform "Calvin"

Background Terminology

Because this work builds on scientific fields with established terminology, I use a number of such terms throughout this thesis. The most important ones are summarized as follows:

Percept

The term “percept” refers to a single datum of input received through an agent’s perception system. Relating to autonomous robotic systems, the term might for example refer to a single laser scan, a camera image, or a point cloud.

Point Cloud

Point clouds are one of the most basic data structures used in applied robotic perception. They describe sets of points specified in a two- or three-dimensional coordinate system. Additional information, such as color, an intensity value or a normal orientation can be attached to each point. Point clouds are generated for example by RGB-D cameras, which provide color information for each point.

Pose

A pose comprises a Euclidean position (a point in space) and an orientation (a direction of view). For example, Calvin’s pose in a room can be described by the location of its center point in space and the direction it faces.

Frame

Every coordinate and angle in a robotic system is given in relation to some coordinate system. Calvin’s pose in the room might be described relative to an arbitrary - but fixed - different pose in the room. A coordinate system of this kind is usually given a name such as “`map`”. Assuming the difference - a transformation - between different frames is known, data can be converted between the two by applying the transformation to each datum separately.

Normal

A normal, or normal vector, of a point on a surface describes the spacial direction orthogonal to the local surface around this point. Thus, it describes the local surface on which the point was measured.

Chapter 2

Related Work

This thesis integrates a variety of different fields of research. Accordingly, there is a variety of related work to each of these fields. However, the exact context of each field and its application in the scope of this work will only become apparent in the discussion of the different modules of the presented framework. Therefore, related work to the specific subtasks in the framework will be discussed in the chapters which present the respective modules of the framework. This chapter, on the other hand, reviews related work that deals with the overall idea of table exploration or the unsupervised formation of object representations. All of these approaches focus on different aspects of complete environment exploration with an autonomous robot.

[RSGB09] present methods to construct 3D models of objects, in the form of aligned point clouds, from a set of 3D laser scans. They expect objects of interest to appear in uncluttered environments to simplify the extraction of object views from complete scans. No restrictions are imposed on the laser scans they analyze except that the same objects are visible in multiple scans. Hence, they do not rely on global scan alignment techniques but associate different views of objects based on keypoint matching, traditional scan alignment methods, and a novel scoring function for aligned views. This function rates the similarity between sets of aligned range images and is used to cluster different partial views to form complete object models.

[NAH$^+$13] present a baseline system for autonomous table observation. The system utilizes a symbolical planner with external geometric reasoners to infer if an observation action is required to achieve a stated goal. It autonomously navigates around a table and collects percepts of the scene. These are stored in persistent memory accessible from within the system and, in a later step, can be aligned with traditional scan alignment methods and analyzed by an object recognition module.

[MM12] address the problem that most experiments in the field of autonomous robotics span only a short period of time. Hence, in these experiments the robot cannot acquire larger models of its environment or analyze changes in the environment over time. They implemented a system to construct a world model from the percepts of an autonomous robot which operates in an environment over a longer time frame. This model contains all tables the robot observed and keeps track of objects lying on top of them. By querying the model, the system is able to analyze changes in the environment and group observed objects by simple criteria. They present a data set that was collected over a period of six weeks containing data of more than 20 hours in which a robot navigates their laboratories and they use this data set to evaluate their system. In later work ([MMP14]) the authors built on this system. After collecting different views of tabletop objects in the model, they associate object views the robot observed within a short period of time at the same location. Such views are taken to represent the same object, assuming the environment did not change during this period. In a second step, they aim at the unsupervised extraction of simple object categories and associate object views if their color histograms and shape descriptors are similar enough. They tuned the respective thresholds on 86 hand-labelled object classes in the data set and report precision and recall values of up to 98.7% and 71.8%, respectively. However, they do not give examples of the different hand-picked classes and it remains unclear what these classes represent exactly.

A different field of research which aims at the unsupervised extraction of object representations considers *object manipulation*. Whereas often objects are only *observed* in percepts, an embedded robotic agent with a manipulator is able to *interact* with many tabletop objects. [KHRF11] and [KCF11] present a framework that enables their robot to explore tabletop objects it can grasp. The robot autonomously grasps uncluttered objects on a table in front and, if this is successful, moves the object in front of the camera to observe it from all sides. The object is tracked and all perceived object views are used to generate a 3D model of it. To reduce partial occlusions of the object due to the manipulator, they also consider putting the object back on the table to grasp it differently.

Going one step further, [MGC$^+$15] present a complete framework for visual Simultaneous Localization And Mapping (SLAM) that incorporates interactive tabletop object learning. They apply an appearance-based object segmentation algorithm to initially generate a set of object candidates on a table in front of the robot. These candidates are then verified by grasping or poking them with a manipulator. Objects in the scene that moved due to the attempted interaction are recognized as objects and are tracked independently of the rest of the world in subsequent manipulation attempts. After gathering enough object views, the system generates complete 3D models of the respective objects.

Contribution

While all research projects discussed above present methods which enable a robot to analyze its environment and build models of objects, they all focus on specific aspects of this process and none of them provide a complete and modular framework for the task. The main contribution of this work is to provide a set of exchangeable components which together enable an autonomous robot to perform curious table exploration. This includes autonomous navigation around tables, object discovery, model generation for tabletop objects and object recognition. For each implemented component different approaches are discussed and methods are presented that integrate well with the overall framework and, in some cases, improve on traditional methods for the same task. Notably, this work presents a novel version of the ICP algorithm to align point clouds in tabletop scenarios and a successful scheme to extend VFH signature matching to sets of signatures.

To facilitate future research to improve the framework, I attempt to explicitly list all major assumptions about the environment or the context of application with each module. This is meant to point out the main challenges for future research aiming at curious table exploration in arbitrary mundane environments.

Chapter 3

Object Discovery

In order to learn about unknown objects from (RGB-D) camera images, the inevitable first step is to mark a structure in the image as an object we can learn about. This step is widely known as *object discovery* or *object segmentation*. Another applicable term, in a broader sense, is *foreground-background segmentation*, given that objects in the camera image specify our "foreground of interest". This is a complex issue, and in fact Elizabeth Spelke, a cognitive psychologist working with infants, wrote in 1990 [Spe90]:

> [. . .] the ability to organize unexpected, cluttered, and changing arrays into objects is mysterious: so mysterious that no existing mechanical vision system can accomplish this task in any general manner.

In the last two and a half decades, major scientific progress has been made in the field of computer vision. Today, it is possible to successfully apply object segmentation and object recognition algorithms to much less restricted domains of images and objects than in the 90s. Yet, it remains a big challenge to select appropriate methods for a given detection problem and ensure their functionality.

3.1 Restricting the Scope

[Spe90] points out some fundamental aspects of objects which are relevant for human object segmentation, including *boundedness*, *object persistence* and *coherent motion*. In other words, we perceive something as an object if (1.) it is a bounded region in space, (2.) which exists over time and (3.) moves as a whole. Due to constraints on computational resources and integrated robotic platforms, only in recent years it became feasible to consider the latter two aspects in online object detection algorithms, and therefore many algorithms focus on the detection of object bounds. This chapter, too, will focus on geometrical properties of objects and their boundaries. In the next chapter, the aspect of object persistence will be addressed when integrating the ideas presented here with an embodied agent.

Although the three aspects mentioned above provide a good description of what constitutes an object, they are too general to readily apply them in current robotics research. To derive an object definition that can be used to discover objects in the scope of this thesis, I will pose a number of assumptions which impose significant restrictions on the definition of an object in this work. Most of these assumptions can be violated to some degree, but the framework explicitly does not account for errors occurring because of such violations.

Assumption 1 (Observability). *Objects are opaque, non-reflecting entities.*

This assumption is dictated by the *light coding* technology used to measure depth information with the RGB-D camera in use: The device projects a near-infrared light pattern onto the scene and captures the projection with a camera. By analyzing the distortions of the captured projection, a depth value can be computed for most of the camera pixels.

This technology provides depth information of good quality in indoor environments, but it requires the environment to act as a projection surface. Transparent and reflecting surfaces fall short of this requirement, because they disperse the light pattern. No depth information will be available for pixels which represent such objects. For this reason these objects (such as metallic cutlery and glasses) are not considered in this work. Note, though, that they usually produce characteristic “holes” in the computed depth image which can be analyzed as demonstrated by [LEB12, AS13].

Assumption 2 (Supporting Planes). *Objects are supported by/lie on top of planar, horizontal surfaces such as tables, i.e. supporting planes, at some height above ground.*

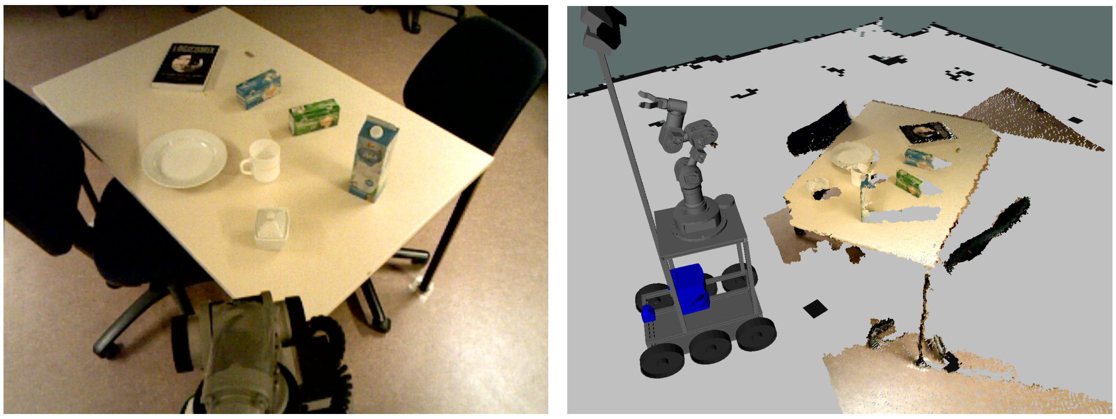
This is a more drastic assumption and it restricts the overall scope of this work almost exclusively to tabletop settings: It excludes e.g. chairs and doors and - in an autonomous setting - also most lamps, waste bins, shoes and a lot of other things people usually do not put on top of tables.[1] Even so, this assumption is quite popular (e.g., [MMP14, RBMB09]). This is, as stated above, because general object discovery is a complex problem and tabletop settings provide a simple and controlled environment, without *too* many restrictions: It is perfectly possible to place about every object relevant to everyday life on a table. Also, the volume of interest in which an object segmentation algorithm has to perform is reduced from the whole field of view of the camera to the planar surface of a table.

Assumption 3 (Light Clutter). *Different objects on a supporting plane can be visually separated by distance. There is a gap between different objects on a supporting plane.*

This assumption is a compromise between two arguments:

- In a domestic environment it is unrealistic to assume that the one, particular object a robot is asked to fetch lies right in the center of a table and, indeed, is the only object on the table.
- Without prior knowledge or the ability to manipulate the scene, it becomes next to impossible to infer one long black stick is the handle of the pan right next to it, but a different black stick is a straw in a glas of lemonade (which technically is an object on its own). To detect something as a standalone object, it is best to observe this object in isolation.

[1]Although most of these things are supported by the floor, the floor is explicitly *no* supporting plane in the scope of this thesis, because it trivially supports everything (including other supporting planes).



(a) (b)

Figure 3.1: A typical percept of Calvin (a) from the camera's point of view and (b) as a third-person view. In the latter, the depth information of the RGB-D camera is used to plot the image as a 3D point cloud.

Note that these two sides roughly correspond to two orthogonal lines of research: On the one hand, object models for robotic manipulation are usually acquired in carefully setup environments, often turntables, with a single object in the camera's field of view (e.g. [MPR$^+$10]); on the other hand, a lot of work in computer vision deals with unsupervised image segmentation algorithms (e.g. [GBS14]). Such algorithms are expected to produce segments which correspond to logical units *especially* in cluttered scenes. Even so, they usually produce more segments than there are objects in the image. To account for both sides' problems to some degree, the assumption allows multiple objects on a table, but maintains a distance between different objects such that they are still recognizable as separate objects.

With these assumptions formulated, we can go ahead and implement an appropriate object segmentation algorithm.

3.2 Tabletop Object Segmentation

As the robot explores the environment, it might perceive the scene illustrated in Figure 3.1. With respect to the assumptions given above, the robot observes seven objects in this scene - in accordance with Assumption 2, the table and the two chairs will not be considered.

The task for the segmentation algorithm is to extract a set of point clouds (i.e. regions of interest in the camera image) from the point cloud in Figure 3.1b, where each element of the set represents one segmented object view. Such a task specification is rather common in the field of autonomous robotics and there exist a number of libraries which facilitate the development of respective algorithms. I decided to utilize the libraries Ecto [ecta] and Point Cloud Library (PCL) [RC11] in this work.

The PCL provides sophisticated data structures to work with point cloud data and supports a wide range of algorithms such as centroid computation, normal estimation, convex hull and mesh reconstruction, histogram computation, etc.

Ecto, on the other hand, implements a computation graph framework especially useful for organizing computer vision algorithms. Usually, those algorithms are expressed as pipelines of relatively simple operations. In Ecto these pipelines are implemented as directed acyclic graph structures over "cells", where each cell performs only one operation and passes on its results. Also, Ecto provides a number of cells which smoothly bridge the gap between ROS' messaging system and the communication between ecto cells. This allows for the integration of Ecto graphs with the existing ROS infrastructure of a robot.

3.2.1 Table Segmentation

Because Assumption 2 defines objects with respect to tables, the first part of the pipeline focuses on the detection of tables in the robot's percepts. Michael Frayn gives an interesting, functional definition of a table in his comical novel "The Tin Men":

> And this is a table ma'am. What in essence it consists of is a horizontal rectilinear plane surface maintained by four vertical columnar supports, which we call legs. The tables in this laboratory, ma'am, are as advanced in design as one will find anywhere in the world.

Because we are not interested in the bottom side of tables, the first part of this definition suffices in our case. Therefore, the aim is to detect "a horizontal rectilinear plane surface". Such a structure can be represented appropriately by a two-dimensional polygon oriented in the environment. Note that this representation still allows for the approximate representation of curved tables. After all, the input point cloud from which the table will be extracted only provides a finite discretization of the curvature.

We are only interested in surfaces *at a minimum height above the floor*, but the most obvious, though not rectilinear, horizontal surface in Figure 3.1b is the floor itself. Hence, as a first step, all points which represent the floor should be subtracted from the cloud. However, this step requires more background knowledge than introduced so far. The point cloud is given in the camera's coordinate frame, i.e. the point (0/0/0) is right on the lens of the camera and (1/0/0) is 1 meter in front of the camera. As a result of this (straight-forward) representation, the location of the floor in the input cloud is unknown. This is especially troublesome if, as is the case with Calvin, the camera is mounted on the robot *upside down.*

Assumption 4 (Known Floor)**.** *The 3D rotation and the height of the camera frame w.r.t. the floor is known. The transformation between the camera frame and some frame whose x-y-plane coincides with the floor is known.*

In the case of the robot Calvin, this transform *could* be considered static and could be inferred from the robot's URDF description[2]. This description provides a transform between the camera

[2]The Unified Robot Description Format (URDF) is the standard format to describe a robot's appearance in ROS.

frame, which is called `kinect_rgb_optical_frame`, and the static frame `base_footprint` right below the robot, whose x-y-plane spans the ground, assuming the robot stands perfectly straight.

However, the robot slightly sways when moving around or running over negligible obstacles such as cables on the floor. It does *not* stand perfectly straight. To account for this, Calvin is equipped with an Inertia Measurement Unit (IMU) to measure its orientation in 3D, a fact that I make use of. Instead of the static transform, I use the dynamic transform between the camera frame and the frame `map` as the required transform, where the latter, among other things, incorporates measurements from the IMU.

After transforming the input cloud to `map`, removing points on the floor is simple: they are at a height of 0 m, i.e. their z-coordinate roughly equals 0. To allow for noise in the robot's measurements and to provide a minimum height for a plane to be considered a *supporting plane* in the sense of Assumption 2, all points below a threshold of 20 cm are discarded. All remaining points might represent supporting planes and objects on top of them.

As a next step, the pipeline should identify the largest of these planes. I utilize the well-known paradigm Random Sample Consensus (RANSAC) [FB81] for this task. Respective algorithms to apply this for plane detection are readily available through the PCL [RC11].

In its basic version, the utilized algorithm proceeds by sampling three points at random from the input cloud. If these points are not collinear, the plane which passes through all three points is computed. Given some distance threshold, it computes all of this plane's inliers as the set of all points which are closer to the plane than the threshold. These steps are repeated for a given number of iterations and eventually the plane with the most inliers is returned. To refine the final result, the calculated plane is usually replaced by the least-squares-fit plane of all of its inliers.

This leaves us with two free parameters which have to be selected with regard to the table detection setting:

- *The distance threshold* specifies above which distance from the table plane points will be detected as outliers that are not part of the table plane. This depends on the noise level in the depth information of the camera input. A (rather high) value of 2 cm was used in practice to reduce false negatives.

- *The maximum number of iterations* before the best detected plane is returned
  To ensure a high probability of $p = 0.99$ to detect the largest plane in the input given uniform random sampling, [FB81] proposes an upper bound of $k = \frac{\log(1-p)}{\log(1-w^3)}$ samples, where w is the fraction of inliers of this plane in the input. Assuming the table plane occupies one third of the input image (ignoring points on the floor), this yields the number of approximately 122 iterations. In practice, 125 iterations were used without noticeable problems.

The application of this algorithm produces the plane equation of the dominant plane in the point cloud. Yet, a table surface is no (infinite) plane but a bounded region. To arrive at the polygonal description discussed above, the inliers of the plane will be analyzed further.

The applied RANSAC algorithm does not impose any structure on the inlier points beyond their low distance to the detected plane. Especially, it does not ensure that all inlier points form a single connected surface. Instead, points on an armrest on one side of an input image might be part of the inlier set of a table on the other side of the image. To make sure that all inliers of the extracted surface are connected, I apply Euclidean clustering with a threshold of 3 cm to all plane inliers. This produces a number of point clusters where different clusters are more than 3 cm away from each other. Only the largest cluster is taken to represent the detected table.

The two-dimensional convex bounding polygon of this cluster finally provides the abstract representation of the detected table/supporting plane. This process systematically overestimates the surface area for concave surfaces and can produce wrongly detected "tabletop objects", which happen to stand right in a cavity of a table. A characteristic example for this could be an office chair standing in front of a corner desk.

Because this table segmentation does not account for such problems, concave surfaces are not considered in this work and are excluded by the following assumption:

Assumption 5 (Convex Supporting Planes)**.** *Supporting planes are convex surfaces.*

To get rid of this assumption, one could for example reconstruct a mesh or an alpha shape from the table inliers and close holes in these shapes, which are due to occlusion.

3.2.2 Object Segmentation

Now that we have the input's dominant supporting surface at our disposal, we can go ahead and extract objects on top of it. Taking into account Assumption 3, this turns out to be a relatively simple clustering task which can be performed in different ways. The ROS package `ecto_opencv` [ectb] applies a simple region growing algorithm to a masked camera image, taking pixels which border on table inliers as seeds. Because in this work input data is represented as a point cloud instead of an image, this method cannot be applied and I implemented a more geometry-based approach using methods from the PCL.

To ensure all detected objects are *above* the table, first all points above the surface polygon are extracted. Note that this is different from extracting *supported* points. This step extracts those points, whose projections onto the table plane lie inside the polygon. Because this still includes points on the room's floor or ceiling which happen to be above/below the polygon, an additional height requirement is imposed and only points at a height between 0 m and 0.5 m above the polygon are being extracted. Although this still does not entail that objects actually *lie on top* of this polygon - they might hover above the table or hang from the ceiling - it is a sufficient approximation for this work.

Assumption 6 (No Flying Objects)**.** *Objects below some maximum height above a supporting plane lie on top of it.*

While this assumption might sound superfluous, it is simple to think of everyday scenarios in which it is violated:

- Somebody deliberately holds his arm over the table.
- A lamp hangs from the ceiling.
- The height of a shelf in a frame might not exceed the specified maximum height, therefore making it a "supporting plane" for the shelf above.

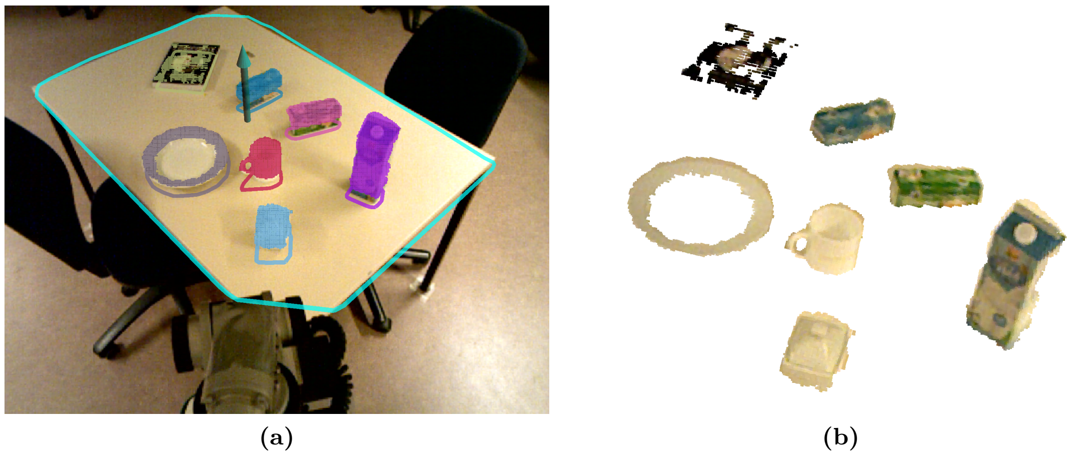


Figure 3.2: The results of the object discovery module on the percept in 3.1. (a) shows the extracted convex hull of the table, the discovered object views on this table, and the two-dimensional convex hull of each object on the table plane. (b) shows the different extracted object views in isolation.

Because Assumption 3 guarantees a gap between different objects, it is now easy to group the set of points above the table into clusters which correspond to individual objects. To do so, I utilize Euclidean clustering with a cluster tolerance of 5 cm to group the remaining points. Often there is still some noise left at this stage, and patches of table points which are too high above the table plane to be recognized as points on the table are now extracted as clusters of usually few points. To remove some of these negligible clusters, all clusters with less then 20 points will be discarded.

The application of the complete object discovery pipeline to the percept in Figure 3.1 yields the segmentation result shown in Figure 3.2.

Chapter 4

Table Exploration

The previous chapter focused on the extraction of object views from single images. Yet, autonomous robots usually do not come into existence in front of a table, take one picture of their surroundings, and vanish again. Instead, they are embodied agents and therefore capable of interacting with their environment.

Most importantly, almost every autonomous robot is able to move around in its environment and change the perspective from which it perceives. Therefore, given the object discovery pipeline from the previous chapter, an autonomous agent can not only discover objects once, but can investigate interesting objects from multiple perspectives. This yields multiple segmented object views which can be stitched together to form more complete 3D models of the inspected objects.

In theory the robot could investigate a dynamic object such as a mouse which runs across the table. However, because the mouse can move around and change its own appearance to some degree it is impossible to represent it by a single static 3D model. Additionally, it might have left by the time the robot moved around to observe it from a different perspective. Because the framework does not account for such cases, I henceforth assume that the overall environment (including all observed objects) does not change.

Assumption 7 (Static Environment)**.** *During the robot's exploration the environment will not change on its own.*

Also, this assumption allows to check for *object persistence* without having to worry too much about *object anchoring*: In dynamic environments it is a rather complex problem to decide whether an object detected in the latest camera image is the same one as an object detected twenty seconds ago from a different perspective. See [CS03] for a complete introduction to the topic. In a tabletop setting, this problem usually invokes methods from the field of *visual object tracking*, which track respective objects through all camera frames. They build a chain of linked detection results of objects over time, which can be used to match currently detected objects with older observations (e.g. [CC13]).

However, visual tracking assumes the relevant objects stay within the camera's field of view. Due to hardware constraints, this is not the case in this thesis: As the robot moves around to acquire new views of objects on a table, the table (and all objects on top of it) will regularly leave

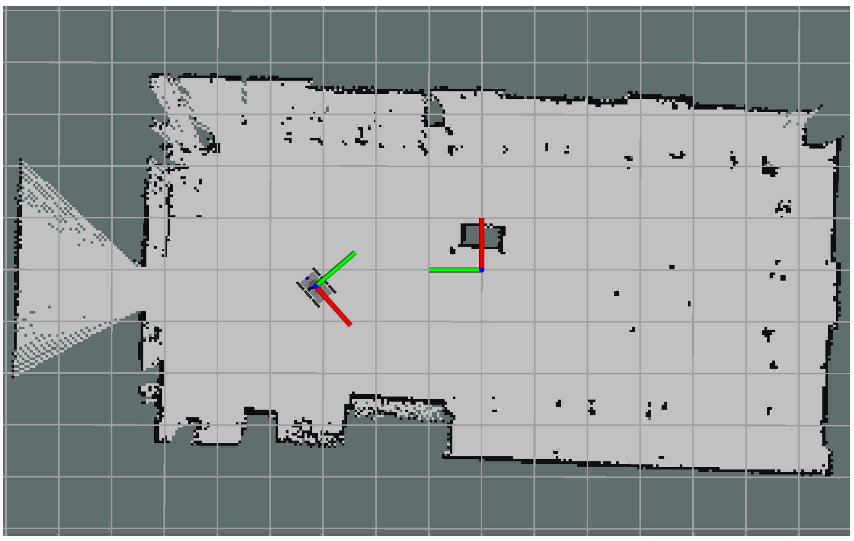


Figure 4.1: A grid map of the laboratory as it is generated by `gmapping`. The frame `map` is depicted by its unit coordinate system. Calvin's current pose in this frame is approx. $(-0.25\,\mathrm{m}, 3.2\,\mathrm{m}, -2.4\,\mathrm{rad})$.

and reenter the field of view. Hence, the assumption above guarantees that objects detected in different camera images are the same if and only if they appear at the same position in the environment as before.

The following section gives a short overview of applied robot navigation for indoor environments and presents the use case of the described navigation framework to explore objects on a table. Afterward, this chapter presents algorithms to merge multiple observations of this table and to extract multi-view object models of the objects on top of it.

4.1 Robot Navigation

Indoor navigation of autonomous agents in known and unknown environments is a well studied problem. Fields of research addressing different aspects of it include: traditional pathfinding such as Dijkstra's algorithm and the A* heuristic [RN10], Adaptive Monte Carlo Localization (AMCL) [Fox01], Occupancy Grid Mapping [GSB07], SLAM [TBF05], and Dynamic Window Approach (DWA) Motion Planning [FBT97]. In practice, these fields boil down to a navigation framework with the following assumptions:

- The world is considered to be a two-dimensional plane.
- The robot's environment is represented by a grid map of some fixed resolution. All maps in this thesis use a resolution of $5\,\mathrm{cm}^2$ per grid cell and one such map is depicted in Figure 4.1.
- The robot perceives its surroundings via a horizontal laser scanner.
- The robot itself is represented only by its center and orientation. To ensure there are no collisions with obstacles in the map, all obstacles are enlarged by the robot's outer radius.

Note that this does not take obstacles into account which cannot be observed in the plane of the horizontal laser scanner. Although there are numerous tables in the laboratory, the map in Figure 4.1 only shows their legs, which at first glance appear to be noise. This is an obvious restriction of the framework, but it also tremendously simplifies the navigation task. Because in this thesis the robot should specifically *move around* tables, tables should also be detected as obstacles to ensure that the robot does not run into them. Therefore an additional method to perceive obstacles is required:

- The robot perceives obstacles in front of its RGB-D camera at a height of more than 15 cm above the ground. All such obstacles in the vicinity of the robot occupy their corresponding map cells of the 2D map.

Using these five assumptions, a number of ROS packages provide implementations of the different navigation aspects mentioned above:

- Given a complete map of the environment and known start & goal poses, traditional pathfinding algorithms are applied by the ROS package `global_planner` to generate a motion plan.
- The robot's pose on the map, i.e. its pose in the frame `map`, is determined probabilistically (via AMCL) based on the percepts generated by the laser scanner and the robot's odometry. The respective algorithms are implemented in the package `amcl`.
- If no map of the environment exists, the required map can be generated using techniques from 2D SLAM research and Occupancy Grid Mapping, which are implemented in the package `gmapping`.
- Assuming an accurate grid map and a discrete motion plan on that map, the package `dwa_local_planner` applies DWA to solve the actual control problem of moving the robot along the planned path. This also takes dynamic obstacles, such as humans walking in front of the robot, into account.

These components combined provide the `move_base` framework, by which a mobile autonomous robot can navigate a planar surface, e.g. a regular room or story. It is able to estimates its own position and can move to arbitrary reachable poses in the room. Even so, there are a number of parameters to each of these components which make it hard to calibrate the system as a whole in order to work reliably.

4.2 Investigating Table Objects

I utilize the robot's navigation capabilities to acquire multiple views of the objects on top of a table and thereby observe as much of each object's surface as possible.

This raises the question from which poses the robot should try to observe the table objects. Especially, which poses of the robot (or its camera) are the most profitable ones to acquire more views of the objects on top of the table? This question has, in a more general setting, already been studied for many years in the field of *Next-Best-View Generation.* Research in this field ranges from purely theoretical research with virtual cameras evaluated against synthetic data sets [BWD+00] to the integration of proposed algorithms with robot arms for intelligent manipulation planning [KCF11]).

Even so, these research results are mostly incompatible with a completely integrated autonomous robot and I am not aware of research which could be readily integrated with the robot setup described in this thesis. For example [BWD+00] imposes the specific requirement for all possible camera poses to lie on a sphere with fixed radius around the center of the region of interest. This is obviously no reasonable assumption to make with a mobile robot which cannot necessarily reach poses at a fixed distance all around an object.

A more severe problem though which is usually not addressed in these works is a requirement imposed by the autonomous setup used in this work. The proposed algorithms usually apply a quality metric based on how much unobserved surface will be perceived from a new camera pose and how costly (often in regard to time) it is to move to this new pose. However, to apply such a metric in practice, the subsequent processing step which integrates all acquired object views into one model requires an accurate estimation of the camera's pose relative to the region of interest at all times. Without this information the different acquired views cannot be properly stitched together. In reality, the robot's pose estimation is quite imprecise and hence succeeding views of the table should always contain sufficient overlap with earlier views to ensure the successful integration of the new view (see Section 4.3 for the integration method applied in this thesis).

A reasonable next-best-view algorithm for this setup should therefore also consider the *safety* of the new pose, with respect to the successful integration of the new view. However, adding such a term to a next-best-view framework turns out to be rather difficult, because it basically changes the traditional scheme from an *exploration* scheme to an *exploration-exploitation* scheme and therefore complicates the overall problem significantly.

Based on these considerations, I decided to frame the problem of reasonable pose generation not as a question concerning the next best view, but instead consider the more simple question of how to *surround* objects on top of a supporting plane, as it is extracted by the pipeline presented in the previous chapter. As the polygonal representation of a supporting plane already provides us with the extracted contour of the table, *contour following* behavior comes to mind. This makes the robot move in parallel to the segment of the contour which is in front of it (in either direction), while it also ensures that the robot keeps an approximately constant and safe distance to the contour.

However, such an exploration behavior faces us with another problem. The goal of this table exploration module is to investigate *objects on top* of a table instead of the table itself. Now imagine the robot investigates a laboratory and encounters a long workbench which it wants to explore. It starts at one end of the workbench and discovers an object it does not recognize yet. If the robot only were to follow the extracted contour of the table, this novel object will leave the camera's field of view soon and the robot would go on to explore the rest of the table which turns out to be empty. Eventually the robot will return to observe the backside of the object. This is no problem in theory (the robot *does* observe the backside of the object), but it generates problems in practice: First and foremost, it changes the scope of the reconstruction from the size of tabletop objects to a larger scale and hence different methods would have to be applied in the reconstruction step. Secondly, assuming a completely static environment (Assumption 7) does allow the robot to explore other things for an arbitrarily long time before coming back to observe the rest of an object, but it is a pretty strong assumption to make with an autonomous robot. Because the robot should eventually be able to explore indoor environments in which humans live or work, it should uphold this assumption only for a short period of time and only within a restricted area. Hence, the individual exploration steps should be as short as possible. *Between* different exploration steps changes in the environment do not influence the exploration's outcome.

Following this idea, I decided to focus the robot's exploration on a single point on the table. This point could, for example, be the center point of a regular-sized table or the center of a specific object the robot should focus on. While the exploration module will try to enforce this point is at the center of the camera's field of view, objects in its vicinity will also be observed and reconstructed. With this approach, the workbench example from above is split up into multiple exploration steps: In a first step, the robot would explore one end of the table, probably focusing on the novel object on top. Only *after* it fully explored this object from all reachable points of view, the robot would turn to the next part of the workbench and might not even fully investigate the rest of it as soon as it discovers this was the only object on the whole workbench.

Given a focus point on the navigation plane $c = (c_x, c_y)$, it is quite easy to compute the pose of a robot from which it can observe c at an angle of θ rad and from a distance of d m:

$$p_\theta^d(c) = \begin{pmatrix} c_x \\ c_y \\ 0 \end{pmatrix} + \begin{pmatrix} d \cdot \cos(\theta) \\ d \cdot \sin(\theta) \\ \theta + \pi \end{pmatrix}$$

Here, the first two entries of each vector specify the two dimensions of the metric position in the map and the third component specifies the orientation angle in radians. As the robot is supposed to face the focus point from observation angle θ rad, the orientation of the robot corresponds to the reversed observation angle $\theta + \pi$ rad.

Because Calvin's camera is mounted on a pole near the robot's center pointing straight forwards, its center frame `base_footprint`, which is used for navigation, aligns with its camera frame `kinect_rgb_optical_frame` in the navigation plane. As a result, the 2D poses of the camera and of the robot are interchangeable without a relevant loss of precision. With a different robot setup, a transformation between these two frames might have to be taken into consideration.

Utilizing the definition from above, I characterize the set of poses from which the robot can observe the focus point as follows:

$$\mathcal{P}_c = \left\{ p_\theta^d(c) \middle| \theta \in [-\pi; +\pi), d \in [d_{\min}; d_{\max}] \right\}$$

To simplify the test for whether or not the focus point is in the camera's field of view from a given robot pose or whether the camera might be too far away from the point to actually perceive anything of interest, $d_{\min}$ and $d_{\max}$ are used in the above definition to specify the minimum and maximum distance from which the table should be observed. In the experiments performed with Calvin these values were set to 0.7 m and 1.5 m, respectively.

Assumption 8 (Observation Distance)**.** *Objects on top of a supporting plane can be reasonably observed by the robot from a distance between some known limits* $d_{\min}$ *and* $d_{\max}$ *facing the plane.*

However, not all of the poses in $\mathcal{P}_c$ will be reachable by the robot. Some will be blocked by the table which the robot tries to explore, others might be occupied by a chair or because there is a wall at one side of the table. To check for reachability, the `move_base` framework offers the service `make_plan`. This service tries to generate a motion plan between two given poses and either returns the complete motion plan or aborts ($\perp$). To make use of this service, the implementation of the table exploration module discretizes the set $\mathcal{P}_c$ along the parameters θ (the discrete set of angles is $\hat{\Theta}$) and d (the respective discrete set is $\hat{D}$) to check which of the resulting poses are reachable. Note that the discretization level influences the maximum number of observations which will be collected in one exploration step. In all experiments I used a discretization level of 0.5 rad for θ. This entails $|\hat{\Theta}| = 12$, i.e. the robot will observe objects from up to twelve different observation angles around the focus point. This choice usually provides enough object views with sufficient overlap between different views. Still, the level is chosen more or less arbitrarily and can be overwritten when the exploration module is invoked.

Because I want the robot to observe as much detail as possible from each observation angle but stay within reasonable time constraints while exploring, I further restrict the set of observation poses. Instead of considering *all* distance values from $\hat{D}$, the robot will only approach the observation poses which correspond to the smallest distance $d \in \hat{D}$ for which `make_plan` successfully returns a motion plan, i.e. it will try to get as close to the focus point as possible:

$$\mathcal{P}'_c = \left\{ p^d_\theta(c) \middle| \theta \in \hat{\Theta}, d = \min\{d \in \hat{D} | \; \texttt{make\_plan}(p_{\text{cur}},\, p^d_\theta(c)\,) \neq \perp\} \right\}$$

The robot's current pose p_{cur} is used as as the start pose for `make_plan`.

Now, in theory the exploration module only has to make the robot approach the different poses in $\mathcal{P}'_c$ and invoke the object discovery module described in the previous chapter after reaching each pose. In reality though, the `move_base` framework is known to either collide with obstacles quite regularly when asked to navigate to a pose in the immediate proximity of this obstacle, or not being able to navigate there *at all*, although this should be perfectly possible. This is mostly due to the simplified representation of the robot as a single point. The simplification implies that the robot can turn on the spot without collisions when in reality it cannot.[1]

To compensate for this problem, I decided to implement a two-step scheme to approach these poses.

$$\mathcal{P}''_c = \left\{ (p^{d_1}_\theta(c), p^{d_2}_\theta(c)\,) \middle| \begin{array}{c} \theta \in \hat{\Theta}, \\ d1 = \max\{d \in \hat{D} | \; \texttt{make\_plan}(p_{\text{cur}},\, p^d_\theta(c)\,) \neq \perp\}, \\ d2 = \min\{d \in \hat{D} | \; \texttt{make\_plan}(p_{\text{cur}},\, p^d_\theta(c)\,) \neq \perp\} \end{array} \right\}$$

Instead of directly sending a request to `move_base` to approach the observation pose, the exploration module requests to move the robot into the line corresponding to the next observation angle θ *at an increased distance* to an intermediate approach pose $p^{d_1}_\theta$. In a next step, the more

[1]There are mechanisms in place in `move_base` to avoid problems with such rotations, but often they do not suffice or trigger too late.

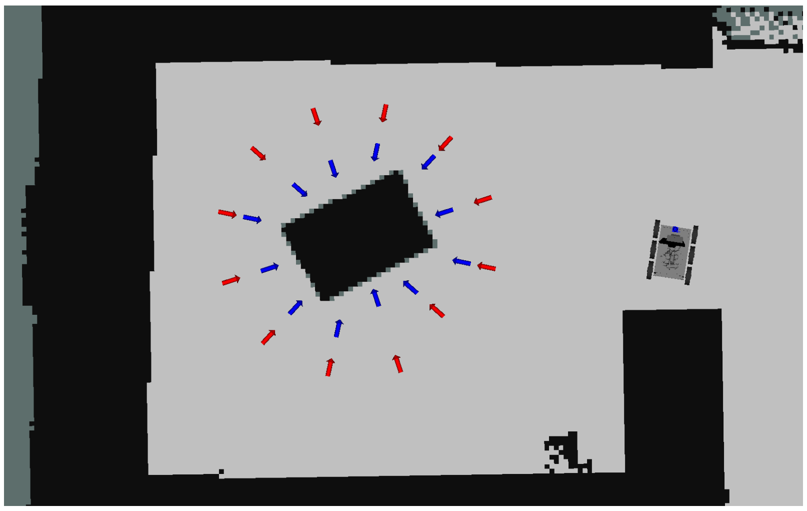


Figure 4.2: An example of $\mathcal{P}_c''$ in practice with the focus point on the table plane. Approach poses are colored red, observation poses blue.

simple navigation module `move_base_straight` is requested to move the robot (as the name suggests) in a straight line to the actual observation pose $p_\theta^{d_2}$. This exploration scheme turned out to be much more robust in practice.

Figure 4.2 illustrates the set $\mathcal{P}_c''$ in practice with a focus point at the center of the table. The complete navigation module for table exploration which is used in this thesis proceeds as follows: It receives a point c which is assumed to lie on a table in its environment. It proceeds by picking the element from $\mathcal{P}_c''$ whose approach pose is closest to the robot's current position. It moves the robot to this approach pose via `move_base` and afterwards approaches the observation pose via `move_base_straight`. There, it invokes the object discovery pipeline described earlier and collects the results. Afterward, it repeats these steps for all other elements in $\mathcal{P}_c''$ counterclockwise one after another.

4.3 Table View Registration

At each exploration step the framework as it was presented up to now yields object views (in the form of colored point clouds) of all objects it discovered on the table together with an estimate of the pose from which the robot has perceived these objects. To combine the views from different observation poses, succeeding views can be transformed into the `map` frame. Visualized together, the result will usually look like Figure 4.3a: Although nothing changed while the robot collected both views, the table appears to have moved (and even tilted a bit) between the two percepts. The perceived mismatch between the views originates in the aforementioned noise in the robot's localization. That is to say, it was not the table that moved, but the robot that moved differently from the framework's estimate. To accurately represent the table content from multiple perspectives and extract multi-view object models from table views, these views have to be properly aligned. An example of a reasonable alignment, which was computed using the Plane ICP method described in the next section, is depicted in Figure 4.3b.

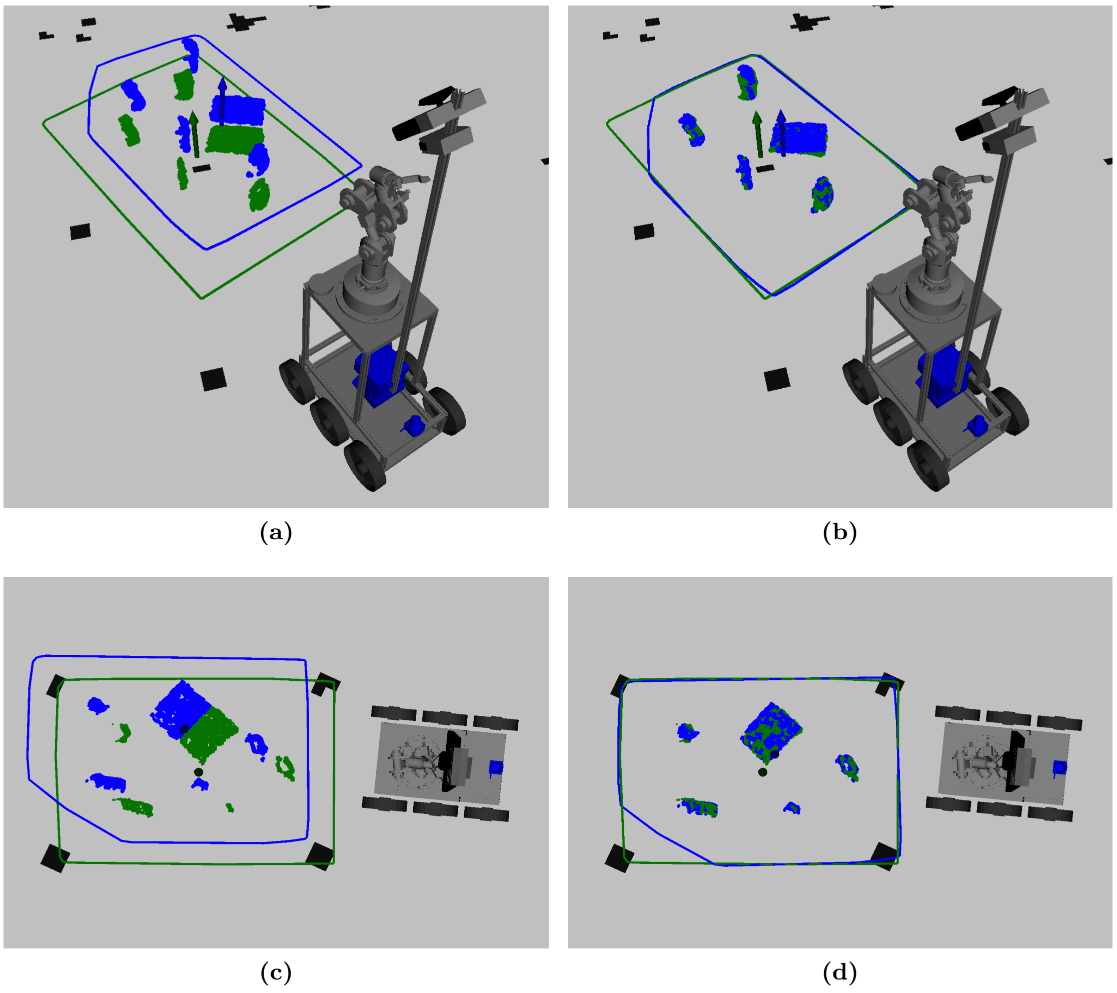


Figure 4.3: Two table views in uniform coloring shown together based on the localization estimate of `move_base` in (a) and (c) and after successful registration via Plane ICP in (b) and (d)

ICP

The task of estimating the mismatch between different 2D or 3D data sets (in our case 3D point clouds) showing the same scene and align them is called *point set registration* and has been a focus of research for the last 25 years. Often, one applies some version of the Iterative Closest Point (ICP) algorithm [BM92] to align the point sets and this work is no exception here.

This algorithm aims to align two point clouds, a *model* cloud and a *data* cloud, for which a rough estimate of the mismatch is already known. The usual task definition is to find a spatial transformation which can be applied to the *data* cloud such that it best aligns with the *model.* Given an initial estimate of this transformation, this estimate is iteratively refined in a way similar to standard Expectation-Maximization (EM) algorithms:

In a first step (Expectation) the current transformation estimate is used to select point pairs, i.e. *correspondences*, from both point sets. Points in a correspondence are taken to represent the same point in the scene and therefore each correspondence assigns a point in one cloud to its closest neighbor (by Euclidean distance) in the other cloud with respect to the current transformation estimate (if there is a neighbor within some maximum correspondence distance).

In the second step (Maximization) a new transformation is computed based on the estimated correspondences. The new transform aligns the two clouds in a way that puts the points in each correspondence as close together as possible. This is usually implemented by optimizing the mean squared distance between points in a correspondence.

After the second step the transformation estimate has changed and hence the correspondences selected in the first step have to be updated. Therefore, these two steps are repeated alternately until either the transformation converged (usually this happens quite fast) or a maximum number of iterations is reached. The final transformation provides the result of the algorithm and, assuming the algorithm converged on a reasonable optimum, the two clouds can be aligned by simply applying this transformation to the *data* cloud.

A lot of variants of the ICP algorithm have been studied in the past which differ in every single aspect from the abstract algorithm described above. See [RL01] for a review of a number of them. Still, the overall two-step scheme always remains the same.

Incremental ICP

On integrated robotic platforms the framework usually provides a stream of percepts - in our case point clouds - over time instead of a pair of percepts. In order to align all of these input clouds, the ICP algorithm can be applied incrementally to each pair of successive clouds, assuming these clouds have a sufficient overlap in the parts of the environment they represent.

Assumption 9 (Overlap of Successive Views)**.** *In the stream of perceived point clouds, each pair of successive clouds partially overlaps in the surfaces they represent to allow for successful alignment of these views without background knowledge.*

The stream-based algorithm to do this is formalized in Algorithm 1. Here, “$\Delta_{i-1} \circ \delta_i$” is used to denote the composition of two transforms where the transform δ_i is applied first and Δ_{i-1} is applied afterwards[2]. $\Delta_i(p_{\text{data}})$ denotes the application of the transformation Δ_i to the cloud p_{data}. This algorithm yields a new stream of point clouds $p_0^{\text{out}}, p_1^{\text{out}}, \ldots$ which are, assuming ICP converged on the right solution in each case, aligned pairwise and, by induction, also globally aligned.

Still, ICP - again assuming it converged on a reasonable solution - usually does not align point clouds from real sensors in precisely the same way a human would align them to represent the real environment. There will be small registration errors in each application of ICP and these errors add up over time to an unbounded global registration error, even though each small set of succeeding point clouds still appears to be aligned correctly. There has been a lot of research to improve on this type of multi-cloud registration by finding other reasonable pairs of point clouds in the input stream which can be aligned. This field of research is usually referred to as *loop*

[2]The composition operator for spatial transformations is not commutative and therefore the order of application matters. Even so, both orderings are used across the literature.

Algorithm 1 Incremental ICP

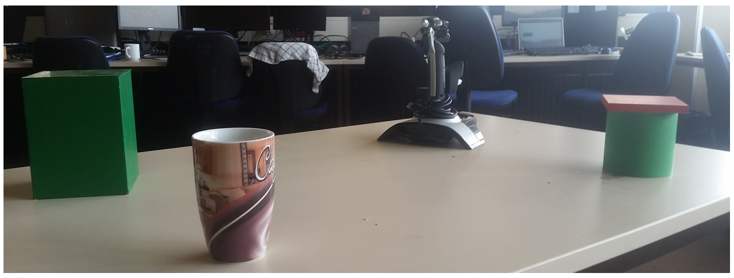
```
input: point clouds $p_0$, $p_1$, ...
yields: $\langle p_0^{\text{out}}, \Delta_0 \rangle$, $\langle p_1^{\text{out}}, \Delta_1 \rangle$, ...
procedure IncrementalICP($p_0$, $p_1$, ...)
    $\Delta_0 \leftarrow$ IdentityTransform
    $p_{\text{model}} \leftarrow p_0$
    $p_0^{\text{out}} \leftarrow p_0$
    yield $\langle p_0^{\text{out}}, \Delta_0 \rangle$
    for $i \leftarrow 1, 2, \ldots$ do
        $p_{\text{data}} \leftarrow p_i$
        $\delta_i \leftarrow \text{ICP}(p_{\text{model}}, p_{\text{data}}, \text{IdentityTransform})$
        $\Delta_i \leftarrow \Delta_{i-1} \circ \delta_i$
        $p_i^{\text{out}} \leftarrow \Delta_i(p_{\text{data}})$
        yield $\langle p_i^{\text{out}}, \Delta_i \rangle$
        $p_{\text{model}} \leftarrow p_{\text{data}}$
```

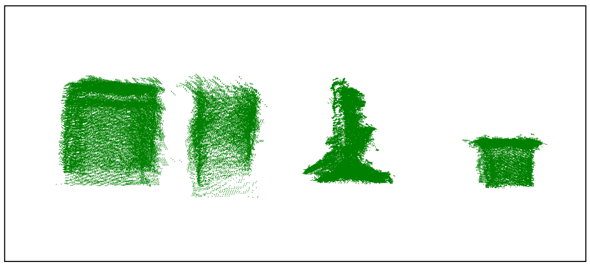
closing and there are a number of different approaches to it. For example, [SRGB11] recognizes previously seen places in the input by comparing oriented visual features in different percepts and aligns percepts which represent the same place. Still, incremental registration usually serves as a base line for such approaches and it is used in this thesis for the sake of simplicity.

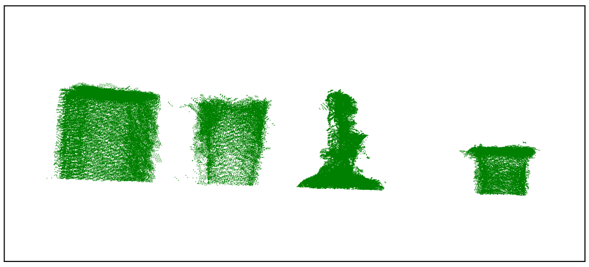
Another way to look at incremental registration is to use it as a form of *frame tracking*. For some iteration k in the algorithm, Δ_k is defined as $\delta_1 \circ \delta_2 \circ \ldots \circ \delta_k$ which corresponds to the aggregated mismatches between all previous clouds, that is to say, the mismatch between the first and the kth cloud. Therefore, this transformation can be used to infer where in the first cloud an object should have been which was detected at some pose in the kth point cloud. Correspondingly, the frame in which the first point cloud was given can be specified with respect to the kth cloud via the inverse transformation $(\Delta_k)^{-1}$. This property can be used to readily transform the first cloud (and everything that is specified in its frame) into the frame of the kth cloud although the incremental ICP algorithm given above transforms all clouds into the frame of *the first* cloud.

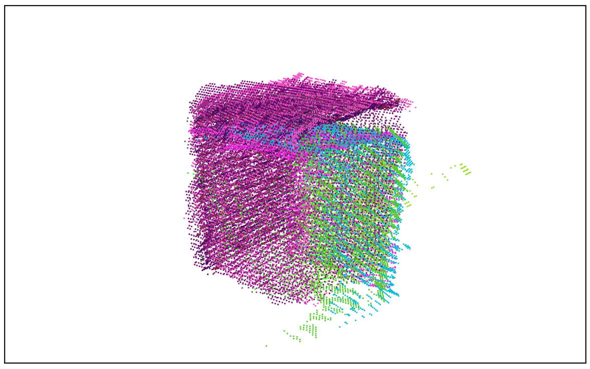
Plane ICP

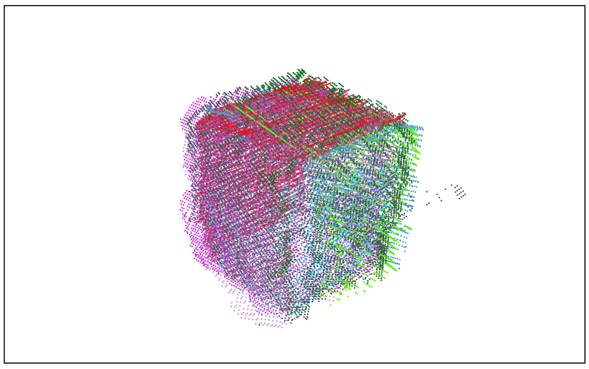
The incremental ICP algorithm presented above can be used to align a stream of table views gathered by the table exploration pipeline. An example of a resulting cloud of aligned views is depicted in Figure 4.4b. As can be seen, the algorithm managed to align the different views such that all views of an object are clustered together. Even so, there is a rather high global registration error that can be best observed with the aligned views of the cube in Figure 4.4d. One apparent mismatch catches the eye immediately: Why is the left part of the cube in Figure 4.4d on a different *height* from the right part? Assuming the object was not partially occluded in some view, the lower ends of all views should align on the supporting surface on which the object was detected. This world knowledge should (and can) be included in the registration process to refine the registration.

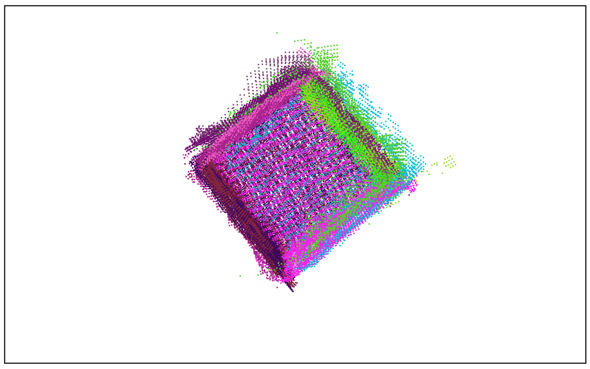

(a)


(b)


(c)


(d)


(e)


(f)

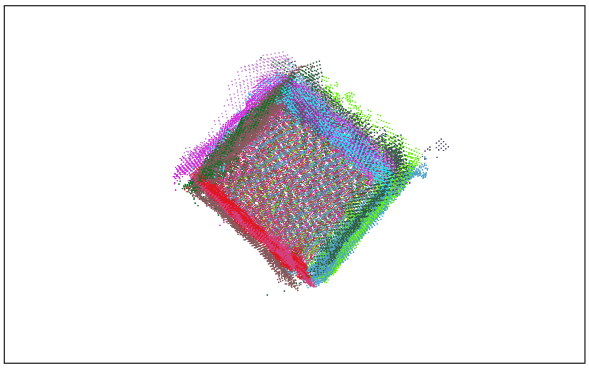

(g)

Figure 4.4: A number of incrementally aligned views of the table scene (a) with four objects depicted from the side and two detailed views of the solid green cube on the left. (b,d,f) were aligned via 6D ICP, (c,e,g) via Plane ICP

The authors of [NAH$^+$13] implemented a two-step registration process to refine their registration of tabletop views: First they run ICP on the *complete* point clouds perceived by the camera, after cutting away only the ground plane from each cloud. They argue that most of the remaining points represent the table plane and therefore the application of ICP to these clouds is biased to align the table planes. In a second step they remove dominant planes (including the plane of the supporting table) and again run ICP on the resulting point clouds where they use the transformations returned by the first ICP as initial estimates.

This procedure will likely produce reasonable alignments, but it has some shortcomings: First and foremost, the second ICP step will reintroduce some of the noise which can be observed in Figure 4.4d. Hence, it fails to fully incorporate the world knowledge discussed above. Secondly, it ignores some of the information which has to be extracted for tabletop segmentation anyway: The table plane of the table of interest is already segmented in each cloud, so instead of implicitly aligning the table planes in different clouds via ICP, this can be done explicitly by aligning the segmented table planes.

But again, this alignment is of negligible consequence if the next step aligns the clouds (containing only the object views) via traditional ICP in 3D space as described earlier. Also, it does not make sense to swap the ICP registration and the plane alignment steps, because the alignment produced by ICP will likely become worse[3] by aligning the planes afterwards. Running ICP a second time with the resulting (plane-aligned) transformation as transformation estimate could yield a completely different alignment.

Because, as stated before, ICP is a modular algorithm in which different components can be exchanged and because ICP can be efficiently implemented for two- as well as for three-dimensional data sets, this circular problem can be avoided by adjusting the ICP algorithm: Instead of computing the optimal spatial transformation in 3D space for the detected correspondences, this computation can be exchanged by a module which computes the optimal transformation in 2D space for the given correspondences. Thus, this computation will ignore the third dimension of all the points in the correspondences and will generate a transformation that moves the point clouds only in the plane which is spanned by the first two dimensions. Note though, this is not the same as running ICP on two-dimensional input data: Point correspondences will still be selected by their Euclidean distance in 3D. Thus, I will refer to this version of ICP as *2.5D-ICP* and consequently call the incremental version of it, which applies 2.5D-ICP in the incremental ICP algorithm given above, *Incremental2.5D-ICP*. Assuming all clouds are aligned in the same plane beforehand, *Incremental2.5D-ICP* can be applied to produce the registration result depicted in Figure 4.4c.

In reality, the detected supporting planes are not plane-aligned beforehand, so an incremental algorithm has to keep track of the most recent table view's plane to align it with the newly detected table. Again, the complete algorithm expects the table views p_i as input, but this time assumes they are given in individual frames $\mathtt{t}_i$ whose x-y plane aligns with the view's supporting plane. Also, it expects a transformation $\lambda_{\mathtt{map}}^{\mathtt{t}_i}$ with each view which describes how points in the view's frame $\mathtt{t}_i$ can be transformed to the global static frame $\mathtt{map}$. The full algorithm is given as Algorithm 2.

[3]with respect to ICP's optimization criterion

Algorithm 2 Incremental Plane ICP

```
**input:** $\langle p_0, \lambda^{\mathtt{t}_0}_{\mathtt{map}}\rangle, \langle p_1, \lambda^{\mathtt{t}_1}_{\mathtt{map}}\rangle, \ldots$
**yields:** $\langle p^{\text{out}}_0, \Delta_0, \Lambda_0\rangle, \langle p^{\text{out}}_1, \Delta_1, \Lambda_1\rangle, \ldots$
**procedure** IncrementalPlaneICP($\langle p_0, \lambda^{\mathtt{t}_0}_{\mathtt{map}}\rangle, \langle p_1, \lambda^{\mathtt{t}_1}_{\mathtt{map}}\rangle, \ldots$)
    Incremental2.5D-ICP iicp
    $\Lambda_0 \leftarrow \lambda^{\mathtt{t}_0}_{\mathtt{map}}$
    $\langle q_0, \Delta_0\rangle \leftarrow \text{iicp}(p_0)$
    **yield** $\langle q_0, \Delta_0, \Lambda_0\rangle$
    **for** $i \leftarrow 1, 2, \ldots$ **do**
        $\lambda^{\mathtt{t}_i}_{\mathtt{t}_0} \leftarrow (\Lambda_{i-1})^{-1} \circ \lambda^{\mathtt{t}_i}_{\mathtt{map}}$
        $\xi^{\mathtt{t}_i}_{\mathtt{t}_0} \leftarrow$ As2DTransform($\lambda^{\mathtt{t}_i}_{\mathtt{t}_0}$)
        $p^{\mathtt{t}_0}_i \leftarrow \xi^{\mathtt{t}_i}_{\mathtt{t}_0}(p_i)$
        $\langle q_i, \Delta_i\rangle \leftarrow \text{iicp}(p^{\mathtt{t}_0}_i)$
        $\Lambda_i \leftarrow \lambda^{\mathtt{t}_i}_{\mathtt{map}} \circ (\xi^{\mathtt{t}_i}_{\mathtt{t}_0})^{-1} \circ (\Delta_i)^{-1}$
        **yield** $\langle q_i, \Delta_i \circ \xi^{\mathtt{t}_i}_{\mathtt{t}_0}, \Lambda_i\rangle$
```

In contrast to the previous incremental ICP algorithm, this one returns *three* results for each input. On the one hand, all input clouds have to be aligned in one common frame whose x-y plane aligns with the table (this is the frame $\mathtt{t}_0$ of the first input cloud), but on the other hand this frame has to be tracked with respect to the `map` frame. The estimate of this transformation is updated with each input and is returned as well. Therefore, the algorithm returns (1.) the aligned cloud q_i, (2.) the estimated transformation $\Delta_i \circ \xi^{\mathtt{t}_i}_{\mathtt{t}_0}$ to align the cloud p_i to the first cloud and (3.) the estimated transformation Λ_i between the frame $\mathtt{t}_0$ and `map`.

To align the table planes of successive table views, the algorithm first computes the relative transformation $\lambda^{\mathtt{t}_i}_{\mathtt{t}_0}$ which can be used to transform the point cloud p_i from its own frame $\mathtt{t}_i$ to the latest estimate of the first cloud's frame $\mathtt{t}_0$. This transformation describes the difference between two frames, whose x-y plane coincides with the surface of the same table. So according to our background knowledge, this transformation must not change the z-component of the points in the cloud and it is only allowed to rotate the cloud around the frame's z-axis (turn the table around, but not flip it over). This corresponds to a regular two-dimensional transformation. Hence, in order to enforce the background knowledge that different views must align on their table planes, the computed transformation $\lambda^{\mathtt{t}_i}_{\mathtt{t}_0}$ can be reduced to a two-dimensional transformation. This is done by As2DTransform in the algorithm given above. This method generates a new transformation by copying its argument, but it ignores the arguments z-component and assumes 0 instead. Also, it extracts only the rotation around the z-axis from the original 3D rotation[4].

This algorithm was used to align the table views in Figure 4.4c. As can be seen from the figure, the registration result improved significantly over traditional incremental 6D ICP and Plane ICP corrected the vertical mismatch discussed above. Even so, the mismatch of the different point clouds in the plane, which can be observed in Figure 4.4f and 4.4g, did not improve significantly. Assuming that the robot has fully circumnavigated the table, this registration could likely be further improved by loop closing techniques.

[4]This is implemented using `tf`'s method `tf::getYaw`

4.4 Extracting Object Representations

The application of the registration method described in the previous section yields aligned point clouds which contain all views of discovered objects on top of the table. The remaining task is to extract reasonable object models from these.

To do so, first we have to extract sets of object views which correspond to the same object. As discussed before, Assumption 7 (static environment) ensures that views of discovered objects from different perspectives represent the same object if and only if the object appears at the same Euclidean position. Even so, this is still no trivial problem because the statement specifies the *object's* Euclidean position. But until the robot has built a complete representation of the object, there is no way to specify a unique point on the object to define its exact position.

Consider the interpretation of the statement to mean "the centers of the different views appear at similar positions". However, object views are made up of points on *one* side of an object. As a consequence, the center of a view of an object is not the center of the object, but instead the center of only one side of the object. Accordingly, centers of different views of the same object can be quite far apart. Indeed, in light clutter they can be *so* far apart that the center of the view of a different object is closer. Hence, distances between the centers of different object views cannot be used to unambiguously group object views into objects.

Because the registration step of the exploration framework in this thesis relies on ICP, it requires a sufficient overlap of the different table views. This does not guarantee that *all* object views overlap, for example some could be missing in a table view due to occlusion. Still, it provides a reasonable interpretation of the statement above to assume that different object views of the same object *overlap*. To check whether or not this is the case, I decided to follow [MM12] and utilize the two-dimensional convex hulls of the different object views. In case an object view overlaps with the convex hull of an earlier object view, these views are associated and new views are tested against the convex hull of the combined views. However, this procedure requires an additional assumption about the arrangement of objects on the explored table.

Assumption 10 (Convex 2D Shapes)**.** *Objects on supporting planes can be separated by their two-dimensional convex hull with respect to the plane.*

To remove this assumption, an alternative grouping strategy could be based on probabilistic data association models as those explored by [WKLP13].

The choice of the two-dimensional over the three-dimensional convex hull helps to associate object views which show only one flat surface of an object. The three-dimensional convex hull of such a view occupies a very small volume even if the view itself is of reasonable size. Thus, it is quite likely that other views of the object will not fall within this volume and, as a result, the views would not be associated. On the other hand, assuming that such a "flat" view is not perfectly orthogonal to the supporting plane, its two-dimensional convex hull will be of sufficient size.

The resulting sets of object views should represent the different objects on the table. But as explained in Chapter 3, the object discovery module sometimes also extracts object views which are actually patches of the table surface that appeared to be too high above the detected table plane due to noise in the input. During the exploration of the table, such erroneous object views usually pop up in one observation step but are not detected in the succeeding step. Therefore, they can be readily detected in the extracted sets of object views, because objects are required to *persist over time.* To remove such erroneous detections, a set of object views is removed if it has been observed less than three times in a row.

Now that all views of the same object are grouped together, these views could be used to construct object representations for various purposes. Quite a number of different representations for 3D objects have been used in the recent computer vision & autonomous robotics literature and all of these have different advantages and focus on different aspects.

The most simple form of representation merges all views of the object into *one point cloud.* This is obviously no memory-efficient model. To work around this problem, such point clouds can easily be grid-filtered to provide more lightweight representations. Point clouds of complete objects can for example be used with the ICP algorithm to fit them into new observations for hypothesis verification (see for example [AWGH11]). Point clouds provide a basis for many other forms of representation.

Traditionally 3D objects are represented as *polygon meshes.* These meshes can be created from point cloud data by various surface reconstruction techniques. They impose structure on the points by connecting multiple points to polygons. These polygons provide an explicit representation of an object's surface and, among other things, allow to do ray casting and grasp-stability analysis.

[KHRF11] build object models as sets of *surfels.* A surfel is a small patch of surface which is described by a center point, a normal vector and a radius. Hence, this representation extends simple point clouds to address some limitations, but keeps the set-like character of them: A single point in a point cloud does not describe a surface. Still, each pixel on an object in an input image represents a patch of surface on that object. Importantly, the size of such a patch on an object observed far away is much greater than the size of a patch observed nearby. This difference is not represented in traditional point clouds. Although this representation does not build an explicit surface structure of the whole object, it still provides some of the advantages of polygon meshes.

[MGC$^+$15] use the *truncated signed distance function* to represent objects the robot tracks. This method represents an object implicitly by a function on a 3D space. Each point is mapped to a value between -1 and 1 where values less than 0 correspond to points *inside* the object and values greater than 0 correspond to points *outside* the object. The surface of the object corresponds to the zero level set of this function (the set of all points which are mapped to 0). In practice this function is discretized at a high resolution and represented as a three-dimensional grid. Together with each function value a weight is stored that represents the certainty of the grid cell's value. Because of the implicit object representation this method can account for uncertainties in the object's surface and allows for incomplex incremental updating. Recently it became quite popular because it is well-suited for data-parallel implementations which run on GPUs. This allows for real-time updates and model construction from input data.

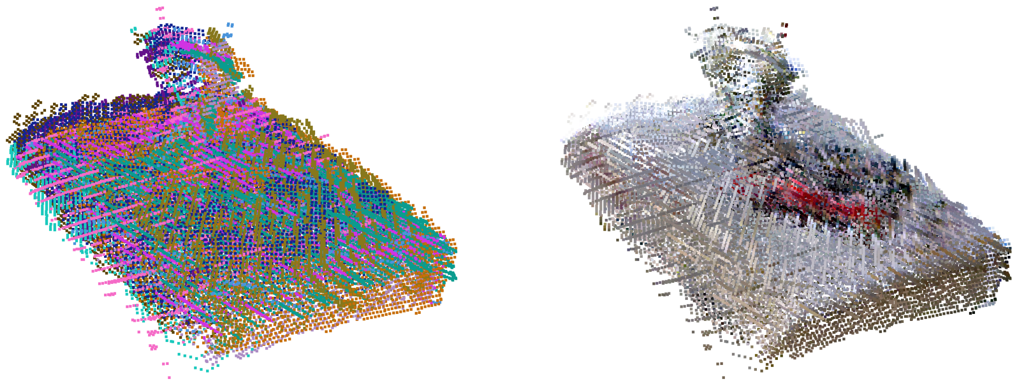
In theory all these representations could be created from the grouped object views. Even so, in this thesis object representations are only required to *recognize* the respective objects in new sets of discovered object views and not to grasp the object or track it in online sensor data. This is a general computer vision problem and many different methods can be used to tackle it. However, most methods require the individual object views as input and thus it is reasonable to utilize a representation which preserves all individual views.

I decided to store the object views in the camera's frame (the perspective from which they were originally perceived). Each object view is associated with a transformation into one common frame of reference at the center of the object, i.e. the center of the set of all aligned object views in the group. This frame retains the orientation of the supporting plane on top of which the views were extracted and it is computed from the result of the table view alignment.

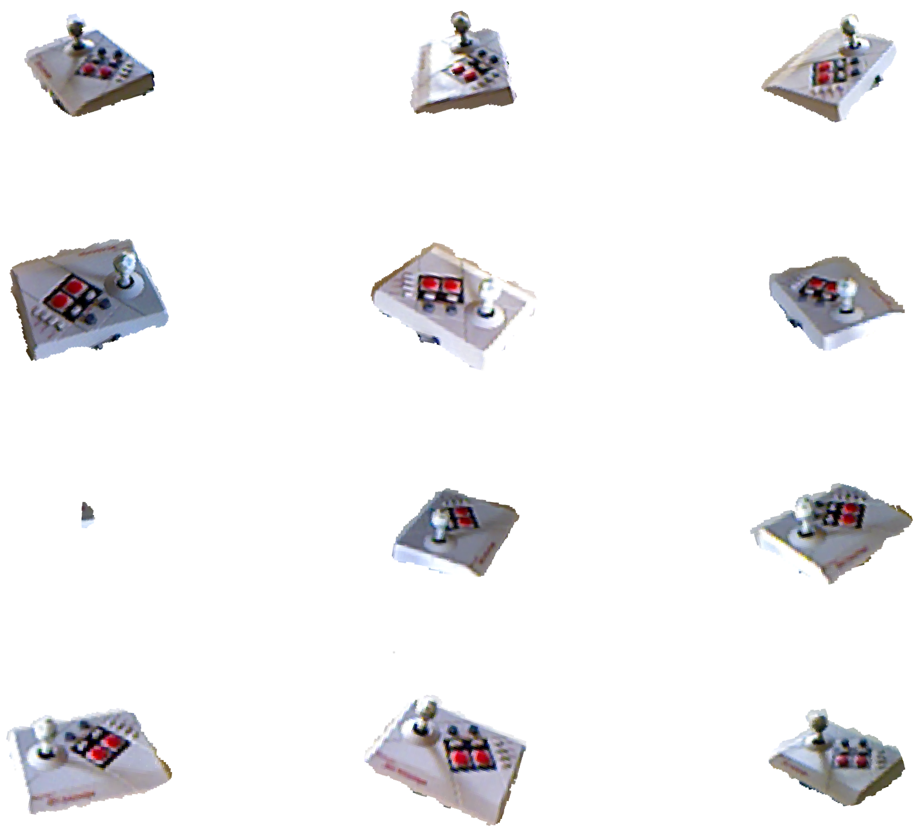
Hence, this multi-view object representation combines all pieces of information about an individual object that were extracted by the exploration module. These include:

1. the original object views
2. their pose on the supporting plane
3. their relation to all other object views

It provides the overall outcome of the table exploration module and Figure 4.5a depicts an example of such a representation. Note that one of the aquired object views in the example only shows one corner of the object because the rest of it was outside of the camera's field of view when the respective table view was perceived. On the one hand, such object views could be automatically removed from the representation. They contain significantly fewer points than all other object views of the representation and were observed at the border of the camera's field of view. Thus, it is simple to detect them. On the other hand, although such object views provide no relevant information about the respective object *in isolation*, such views can still help to refine the 3D object model. Therefore I decided to retain such partial object views in the generated multi-view object representations.



(a)



(b)

Figure 4.5: Multi-view object representation: (a) the registered point clouds colored by object view and in original input colors; (b) all views of the object

Chapter 5

Object Recognition

After the robot has explored an object it encountered in the environment, an obvious next question to ask is how this object and similar looking objects can be recognized in new observations. This ability is essential to form useful symbolical representations of the explored objects and it is a basis for various applications. These range from object-search related questions like “Where is the object that usually lies over here?” and “Have you seen my keys?” (assuming the robot has been told which of the objects it explored corresponds to “my keys”) to much more abstract questions like “Which object is missing on the table to make this a complete place cover as you have seen many times before?”. All of these tasks require a *symbolical* representation of an object (or a type of objects) which is *grounded* in actual observations of these objects and can be recognized in new percepts.

Object recognition from visual input data is a vast field of research which is studied since the mid-sixties. Hence, it is not in the scope of this thesis to give an overview over the entire field. Instead, I focus on object recognition from point cloud data. Keep in mind, though, that the object views collected by the table exploration module in this thesis can also be interpreted as image patches and therefore also support object recognition algorithms based on images.

Because of the autonomous exploration process and the task to recognize explored objects which are observed several times in different contexts, this task requires *unsupervised* recognition methods. In the context of point cloud data, recognition approaches can be divided into those which are based on matching local 3D features around single points in the data and those based on global descriptions of a cloud’s complete shape - usually in terms of histograms.

The first group of methods extracts a set of keypoints from the input cloud and associates each point with a description of its local surrounding in the cloud. The individual descriptions can be compared with a database of descriptions extracted from known objects. If enough descriptions of points in the input cloud match with descriptions of an object in the database, this object is recognized in the input cloud.

[TSS10] and [CC12] demonstrate two such approaches. The former presents the SHOT descriptor. This descriptor is made up of a set of histograms of angles between the normals of its center point and the points in its vicinity. The latter is based on *pairs* of keypoints in the input instead of single keypoints and their descriptor is made up of the points’ distance, the relative orientation of their normals and the points’ color values.

Recognition based on local features is usually not affected by clutter in the input because the cluster of descriptions of points which are on the object of interest should still match successfully regardless of the presence of additional points which are not part of the object. Also, to some extend methods relying on local features are robust to occluded parts, because if only half of the object is visible in the input cloud, this part might still contain enough descriptions to recognize the object. However, the downside to approaches based on local features is their relatively high sensitivity to noise. Because a description is based only on the immediate surroundings of a point, these descriptions change considerably if the input is locally distorted by noise. If they change too much, they do not match with the correct descriptions stored in the database anymore and recognition fails.

Global approaches on the other hand tend to be relatively stable with respect to local noise. These approaches compute a single description, a *signature*, of fixed size from the whole input cloud. Again, this signature can be compared with a database of signatures of point clouds of previously observed objects. If a sufficiently similar signature is found in the database, *the whole input point cloud* is recognized as the object whose view generated the signature. Hence, these approaches are sensitive to clutter and occlusion in the input point cloud - additional points in the input which are not part of the object as well as missing object points influence the overall signature of the cloud - and therefore they rely on an accurate object segmentation and mostly complete object views.

Two examples of such global recognition approaches are extended shape distributions [WV11] and the Viewpoint Feature Histogram (VFH) presented in [RBTH10]. Extended shape distributions describe a point cloud in terms of a number of histograms over the distance between randomly chosen points from the point cloud. This description is easy to compute and tries to approximately capture the overall shape of the point cloud. By contrast, VFHs represent an object as a set of histograms over functions of the points' normals. Hence, this description requires normals for all points in the cloud and is more expensive to compute.

Because the exploration module in this thesis already provides segmented object views, I decided to utilize a global descriptor approach for object recognition and used the implementation of VFH which is readily available in the PCL. Yet, global descriptors can only be used to recognize very similar point clouds and do not allow for structural changes in the cloud. While it does not overly influence local keypoint methods whether a package of cereal is open or closed as long as enough keypoints which are not on the lid are detected, this changes the signature of a global descriptor method considerably. To successfully recognize objects with this method, these objects are expected to retain their structure over all exploration steps.

Assumption 11 (Rigid Objects)**.** *Objects are rigid.*

5.1 Viewpoint Feature Histograms

As stated above, VFHs are fixed-size descriptions of a complete point cloud (of an arbitrary number of points) which are computed from the normals of this cloud. They were originally described by Radu Rusu et al. in [RBTH10] to support the mobile manipulation capabilities of the PR2 robot. They reported a recognition rate of 98.5% on their data set of over 60 household objects and 54000 object views generated with a turn table.

To compute this descriptor for segmented object views collected by the exploration module, first of all, normals have to be estimated for each point in these views. To do so, one considers the neighborhood, the *support*, of a point in the cloud. Assuming all points in the support lie on the same surface, the normal of this point can be approximated by the smallest eigenvector of the covariance matrix of its support. This eigenvector corresponds to the dimension that exhibits the least spread of points in the support, which should correspond to a direction *away* from the surface. The *direction* of the normal along this dimension, however, is not specified by this approximation. Because the collected object views were generated by an RGB-D camera from one single point of view, all surfaces which were observed from this perspective must face the camera. Therefore, the direction of the normal along the smallest eigenvector is chosen to make the normal point towards the camera.

To generate adequate normals which correspond to the real surface of the object from which the point cloud was generated, this leaves us with the crucial choice regarding the size of a point's support. If the size of the support is chosen too small, only few points are used to estimate the normal and the resulting normal is strongly affected by noise in the data. If the size of the support is chosen too big, local structures in the object's surface do not influence the normals anymore and are ignored. Considering the size of characteristic structures of tabletop objects and the resolution of the RGB-D camera, all points within a search radius of 1 cm were used as a point's support to estimate normals for collected object views. This value turned out to be a reasonable compromise for this use case.

Now, to compute the global histogram, VFHs require a keypoint in the point cloud that can be used as a reference point for the computation. For this purpose, the centroid p_c and the mean normal n_c of the point cloud are used. The combination of both can be used to specify a well-defined coordinate system for each point in the point cloud.[1] The three orthogonal dimensions u_i, v_i, w_i of this coordinate system with respect to a point p_i are defined as follows.

$$\begin{aligned} u_i &= n_c \\ v_i &= \frac{p_i - p_c}{||p_i - p_c||} \times u_i \\ w_i &= u_i \times v_i \end{aligned}$$

Note that the normal n_i of the point p_i is not used to specify this coordinate system. Instead, the normal can now be specified with respect to this coordinate system by two angles:

$$\begin{aligned} \cos(\alpha_i) &= v_i \cdot n_i \\ \theta_i &= \operatorname{atan2}(w_i \cdot n_i, u_i \cdot n_i) \end{aligned}$$

Additionally, the vector between p_c and p_i is characteristic of this point in the cloud and one can compute an angle between this vector and the coordinate system's axis u_i:

$$\cos(\phi_i) = u_i \cdot \frac{p_i - p_c}{||p_i - p_c||}$$

[1]This is called a Darboux frame in [RBTH10], but the definition of a Darboux frame relies on the point's normal and the principle curvatures of a point on a surface [HS03] and these are not used here.

Because the coordinate system's v_i axis is computed to be orthogonal to $p_i - p_c$, one angle already fully specifies the angular position of the vector $p_i - p_c$ in the coordinate system and it does not make sense to compute a second angle as done above for the normal n_i. All of these angles are geometrical properties of the point cloud and do not depend on the point of view from which the cloud was observed, as long as the set of observed points does not change. The values of $\cos(\alpha_i), \theta_i$ and $\cos(\phi_i)$ are computed for each point and represent parts of the VFH signature. Each angle is discretized into 45 bins and three histograms are created for them over all points. The resulting 135 values represent the first part of the signature.

Although the invariance to the point of view can be seen as a useful property of the signature, [RBTH10] considers it to be a shortcoming. Because the authors aim at a descriptor which can be used to identify not only the category, but also the *pose* of an object, they decided to include an additional value in the signature which explicitly depends on the point of view from which the point cloud was observed. To intertwine the point of view with information in the point cloud, they compute another histogram over angles between each point's normal and the vector between the camera's origin and p_i:

$$\cos(\beta_i) = n_i \cdot \frac{p_c}{||p_c||}$$

The respective histogram is discretized to 128 values[2] and appended to the signatures. This constitutes the official VFH signature which now consists of 263 values. In order to generate this signature invariant of the resolution of the point cloud, the final signature is normalized by the number of points in the point cloud.

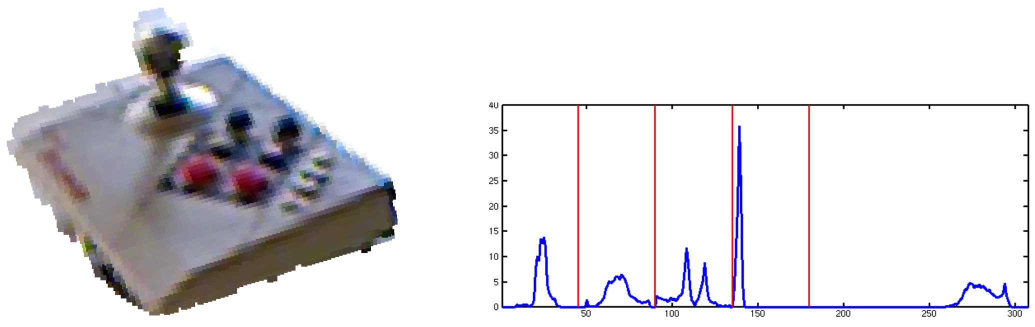
However, in a follow-up publication [AVB$^+$11], Aldoma et al. introduce an additional component to the signature because it concentrates on the normals only and does not capture the point distributions of the overall point cloud. According to them, this can result in very similar signatures for elongated and "more compact" planar surfaces. They add a *shape distribution component* to the signature to enhance it and remove this fault. The version of this component that is implemented in the PCL describes the normalized distance of each point to the centroid:

$$SDC_i = \frac{||p_c - p_i||}{\max_j(||p_c - p_j||)}$$

The computed values of this component are discretized to 45 values and the respective histogram is appended to the signature. The resulting VFH signature consists of 308 floating-point numbers. An example of this descriptor is given in Figure 5.1.

Since the original VFH signature was published, modified versions of it have been proposed as CVFH [AVB$^+$11] and OUR-CVFH [ATRV12]. These versions introduce a number of changes to the signature which mainly address the use of the centroid as a keypoint. Even so, they retain the original idea and the original four components of the signature. The main problem with the use of the centroid is that the centroid changes considerably even if only an insignificant part of

[2] As the authors note in a later paper [ATRV12], only half of these values are actually used in practice with RGB-D camera views because, as explained earlier, the normals of such point clouds are guaranteed to point towards the camera. This restricts $\cos(\beta_i)$ to positive values whereas they originally considered the whole function range $[-1; 1]$.



(a) (b)

Figure 5.1: (b) shows the VFH signature with shape distribution component of the point cloud in (a) (depicted from the perspective of the camera). The five different components are marked and ordered as $\theta, \cos(\alpha), \cos(\phi), SDC, \cos(\beta)$.

the object is occluded. Because the whole signature is built on top of the centroid, the signature changes drastically in this case. The proposed new descriptors try to extract a more stable keypoint and also generate multiple signatures in case no single keypoint in the point cloud is unique and/or stable enough. Because partial occlusions are no serious problem with lightly cluttered tables that are observed from multiple perspectives, I decided to use the more simple VFH signature in this thesis. Even so, it might be of interest to test these modified signatures in the future.

To decide which object view corresponds to a point cloud, [RBTH10] compute the nearest neighbor of all signatures of pre-labelled object views in a kd-tree (using L2-norm) and return the class and pose of the object view that gave rise to the nearest neighbor.

This procedure is mostly compatible with the setup in this thesis. However, the unsupervised character of this work leaves us with a single parameter to choose: Because there are no pre-labelled object views in our case, a threshold parameter γ has to be applied to decide whether an observed object is an *unknown* object or whether its nearest neighbor is close enough to recognize it as this object. The behavior of the complete object recognition module with different values of γ is discussed in Chapter 6.

Note that, in order to select one fixed value for γ, it is important to use signatures which are normalized with respect to the number of points of the corresponding point cloud. With unnormalized signatures, the influence of local (per point) noise in the input point cloud on the signature is proportional to the number of points in the cloud. A reasonable choice of γ should still recognize point clouds when they are locally perturbed and therefore would have to be selected proportional to the size of the respective point cloud when the signatures are unnormalized.

5.2 Multi-View Object Recognition

VFH signatures can be used together with a distance measure and a distance threshold γ to cluster and recognize single object views. However, objects in this thesis are reconstructed as multi-view object representations and those representations comprise more than one object view. Therefore, the object recognition module can make use of the additional knowledge these representations provide.

A simple way to extend VFH signature matching to multi-view object representations is to cluster all object views which are part of the same representation. While the signatures of object views of the same object from different perspectives usually differ quite a lot, these views still belong to the same object. In order to recognize a new object representation with this scheme, a single-linkage criterion can be applied. The signatures of all of its object views are compared to all previously collected signatures. If the shortest distance of any of the new signatures to a previous signature is below the distance threshold γ, the object representation is recognized as the same as all other representations in the cluster of this signature. Afterward, the object representation (with all of its object views) is added to this cluster. I call this the “shortest distance” approach to multi-view object recognition.

However, this approach still ignores a lot of information especially from the newly acquired object representation. Because eventually recognition is based only on one of the new object views, the one that is closest to a previously observed object view, all other views do not influence the final decision at all. This makes the recognition more prone to error and it does not take into account that different objects might look similar from a single perspective.

To account for these shortcomings, I applied a different distance measure between a new object representation and a cluster of object views. Instead of taking the overall-shortest distance between a new view and its nearest neighbor to represent this distance, one can compute the *mean* shortest distance to a view from the cluster over all object views of the new representation - on average, how far away are all new object views from any view in this cluster? This distance can be computed between a new object representation and all previously constructed clusters. If the shortest of these mean distances is below γ, the object representation is recognized as the object corresponding to the cluster and its object views are added to the cluster.

While this approach ensures no single object view dominates the recognition process, it also raises a barrier between different object representations: If an acquired object representation of an object does not contain a view from a certain characteristic perspective, it will never be associated with a new object representation of the same object which contains this view. In order to retain the benefits of a more stable recognition but to lower this barrier, the applied distance measure does *not* describe the overall mean shortest distance. Instead, it describes the mean shortest distance of only those n percent of the new views which are closest to an object view in the cluster.[3] This measure enforces that a new object representation is only close to a cluster of object views if it looks similar to these views from several points of view. But it also allows clusters to incorporate new object views which differ from all views in the cluster so far if these new views belong to an object representation that is otherwise very similar to the cluster. I call this the “mean distance” approach to multi-view object recognition.

[3] a value of 66% was used in all experiments

5.3 Priming

A general problem with many object recognition algorithms concerns their scalability in terms of the number of object classes that can be recognized by the algorithm. While many proposed recognition algorithms perform nicely with a relatively small number of classes, they become computationally intractable when more classes are added. This also poses a problem to the recognition module presented above: The mean shortest distance between an observed object representation and *each known cluster* has to be computed to decide if the representation belongs to one of these clusters. Even worse, the number of clusters, i.e. unsupervised object classes, is not limited and is bound to increase over time as the robot explores its environment and encounters new objects.

To limit the computational overhead of the recognition process, I implemented a priming mechanism: First of all, the module will compute the k nearest neighbors (k=1 proofed sufficient in practice) of each view of the new object representation in the set of *all* previously observed object views. This can be accomplished efficiently by using a kd-tree. Afterward, only the distances to the clusters which correspond to each of these neighbors are computed and evaluated.

This imposes an additional requirement to recognize an object as a known cluster: Leaving aside the mean shortest distance to the cluster, this cluster also has to contain at least one object view which is more similar (in terms of signature distance) to a view of the new object than any other object view that was observed before. While this procedure might miss a cluster correspondence that would be detected by computing the distance to all clusters, it reduces the computational complexity of the recognition module tremendously.

Chapter 6

System Integration

The complete proposed table exploration system was tested in our laboratory with a set of 30 different objects which are shown in Figure 6.1. These objects range from simple geometrical shapes like a ball and a cube to everyday objects like a plate, different cups, a package of cereal, a book, and different game controllers. I recorded 17 successful exploration steps of the robot around a table with random tabletop arrangements of four to nine of these objects and the exploration system observed each of these setups from 12 perspectives. The exploration steps took on average 323 seconds. Each object appeared in exactly four of the 17 arrangements. Thus, the system extracted a total of 120 multi-view object representations from 204 table views. One of these representations is shown in Figure 4.5.

This chapter evaluates the performance of the different subsystems with respect to the collected data set. It demonstrates advantages of the utilized methods over alternative approaches and discusses systematic sources of error in the different modules. For some of these errors, I propose stratagies that could be used to overcome or reduce them.

Object Discovery

Over the entire data set, all tabletop objects visible in a percept were successfully discovered. However, due to noise in the depth information of the RGB-D camera patches of table surface are also extracted as object views. As explained in Section 4.4, object views are ignored if the corresponding tabletop object is not detected in the succeeding two table views as well. This procedure removed all but one falsely extracted cluster of object views. The remaining case constitutes an object representation which does not correspond to a real object and it comprises exactly three views of points near an edge of the table. One of these views contains points along the whole edge and therefore overlaps with two smaller views extracted on different parts of the edge. During the experiments, a lot of falsely extracted object views were observed along the edges of a table. This suggests a systematic error that could be investigated. It might make sense to ignore object views extracted by the discovery module if most points of the view are close to the edge of the detected supporting plane. On the other hand, this might remove valid object views of, for example, a pen or other elongated objects.

Another issue with the object discovery module which appeared in practice concerns the detection of points on the table plane. Points on an object which are closer to the table plane

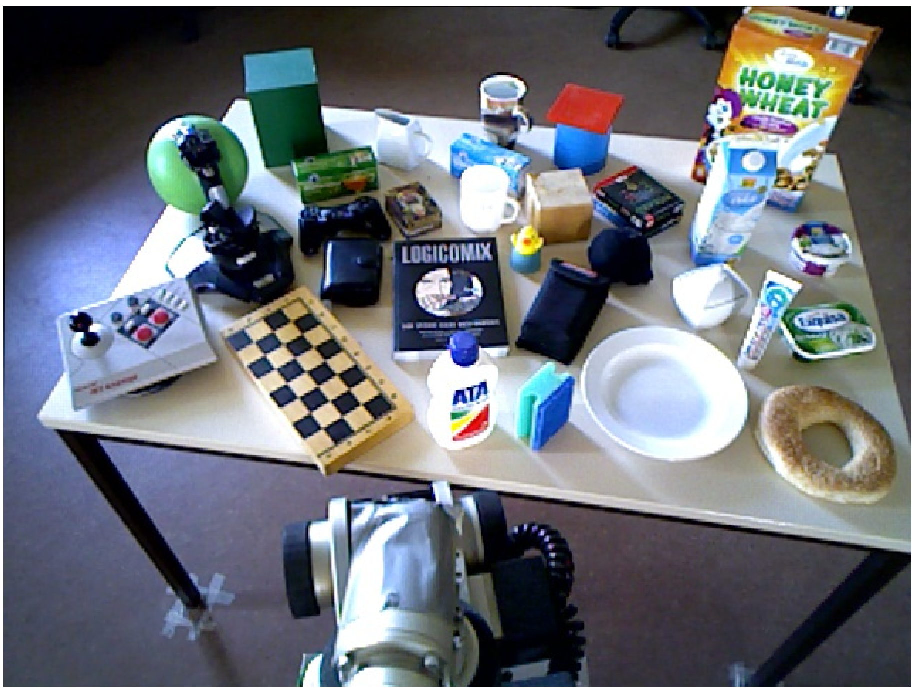


Figure 6.1: The set of objects used for evaluation of the system from the robot's point of view

than the distance threshold of the RANSAC algorithm (2 cm) are considered to be part of the table plane and therefore are not extracted as part of the object view. This is a systematical shortcoming of the implementation and it slightly crops all object views. An illustrating example of this effect can be observed with the aligned object views of the plate in Figure 6.3. Because the bottom of the plate is directly on top of the table, it is not part of the extracted model of the plate. Essentially, the extracted model consists of the plate's edge only. However, the same figure illustrates the tremendous challenge for an object discovery module which tries to resolve this problem: The sesame bagel on the same table indeed has a hole in its center.

View Registration

This thesis proposes a modification of the traditional incremental 6D ICP which improves its performance in tabletop scenarios. Figure 6.2, 6.3, and 6.4 depict registration results for three of the 17 tables in the data set. Each figure exemplarily shows a different behavior of 6D ICP whereas Plane ICP produces relatively stable results.

In Figure 6.2, 6D ICP yielded a significant vertical mismatch in the registration. This is the exact problem that inspired Plane ICP and, accordingly, Plane ICP computed a registration result without this mismatch. Even so, an obvious horizontal mismatch remains.

Figure 6.3 demonstrates a different situation: In the second incremental registration step of the table exploration, the estimated mismatch between the two table views exceeded the maximum correspondence distance of the ICP algorithm. This distance is set to 5 cm in all experiments to avoid (wherever possible) unreasonable correspondences between points on dif-

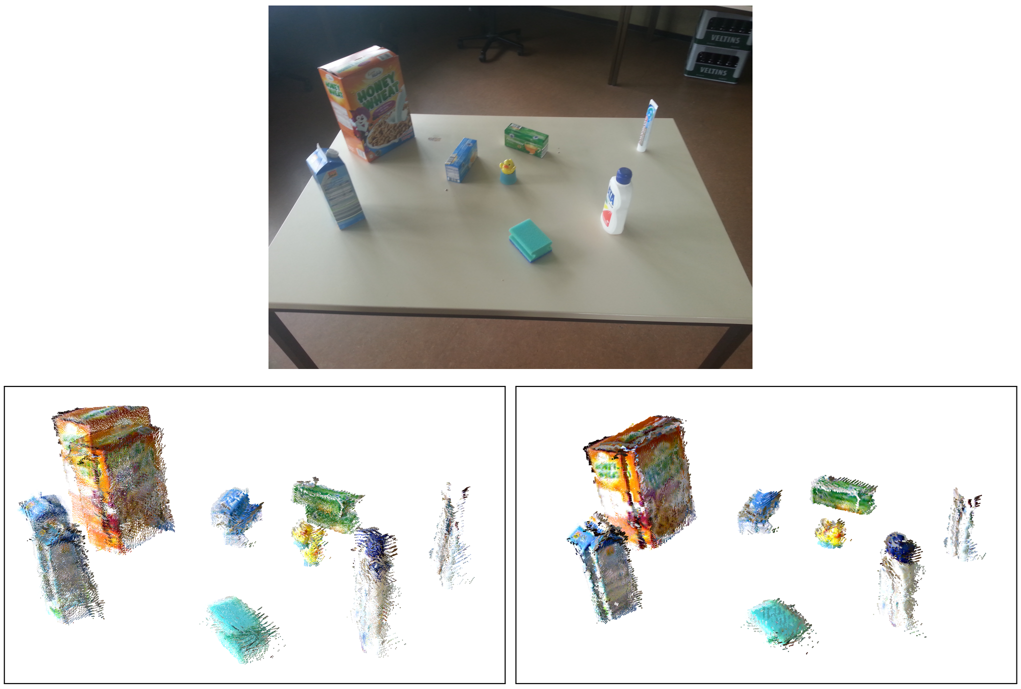


Figure 6.2: Exemplary table arrangement with the results of 6D ICP registration and Plane ICP

ferent objects. While the registration with 6D ICP failed in this case, Plane ICP corrects the vertical part of this mismatch beforehand and the registration succeeded.

The third table setup in Figure 6.4 depicts a scenario where all table objects are of relatively small height. Given roughly aligned point clouds which vary mostly in two of their three dimensions, 6D ICP behaves similar to Plane ICP and generates a comparable registration result.

Table Exploration

To record the 17 successful exploration steps, a total of 47 attempts were made. Hence, 30 attempts failed. This is mainly for two reasons.

Whereas `move_base` is a well-tested framework which successfully navigates a robot in free space and autonomously recovers from a number of error situations, `move_base_straight` often fails to recover from error situations. While the latter often works perfectly fine and is the better choice to approach a table, a problem in the interaction with `amcl` regularly makes the robot turn away from the table at the very last moment during the approach. This way, the robot only registers a very small proportion of the whole table surface (or no table at all) from the respective angle. This breaks the incremental view registration used in this thesis and the next view registration likely fails to correctly align the new view. A more robust implementation of a module which can move the robot without collisions by a specified distance in a straight line would likely reduce the number of failed exploration attempts by two-thirds.

The second source of error is a well known problem with the ICP registration algorithm: The algorithm converges on a local optimum only. In case the estimation of the mismatch between

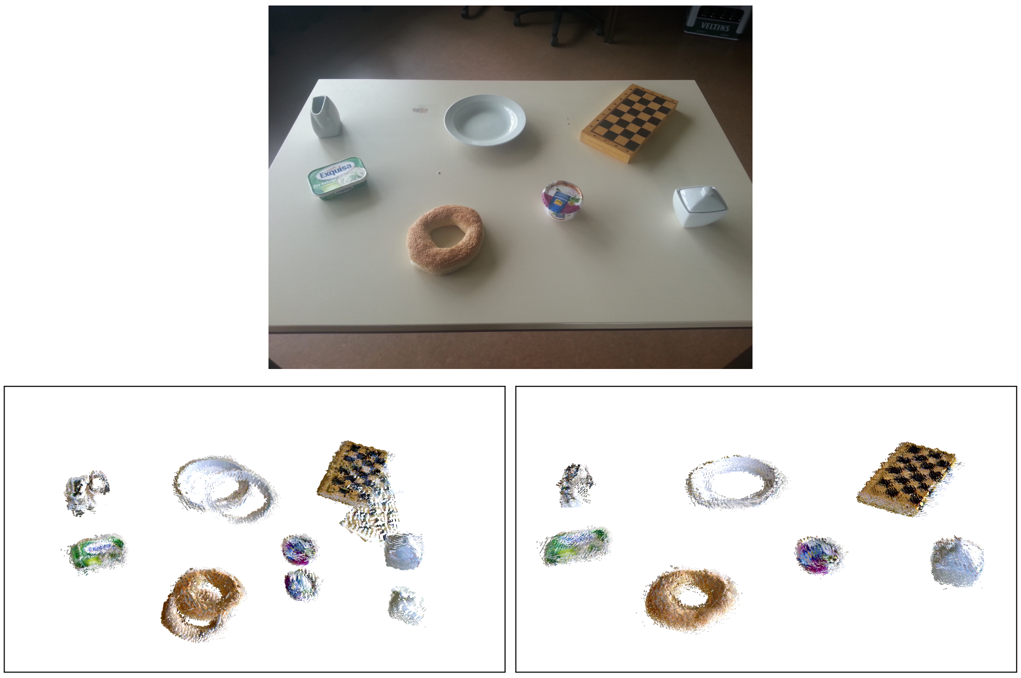


Figure 6.3: Exemplary table arrangement with the results of 6D ICP registration and Plane ICP

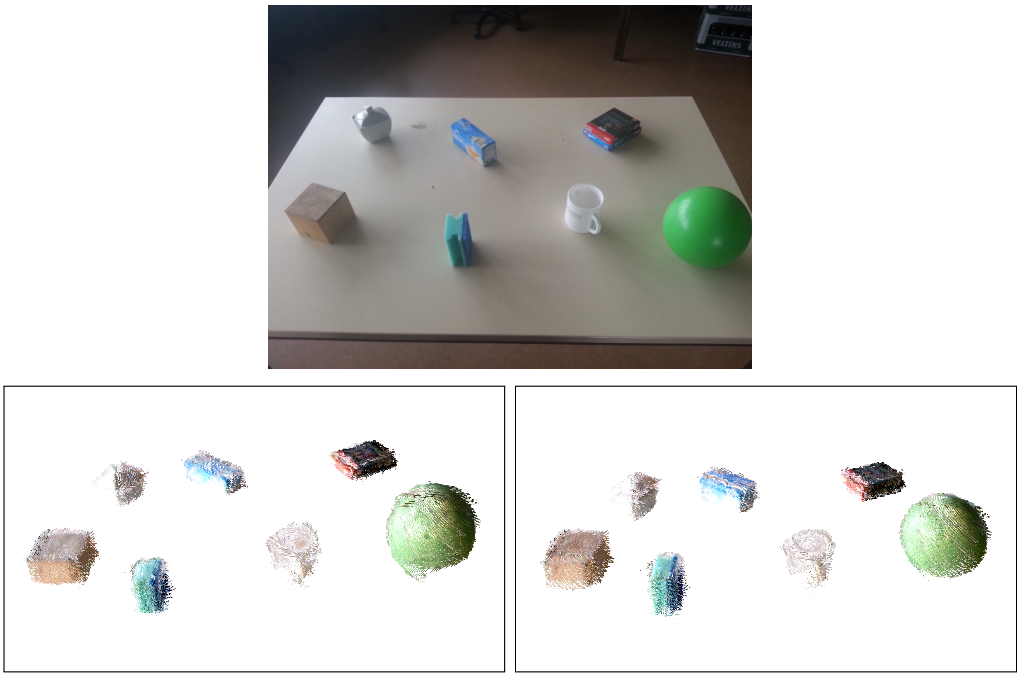


Figure 6.4: Exemplary table arrangement with the results of 6D ICP registration and Plane ICP

the last table view and a new view is too far away from the true alignment, the algorithm often converges on a solution that wrongly associates object views of different objects on the table. In this case, the reconstructed convex hull of the supporting plane suddenly increases in area and the reconstructed object representations contain object views from different objects. Thus, the table exploration failed.

Whereas the proposed Plane ICP algorithm only incorporates the information about the *plane* of a table, an improved registration method might also incorporate the information about its *convex hull* as well to detect or avoid more of these registration failures. Additionally, the navigation framework provides a sampled probability distribution over the current robot pose instead of only the one pose that currently matches the laser scan data best. To account for a more uncertain robot pose while collecting the table views, this distribution could be used to test multiple likely estimates of the mismatch between different table views as initial estimates for the ICP algorithm. Thus, even if the currently most likely pose of the robot results in a wrong view registration, a slightly less likely pose might yield the correct registration. To detect the correct alignment in the different registration results, choosing the registration result which increases the area of the reconstructed supporting plane *least* might prove to be a good heuristic.

Object Recognition

The performance of the object recognition module crutially depends on the distance threshold γ that has to be specified beforehand: With a very low value of γ, each new observed object representation creates a new cluster and no previously observed object representation is recognized due to noisy data and slightly different observation perspectives. A very high value of γ, on the other hand, entails that all object representations become part of the same cluster and the recognition module does not differentiate between different objects at all.

Because the dataset comprises only 17 table arrangements, which is rather few from a data-analyst point of view, I decided not to split off a test set to tune and test the algorithm with specific values of γ. Instead, this section investigates the overall performance of the recognition module with respect to different distance thresholds.

Usually, object classification tasks are evaluated by their *precision* and *recall* on the different classes. These values correspond to the proportion of correct positive classification results out of all positive classification results for a class and the proportion of correct positive classification results out of all positive samples of a class, respectively. However, neither the overall number of classes nor their labels are known beforehand in the recognition module. Hence, these metrics cannot be computed directly to assess the performance of the module with respect to a specific value for γ.

Instead the clustering task in this thesis can be interpreted as a *data-association task*. This way, the actual task is not to form clusters of object representations, but instead to decide whether *each pair* of object representations in the data set is associated (they share the same cluster) or not (they are part of different clusters). Because this is a binary decision problem, it is easy to compute precision and recall for this classifier with different values for γ. These two metrics are also called *pair-counted precision* and *pair-counted recall.* Figure 6.5 presents both metrics for the recognition module as functions of γ. Keep in mind though that pair-counted metrics yield less intuitive results than their traditional equivalent because there is a quadratic dependency between the set of sample object representations and the set of pairs of object representations which is used to compute the metrics: A single wrong association between an object representation and a cluster will influence multiple decisions of the binary pair classifier.

To illustrate the clustering results of the algorithm, Figure 6.7 depicts a number of clustered object views which were generated with a distance threshold $\gamma = 16.4$. While this value still provides a high precision of 0.89, it also demonstrates the first failure cases of the clustering algorithm. Several clusters were formed which contain all (and only those) object representations of the cup, the NES game controller, the sesame bagel and the wooden cube. Because the VFH signature is based only on the geometry of an object and does not include information on the texture or the color of the object, the module confuses a box of peppermint tea bags with a box of camomile tea bags. Apart from their color and texture, these two boxes look the same. Also, object representations of the box of a card game and a sponge were clustered. Although the texture and color of these two objects are quite different, they share the same dimensions. On the other hand, the shorter edges of the sponge are curved whereas those of the box are straight. However, these edges are systematically cropped in all object views because they are directly on top of the table plane and, hence, the object recognition module cannot use them to differentiate between the objects.

In this thesis, I also discussed two possible schemes to extend signature matching for point cloud recognition to allow for the recognition of multi-view object representations. On the one hand, there is the single-linkage “shortest distance” approach which recognizes an object representation as part of a cluster if any signature of an object view of the object representation is closer than γ to a signature that belongs to the cluster. On the other hand, I proposed the “mean distance” approach which only recognizes an object representation as part of a cluster if the mean distance of a high percentage of (but not all) signatures of the representation to signatures of the cluster is less than γ.

To compare both approaches with respect to the data set, I benchmarked both with various values of γ and computed their respective pair-counted true positive and false positive rate. These values describe the percentages of truly associated pairs of object views which are classified correctly and of truly unassociated pairs which are falsely classified as associated. The results of this benchmark are depicted in the ROC curve in Figure 6.6. As can be seen, the described “mean distance” approach strictly dominates the more simple approach and hence describes a more successful scheme to extend distance-based recognition methods for single object views to multi-view representations.

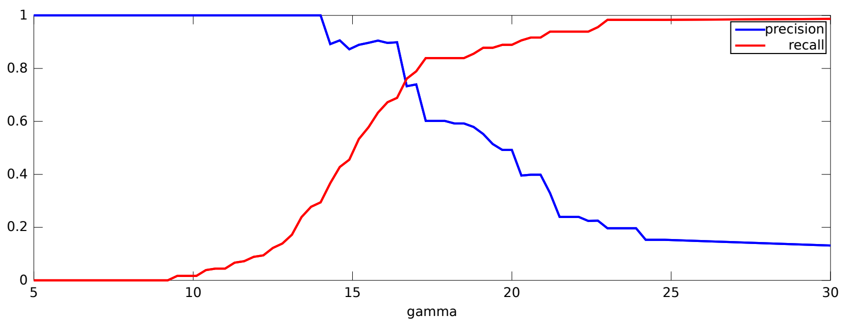


Figure 6.5: Pair-counted precision and recall for multi-view object recognition as functions of the distance threshold γ

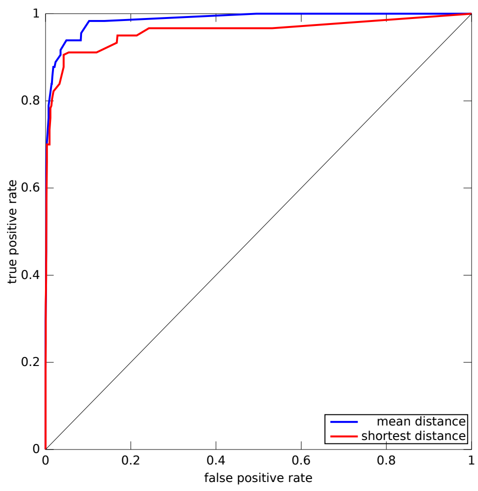


Figure 6.6: Pair-counted ROC curves of the two proposed recognition schemes for multi-view object representations on the data set.

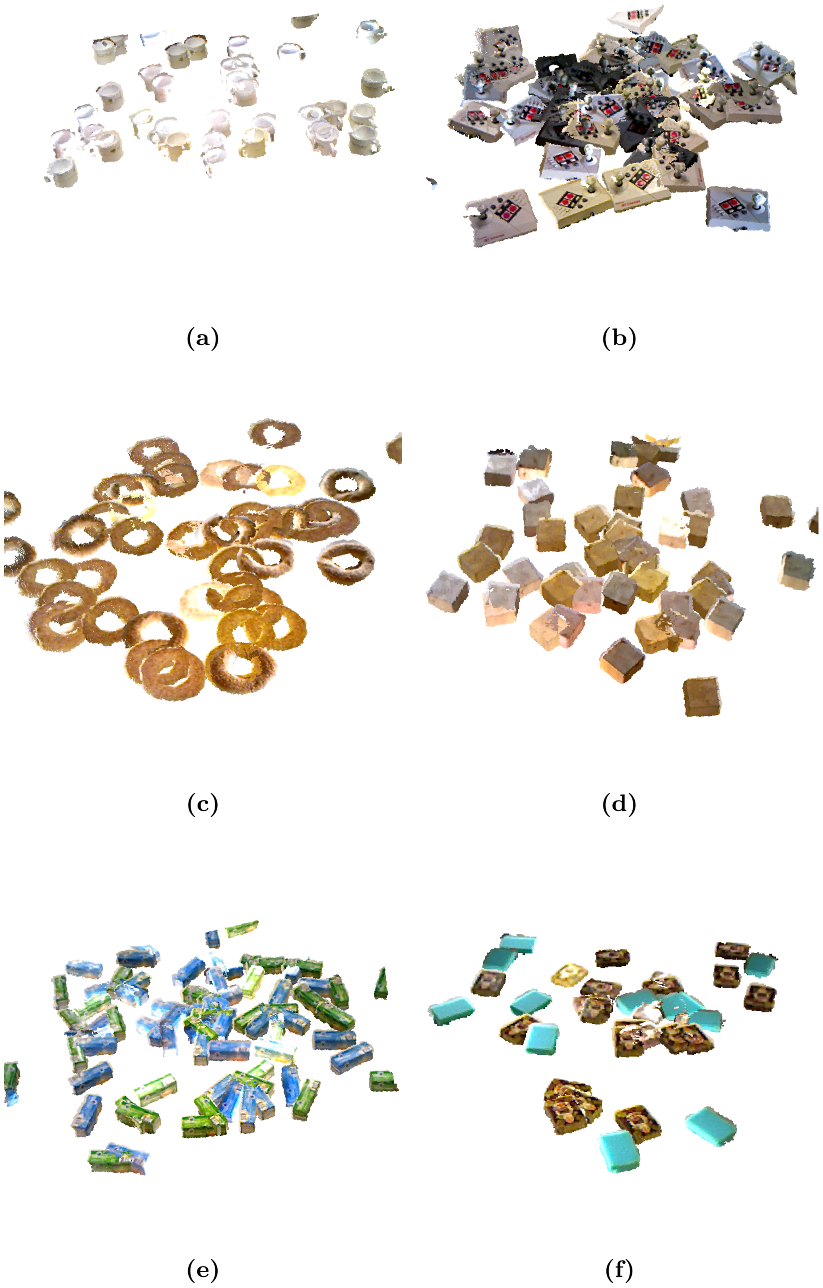


Figure 6.7: Examples of clustered object views with $\gamma = 26.4$. While the first four clusters each represent exactly one object, the system confused (e) different boxes of tea bags and (f) a sponge and the box of a card game

Chapter 7

Conclusion & Future Work

This thesis presented a complete software framework that implements the idea of curious table exploration. By integrating a number of different software systems, the framework enables a robotic platform to explore tables in its environment, discover objects on top of them, and construct simple 3D models of the observable surface of these objects. As demonstrated in the evaluation, the model of an explored object can often be recognized in new percepts and therefore provides a grounded symbolical representation of this and similar-looking objects. Hence, the presented framework could be of use to most autonomous robotics projects which act in mundane environments to (1.) reduce the workload of data aquisition for tabletop objects and (2.) enhance autonomous capabilities of the robots in use. It provides one step towards the goal of taking the programmer out of the loop to use robots in unknown mundane environments.

Because of the integrative nature of this work, there is a multitude of ways to improve on it. I explicitly listed a number of assumptions that restrict the environments in which the exploration module can operate successfully. To enable a robot to explore arbitrary indoor environments, including mundane households and untidy laboratories, future research should aim to overcome some of the software restrictions that require these assumptions.

Also, up to now the framework is only able to explore known environments, because the framework can only explore tables whose position is already known. To work around this limitation an exploration module could be added that searches for suitable tables in the environment.

For most of the topics discussed in this work there exists a lot of background literature and articles on proposed techniques which could improve the performance of the overall framework. Hence, future integrative work could focus on any specific component of the framework and try to exchange it for a different/better/more general/less fault-prone method. Most importantly, the new component should be demonstrated to perform well within the context of the whole framework and *should not* impose new assumptions. This procedure will often point out shortcomings of the proposed methods with respect to autonomous robot setups which then can be discussed and improved upon.

Moreover, this work provides a basis for robotic methods that build or make use of a grounded ontology, i.e. a symbolical representation of the world. Future research with respect to place recognition could investigate the different contexts in which objects appear and form concepts of places in which the same objects appear, aiming for concepts such as e.g. “breakfast table”, “workplace”, “workbench”, “Michael’s table”, “Jochen’s office”, ... By using the context of the explored objects one could investigate categories of objects that appear in similar contexts, aiming to autonomously generate more general grounded categories of terms such as “drinking vessel” or “cutlery”.

Lastly, the autonomous formation of symbols for objects poses a challenge to the field of Human-Robot-Interaction. In order for humans to be able to interact with robots which have explored their environment, these robots need to associate natural language symbols with the objects they explored. Otherwise the example question “Where are my keys?” could not be answered, even if the robot has just observed these keys.

Appendix A

Contributions to Free/Libre Open-Source Software

This work heavily builds on the free-software robotic framework ROS. In preparation for and over the course of this thesis I contributed to various free-software projects related to ROS. The following list summarizes a number of contributions.

- All software I wrote in the context of this thesis is licensed under the GPLv3 free software license
- `object_recognition_ros_visualization`: I fixed a segfault of RViz when a Table message with an empty hull is received.
- `perception_pcl`: I enabled the module to link properly with an overlayed catkin workspace.
- `katana_driver`: The module now supports the current version of the simulation software Gazebo.
- `PCL`: I repaired the computation of normals for sparse point clouds. A problem with the generated normals broke VFH feature estimation.
- `geometry_experimental`: Now, the standard ROS-Python function `canTransform` actually works on real hardware.
- `rviz`: Now, the module builds with the version of Python that is used with catkin where it otherwise failed.
- `rviz`: I fixed a bug in the visualization that broke the module when an empty point cloud was received.
- `frontier_exploration`: Exploration now works with ROS indigo's `costmap_2d` module.
- `boost ublas`: I submitted a patch for a syntax error that was officially released with boost. This error broke the entire boost module.

- `PCL`: I contributed a PCL class for incremental ICP registration that simplifies the application of incremental registration in terms of lines of code and readability.
- `diffdrive_gazebo_plugin`: I adjusted the module to adhere to Gazebo's official interfaces. This removed a number of warnings that polluted the log files.
- `sick_tim`: I submitted a patch to allow the driver for a number of laser scanners to receive multiple laser scans. The module lost many scans when the robot's CPU was busy.
- `object_recognition_core`: I fixed a segfault of RViz when it received OrkObject messages that are not connected to a database.
- `move_base`: The module broke when asked to navigate to a pose with no tolerance. I submitted a patch to fix this problem.
- `PCL`: I contributed a patch to fix a broken raytracing function. A bug made points outside a table polygon appear to be within the polygon (and made a chair appear to be a tabletop object).
- `object_recognition_ros_visualization`: I contributed code to enable RViz to display multiple tables in different colors to differentiate between reconstructed and observed supporting planes.
- various ROS modules: I contributed a number of patches to support boost 1.57+ to `object_recognition_ros`, `ecto`, `moveit_ros`, RViz and others.
- `ecto_pcl`: I contributed two ecto cells to support cluster extraction via indices.
- `PCL`: I submitted a patch that makes the library build with more current versions of VTK.
- `ecto_pcl`: I fixed two bugs that made the framework fail to extract the convex hull of a table.
- `laser_filtering`: I fixed a bug in the module because of which it failed to filter laser scans correctly if they contained NaN values.
- `warehouse_ros`: I contributed a patch to support a modern `libmongdb.so`
- `PCL`: I contributed a patch that fixes the library's usage of `libqhull`. A bug made it segfault on all Non-Ubuntu systems.
- `tf2_ros`: I contributed a patch to allow the `tf` module to handle loops when playing bag recorded data. Before, the library produced a deadlock in this case.
- `catkin-genmsg`: I contributed a patch and a thorough analysis of a problem with partially overlayed catkin workspaces. The patch was not merged, but the analysis provided the basis for a better solution to the problem.

List of Assumptions

Abbreviations

AMCL	Adaptive Monte Carlo Localization
DWA	Dynamic Window Approach
ICP	Iterative Closest Point
IMU	Inertia Measurement Unit
PCL	Point Cloud Library
RANSAC	Random Sample Consensus
SLAM	Simultaneous Localization And Mapping
URDF	Unified Robot Description Format
VFH	Viewpoint Feature Histogram

Bibliography

[AS13] S. Albrecht and Marsland S. Seeing the unseen: Simple reconstruction of transparent objects from point cloud data. In *Proc. RSS, 2nd Workshop on Robots in Clutter*, 2013.

[ATRV12] Aitor Aldoma, Federico Tombari, RaduBogdan Rusu, and Markus Vincze. Our-cvfh – oriented, unique and repeatable clustered viewpoint feature histogram for object recognition and 6dof pose estimation. In Axel Pinz, Thomas Pock, Horst Bischof, and Franz Leberl, editors, *Pattern Recognition*, volume 7476 of *Lecture Notes in Computer Science*, pages 113–122. Springer Berlin Heidelberg, 2012.

[AVB$^+$11] A. Aldoma, M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R.B. Rusu, and G. Bradski. Cad-model recognition and 6dof pose estimation using 3d cues. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 585–592, Nov 2011.

[AWGH11] Sven Albrecht, Thomas Wiemann, Martin Günther, and Joachim Hertzberg. Matching CAD object models in semantic mapping. In *Proc. ICRA 2011 workshop: Semantic Perception, Mapping and Exploration, SPME '11*, Shanghai, China, 2011.

[BM92] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, Feb 1992.

[BWD$^+$00] Joseph E Banta, Laurana M Wong, Christophe Dumont, Mongi Abidi, et al. A next-best-view system for autonomous 3-d object reconstruction. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(5):589–598, 2000.

[CC12] Changhyun Choi and H.I. Christensen. 3d pose estimation of daily objects using an rgb-d camera. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3342–3349, Oct 2012.

[CC13] Changhyun Choi and H.I. Christensen. Rgb-d object tracking: A particle filter approach on gpu. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1084–1091, Nov 2013.

[CS03] Silvia Coradeschi and Alessandro Saffiotti. An introduction to the anchoring problem. *ROBOTICS AND AUTONOMOUS SYSTEMS*, 43:85–96, 2003.

[ecta] Ecto - a c++/python computation graph framework. `http://plasmodic.github.io/ecto`.

[ectb] Ecto bindings for common opencv functionality. `https://github.com/plasmodic/ecto_opencv`.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[FBT97] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4, 1997.

[Fox01] D. Fox. Kld-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[GBS14] Mahsa Ghafarianzadeh, Matthew Blaschko, and Gabe Sibley. Unsupervised Spatio-Temporal Segmentation with Sparse Spectral Clustering. In *British Machine Vision Conference (BMVC)*, Nottingham, United Kingdom, September 2014.

[GSB07] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:2007, 2007.

[Gö15] Michael Görner. Autonomous Tabletop Object Learning. Master's thesis, Osnabrück University, August 2015.

[HS03] E. Hameiri and I. Shimshoni. Estimating the principal curvatures and the darboux frame from real 3-d range data. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(4):626–637, Aug 2003.

[KCF11] Michael Krainin, Brian Curless, and Dieter Fox. Autonomous generation of complete 3d object models using next best view manipulation planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5031–5037. IEEE, 2011.

[KHRF11] M. Krainin, P. Henry, X. Ren, and D. Fox. Manipulator and object tracking for in-hand 3D object modeling. *International Journal of Robotics Research (IJRR)*, 30(9):1311–1327, September 2011.

[LEB12] 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*, Sydney, Australia, July 2012.

[MGC+15] Lu Ma, Mahsa Ghafarianzadeh, Dave Coleman, Nikolaus Correll, and Gabe Sibley. Simultaneous localization, mapping, and manipulation for unsupervised object discovery. In *ICRA 2015*, 2015.

[MM12] Julian Mason and Bhaskara Marthi. An Object-Based Semantic World Model for Long-Term Change Detection and Semantic Querying. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[MMP14] Julian Mason, Bhaskara Marthi, and Ronald Parr. Unsupervised Discovery of Object Classes with a Mobile Robot. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2014.

[MPR+10] Zoltan-Csaba Marton, D. Pangercic, R.B. Rusu, A. Holzbach, and M. Beetz. Hierarchical object geometric categorization and appearance classification for mobile manipulation. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 365–370, Dec 2010.

[NAH+13] Tim Niemueller, Nichola Abdo, Andreas Hertle, Gerhard Lakemeyer, Wolfram Burgard, and Bernhard Nebel. Towards Deliberative Active Perception using Persistent Memory. In *Proc. of IROS 2013 - Workshop on AI-based Robotics*, Tokyo, Japan, 2013.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[RBMB09] R.B. Rusu, N. Blodow, Z.C. Marton, and M. Beetz. Close-range scene segmentation and reconstruction of 3d point cloud maps for mobile manipulation in domestic environments. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1–6, Oct 2009.

[RBTH10] 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)*, Taipei, Taiwan, 10/2010 2010.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. `http://pointclouds.org`.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *INTERNATIONAL CONFERENCE ON 3-D DIGITAL IMAGING AND MODELING*, 2001.

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A modern approach.* Pearson, 2010.

[RSGB09] Michael Ruhnke, Bastian Steder, Giorgio Grisetti, and Wolfram Burgard. Unsupervised learning of 3d object models from partial views. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 801–806. IEEE, 2009.

[Spe90] Elizabeth S. Spelke. Principles of object perception. *Cognitive Science*, 14(1):29–56, 1990.

[SRGB11] Bastian Steder, Michael Ruhnke, Slawomir Grzonka, and Wolfram Burgard. Place recognition in 3d scans using a combination of bag of words and point feature based relative pose estimation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1249–1255. IEEE, 2011.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[TSS10] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *In Proc.of the European Conf. onComputer Vision(ECCV)*, 2010.

[WKLP13] Lawson LS Wong, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Constructing semantic world models from partial views. In *Robotics: Science and Systems (RSS) Workshop on Robots in Clutter*, 2013.

[WV11] W. Wohlkinger and M. Vincze. Shape distributions on voxel surfaces for 3d object classification from depth images. In *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*, pages 115–120, Nov 2011.

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Osnabrück, November 2015