Universität Bremen
Faculty 3, Mathematics and Computer Science

Bachelorthesis

Towards robots executing observed manipulation activities of humans

Ausführung von beobachteten Manipulationsaktivitäten der Menschen durch Roboter

Alina Hawkin
Supervisor: Prof. Michael Beetz PhD
Second Supervisor: Dr. René Weller
Advisor: Gayane Kazhoyan

April 23, 2018

Declaration of Authorship

I, Alina Hawkin, declare that this thesis, titled "Towards robots executing observed manipulation activities of humans" and the work presented in it are my own and has been generated by me as the result of my own original research. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Bremen.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Alina Hawkin

Date

Abstract

Having robots which can accomplish everyday household activities is an idea and dream many people share. Archiving that is not so easy, since just giving robots instructions of what to do, in the same way as we would give to fellow humans, has been proven to be not sufficient enough. There are a lot of implications that come with instructions, which we take for granted, but which are unknown to the robot.
A solution might be to teach robots by demonstration. Until recently however, there was no viable tracking system available, which would allow for natural human interaction and the achievement of precisely recorded data describing the positions of the human's hands and some description of the activities performed.
This however, has recently changed with the development of affordable Virtual Reality systems, which allow precise tracking of the positions of the handheld controllers, with which a human can interact with a virtual environment, being almost completely immersed in it, allowing for very natural and intuitive interaction.
It is explored in this thesis how this newly gained data can be used to teach robots to perform everyday household activities, if it is possible at all with the here suggested setup, and if it is, where the boundaries of it lie.

Abstrakt

Roboter, die alltägliche Haushalts-aktivitäten durchführen, ist eine Idee oder sogar ein Traum, den sich viele Menschen teilen. Allerdings ist dies nicht ganz so einfach zu erreichen, da Anleitungen die beschreiben was zu tun ist, die man normalerweise anderen Menschen geben würde, für Roboter nicht ausreichen. Dies kommt daher, dass Anleitungen viele Aktivitäten implizieren, die wir für selbstverständlich halten, die dem Roboter aber unbekannt sind.
Ein Lösungsansatz ist es, Roboter mithilfe von Demonstrationen zu lehren. Aber bis vor kurzem, gab es nicht wirklich die Technologie, die in der Lage wäre, die Bewegungen eines Menschen so zu verfolgen, dass z.B. die Positionen der Hände so verfolgt werden, dass ein Roboter genau das was der Mensch getan hat, auch nachahmen kann. Außerdem erlaubten die Systeme die es gab, keine wirklich natürliche Interaktion.
Dies hat sich allerdings nun geändert, denn es gibt nun die Virtual Reality Technologie, die es dem Benutzer einerseits erlaubt, immersiv, intuitiv und auf natürliche Art und Weise, mit der Virtuelen Realität zu interagieren. Außerdem ist es möglich sehr präzise die Positionen der Controller nach zu verfolgen, mit denen der Spieler mit der Virtuellen Realität interagiert.
In dieser Arbeit wird erforscht, wie diese neu gewonnenen Daten verwendet werden können, um Robotern beizubringen, alltägliche Haushaltsaufgaben zu bewältigen, ob dies überhaupt mit dem vorgeschlagenen Setup möglich ist, und falls ja, wo die Grenzen dieser Möglichkeiten liegen.

Contents

1 Introduction

1.1 Motivation

In order for the development of robots which assist humans in every day household activities to be possible, the robots have to be able to perform complex manipulation tasks in fairly complex environments. One of the most promising ways of teaching robots to perform such tasks is by allowing them to observe and learn from us humans. However, this requires the robot to be able to recognize which information is crucial to the performance of a task(Bates et al. 2017), and also track the movements of the human. The last aspect is something that is fairly hard to do with cameras alone(Welschehold, Dornhege, and Burgard 2016), and better results can be achieved when using some sort of motion capture system. These systems have, until recently, been rather expensive or troublesome to set up. Some require the usage of full body suits or markers placed on the hands of the user. Luckily, due to recent developments in the field of Virtual Reality, an affordable and precise solution can now be used, namely systems with head mounted displays and hand held controllers, which can be precisely tracked by two base stations(A. Haidu and M. Beetz 2016a).
Now that such technology is available, a way needs to be found and tested to make it usable for robots. There have been different approaches to this, from Virtual Reality games in order to collect data (Kunze, A. Haidu, and M. Beetz 2013), to setting up virtual every day environments and learning from users interacting with them by performing specific tasks, and applying learning algorithms on that data (Bates et al. 2017).

Now that the data can be collected, it has yet to be seen if it can be used for robots in a way so that they can first copy the activity a human has performed within the Virtual Reality into the real world. Which is where this thesis comes in. The approach presented in *Automated Models of Human Everyday Activity based on Game and Virtual Reality Technology* (Andrei Haidu and Michael Beetz 2018) is used in this thesis. The main goal is to see to what degree the recorded Virtual Reality data can be used, if it needs to be adjusted in some way, what issues might occur with the direct usage of the data that one does not expect, etc.
If it would be possible to recreate an action the human performed in Virtual Reality, within a simulator and later in the real world on a real robot, it would be a big step into the direction of robots performing every day activities on their own, since they could be trained on this data, and as suggested by Kunze, A. Haidu, and M. Beetz 2013, this approach could be used to create games, which would help to collect much more data in a fun, interactive and inexpensive way.

1.2 Hypothesis

Can the data recorded in an Virtual Reality environment of a human performing every day kitchen tasks be used by a robot to perform the same tasks? It is assumed that some of the data might be usable, like the positions of the objects within the virtual kitchen can help the robot look at the right location for a certain item. The poses of the hands of the human, or rather, the handheld controllers, could contribute to teaching the robot how to grasp an item it has never grasped before. However, it might also be that the

recorded hand poses are not precise enough to do so.

1.3 Contribution

Functions are written, which acquire data that has been previously recorded in the Virtual Reality environment, from the knowledge system, and process it in such a way that it is usable by a robot. Also, a process pipeline is developed, which leads from the Virtual Reality data to a lightweight simulator of a robot being able to perform the same task that the human performed in Virtual Reality to some degree. It is also tested and evaluated, how the Virtual Reality data can be used, and where it becomes insufficient. These functions are incorporated into the planning level of a robotic system in a way that it is robot platform independent, although for testing purposes, a simulated PR2[1] robot will be used.

1.4 Structure of this Thesis

In the following chapters, the approach taken to make Virtual Reality data usable to a robot will be described, as well as how the data was used. The contribution of this thesis will also be evaluated and tried within a light weight simulator.

Ch.2 Related Work: will describe other approaches to this topic as well as compare them to the approach taken in this thesis.

Ch.3 Foundations: this chapter will present all the tools used within this thesis, and the research on which this approach is based.

Ch.4 Approach and Implementation: will discuss how the tools described in the previous chapter interact with one another, and also how the problems arising in this topic were solved.

Ch.5 Experimental Evaluation: describes which tests have been performed in order to evaluate the presented approach and discusses their results.

Ch.6 Conclusion: will give a brief summary of the topic presented and how it can be improved in future related work. It will also give insight into the hardships which have occurred during the research of this topic.

[1] Willow Garage PR2 robot: `http://www.willowgarage.com/pages/pr2/overview`

2 Related Work

Teaching robots to do tasks based on some form of data from humans is a wide spread idea, and many approaches have been tried already. The general benefits of the method is that instead of hard coding what exactly the robot has to do, a human can show the robot how and when to do what. However, this is not as trivial as it might seem at first glance. In this chapter, several approaches are introduced of the general idea of making robots learn from human beings. Kinesthetic learning is, however, not taken into account, since this way of learning needs the human to directly manipulate the robot into a desired position, and since this thesis's focus is on learning from recorded human positions directly without the human having to touch the robot, that approach is discarded beforehand.

The paper "Learning manipulation actions from human demonstrations" (Welschehold, Dornhege, and Burgard 2016) approaches complex tasks like opening and closing drawers, via teaching the robot through observation of a human. The robot observes the human perform a task with its RGBD camera. The movement of the hand is tracked with the help of a marker, the movement of the object - via SimTrack.[2] The result are two trajectories, consisting of poses, containing translation and rotation data with a time stamp, so that both trajectories can be matched to one another. Welschehold, Dornhege, and Burgard 2016 also implements techniques to interpolate poses for cases where the view of the object or human might have been momentarily obscured, and therefore, gaps in the trajectories occurred.
Additional background information, such as the trajectories have to be smooth or that during manipulation the relative pose of the hand and object is fixed, is being used to formulate a hyper-graph optimization problem. The solution of that problem are two corrected trajectories. Also, the grasp transformation, which is the pose of the hand in the objects frame, is estimated within the optimization process. Based on these optimized trajectories, grasping poses are being computed with respect to the robot's capabilities and the human demonstrations. In this way, gripper trajectories can be gained which then can be used to learn a motion model for the robot.

The experiments compared the use of the observed trajectories directly, without any optimization or adaptation and with the optimization. Without optimization the robot failed to grasp the object since the "human hand pose does not accurately reflect a proper grasp for the robot" (Welschehold, Dornhege, and Burgard 2016 p.3776). However, after applying the optimization and adaptation process, the experiments of grasping objects were successful.

In a later publication (Welschehold, Dornhege, and Burgard 2017), the torso of the human was added to the tracking, so that the robot's base could be moved accordingly. This allows the robot to perform even more complex tasks, since the robot can now position himself according to an action's requirements.

The approach presented in this thesis differs since here no trajectories will be used to perform the manipulation tasks, but only the poses of the objects and the human.

[2] K. Pauwels and D. Kragic. "SimTrack: A simulation-based framework for scalable real-time object pose detection and tracking". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, pp. 1300–1307. DOI: 10.1109/IROS.2015.7353536 from Welschehold, Dornhege, and Burgard 2016

However, an assumption is that grasping might has to be adapted and that the human grasping pose might not be viable to be used directly.

Zhang et al. 2017 uses a Virtual Reality set for teleoperating a PR2 robot in order to perform complex manipulation tasks and to use that data for imitation learning. This kind of learning requires high-quality demonstrations which the VR set can provide, by tracking the hands and the head of the user. In this scenario the human uses the VR headset to perceive what the robot would be able to see with its RGBD camera. The tracked VR set controllers are used to teleoperate the hands and grippers of the robot. "This setup ensures that the human and the robot share exactly the same observation and action space, eliminating the possibility of the human making any decision based on information not available to the robot, and preventing visual distractions [...]" (Zhang et al. 2017, p.1) The data was then used to train deep visuomotor policies, which map pixels directly to actions while using behavioral cloning. (Zhang et al. 2017) The results showed that a lot less data was needed for the imitation learning to be successful then anticipated. Only 30 minutes of demonstration were sufficient.

The difference to the proposed approach of this thesis lies in the means of how the data is processed and how much of it is needed to achieve an manipulation task. If it works, even less then 30 minutes of recorded data would be needed in order for the robot to perform a task. Also, no learning algorithms are applied, but instead the data is being stored and processed differently.

While this thesis focuses on directly performing pick and place tasks on a simulated robot, which have been demonstrated in Virtual Reality, there is another approach in the "On-line simultaneous learning and recognition of everyday activities from virtual reality performances" Bates et al. 2017 paper, which focuses on actually learning what kind of actions the human performed within the Virtual Reality. The setup is similar, as that it uses a HTC Vive headset and two handheld controllers for the user to interact with the virtual environment. Also, the chosen virtual environment is a kitchen. The fundamental idea is similar, that in order "to improve human-robot interaction and the understanding of how to behave in human spaces, it is important to train robots with examples of real human behavior. "(Bates et al. 2017). In this case, their focus lies on the recognition of and learning of the activities.
The task on which this approach was tested was a simple "washing dishes" scenario. In order for this to be possible, a "property system"(Bates et al. 2017) was introduced, which allows objects like e.g. plates, to have a property of being "dirty", while another object, such as a sponge could become "wet" when it came in contact with water, and as a wet sponge, gain the property of being able to make "dirty" dishes "clean" again.
For knowledge representation, KnowRob (A. Haidu and M. Beetz 2016b) is used. (KnowRob is going to be introduced later, within the foundations chapter, since it is also used in this thesis.)
One of the key aspects was the "Continuous Motion Segmentation"(Bates et al. 2017). This is necessary, since it is impossible to know beforehand, what duration an action will have. That feature was needed as part of the system in order to recognize actions. Another addition of the system is a recording and playback system, which allows to play back the data in such a way, that it is not possible to differentiate between a playback and a real time activity, which means, that through the "simulated sensor system for the

robot agent to perceive and interact with the environment"(Bates et al. 2017), the robotic agent can perceive what the user could as if the robot was in the same body.
The results of the action recognition were very good, with 86.8% being the worst trial out of 12. A difficulty for this system was that sometimes a user would perform pick and place actions fairly fast, without really "stopping" their hands when picking up something. This is something that every human does in every day activities, when the objects are fairly lightweight and small. This was happening in the Virtual Reality even more often since it is yet not really possible to accurately simulate the weight of objects in a way that it would influence the users performance. Overall, with a rate of 92% the system was successful at "employing a continuous motion segmentation model and an ontology based reasoning method for classifying activities.

3 Foundations

This section describes all the tools and frameworks that have been used for this thesis, and also the way they have been used. The following figure will give an overview of the process pipeline and which tools have been used for which step of the process. Then, the tools will be explained in detail within this chapter.

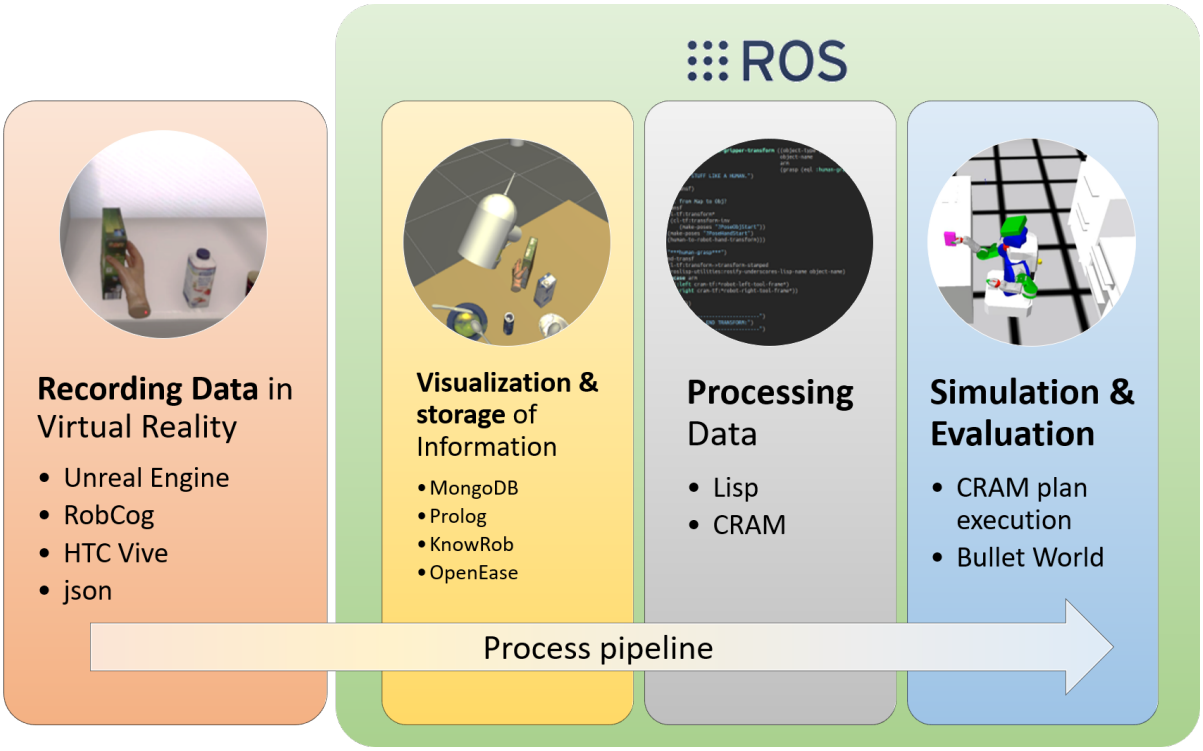

Figure 1: *Overview of the process pipeline, showing the individual steps of the process and the tools used to perform these.*

As can be seen in Figure 1, the first step to make a robot execute the same manipulation actions as a human within a Virtual Reality environment, is to acquire suitable data from the Virtual Reality. For this step the Unreal Engine, which is a game and physics engine, is used. It allows for a 3D environment to be set up, within which the human can move and interact with 3D-objects via an HTC Vive headset and two handheld controllers. The RobCog package allows to record these interactions in a way that they can be processed further, namely as json(JavaScript Object Notation) files, which are structured text files, containing a textual object representation describing them in attribute-value pairs and arrays.

All the following tools are interconnected with one another through the ROS (Robot Operating System) framework. ROS allows to write programs in a wide selection of programming languages, so the language which is most efficient for a task can be used. Such a program represents a node within the ROS system. One ROS master manages all the nodes, and their connections to one another via topics, which have specific types and to which nodes can subscribe or publish data.

For the second step, the previously recorded data is put into the MongoDB, which is a

database. There it can be used by KnowRob, which is a knowledge processing system, capable of reasoning about the data and extract crucial information from it for the robot with the help of queries written in prolog, which is a language based on logic, consisting of rules describing the relations of individual objects, which then can be referenced to acquire information. KnowRob serves also as the backend to OpenEase, which adds a web interface on top of KnowRob allowing for management of different experiment data and its visualization.

In the third step the data is processed in CRAM, which allows to write high level robot unspecific plans since it infers the low level data needed at run time, using Common Lisp. In the last step, the CRAM plans are executed and visualized within the CRAM bullet world, which is a lightweight simulator.

3.1 Recording Data in Virtual Reality

According to the Virtual Reality Society "[...] virtual reality entails presenting our senses with a computer generated virtual environment that we can explore in some fashion."[3] For this thesis the virtual environment used consists of an interactive 3D kitchen, which was built similar to the real robot laboratory kitchen and imported into the **Unreal Engine**[4] , which is a game engine providing physics to the objects, meaning that they can bump into one another, fall or even be stacked on one another. The Unreal Engine allows to run the kitchen simulation in Virtual Reality using an **HTC Vive** set, which includes a headset and two handheld controllers, of which the positions can be tracked with the help of two base stations within a designated area. The positions of those controllers are mapped to virtual human hands, with which the user interacts with the Virtual Reality Environment. The approach of this thesis is based on AMEvA (Automated Models of Everyday Activities), which is a "special-purpose knowledge acquisition, interpretation, and processing system for human everyday manipulation activity." (Andrei Haidu and Michael Beetz 2018 p.1) The core idea is to use modern simulation-based game technologies in order to acquire commonsense and naive physics knowledge. AMEvA is able to create and simulate virtual everyday living environments, such as kitchens, apartments or other living or working rooms. The simulations are photo-realistic and the contained objects posses physics, in the sense that drawers can be opened, objects can be stacked on one another or knocked over, etc. The environment is trying to make the interaction experience for the user as natural as possible. The system is also able to record all the manipulation activities the human performs within this virtual environment, and the effects of these activities on to the environment. AMEvA can also segment the recorded data into categories of motions and represent these activities in KnowRob on a symbolic level. The acquired knowledge can describe not only what has to be done, but also how it can be done. For example, how to set a breakfast table for two people. The robot could learn what objects are needed and where they need to be placed, what the objects most likely location is, for example, a milk carton would usually be in the fridge, how to interact with these objects, where to position oneself in order to be able to see and

[3]Virtual Reality Society:`https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html`

[4]UnrealEngine:`https://www.unrealengine.com/en-US/blog`

grasp these objects, is it worth to put the things on a tray first and carry it over etc. In order to record such data and have a human interact as naturally as possible with the environment, a Virtual Reality setup is used.

The recorded data can be visualized and interacted with via OpenEase(which will be de described in more detail in the "OpenEase" section of this chapter.). Some table setting experiments were conducted and the data is now available online in OpenEase.[5]

Adding new objects into VR

Five objects were chosen for this thesis on which the human and eventually the robot, would try to perform grasping and manipulation tasks. The objects are: a cup with no handle, a muesli box, a bowl, a small bottle of milk and a plastic fork. Manipulating an object is challenging in it's own way. The cereal box might be the easiest, since it is rather big and square, allowing for good perception and a lot of surface to grasp it at, and can also be grasped in different ways, e.g. from the front, back or the top side. The milk carton is also square-like, but smaller and symmetrical. The cup is even smaller and can be grasped at the top rim as well as from the side. The bowl has to be grasped at the rim, since it is too wide to fit into the human's hand or the robot gripper, while in order to grasp the fork, a pinch grasp is necessary.

Figure 2: *View of the Virtual Reality setup of the kitchen, and the objects to be interacted with, from left to right: cereal box, small milk carton, cup, bowl and a fork.*

Recording the VR data

With the VR setup a human can navigate inside the virtual kitchen and perform pick and place tasks on the selected objects, as if they were in a game. There is a version

[5]OpenEase online:`http://data.open-ease.org/user/sign-in`

of the VR kitchen where one can navigate with a keyboard and a mouse (Andrei Haidu 2018), but since that would give not quite as natural human positions as a VR set could, the later is being used instead. The human can move around within an area inside the laboratory and virtually navigate around in the virtual kitchen, and interact with the objects with the help of the two controllers, where movement navigates a set of virtual hands. The human can then pick up objects by using the trigger on the back side of the controller to close the virtual hand. However, the physics here are not very accurate in the sense that the object snaps to the hand rather then being grasped. This can lead to weird grasps, which are possible in the virtual environment, but, are not possible for the real human hands and not to mention impossible for the robot's grippers. Every move the human makes is recorded, and stored in a .json file. The start setup, meaning where the objects are in the beginning of an episode, and the position of the kitchen itself, every shelf and every drawer, are stored as an .owl file. Together, these files contain everything that happened during the recording session. The logged data can be categorized into symbolic and subsymbolic data. The first meaning all the contacts of objects with one another, grasping and touching events, closing or opening of doors and drawers, and the later the position and orientation of every part of the virtual world and all the objects within it. Every recording session, produces an **episode** of data, which is basically the .json file containing all the symbolic and subsymbolic data. The episode is organized in Events. An event can be, for example, the human touching an object or grasping an object. The according events would be *TouchingSomething* and *GraspingSomething* events.

3.2 ROS

ROS (short for Robot Operating System) is a middle ware for writing robot software and is the component, which allows the communication between OpenEase and CRAM to take place. ROS provides a message passing interface which has to be implemented by all modules in a clean and simple way. The messages which are passed through topics can be synchronous, in which case they are called services, and asynchronous in which nodes can subscribe to a topic to listen to it and process the data, or publish the data themselves. Nodes are essentially the programs a developer gets to write. The ROS Core or ROS Master has to manage all these topic communications, and also the global parameters, which can be used throughout the system. The advantage of such a modular system is that one can choose to use a node or package providing a certain functionality, or one can develop it oneself, in order to understand the algorithms behind it. The packages can be written in different languages, as long as they support the communication interface required by ROS. This gives a lot of flexibility to developers, since instead of having to compromise on one language in which to write an application, each module can be written in the language that fits the task best. Some of the currently supported languages are Python, C++, Lisp, Java[6] and Lua.

ROS also provides many useful tools, such as rviz[7], which allows for visualization of the robot, sensor data and map. The tf2[8] package, which keeps track of all the coordinate frames of the robot and it's environment, and allows the user to compute transformations

[6] Java used to be supported but the support ended recently. The last ROS versions to support it are ROS Kinetic and ROS Indigo. Source: `http://wiki.ros.org/rosjava`

[7] rviz: `http://wiki.ros.org/rviz`

[8] tf2: `http://wiki.ros.org/tf2`

between frames. The later being crucial for this thesis, as seen in the implementation section.

3.3 Visualization and Storage of Information and Data

Since OpenEase as the system responsible for managing, visualizing and reasoning, has a lot of dependencies and tools it is build on top of, it's most important components will be explained first in this chapter. Then, OpenEase itself will be explained. However, an overview of this system can be seen in the following figure:

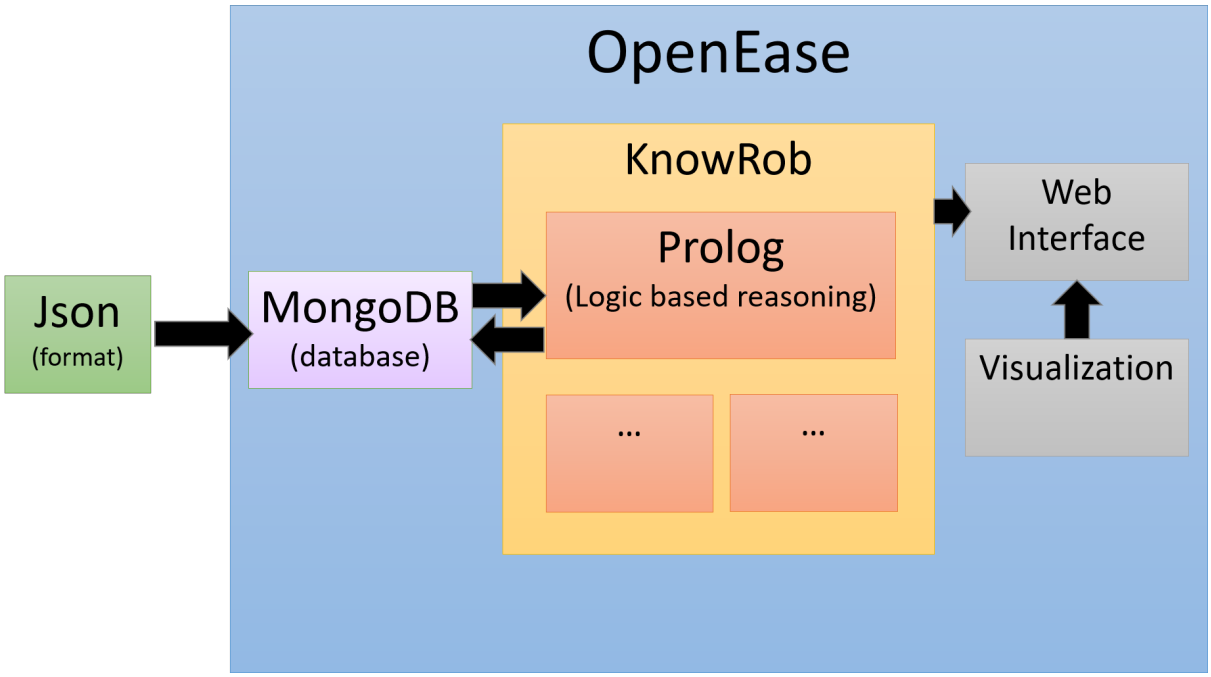

Figure 3: *Overview of the inner structure of OpenEase and the components most important to this thesis. KnowRob possesses more components then are listed here, but describing all of them would go beyond the scope of this thesis.*

Json and Bson

As briefly mentioned previously, json(JavaScript Object Notation) is a text based type, which describes objects with attribute-value pairs and arrays. It is language independent format, and is the format used to record the Virtual Reality data. Bson(binary json) is the binary version of the json format, used by databases to represent data structures, such as objects in the form of maps (key-value pairs).

MongoDB

MongoDB is a database which stores documents in the bson file format, but can read json and bson. It is used as the main container to hold the data which has been recorded in the Virtual Reality.

Prolog

Prolog is a programming language which is based on logic. Programs written in prolog can be seen as databases, which consist of facts and rules. A question in the form of a prolog query can be posed to the program. A result is, essentially, the logical deduction of the rules within the database. If the result is nil, then the database does not contain enough data and does not express enough relations, so that no result can be found.

Json Prolog

As the name suggests, there exists an interface between json and prolog, so that json files can be used by prolog and therefore reasoned about. This means that json terms can be automatically converted into prolog.

KnowRob

KnowRob is the knowledge system of the robot and the backend of OpenEase. It provides knowledge representation, allows for reasoning methods and for acquiring new knowledge. It contains all the information the robot has about the world around him, the objects within it and how these objects have to be handled in order to achieve a plans goal.

KnowRob is the question answering system for the robot. Json-prolog queries used from the planning CRAM level, get resolved by this system. This is how the data about all the episodes can be extracted and used inside of planning. KnowRob can, based on the episode data, provide the answers to questions such as “when did the human grasp an item?”, “Where was the human’s hand while he was grasping?”, “Where was he looking?”, “Which hand was used?”, “What object was grasped?”, “What was the position and orientation of the hand during the grasping process?”, “When did this event occur?”, “When did it end?”
All of these questions can be answered by KnowRob and based on this data extraction, the planning module can make the robot act. KnowRob is implemented in Prolog[9], while the knowledge within KnowRob is represented in the *Web Ontology Language*(OWL[10]) (A. Haidu and M. Beetz 2016b).

3.4 OpenEase

OpenEase is a web based service, which provides data recorded from robot or virtual reality experiments in a way that can be visualized and queried with the help of json-prolog and KnowRob. The main idea is that robots from all over the world could contribute experiment data, and other robots could learn from this data to perform tasks, they have never done before. OpenEase can be used by robots via a WebSocket API or by humans using a browser-based interface, which allows for the visualization of data and for knowledge quiring. The queries can provide answers to questions like: “what the robot

[9]SWI-Prolog: `http://www.swi-prolog.org/`
[10]OWL: `https://www.w3.org/OWL/`

saw, reasoned, and did as well as how the robot did it, why and what effects it caused."
M. Beetz, M. Tenorth, and Winkler 2015

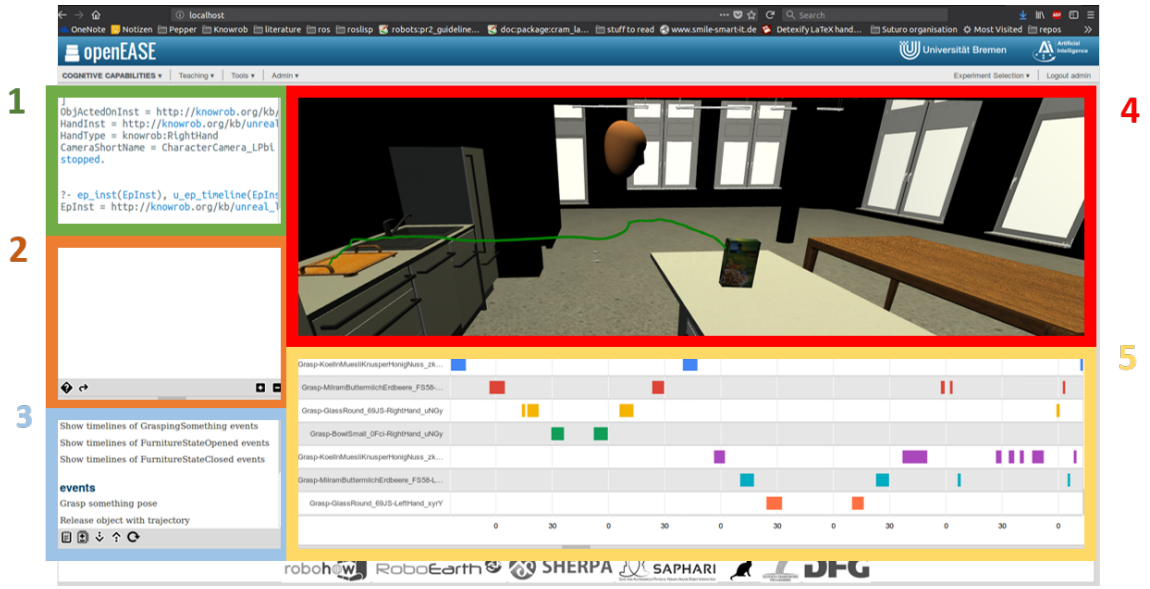

Figure 4: *OpenEase webinterface. containing most important views for this thesis, which are: 1. result of posed queries, 2. queries editor, 3. predefined queries, 4. visualization of the semantic map and events, 5. visualization of the timeline, describing when which event happened and how long it lasted.*

For this thesis, a dockerised version of OpenEase and KnowRob was used.

With the help of a rosbridge[11] ROS node a remote connection can be established from the dockerized OpenEase to the local ROS system, so that queries can be asked and their results can be used within other ROS nodes.

Whenever new data is recorded in the Virtual Reality, it has to be imported into OpenEase so that it can be used for planning. To achieve this, MongoDB is used, and will be explained in the MongoDB chapter.

KnowRob Addons - KnowRob Roslog Launch

This is the package which allows OpenEase to communicate with the rest of the ROS system. It is needed since OpenEase is run within a docker container, which is separated from the rest of the system, and therefore can not interact with the rest of the ROS system directly. Internally, it uses the rosbridge[12] to create the connection.

KnowRob Addons - KnowRob RobCog

This package is the foundation for this thesis and provides two main functions at different ends of the system:
First, it provides a plugin for the Unreal Engine, so that the virtual environment build

[11]rosbridge: `http://wiki.ros.org/rosbridge_suite`
[12]rosbridge: `http://wiki.ros.org/rosbridge_suite`

with it can be saved as a semantic map, which then can be loaded in OpenEase and KnowRob and queried for information. RobCog also makes it possible to record the Events happening in the virtual reality in the form of time lines and .json files, which also can be imported into OpenEase and queried for information. The "relevant abstract Events"A. Haidu and M. Beetz 2016a to be recorded, determined "by checking for specific contacts from the physics engine."A. Haidu and M. Beetz 2016a The physics engine in this case being the Unreal Engine.
Second, as an KnowRob addon, it allows KnowRob "to reason on recorded episodes."A. Haidu and M. Beetz 2016a It also provides specific queries which allow to parse and visualize the data for OpenEase.

3.5 CRAM - Cognitive Robot Abstract Machine

The Cognitive Robot Abstract Machine (CRAM) is a collection of distinct but interconnected tools, which enables robots to perform manipulation tasks and have cognition-enabled control. CRAM uses ROS as a middle-ware. CRAM has lightweight reasoning mechanisms, which allow the robot to infer decisions instead of them being preprogrammed. The main tool behind this functionality are the *designators* Kazhoyan and M. Beetz 2017 which essentially are symbolic descriptions of actions, locations or objects. *Process Modules* Lorenz Mösenlechner 2016 provide an interface to lower-level control processes and generate the parametrization for the low-level control routines, while taking the current belief state into account. Rittweiler 2010

CRAM is used for high-level planning since the designators allow for a high abstraction level. Also, it is used to process the data which is received from OpenEase through KnowRob using prolog queries from the Lisp level. Also the calculation of the grasping transforms for the robot are done with lisp and CRAM.
Since CRAM is mostly written in Common Lisp[13], most of this thesis is also written in Lisp, since CRAM is the most used tool. Also in order to combine all these different tools, the planning level seemed to be the reasonable level for the approach.

CRAM Json Prolog

cram_json_prolog is an implementation of a ROS json prolog client in lisp, which allows the use of json-prolog within lisp. Meaning, that prolog queries in the json format can be sent from lisp via ROS to KnowRob. The result can then be processed further in lisp and fed into the plans and designators.

Roslisp: cl-tf

Roslisp is the library allowing one to write ROS nodes in lisp[14]. *cl-tf*[15] is the lisp implementation of the ROS tf package. It is used for everything that requires some sort of pose or transform manipulation or creation. It is mostly used here to create poses and transforms with a translation consisting of three values as a 3D-vector, giving the

[13]Why Common Lisp was chosen, can be found here `http://cram-system.org/doc/package/why_lisp`
[14]Roslisp:`http://wiki.ros.org/roslisp`
[15]Cl-tf:`http://wiki.ros.org/cl_tf`

coordinates x, y and z, and a rotation, which is a quaternion consisting of x, y, z and w values. It is also used to convert these poses and transforms, into pose-stamped or transform-stamped, which means that in addition to the translation and rotation, a time stamp has to be added, and in the case of the later, the frame in which this transformation is. Further, transform multiplication can be performed thanks to this package, and also the calculation of the inverse of a transform is possible. The exact usage of this package can be seen in the Implementation section.

CRAM Designators

CRAM designators are entity descriptions which contain key-value pairs. These are used to represent a symbolic description of components for a function or skill that the robot should be able to perform. The advantage and strength of the designators is that no concrete parameters have to be specified when writing the designators, since these parameters can be automatically inferred by the system, thanks to rules which have to be specified by the user. The inferring of the parameters is called "[de]referencing" Kazhoyan and M. Beetz 2017.

There are three types of designators defined at the moment, although more can be defined by the user if needed:

object: the key-value pairs are used to describe objects in a abstract aka. symbolic way, e.g. what kind of type is the object, what can this object be used for, how does it look like, how can the robot interact with this object,etc. It is used in this thesis to represent the objects the robot is interacting with. The data describing the type of the object is derived from OpenEase, while the designator itself is resolved by the perception system.

location: used to describe locations in an abstract way. Can take constraints into account, e.g. is the object visible, can it be reached by the robot from that position, can the robot drive and reach this location taking the robot's size into account, are there collisions etc.

action: used to symbolically describe an action the robot should perform. It can be any manipulation, navigation or detection action. The referenced designator is then passed on to the process module for execution. Internally, prolog is being used as an inference engine in order to convert the symbolic description into subsymbolic data that makes sense for the robot.

All these designators combined can describe most of the tasks a robot might need to perform. For this thesis, these three are more then enough and no other designator classes need to be defined.

In order for these designators to be able to obtain the grounded data needed by the process module and the lower level functions to perform a certain task, the designators need to be resolved, which means that the necessary parameters and values need to be generated

from the key-value pairs given in a designator. Since there are currently three kinds of designators, there are three ways in which a designator can be resolved, depending on the type.

Object designator resolution: essentially, since this designator represents an object with which a robot needs to interact, in order to obtain any information about such object, the robot needs to perceive it with its sensors. Meaning this designator represents an interface to the perception system of the robot. The perception system will not be explained here further, since it is a huge topic on its own, and goes beyond the scope of this thesis. It can be viewed as a black box which allows the robot to determine where a certain object is within the map by looking at it and what kind of object it is. Within the CRAM bullet world, the position of any object can be determined easily by simply asking the world where it is. In the real world, complex perception frameworks and algorithms need to be used, such as RoboSherlock, which "can answer tas-relevant queries about objects in a scene, boost object recognition performace by combining the strengths of multiple perception algorithms, support knowledge-enabled reasoning about objects and enable automatic and knowledge-driven generation of processing pipelines."M. Beetz et al. 2015a This kind of designator is also used within this thesis in order to represent the object the robot is interacting with. The type of the object is given to the designator as a symbolic description. After the detecting action, which essentially executes the perception, the designator contains an object instance which describes the perceived object, with it's position, type and name.

Action designator resolution: the goal of this resolution is to convert the key-value pairs which represent the symbolic description of an action, into parameters for an action which is executable on a specific robot. In order for the resolution to be possible, an inference engine is used, which in this case is prolog. This means that rules need to be specified, in order for prolog to be able to find a solution. These rules are contained in fact groups. If a match is found and the designator gets resolved, it can be passed on to a process module, which handles the robot specific execution. Action designators were used for most of the tasks the robot has to perform for this thesis, like navigating to a table, detecting an object of a certain type, grasping and placing it.

Location designator resolution: a location designator has to convert the symbolic description given into an actual pose within the environment which is reachable for the robot base or the grippers. Determining such a pose can be fairly complex, since the given symbolic representation could be something among the lines of "a position to stand in order to open the fridge". It has to be determined, where the fridge is within the kitchen, how the robot needs to stand in order to be able to open the fridge, since it opens to a side, the robot cannot stand right in front of it, but has to stand slightly at a side, which side does the fridge open to, how close can the robot be to the fridge in order to be able to open the fridge door enough to be able to reach within it with the other arm etc. all of these constraints have to be taken into account when calculating the pose, and probably even more.
There are two steps in order to obtain a valid pose. First, a generation function is implemented by the user, which generates candidate poses in a lazy list. The second step is a validation function, which checks if a generated pose is a valid solution and fulfills all the requirements employed by the constraints. Both of these steps can be fairly

complicated, depending on how autonomously the pose calculation shall be. In the case of this thesis, the poses are calculated depending on the ones the human has used in the virtual reality. Therefore their calculation is not too complex for the simulation. This will be explained in the implementation section.

Examples for Designators: For better understanding of the designators, the following example is given:

Listing 1: example: action designator

```
(exe:perform
        (desig:an action
                  (type picking-up)
                  (arm ?arm)
                  (object ?obj-desig)))
```

This is a designator of type *action*. The action to be executed is a *picking-up* action. The arm is set by a variable *?arm* and an object designator is passed as a variable, *?obj-desig*. In order to obtain the object designator, a perception action can be performed in the following way:

Listing 2: example: action designator containing an object designator

```
(exe:perform (desig:an action
                       (type detecting)
                       (object (desig:an object (type ?type)))))
```

This action designator would perform an action of type *detecting*, which means that the perception needs to be asked for the data. The other parameters given describe what kind of object the perception system should look for, which is set by the *?type* variable.
A CRAM plan would consist of multiple of such designators, depending on what needs to be done. Since they are defined in this very abstract way, the same CRAM plans can be executed in the bullet world and the real robot. The only thing that needs to be changed, is that in the beginning of the plan, the following line of code is set:

Listing 3: example: projection environment

```
(proj:with-projection-environment pr2-proj::
   pr2-bullet-projection-environment
[...]
```

which means that the plan is supposed to be executed in the simulation. For the real world, this line has to be changed to *with-real-robot*, and the same plan would be performed on the real robot. This allows for the assumption, that everything tested within the simulation, will most likely also work in the real world.

CRAM Process Modules

The CRAM process module is the component, which essentially makes it possible to write high level plans independent of the robot's hardware, since the process module

is the level which separates robot specific instructions from the abstract and unspecific high level commands of the plan. This means that this is the component which sends the commands to the lower level modules of the robot, which for example, control the navigation, perception or manipulation. For the high level plans which contain the designators, the process modules are black boxes, which get an action designator as input. This designator is then resolved into low level commands and parameters needed by, for example, the navigation module, which performs the desired action of, in this case, moving the base of the robot to a desired position. If this was successful, the success can be returned to the higher level as such. If the action failed, the failure will be passed on to the high level plan, which has called this action. It can then try to execute the action again with a slight change of parameters. Depending on the implemented failure handling, the action can be retried several times with different parameters, for example, different navigation or grasping poses. If it still fails then, the action is deemed unreachable and an error can be thrown. This approach is only viable for non critical actions, where it can be safely assumed, that a failed execution won't harm the environment, robot or human interacting with the robot.

CRAM Bullet World

The bullet world of CRAM is a lightweight simulator which can be used to test software/plans written in CRAM. It can use the same semantic map as OpenEase, and the same object meshes can be used, which makes it ideal for testing. Also bullet world can represent the internal belief state of the robot, and therefore can be used in real time by the robot to perform testing and evaluating of plans, before actually executing them in the real world.
This light-weight simulator is crucial for this thesis, since it is used as the main tool for the evaluation. It allows to see if all the positions and transforms for the objects and robot have been calculated correctly, so that, for example, grasping and placing positions are reachable for the robot. It was also very useful for the debugging process, since the self modeled axes of the multiplied transforms can be visualized, allowing to see what might be wrong in the transform multiplication chain.
The strength of CRAM and bullet world, lies in the fact that real world plans written and tested on the real robot, can be executed in the bullet world and plans written and tested in the bullet world, can be executed on the real robot without having to make any adaptations within the plans. This is of course not a guarantee that everything that works in the simulator will work in the real world, since no physics like friction or other forces (except gravity) are simulated, but since the bullet world environment uses a semantic map which tries to replicate the kitchen environment of the real world as close as possible, it can be assumed that the plans which seem to work within the simulation, have a reasonable high probability to work in the real world as well.

4 Approach and Implementation

4.1 Architecture

The system consists mainly of three parts: the Virtual Reality, OpenEase and CRAM. In the Virtual Reality, the human can perform pick and place tasks with five predefined objects; A cereal box, a small milk carton, a cup, a bowl and a blue fork, in a virtual version of the kitchen from the IAI Lab. The Virtual Reality is set up using the Unreal Engine. The performance is being recorded using an HTC Vive setup, consisting of a headset and two handheld controllers, which are being precisely tracked within a predefined area. This data is recorded, and represents a so called *Episode*, which is imported into OpenEase, so that it can be queried via prolog. CRAM is then used to process this data, meaning mostly to use it in higher level plans to make the robot perform pick and place actions. These plans are then visualized and tested in the bullet world simulator.

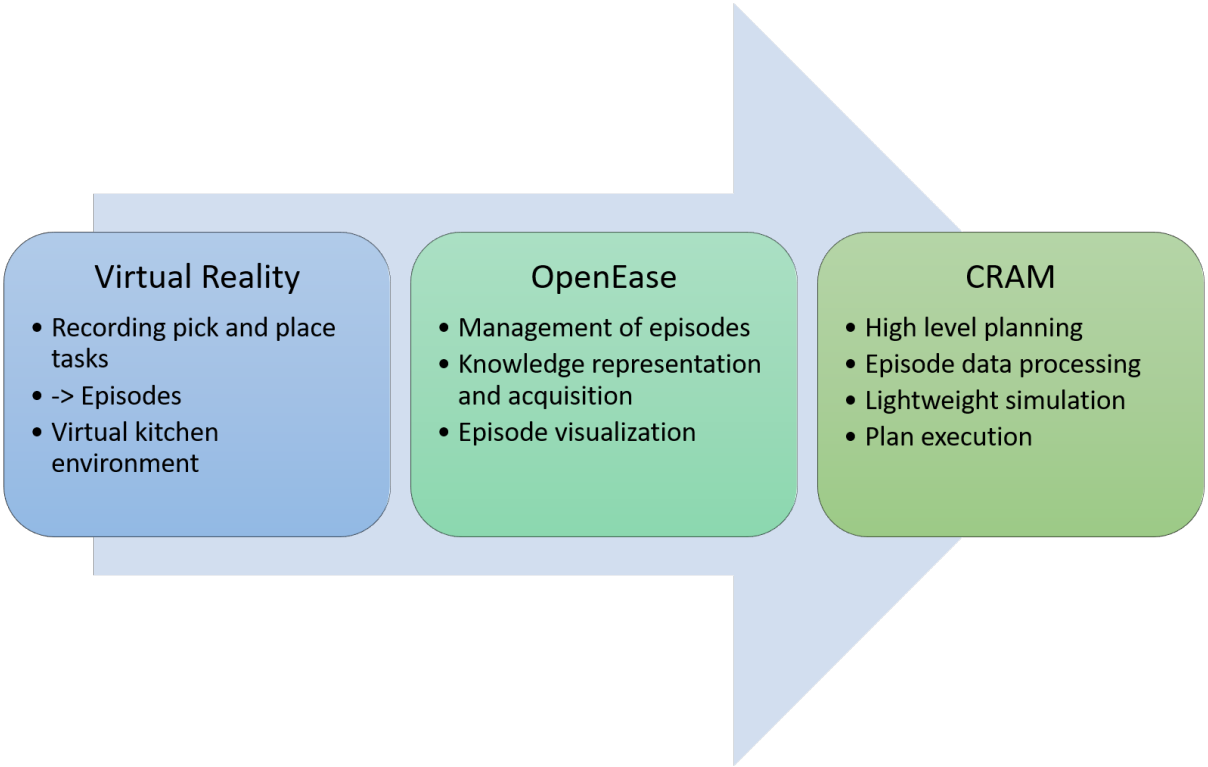

Figure 5: *Overview of the three main systems on which this thesis is founded.*

4.2 Implementation

In the following it will be explained, what the solutions are to the problems and tasks that were faced in the research process of this thesis, and how they were implemented. This chapter will also answer the question of "How is the data read out and adapted, so that it is usable for the robot?"
In brief: The Data is read out from the MongoDB it is stored in, with the help of KnowRob running within Open Ease. Since the dockerbridge is active, KnowRob can be queried even from the emacs/Lisp level with the help of JSON Prolog. The Event Data can be

read out, the grasping events, who grasped which object, with what hand, where did they drop the object, what was th position of the object, what was the position of the Actor, etc.

4.3 Generating Episode Data

The data that was recorded in the Virtual Reality consists essentially of a .json and a .owl file. In the latter, SemanticMap.owl the setup, meaning the initial state, of an experiment is recorded: which objects exist in the scene, which objects are connected to one another (ex. drawers to cupboards), where are the objects within the scene, how large they are, how are they oriented, what exact entity of a specific type of object exists in the scene and what is its ID, what is the path to the cad model used to represent this object. The semantic map can be created with the help of the RobCog package directly from the Unreal Engine graphical user interface. If a semantic map already exists in the same path, no new map will be created unless the old one is removed or renamed.

Figure 6: *This picture shows a part of the Virtual Reality setup. All objects which are interected with can be seen.*

The RawData_ID.json file contains the exact positions of everything in the semantic map at a given time. Since recording the positions of everything at every point in time would make the data unnecessarily huge, only changes are recorded. This means that the data can be organized in time lines, which show how long an object was participating in an event. An Event occurs, when the position or state of an object changes. For example, when the human grasps something within the virtual reality with one of the virtual hands, a so called “GraspingSomething” event is created, describing which object has been acted on, by which hand, when this interaction started, when it has stopped and the position and orientation of the object acted upon, the hand used for that interaction, and the camera/head of the human performing this interaction. If the object is moved during the time of interaction, all its poses are tracked in certain time intervals, which makes it possible to visualize trajectories of movement. There is also an event called

"TouchingSomething" which is used to describe what is attached to what.
The "GraspingSomething" events are the ones used most in this thesis.

Import of the Episode Data into OpenEase with MongoDB

In order to be able to use the episode data, which means being able to query for its contents using prolog queries, it first needs to be imported into OpenEase. This can be done by placing it into the "episodes" folder. The folder contains all of the episode data, which is sorted by the kind of the experiment, e.g. "Chemical-Laboratory", "Pancake-Making" etc. The directory for this thesis was named "Own-Episodes". It contains the "set-clean-table" folder, meaning the data inside are episodes, in which a table is set up for an event, in our case breakfast, and cleaned again. The episodes then are organized in folders named "rcg_a, rcg_b" etc. This naming convention comes from the episode data the thesis was started with, which was not self recorded, and has been taken over. Also this level contains a "queries.json" file, which consists of all the prolog queries which are supposed to be seen and suggested in the visualization of OpenEase. There, they just appear as clickable entities. Once clicked, the entire query unfolds in the OpenEase editor window and can be modified by the user. The same queries can be applied to all of the episodes. The queries from the already given "Virtual-games/clean-table" episodes, have been slightly adapted for the use of this thesis. They were mainly used to check if the data was correctly imported or not. All the self written queries were written within the scope of the lisp implementation of prolog and will be therefore discussed later in the thesis.
The next directory level contains the semantic map file, which has been described above, and the RawData_ID.json or .bson. Now, the recorded episode data is originally a .json file, as can be seen in many of the other episode folders. However, if it reaches a size of more then 16mb, it can not be imported into the database, and errors would get thrown by MongoDB. Upon further inspection of the .json file, which can be done with the sublime[16] editor - yes it has to be sublime since other editors might not be able to open such a huge .json file and will crash or take a very long time to load - one can see that the data in it is readable, but also well spaced. This means that for almost everything a new line is used, which uses a lot of space. This is not necessary, since as a human, one usually doesn't need this file to be readable anyway, and this is something that can be used to compress the file. Also, it is necessary to split the file into multiple ones, which then are small enough to be imported directly into the MongoDB. The following solution was found to split the files with just one line of shell commands:

```
cat ../RawData_ydPt.json | jq -c -M '.' | split -l 2000
```

The "cat" command is used to open the file and direct it's contents into the "jq"[17] command, which is a command line filter program for .json files. The chosen flags are "-c" which removes the pretty print and forces each json object on one line, "-M" disables color output and the "." is the identity operator, meaning that everything in the file will be passed to this function, and not just a specific field. "split" allows to split the file into multiple ones, each having a size of 2000 lines, which is set with the "-l 2000" parameters.

[16] sublime editor:`https://www.sublimetext.com/`
[17] command line JSON processor:`https://stedolan.github.io/jq/manual/`

The pipes concatenate the commands, so that the result of one can be fed to the other. After this is done, one ends up with multiple files names " xaa, xab, xac etc." All of these need to be imported into the MongoDB database, with the help of the "mongoimport" function:

```
mongoimport --db s3Tg --collection vr --file xac --jsonArray
```

The "–db" parameter denotes the name of the database, into which the file shall be imported. A good solution was to use the ID tag of the Episode, in order to later on still know which database belongs to which episode. Since each database can contain multiple collections, the name of the collection needs to be specified after the "–collection" flag. Otherwise, the name of the file is going to be used as the collection name, and one will end up with multiple collections instead of the needed one. The name should be the same for all the files of one episode, but otherwise it does not matter what exactly the name is. The "–file" flag specifies which file needs to be imported into the database, which are all of the ones received from the split command in the precious step. The last flag, "–jsonArray" lets the MongoDB know that the file is structured as an array of json objects, instead of the usual json format. If this flag is not set, errors will occur.

The last step is to dump the newly created MongoDB, in order to receive a .bson file:

```
mongodump --db s3Tg
```

Within the database dump, the .bson file needs to be copied into the episodes folder. It also needs to be renamed into RawData_ID.bson so that OpenEase would recognize it without any issues. This way, one can make sure that all the other files which came from the recording, which will be explained in the next section, can be linked to one another and that no errors will occur.

The other data which was recorded and needs to be loaded into OpenEase, is contained in the "EventData_ID" directory. It contains an event data owl file, which describes when a "GraspingSomething" event took place and which entities participated in it. It also contains property and class definitions. The other file is a visualization of the time lines, which can be viewed with a browser and where all the events are represented graphically.

After all of this, the only thing left to do is start OpenEase, log in as admin, and through the GUI *admin* → *mongo* → *synchronization* synchronize the "Own-Episodes" data.

If the queries performed in OpenEase give all results except the poses of the objects, the solution is to mongoimport the split files again but this time into the MongoDB which OpenEase uses, which is only visible when OpenEase is running. The import can be done as following:

```
mongoimport --db Own-Episodes_set-clean-table --collection
  RawData_cUCM --file xak --jsonArray
```

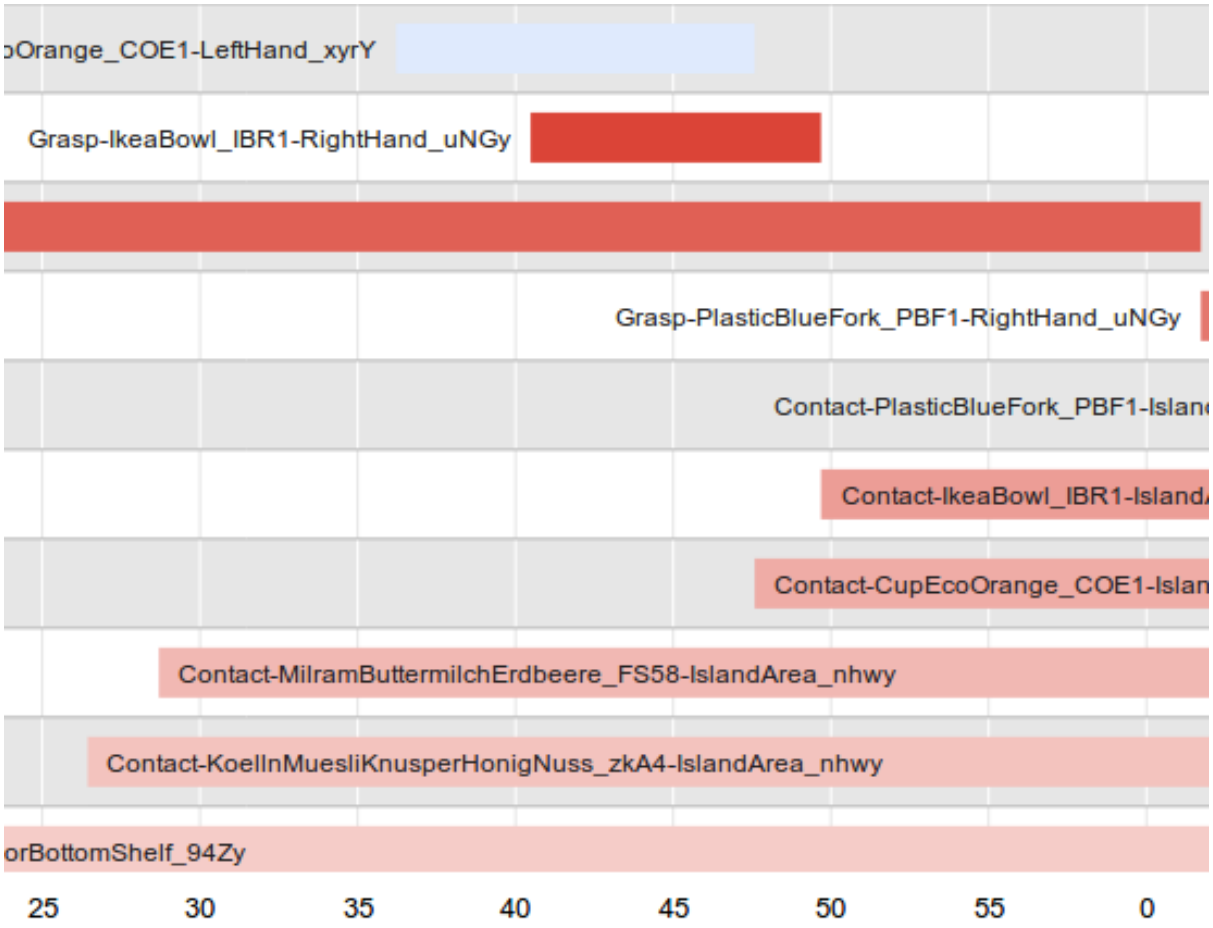

Figure 7: *An excerpt of a timeline, which shows when e.g. a bowl was grasped, and the milk had contact with the island area of the kitchen.*

All the commands have been previously explained. The only change is the name of the database. It could be that this error only occurs, due to a different MongoDB version thann the one OpenEase used when it was developed. However, there was no time to look into this error further, and it was also not needed, since a solution - even if probably not the most elegant one - has been found.

Usage of the Episode Data

First, the necessary data needs to be obtained via prolog queries within lisp. Before this can be done, a connection between OpenEase and emacs, the IDE used to write and execute the lisp and CRAM code, needs to be established. After starting up all necessary nodes, which are the *knowrob_roslog_launch knowrob_ease.launch* node, allowing OpenEase to reach beyond docker and access the local ROS system, and *roslaunch cram_bullet_world_tutorial world.launch* node, which launches the bullet world simulator node.

The “init-set-clean-table” function starts a new ROS node through emacs, and sends the necessary queries to prolog in order to have OpenEase load the necessary episode

data. E.g, the ros package is registered, which is the "knowrob_robcog" package, the episode data is loaded and the owl file is parsed. Then, a connection to the MongoDB is established and the map-markers are initialized. This setup is basically a summary of the setup OpenEase itself suggests, when for example, loading the "Virtual-Games, clean-table" episode, just that the queries were put into one function and adapted in order to load own files.

Accessing stored Data

The query to obtain all the necessary data from OpenEase, basically asks for an event instance within the episode instance, of type "GraspingSomething", when it started and when it was over. With the help of the these points in time, it can then ask what an actors pose was, within a certain episode instance. The actors in this case are the hand the human used to pick up an object, the head of the user and the object on which the grasping action has been performed. The results are the poses of these objects, for both points in time, meaning the beginning and the end of the interaction. The poses at the beginning of the interaction, are used to determine all the necessary grasping poses. The poses at the end, are needed for placing. The result of this query is one big lazy list, which is a list that can be expanded once needed, meaning only the first set of positions and objects are shown until the list is expanded further, containing all the information prolog has inferred through the queries.

This list is saved in two global variable $*orig-poses-list*$ and the first entry of the list is saved into the variable $*poses-list*$, so that it can be acted upon directly, without having to deal with the entire list. The first entry contains all the necessary information to perform one pick and place task on one object.

However, since the robot should perform pick and place tasks for more then one object, more then just the first set of poses need to be accessed. Therefore, there exists a function which allows to ask for other elements of the list, which would contain all the necessary information to pick and place another object.

This approach was chosen since quering OpenEase takes a lot of time. This way, it needs to be done only once. A different approach would have been to query for each object separately, which potentially would be a lot slower.

Adapting the Data to the Robot

Now that the episode data is available within lisp in form of a lazy list element, the necessary information needs to be extracted from it and converted into a form which can be useful for CRAM's high level plans and the simulation environment, which is the bullet world. The first thing to do would be to extract the translation and rotation of an object. The position and orientation of an object arrive in the form of one list with all the seven values in it. Therefore, it needs to be parsed in order to become a transformation CRAM can work with. Also, OpenEase uses a slightly different way of storing the quaternion which describes the orientation of an object. Instead of the order " x y z w", which CRAM

and cl-tf use, OpenEase uses “w x y z”. This means that the quaternion has to be ordered correctly first, before it can be used. Otherwise the orientation will look very weird within the simulation and be unusable.

The Bullet World itself

Another issue was that the semantic map of the Virtual Reality environment, was different to the one used within the bullet world simulator in the sense that the tables were positioned at a different position and the entire map seemed to be rotated around 180°. In order to fix that, the transformation which describes this offset, needed to be found and multiplied with all the other transformations used, meaning the ones that describe the positions of the object interacted with, the human hand, the camera... etc. In order to find this offset, cereal boxes were placed on the edges of the kitchen island table, and milk containers on the table with the sink.

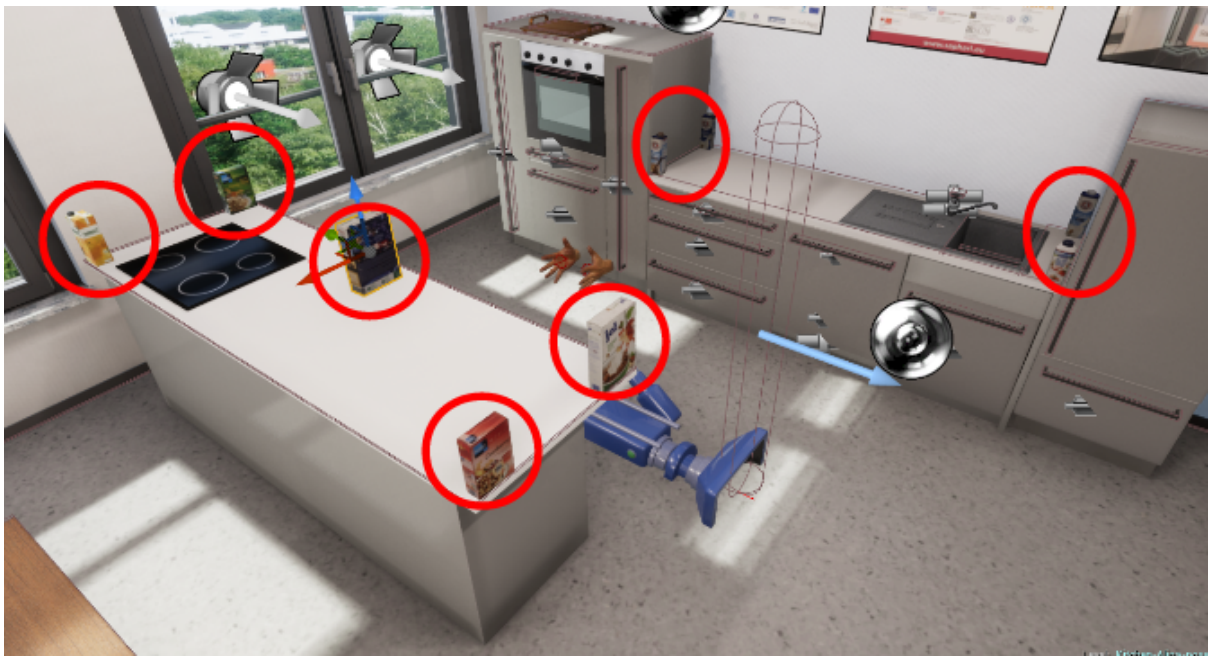

Figure 8: *Shows all the cereal and milk containers placed at the edges of the tables in order to find the offset of the Virtual Reality environment to the bullet world. The red circles mark the objects meant, since some of them are quite small and hard to see in the picture.*

The via experimentation and trial and error found transformation adds an offset for $x : -2.65$ and $y : -0.7$ to the transformation received from OpenEase. This offset describes by how far the bullet world’s map origin is set at a different point then the one of the map used in the Unreal Engine for the Virtual Reality. If this offset was not to be set, objects that were spawned in the Unreal Engine on a table, would float in mid air in the bullet world. Also, the poses are rotated around the z axis for pi, which means 180°.

It was also found that the kitchen island is placed at a slightly different distance from

the rest of the kitchen in the Unreal Engine map then the bullet world one. For placing poses, a solution was found, which basically includes the following transformation chain: ${}_{map}T_{objectPlacingPose} = {}_{map}T_{bullet_table} * ({}_{map}T_{openease_table})^{-1} * {}_{map}T_{objectPlacingPose}$
This needs to be applied to the placing position, in order to make it relative to the bullet world table, instead of the OpenEase one.

Finding the right Positions for the Robot

Moving the robot within the bullet world, is a bit more tricky. This is due to the Virtual Reality data only being able to track where the hand held controllers and the head aka. the headset on the human are. The pose of the feet is unknown, but it can be derived from the pose of the headset, which is called $PoseCameraStart$ or $PoseCameraEnd$, depending on if it is the pose the human started the grasping event with or where it has already ended. Since the head of the human is not on the floor, it's transformation is going to contain a value for the z coordinate, which is not needed for the robot, since the transform needed has to be on the floor and not somewhere floating in the air, so that the robot base can reach it. Therefore, the pose of the camera can be projected onto the floor, so that the base of the robot can move there. Simply, the z coordinate is removed.

In the case of the quaternion describing the orientation, it is a bit more complicated. The rotation which needs to be removed, is the head looking "down" on the object, since the robot cannot tilt itself into the floor. So the quaternion is converted into a matrix with the help of the $quaternion -> matrix$ function from the $cl - tf$ package. Then the rotations x and y around the z axis are extracted, and with the help of the $axis - angle -> quaternion$ function, a rotation around the z axis calculated by the arc tangent of the y and x rotation taken from the matrix earlier. This results in a quaternion which only consists of the rotation around the z axis, which is important so that the robot can face the right direction. This quaternion is then used in the returned pose, to which the robot can safely navigate.

Another feature is that a $*human - feet - offset*$ can be added to the x coordinate of the pose. This was useful in the very beginning of this thesis, since back then only the already recorded, older Virtual Reality data was used, and there the human recording it happened to stand very close to the table. And since the humans feet are a lot smaller then the robots base, often times the robot would be colliding with it's base and the base of the kitchen counter. Later when new data was recorded, it was no longer an issue, and is currently just set to 0, but was kept in the code for now, just in case it might be needed again.

How to "human-like" Grasp an Object

In order for the robot to be able to grasp an object, three poses need to be calculated. Two pre-grasp-poses, which allow the robot to approach the object, and the grasping pose itself, which describes where the gripper has to be positioned and closed. This is usually a transform which is calculated from the position of the object relative to the robots base, to the robots tool center point, which is a virtual point in the middle of the

robot's gripper. Usually, for every object the robot should be able to grasp, there are functions which include hard coded transforms, on which the grasping is based. There are separate functions, depending on what kind of grasp to perform. Kinds of possible grasps are based on which direction the robot's gripper should approach the object, for the example of a cereal box which faces the robot with the thinner side, there would be a front grasp. A back grasp, if the robot can try and reach from behind the box and a top grasp, meaning grasping from the top. Which grasp is chosen, depends on the position of the box, and on the rules which give preference to certain grasps and of course the context of the situation. All of these grasps need to be predefined, which can result in many very similar but slightly different functions.
In this approach, the grasp which the human performed within the Virtual Reality is supposed to be used instead. For this, the transformation of the human hand from Virtual Reality can be used. In this case, the differentiation between top, back or front grasp, is no longer needed, and is replaced just by a "human-grasp."
The most important part of grasping, is the $get-object-type-to-gripper-transform$ function, since as the name suggests, it calculates the grasping transform depending on the type of object, which is going to be important very soon.

$$_{object}T_{robotStandardGripper} = (_{map}T_{object})^{-1} * {}_{map}T_{humanHand} * {}_{humanHand}T_{robotStandardGripper}$$

This transform is then converted into a stamped transform, with the given parent frame name being the name of the object, the child frame is the robot's fingertips frame which depends on the hand. The time stamp is once again 0.0. From this grasping pose, the pre-grasping poses are being calculated, by just adding offsets to the transformations depending on the object's type. These offsets create poses slightly in front of the desired end grasping pose. Meaning the robot approaches the object.

4.4 Using Knowledge from Humans to Execute Plans on the Robot

The plans to execute the pick and place task, use CRAM designators in order to determine the low-level data needed, to perform the action needed. However, the underlying functions on which the designators base their results are all the ones described above.
With these plans, the robot is able to perform pick and place tasks, meaning that he can bring an object from the kitchen sink counter, to the kitchen island table.

4.5 Useful Utilities: Debugging with Axes

There is one special object, which was self modeled with blender for this thesis specifically - and that is the axes object. It is used to view the axes of a certain pose and the orientation of a quaternion. It played a major role in the debugging process, when it came to calculating and debugging transformation, since one could now see them, and therefore figure out in which direction an object needs to be rotated, and if it is facing the right direction. Since the bullet world allows to set colors only for one object entirely, letters were added to the tips of the axes so that they could be told apart.

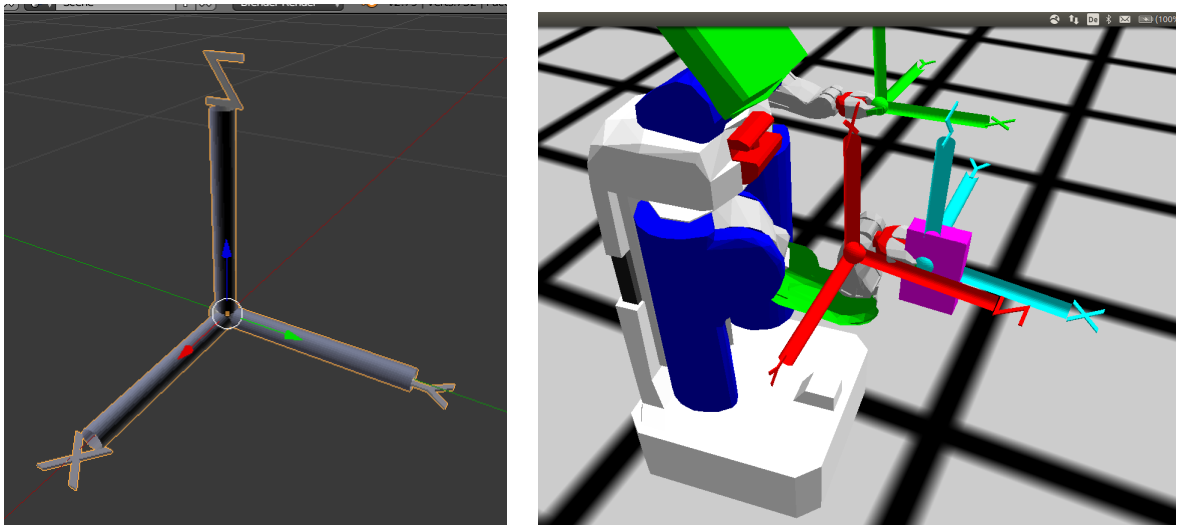

Figure 9: *On the left: The axes object which has been used for all the debugging of the transformations, which has been created entirely for that purpose. On the right: How the axes were used to visualize in order to determine if the grasping transformations were correct.*

5 Experimental evaluation

5.1 Simulation

Different kinds of experiments were performed within bullet world in order to prove this approach. The following tables are organized as follows: *1* means the pick and place action was succesful, *0* means it has failed. If a *C* is added to the result, it means that there was a potential collision, for example the elbow of the robot was inside the table after placement of the object or similar. *P* means that the grasping was successful, but the placing failed. For example, the bowl fell seemingly through the table. *W* means that something else unexpected happened, for example, the object was turned upside down while carrying, and lastly - means the object was not present in this Dataset. The used dataset aka. episode is named *rcg_**

Sign	**meaning**
1	Success
0	Failure
C	Collision
P	Grasping successful, Placing failed
W	Successful but weird
-	Object not present in the recording
U	Object unreachable
V	Object not in view, detection had to be repeated.

Table 1: Overview over the different result possibilities and their shortcuts.

5.1.1 Experiment 1 - Using the data from VR directly

First it was tested, if the robot can pick-and-place the objects using the poses from the Virtual Reality directly. Meaning, the objects to perform the actions on, were exactly in the same locations as in the virtual reality, with only the offsets added which were needed to match the bullet world kitchen environment to the Virtual Reality one.
The data includes picking objects up from the kitchen sink table, and placing them on the kitchen island.

rcg_eval1:
Cup: The grasping of the cup was successful, but it was turned upside down for carrying. The elbow was slightly brushing the table and the end of the placing pose.
Cereal: Works, but since the position for placing the cereal was fairly far within the

Object	**milk**	**cereal**	**cup**	**bowl**	**fork**
Result rcg_eval1	1	1C	1CW	1P	1
Result rcg_eval2	1C	1	1W	1	1
Result rcg_f	1C	1	1C	0C	-

Table 2: Results of performing a pick and place task with different episode data.

table, the elbow of the robot ended up within it.
Bowl: Pick and place both worked, but the bowl was placed on the very edge of the table. It even looked like it was slightly placed within the table.

rcg_eval2:
Fork: The gripper looks like it is partially in the table.

rcg_f:
Cup: Collision of robot base with the furniture, meaning the human was standing too close.
Bowl: It "worked" but the bowl was grasped "above" itself. It was close enough to work in simulation but this would be impossible to work on the real robot with the real environment.

5.1.2 Results of Experiment 1

One can see that most of the pick and place tasks seem to be successful, at least to some degree. The first two sets, the *rcg_eval1* and *rcg_eval2*, are the ones recorded most recently, while *rcg_f* was recorded a while ago. This is important, then the more cycles of recording data and testing it in the simulator are being made, the more does the user performing these tasks adapt to what the robot seem to need in order to be successful.
Overall, it seems that this approach is working. However, there are still a few collisions between the elbow of the robot, and the table happening, mostly during placing. These can be avoided by setting more constraints to the low-level motion planner which calculates these poses. Mostly those collisions are the result of the human standing fairly close to the table - humans have shorter arms then the PR2 robot - and the simulated PR2 using a grasping pose which puts his elbow in front of him. The hight of the object also plays a role, since the biggest object, which is the cereal box, seems to be a lot more collision free and successful then the smaller objects, where the robot has to ultimately place his arm and gripper a lot lower and closer to the table.
Some of the grasps however, look a bit imprecise. Meaning that when e.g. the bowl is grasped, it looks like the gripper comes too far down to grasp it, and misses the rim. One could argue that this is due to the nature of the simulator not really having physics behind it when executed the way it is now, so that the robots gripper can go through objects, like the bowl or table. On the other hand, the argument could be that the real robot would get his gripper rightfully "stuck" on the rim of the bowl and could not grab past it.
It could also be argued that this might be a simple calibration issue, where the pre-grasp-pose offsets are not correctly set.
It shows also, that the grasps within the Virtual Reality environment have to be fairly precise to begin with and this is something that comes with the experience of the user, since when not knowing how exactly the robot would grasp things, one would tend to grasp "however and faster it seems to work" rather then keeping in mind the position and orientation of the axes of the hand.

One feature of this approach is, that the object orientation does not matter. In the *eval2* dataset, the objects were rotated. This would usually mean that the robot has to decide between using a front or back grasp, depending on the objects rotation, but as

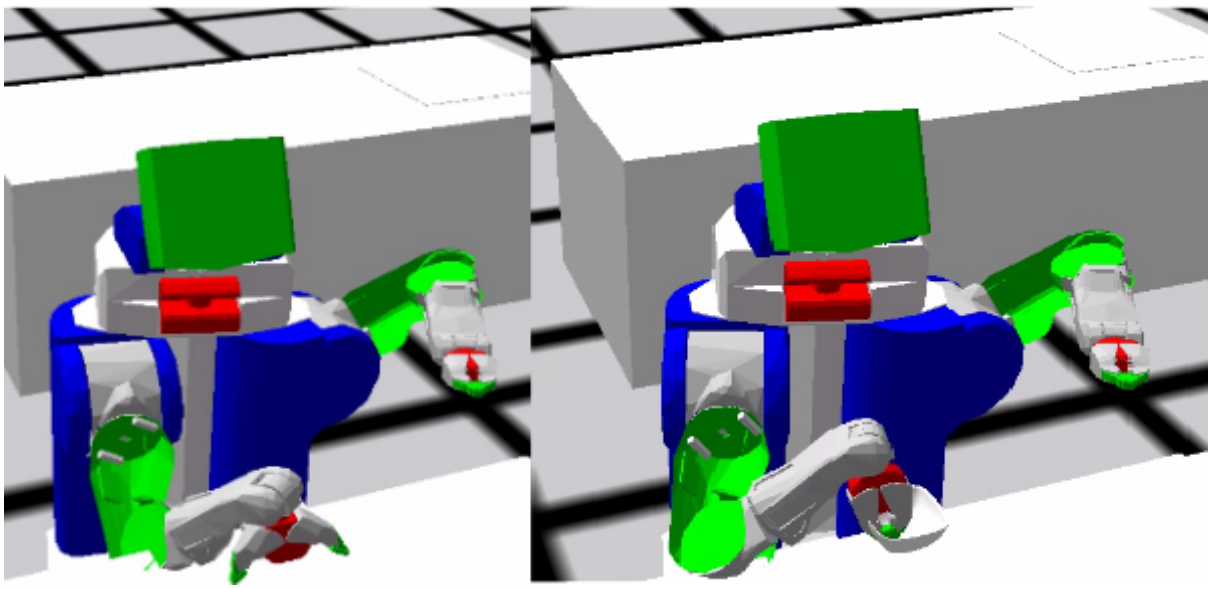

Figure 10: *The comparison of the two pre-grasping poses when grasping a bowl.*

can be seen, the robot did not need to do this differentiation in this approach, since this decision is made by the human performing the grasp, and the human just usually grasps the closest edge of an object.

Each of the oick and place tasks were performed at least 5 times, and looked like the robot did exactly the same thing over and over again, without any noticeable difference.

5.1.3 Experiment 2 - Using the Positions of Objects from VR with slight displacement

In this experiment it is tested, with how much of an location offset can an object have and still be recognized and grasped. This is important since there is no way for an object in the real world to stand on exactly the same spot with exactly the same orientation as the object in the Virtual Reality. This experiment will therefore show, how flexible this approach is. *rcg_eval2* was chosen as the dataset aka. episode to perform this, since according to the previous experiment, it is the set that had the least collisions.
The object chosen first is the cup, since it is in a position that can be easily moved sideways, along the y axis and has a lot of free space around it. The x offset is basically how close the object is to the edge of the table, towards the robot. Starting at the original position of the cup it is very close to the edge. The y offset represents the directions left to right.
The z axis is in this case not important, since the object will be on a surface and not floating in mid air, therefore an offset to the z axis can be neglected.
All the pick and place tasks were performed with the left hand.

5.1.4 Results of Experiment 2

A table was composed to summarize, all the performed grasps.

↓ x/y →	-0.4	-0.3	-0.2	-0.1	0.0	0.1	0.2	0.3	0.4	0.5
0.0	0V	1	1C	1C	1C	0	0	1	1C	0V
0.1	0V	1	1C	1C	1C	1C	1C	1C	1C	1C
0.2	1C	1	1C	1C	1C	1C*	1C	1C	1	1C
0.3	0UV	1C	1CV	1CV	1C	1	1C	1C	1C	1C
0.4	0UV	0UCV	1CV	1CV	0UC	1	1C	1C	1C	0UC
0.5	0UV	0UCV	0UVC	0UCV	0UC	0UC	0UC	0UC	0UC	0UC

Table 3: Results of the experiment: grasping the cup with an offset.

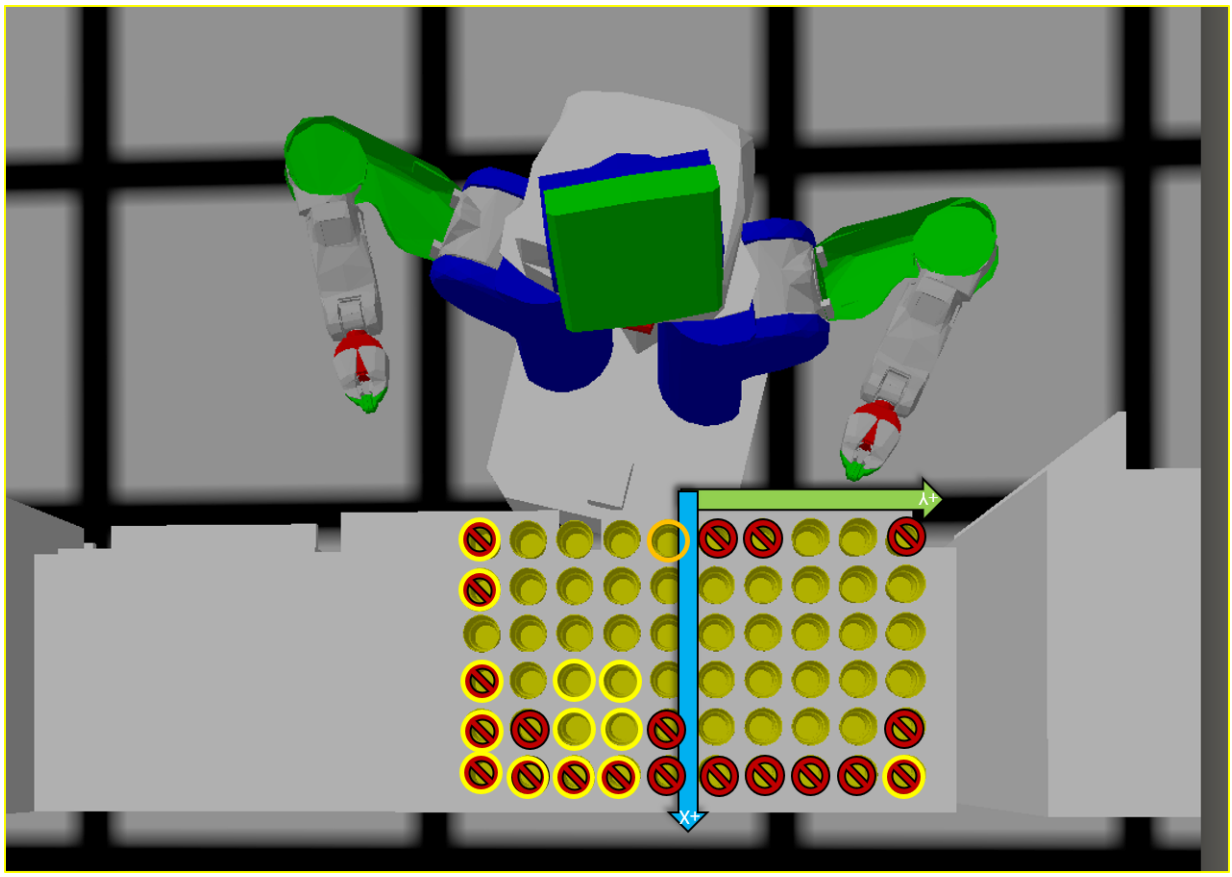

Figure 11: *Graphical representation of the table above. The orange circle marks the original cup pose, the yellow ones marks cup positions that needed one extra execute in order to be seen. The dark red marked out ones are the ones the robot could not reach.*

One can see in fig.11 which of the cups could be grasped and which were out of reach. There are a few interesting cases, e.g. the cups which are very close to the robot and to his left side, (cups one ad two to the left of the origin cup) were not reachable, since the robot is too close to the table and would have to turn his left torso part, he would be in his own way.

It can also be seen that from the way the PR2 is positioned, that he looks slightly to the left. Allowing him to see the cups offset to his left side better, then the ones to the right. The further away the cups are and the more offset to the right, the harder they are to see on the first try. Since the robot first looks directly at the origin location of the original cup, these ones are simply out of the field of view. However, once the detecting action gets executed a second time, the robot slightly lifts his head to look a bit further, and then he is able to see the cups.
At some point, since he is grasping them with his left hand, the reachability becomes a problem, so that the furthest cups cannot be reached anymore.

Since this was mostly an experiment to see how far the objects can be from their original location in order for the robot to still be able to detect and interact with them, it can be called a success, since the field of view seems to be rather broad. Therefore the robot should be able to find objects within a reasonable large radius and be able to perform the action the human performed in Virtual Reality.
One drawback is that collisions occur, but again, this might be solvable by the movement planning system of the real robot.

Sometimes the robot would end up "knocking the cup from the table" while moving the gripper away from it. Since this has not happened in the prior Experiment, it seems like due to the different position of the cup, it is grasped slightly different each time, and therefore might be grasped in such a way that it can get knocked over while being placed, due to the gripper being too low.

5.1.5 Experiment 3 - Grasping Limits on One Object

In this experiment the human was grasping only one object, the cereal box. However, it was grasped multiple times from slightly different angles, with both hands one after the other, and even from the top. In order to be able to visualize all the grasps, axes have been spawned in the locations of where the human hand was.
One should keep in mind about this figure, that the variation of the cereal box varies also, since it is permanently picked up and put down again. in order to not make the image even more noisy, only one cereal box is shown, instead of 10.

In the episode data, the human basically was trying to grasp the object from the front side, as usual, but was varying the depth of the grasp as well, "going further into the object," so to speak. After that, a top grasp, and grasps with the other hand were tried. Also, the distance to the object was varied, and one of the handheld controllers was turned upside down at one point, to try if this kind of grasp would still be possible.

Results of Experiment 3

Overall the robot was able to grasp the object in 8 out of 10 cases, including the upside-down rotation of the hand, since the robot can rotate his forearm indefinitely. The only cases which he could not grasp at all, were the last two ones, where the object

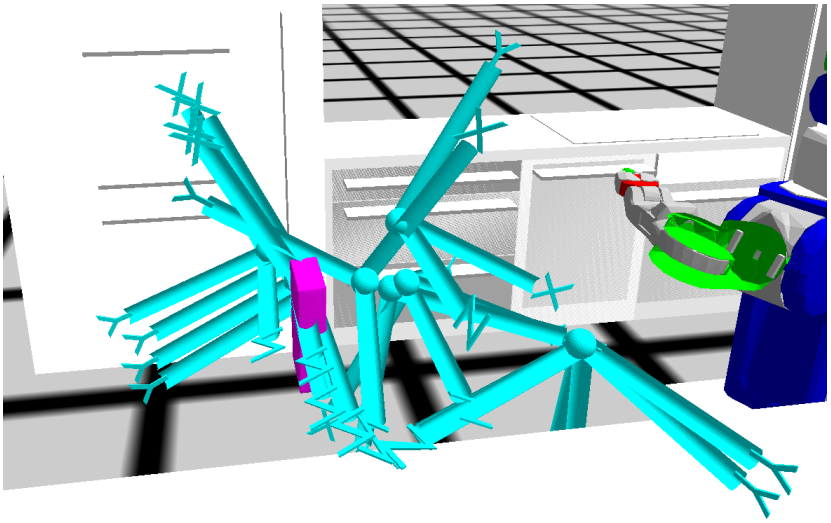

Figure 12: *Even if chaotic, it is possible to see all the different positions the robot would try to grasp the object from.*

was simply far too close to the robot's torso, and a back grasp was attempted, meaning grasping the cereal box from behind. If the robot would have turned enough to try to perform this grasp, he would very likely knock the object over with his torso or shoulder part. Also, the object being so close, it was not possible for the robot to detect it properly.

6 Conclusion

6.1 Summary

It was tried to record data in a Virtual Reality setup and use it as a baseline to make a robot perform every day household activities, for example, setting up a table for breakfast, by bringing the necessary items from one table to another. Such data needed to be recorded, processed into data that can be used by the robot, and then tested by having a simulated robot act on the grasping, placing and positioning poses from this data.
Overall the goal was achieved, as functions were developed to query for the necessary data from the knowledge database, for converting it into a robot useful state and then executing plans on it.
It can be said that the approach is mostly successful, since in most of the experiments performed, it looks like the robot can perform the necessary grasps. It was also explored if this approach would still work, even if the object the robot is looking for is misplaced and that was also successful.

The so far not mentioned but most fragile part of this system is the setup of this system itself. Very specific versions of software were needed in order to set this up and as soon as updates came, there were compatibility issues. One of the strongest drawbacks was rosjava. The repository where rosjava and gradle pull their dependencies from was restructured in such a way, that the dependencies could not be found anymore and therefore, KnowRob was not able to build properly. Sometimes it would still work, but not reliably in the long run. After a while and a gradle update, which was needed in order to set it in such a way that it can use a different dependency repository, it was finally repaired.
Similar issues occurred with docker and OpenEase. The version of docker did not match at first, and so it required an update. Later on, after an OpenEase update which was needed for another project, this setup fell apart. After fixing OpenEase, there were issues with the MongoDB, in a way that it could not import episode data properly at first, until the right approach was found which was also described in the implementation chapter of this thesis.
All these issues stole away valuable time, which could have been put into finessing the code of this thesis and making it more generic, reliable and easier to use.

6.2 Discussion

According to the experiments performed, it seems that the data recorded in the Virtual Reality is very much usable for the robot to perform the same tasks the human did in VR. However, it should be noted that after a while after going back and forth between recording data and testing with it, one tends to interact and grasp things within the virtual environment in such a way that would benefit the robot. Therefore, one could say, there is a slight bias.
Bigger objects with lots of surface, like cereal boxes and milk cartons, are generally easier to grasp in the Virtual Reality and in the robot simulation, than smaller thinner ones. Grasping the bowl at its rim, though natural for a human, proved to be hard to do for the robot. The controller needed to be held in a rather unnatural way, namely pointing downwards, which is very uncomfortable and is not really a natural human grasp.

Grasping the fork within the Virtual Reality proved to be harder then expected, since the virtual hands can not really perform a “pinch” grasp. From the successful results of the fork grasp within the simulation, it looks like the robot would press into the table a lot, in order to pick up the object, which is not optimal.

One potential advantage of this approach is that it is not necessary to differentiate between top, side or back grasps anymore, which are usually used hard coded grasps, since it can all be summed up as a “human grasp”, which is in a way universal. Depending on the situation, it can be instead queried for episodes, in which the objects had similar position and orientation and the human grasp from that episode can be applied. It looks very promising in the simulation, but only experiments in the real world can tell if this is truly so, and if it can be used to some advantage.
The pose which is recorded from the virtual hands, can sometimes be fairly missleading, since it is at the outer side of the wrist, which tends to point right next to the object, since this is where the palm of the hand is while grasping.

6.3 Future Work

In the future there are a few things that could be improved and researched further. For example, the code could be made more generic and automatic. Experiments could be performed, where multiple people without any knowledge about robots are asked to perform pick and place tasks within the VR environment, in order to not be biased about how to grasp an object in a way that a robot will be able to do it also.
The semantic maps between the VR, bullet world and even the real world could be adjusted to one another, so that no weird offsets need to be calculated. The setup of all the tools necessary for this system could be built in a way that makes it is easier to install. The import and export of episode data directly from the VR to OpenEase could be automated.
Instead of using the poses from the recordings directly, gaussian maps could be used to determine them. Incorporating this into the system has already been started.
Also, a lot of the knowledge learned through this thesis can simply be documented and noted down, which could start a documentation of KnowRob and RobCog queries, which would make it a lot easier for other people, who are trying to get into this material, to do so, instead of just going by trial and error.
It would be also very interesting to see, if more complex tasks can be performed. For example, opening and closing of drawers, picking and placing things in them, or even trying to open a dish washer or a fridge.
It could be also investigated, how using more detailed poses of the virtual hands within VR could benefit the robot’s grasping functionality, since the poses of individual finger joints can be retrieved from the episode data, instead of just using the overall hand pose.

7 Appendix

7.1 Bibliography

[] URL: https://www.unrealengine.com/en-US/what-is-unreal-engine-4 (visited on 04/08/2018).

[] *rosjava*. URL: http://wiki.ros.org/rosjava (visited on 04/08/2018).

[] *SWI-Prolog*. URL: http://www.swi-prolog.org/ (visited on 04/15/2018).

[] *Web Ontology Language*. URL: https://en.wikipedia.org/wiki/Web_Ontology_Language (visited on 04/15/2018).

[Bat+17] T. Bates et al. "On-line simultaneous learning and recognition of everyday activities from virtual reality performances". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 3510–3515. DOI: 10.1109/IROS.2017.8206193.

[Bee+15a] M. Beetz et al. "RoboSherlock: Unstructured information processing for robot perception". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 1549–1556. DOI: 10.1109/ICRA.2015.7139395.

[Bee+15b] M. Beetz et al. "RoboSherlock: Unstructured information processing for robot perception". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 1549–1556. DOI: 10.1109/ICRA.2015.7139395.

[BTW15] M. Beetz, M. Tenorth, and J. Winkler. "Open-EASE". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 1983–1990. DOI: 10.1109/ICRA.2015.7139458.

[Fou] Open Source Robotics Foundation. *Bullet world demonstration*. URL: http://cram-system.org/tutorials/intermediate/bullet_world (visited on 04/09/2018).

[Hai] Andrei Haidu. *Robot Commonsense Games*. URL: http://www.robcog.org/games.html (visited on 04/08/2018).

[HB] Andrei Haidu and Michael Beetz. *Automated Models of Human Everyday Activity based on Game and Virtual Reality Technology*. (Visited on 04/01/2018).

[HB16a] A. Haidu and M. Beetz. "Action recognition and interpretation from virtual demonstrations". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 2833–2838. DOI: 10.1109/IROS.2016.7759439.

[HB16b] A. Haidu and M. Beetz. "Action recognition and interpretation from virtual demonstrations". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 2833–2838. DOI: 10.1109/IROS.2016.7759439.

[KB17] G. Kazhoyan and M. Beetz. “Programming robotic agents with action descriptions”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 103–108. DOI: 10.1109/IROS.2017.8202144.

[KHB13] L. Kunze, A. Haidu, and M. Beetz. “Acquiring task models for imitation learning through games with a purpose”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 102–107. DOI: 10.1109/IROS.2013.6696339.

[MB11] L. Mösenlechner and M. Beetz. “Parameterizing actions to have the appropriate effects”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2011, pp. 4141–4147. DOI: 10.1109/IROS.2011.6094883.

[MB13] L. Mösenlechner and M. Beetz. “Fast temporal projection using accurate physics-based geometric reasoning”. In: *2013 IEEE International Conference on Robotics and Automation*. May 2013, pp. 1821–1827. DOI: 10.1109/ICRA.2013.6630817.

[Mös16] Lorenz Mösenlechner. “The Cognitive Robot Abstract Machine”. Dissertation. München: Technische Universität München, 2016.

[PK15] K. Pauwels and D. Kragic. “SimTrack: A simulation-based framework for scalable real-time object pose detection and tracking”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, pp. 1300–1307. DOI: 10.1109/IROS.2015.7353536.

[Rit] T.C. Rittweiler. “CRAM. Design Implementation of a Reactive Plan Language”. Thesis. (Visited on 05/05/2010).

[TB] Moritz Tenorth and Daniel Beßler. *KnowRob*. URL: http://knowrob.org (visited on 04/15/2018).

[WDB16] T. Welschehold, C. Dornhege, and W. Burgard. “Learning manipulation actions from human demonstrations”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 3772–3777. DOI: 10.1109/IROS.2016.7759555. URL: http://ieeexplore.ieee.org/document/7759555/ (visited on 11/23/2017).

[WDB17] T. Welschehold, C. Dornhege, and W. Burgard. “Learning mobile manipulation actions from human demonstrations”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 3196–3201. DOI: 10.1109/IROS.2017.8206152.

[Zha+17] Tianhao Zhang et al. “Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation”. In: (Oct. 2017). URL: https://www.researchgate.net/publication/320371149_Deep_Imitation_Learning_for_Complex_Manipulation_Tasks_from_Virtual_Reality_Teleoperation.

List of Figures

List of Tables

Listings

7.2 Acronyms

bson Binary JavaScript Object Notation

CRAM Cognitive Robot Abstract Machine

json JavaScript Object Notation

owl Web Ontology Language

RobCoG Robot Commonsense Games

ROS Robot Operating System

VR Virtual Reality