

The Use Of Coreference Resolution For  
Understanding Manipulation Commands For The **ARCHIVES**  
PR2 Robot

by

Dimitar N. Simeonov

B.A. Massachusetts Institute of Technology (2011)

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of


Master of Engineering in Electrical Engineering and Computer Science



at the



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012

© Massachusetts Institute of Technology 2012. All rights reserved.

Author  .....  
Department of Electrical Engineering and Computer Science  
February 01, 2012

Certified by  .....  
 .....  
Nicholas Roy  
Associate Professor  
Thesis Supervisor

Accepted by  .....  
 .....  
Prof. Dennis M. Freedman  
Chairman, Masters of Engineering Thesis Committee



# The Use Of Coreference Resolution For Understanding Manipulation Commands For The PR2 Robot

by

Dimitar N. Simeonov

Submitted to the Department of Electrical Engineering and Computer Science  
on February 01, 2012, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

Natural language interaction can enable us to interface with robots such as the Personal Robot 2 (PR2), without the need for a special training or equipment. Programming such a robot to follow commands is challenging because natural language has a complex structure and semantics, a model for which needs to be based on linguistic knowledge or learned from examples. In this thesis we first enable the PR2 robot to follow manipulation commands expressed in natural language by applying the Generalized Grounding Graph ( $G^3$ ). We model the PR2's actions and their trajectories in the physical environment, define the state-action space and learn a grounding model from an annotated corpus of robot actions aligned with commands. We achieved lower overall performance than previous implementations of  $G^3$  had reported. After that, we present an approach for using the linguistic technique of coreference resolution to improve the robot's ability to understand commands consisting of multiple clauses. We constrain the groundings for coreferent phrases to be identical by merging their nodes in the grounding graph. We show that using coreference information increases the robot ability to infer the right action sequence. This brings the robotic capabilities of modeling and understanding natural language closer to our theoretical understanding of discourse.

Thesis Supervisor: Nicholas Roy  
Title: Associate Professor



## Acknowledgments

First, I want to thank my thesis supervisor, Nicholas Roy, for giving me the opportunity to pursue MEng and for his guidance, making me strive to always do my best. I learned a lot through the experience and am grateful that he helped me find the right way to do my work — thoughtfully and systematically. This is a great lesson and I will do my best to apply it in my future life.

I also want to thank everybody in the Robust Robotic Group (RRG) for being inspiring examples who I tried to follow, for participating in discussions and for giving me their feedback. Special thanks to Emma Brunskill, Thomas Kollar, Ashis Banerjee and Stefanie Tellex, who supervised me on projects through my years in RRG, and from whom I learned a lot.

I want to thank my parents and family for being close to me even when the Atlantic ocean separated us. I also owe a lot to all my friends, who always believed in me and who, through their confidence in me, turned their wishes into self-fulfilling prophecies. Last, but not least, I also want to thank Manal for keeping me sane and happy.



# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Motivation . . . . .	13
1.2	Challenges and approach . . . . .	15
1.3	Scope . . . . .	18
1.4	Contributions . . . . .	18
1.5	Organization of this document . . . . .	19
<b>2</b>	<b>Related Work</b>	<b>21</b>
2.1	Generalized Grounding Graph ( $G^3$ ) overview . . . . .	21
2.2	Natural Language Modeling and Processing . . . . .	30
2.3	Coreference resolution as a part of $G^3$ . . . . .	34
<b>3</b>	<b>Applying Generalized Grounding Graph (<math>G^3</math>) to PR2</b>	<b>35</b>
3.1	PR2 and ROS overview . . . . .	36
3.2	Perception . . . . .	45
3.3	Actuation . . . . .	47
3.4	Planning . . . . .	49
3.5	Evaluation technique . . . . .	59
3.6	Summary . . . . .	63
<b>4</b>	<b>Coreference resolution</b>	<b>65</b>
4.1	Graph merging algorithm . . . . .	65
4.2	Node ordering algorithm . . . . .	68

4.3	Implementation . . . . .	70
4.4	Comparison of coreference resolvers . . . . .	75
4.5	Summary . . . . .	77
<b>5</b>	<b>Conclusion</b>	<b>79</b>



# List of Figures

1-1	Photographic description of the PR2 robot . . . . .	14
2-1	2D rendering of forklift physical context . . . . .	28
2-2	3D rendering of forklift physical context . . . . .	29
2-3	Merging coreferent $\gamma$ nodes. . . . .	33
3-1	Rendering of the PR2's depth sensor perception abilities. . . . .	39
3-2	Rendering of the PR2 object detection . . . . .	41
3-3	Rendering of the PR2 about to pickup an object . . . . .	42
3-4	Rendering of the PR2 attempting to place an object on a table. . . .	44
3-5	Rendering of some of the hand and object trajectories in the trajectory database . . . . .	52
3-6	Quantitative analysis of the trajectory database . . . . .	53
3-7	Displaying predicted hand trajectories . . . . .	54
3-8	Screenshot of the annotation interface we used to annotate the groundings	56
3-9	Rendering of a partial path . . . . .	56
3-10	Rendering of an example video from the corpus . . . . .	57
3-11	Screen-shot of an example task that we gave to the subjects in Me- chanical Turk . . . . .	58
3-12	Plot of correct examples among the top confident inferences . . . . .	62
4-1	Performance plot of the ReconcileNPS engine . . . . .	72
4-2	Performance plot of the ReconcileNPSAnaphora engine . . . . .	74
4-3	Plot of correct examples among the top confident inferences . . . . .	77



# List of Tables

2.1	An example of an Extended Spatial Descriptive Clause (ESDC) . . .	23
2.2	An example of a physical context. . . . .	27
3.1	An example of a ROS message type definition — <code>TabletopDetectionResult</code>	37
3.2	Performance on predicting the correspondence variables $\phi$ in the testing dataset. . . . .	60
4.1	Object grounding performance on the training dataset . . . . .	75
4.2	Object grounding performance on the test dataset . . . . .	75
4.3	Plan correctness performance of different coreference resolvers on the examples of the dataset which include coreference . . . . .	76
4.4	Plan correctness performance of different coreference resolvers on the whole dataset . . . . .	76



# Chapter 1

## Introduction

The work presented in this thesis consists of two main parts. The first part is the application of the Generalized Grounding Graph ( $G^3$ ) framework for following natural language commands on the PR2 robot [12] and the second part is the incorporation and empirical evaluation of coreference resolution in the language model of  $G^3$ . In this chapter we present the theoretical and practical motivations for accomplishing these two tasks and provide brief descriptions of the challenges and the approaches in each of them and the resulting contributions. We also give an overview of the thesis.

### 1.1 Motivation

Mobile and dexterous robots such as the PR2 (Figure 1-1) have the physical capabilities to accomplish a wide variety of manipulation tasks. In the recent years their software capabilities have advanced as well — for example the PR2 can open doors [51], pick and place objects [10] on a table, plug its power cord into a power outlet [11], bring beer [3], traverse a cluttered indoor environments [42], bake cookies [43], fold towels [18], identify and remove potentially offensive objects [21], solve the Rubik’s cube [13], and build composite skills based on simpler ones [33, 49]. There are also a lot of robotic capabilities that require only a subset of the PR2’s physical hardware to operate, or a similar physical hardware, such as getting a robot to push things out of its way in order to grasp an object [23] or playing chess in a natural way [45].

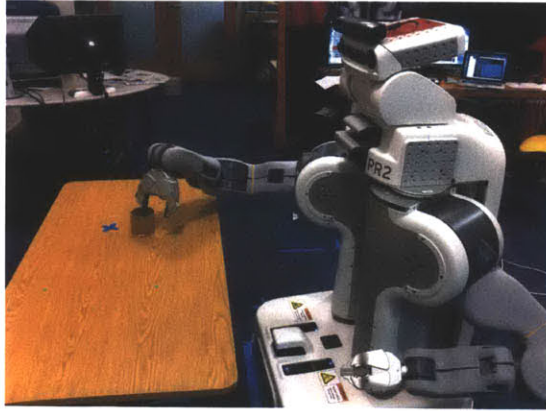


Figure 1-1: The PR2 robot has two arms which it can use to manipulate objects such as books or phones. It also features cameras and a point-cloud sensor on the head which allow it to perceive its surroundings.

Even though such robots could be useful for a wide range of applications, interacting with them is challenging exactly because of the inherent variety and number of the robot abilities. The human needs to somehow specify what they want the robot to do. Natural language is a promising modality for interaction because it does not require extensive training or physical contact with the robot and is highly expressive. The practical usefulness of natural language as a medium to express commands has already been validated in its use in touchscreen devices such as iPhone 4S [1]. When the company Willow Garage introduced the PR2 robot via the PR2 Beta Program [4], one of the proposed research directions was human-robot interaction via natural language [14], and the only other project about natural language interaction on the PR2 that we are aware of is [5], which at the time of writing of this thesis has only been published as an online article and video. Therefore, to allow the PR2 to understand natural language commands in the first part of this thesis we apply the  $G^3$  natural language interaction framework on the PR2 robot, allowing it to follow simple natural language commands. Another motivation for the first part of the thesis is collecting a corpus of natural language commands paired with robotic actions, which we created in order to learn the grounding model of  $G^3$ .

$G^3$  follows commands one sentence clause at a time, but language is not just a sequence of sentence clauses. Between consecutive clauses there are semantic con-

nections, some of which manifest themselves through coreferent phrases across the sentences. For two phrases to be *coreferent* it means that they refer to the same entity. For example, in the command “Pick up the bottle. Place it on the table,” the phrase “the bottle” and the phrase “it” must refer to the same physical object and are therefore coreferent. Theoretical models of discourse such as Discourse representation theory (DRT) [38] include those semantic connections and represent the value of phrases about coreferent entities with the same variable.  $G^3$  considers commands in a context-free manner; every command clause is processed in isolation of the preceding and following ones, which does not take advantage of previous mentions of the same entities. By not accounting for such coreferences  $G^3$  could miss important information about the groundings and achieve inferior overall performance. In [53] we showed theoretically that we can incorporate linguistic information about coreference into  $G^3$  with only minimal extensions to the model, thus opening the possibility to use  $G^3$  for understanding longer sequences of commands as well as question-asking dialog. The task to evaluate this approach in practice motivated the second part of this thesis in which we implement coreference resolution in  $G^3$  and empirically show that it indeed improves the overall inference performance on longer sequences of commands.

## 1.2 Challenges and approach

### 1.2.1 Applying the $G^3$ framework

$G^3$  was originally designed and first implemented on a robotic forklift [55] and when applying  $G^3$  to a new type of robot we encountered differences which made the application process non-trivial. Here, we present what challenges arose from the different ways robots perform their actions, from the software differences and from differences in the physical environment, highlighting the approach we took to solve them.

## Grounding robotic actions

$G^3$  models the robot as a single prism which is not sufficiently descriptive of the PR2's actions. Since a robotic forklift, except for the forks, is a rigid body and does not have a posture, it made sense that there the robot was represented as a prism. The PR2, however, has two hands which it uses to manipulate the objects. For tabletop manipulation, the PR2 body does not move and its trajectory would not be descriptive of the action performed. Consequently, we cannot learn and infer a representation of a verb phrase just using the PR2's body representation.

To overcome this challenge we added the arms of the robot and their trajectories to the grounding model. At learning time we annotate each action with the arm performing it and its trajectory for the duration of the action. At inference time we infer which arm is doing the action and predict its trajectory for the duration of the action.

## Predicting the effect of the robot actions

The output of  $G^3$  is a sequence of actions, which then the robot performs in an open loop fashion. Before a robot commits to perform an action, it needs some way of knowing that the action is right for the command before executing it.  $G^3$  scores each plan by assigning a cost to it, which is the negative logarithm of the probability that the actions correspond to the command. This cost function does not just score how much energy it will spend performing the task or what distance it will traverse, but how closely the action trajectory and the objects involved correspond to the ones described in the command.

To assign a cost on a plan,  $G^3$  needs to know what will happen if the robot executes the plan and we model that knowledge according to recorded trajectories. This knowledge about those trajectories could come from simulating the robot's planned actions. However, such a simulation would not guarantee exactness, as the robot would not have exact knowledge of the environment for creating a perfect simulation. Simulation is also very computationally costly in the case of PR2. Therefore,



to predict the effects of the robot actions efficiently we use a database of previously recorded robotic actions. We also note that for the actions that PR2 accomplishes we can know exactly the starting position of the robot hand performing the action and approximately the furthest position from the start in its trajectory. We used these positions as a similarity metric to select a recording from the database in which the starting and furthest positions are closest to the ones we expect, and use the trajectories from this recording as a prediction. We used the start and far trajectory points not only because we could estimate them, but also because they are descriptive of what happens in the actions.

### 1.2.2 Using coreference resolution

While coreference means that multiple phrases in the text have the same meaning, *coreference resolution* is the problem of automatically extracting those coreferences from text. Coreference resolution is a well-studied problem in computational linguistics [32, 48, 34, 22], but the current state-of-the-art coreference solvers are better designed and trained for language such as the Wall Street Journal articles rather than grounded robotic manipulation commands and are therefore far from perfect. When applying coreference resolution to the problem of grounding natural language, the use of coreference information could improve grounding performance, but the reverse is true as well if the resolutions are erroneous. It is unclear what is the most effective way to combine the coreference information about the command with the grounding process. Considering the coreference and grounding problems jointly may lead to a better overall performance than if the result of the coreference is just inputted into the grounding algorithm. Our approach uses the information from coreference resolution to improve grounding, and *not* vice versa, because it is simpler, more decoupled approach, and only requires minimal changes in  $G^3$ . We therefore take the coreferent phrases as given by a coreference resolution engine and merge the corresponding nodes in the graph.

### 1.2.3 Evaluation

Evaluating whether the robot actions follow a command is hard, because there could be multiple sets of robotic actions with differences in some details which follow the command correctly. Creating an exact representation of the set of all possible things that the robot can do and follow the command correctly is hard, and not necessarily an easier problem than following commands. However, it is easy for a human to judge whether the robot actions made sense for the command. Therefore, we resort to manual evaluation of whether the actions for each command are correct.

## 1.3 Scope

Even though there are multiple possible modalities of interaction such as voice interaction, gestures, remote control through a computer or a computerized device we do not explore the question of which is the best modality or a combination of modalities for interaction, but instead focus on improving the abilities of robots to understand natural language commands. We also do not address any speech recognition in this thesis. We assume plain text as the format of the natural language.

We also do not address the question of what is the right level of autonomy for the robots, for example, should a robot only wait to be explicitly commanded by its human supervisor to perform a task, or it should proactively decide what to do based on its understanding of the human’s preferences.

We only show that the use of coreference resolution information improves grounding. We do not attempt to improve the coreference resolution through grounding information.

## 1.4 Contributions

This thesis has three main contributions:

1. Application of the  $G^3$  model on a PR2 robot, allowing a two-handed robot to follow natural language commands when next to a tabletop.

2. Collection via crowdsourcing and annotation of a corpus of natural language commands paired with robotic actions.
3. The introduction of coreference resolution as a part of the natural language model to improve the robot’s ability to ground the words of the command when presented with a multi-sentence command.

## 1.5 Organization of this document

Chapter 2 provides a review of the background, including  $G^3$  overview and related work, and describes what challenges and opportunities emerge from analyzing it. Chapters 3 and 4 describe the work and procedures carried out in the first and the second part of the thesis. Chapter 3 provides background on the PR2 system and describes how we apply the  $G^3$  framework on the PR2 robot and what adaptations were needed to move from the forklift platform to a dexterous robot. Chapter 3 also describes how we collected and annotated a corpus from which we learned a grounding model for the new domain, and how we evaluated the first part of the thesis. Chapter 4 explains how we introduce coreference resolution as a part of the language model in  $G^3$ , what novel challenges this creates, how we adapt the inference procedure and how we evaluate the usefulness of the approach. Chapter 5 concludes the thesis, summarizing the contributions and their impact and providing suggestions for future work.



# Chapter 2

## Related Work

In this chapter we first present an overview of  $G^3$ , introducing the necessary taxonomy and concepts, then we discuss previous work in grounded natural language, and finally we describe the theoretical work we did on introducing coreference resolution in  $G^3$  ([53]).

### 2.1 Generalized Grounding Graph ( $G^3$ ) overview

The Generalized Grounding Graph ( $G^3$ ) is a framework for following natural language directions in human-robotic interaction which frames the problem of following instructions as inferring the most likely robot state sequence from a command and knowledge about the environment. To perform the inference  $G^3$  uses a language model (to decompose the command), a robot state model (to describe the environment and the robotic actions) and learns a model for grounding the parts of the language to entities in the robotic state. The language model is a *grounding graph*, a probabilistic graphical model, and the grounding model is the probability distribution of the grounding graph.

### 2.1.1 Language Model

The grounding graph is constructed from Extended Spatial Descriptive Clauses (ESDCs) — hierarchical structures used to describe the semantics of the command. As their name suggests ESDCs are an extension of Spatial Descriptive Clauses (SDCs) — a formal representation of the semantics of natural language directions. Here, we provide background on SDCs and ESDCs and the automated construction of a grounding graph from ESDCs

#### Spatial Descriptive Clauses (SDCs)

Kollar et al. ([36]) first introduced SDCs to formally describe the semantics of natural language navigational directions. In [36], every navigational direction is represented as a sequence of SDCs. Each SDC consists of a *figure*, a *verb*, a *landmark*, and a *spacial relation*. Each of those fields could be either a part of the text or a nested SDC, but [36] only used SDCs in a sequential way. The figure of an SDC represents the subject performing the action, the verb represents the action, the landmark can be any physical landmark in the environment and the spatial relation represents a geometrical relation between the figure and the landmark. For example, in the sentence “You go past the elevator,” the word “you” is the figure, “go” is the verb, “past” is the spatial relation and “the elevator” is the landmark.

#### Extended Spatial Descriptive Clauses (ESDCs)

Tellex et al. ([55]) introduced ESDCs and represented every manipulation command as a sequence of ESDCs, most of which typically contain nested ESDCs themselves (Table 2.1). There are few differences between ESDCs and SDCs. Firstly, ESDCs, unlike SDCs also have a type which could be OBJECT, PLACE, PATH or EVENT. Secondly, in ESDCs a single *relation* field replaces the verb and the spatial relation fields of an SDC. Finally, an ESDCs can also have a second landmark field. While SDCs handle sequential directions, ESDCs aim at better understanding of hierarchical language.

Table 2.1: ESDCs are usually nested in each other. Here fields of a given ESDC are indented once more than the ESDC, and nested ESDCs are indented once more than the field they occupy in their parent ESDC.

<pre> - Put the black item on the table - - EVENT:   r: Put   l:     OBJECT:       f: the black item   l2:     PLACE:       l: the table       r: on </pre>
---

OBJECT ESDCs can represent a variety of things in the world — most typically objects that the robot can manipulate, but also other landmarks, humans and even the robot (or parts of the robot). Leaf OBJECT ESDCs are those object ESDCs which do not have any nested ESDCs. The landmark fields of an OBJECT ESDC can only be a nested ESDC, and therefore leaf OBJECT ESDCs have empty landmark fields. An example of a leaf OBJECT ESDC is the object with figure “the black item” in Table 2.1. When an OBJECT ESDC has a landmark or a relation field, its figure is also considered a leaf ESDC. PLACE ESDCs represent locations in the physical space. PATH ESDCs represent paths or path fragments in the physical environment. EVENT ESDCs represent action sequences.

### Grounding graph definition

*Grounding graphs* are a data structure based on factor graphs. As such, they are probabilistic graphical models as well. Every grounding graph is a factor graph, but also contains some additional meta-data and imposes constraints on the structure of the graph.

Formally, a grounding graph can be defined as a tuple  $\{G, Z, \Psi\}$  in which  $G$  is the factor graph,  $\Psi$  is its potential function and  $Z$  is the set of observed data.

$G$  as a graph contains nodes and edges and can formally be described by the tuple  $\{V_N, V_F, E\}$  - nodes, factors and edges.

The set of nodes  $V_N$  satisfies

$$V_N \subset \text{IDs} \times \{\lambda, \gamma, \phi\} \times \text{Types},$$

meaning that every node has a type which is a combination of one of  $\lambda$ ,  $\gamma$  or  $\phi$  and possibly an additional type modifier. Only  $\gamma$  nodes have type and the set of possible types are the same as the types of the ESDCs.  $\lambda$  nodes are text nodes and are observed during inference.  $\gamma$  nodes are world groundings (such as a specific object) and  $\phi$  nodes are correspondence variables which have to be **True** or **False**. Each  $\phi$  node stands for the random variable describing whether the groundings in the factor's  $\gamma$  neighbors correspond to the text in the  $\lambda$  neighbors.

The set of factors  $V_F$  satisfies

$$V_F \subset \text{IDs} \times \text{Types},$$

which means that every node and factor has a unique ID, and there is exactly one  $\phi$  node adjacent to each factor and positive number of  $\lambda$  and  $\gamma$  nodes. For every factor there is an assigned ESDC from the text, and the type of the factor is the same as the type of the assigned ESDC.

The set of edges  $E$  satisfies

$$E \subset V_N \times V_F \times \text{Labels}$$

which means that every edge connects one node and one factor and has a label. The edge labels could be **top**, **phi**, **f**, **r**, **l** or **l2**.

$Z$  is a set of observations where each observation is a pair of some node  $V_N$  and some grounding.  $Z$  could be empty, but during inference all  $\lambda$  nodes are observed and all  $\phi$  nodes are set to **True**.  $\Psi$  is a non negative factor potential function with inputs consisting of a factor in  $V_F$  and observations for all its neighboring nodes and



it satisfies  $\Psi \in \mathbb{R}$ . It is usually learned as a log-linear model of a set of features over the set of the neighboring nodes. In  $G^3$ , the potential function  $\Psi$  represents the probability

$$p(\phi = \text{True}|\Lambda, \Gamma)$$

that the groundings of the  $\gamma$  nodes correspond to text in the  $\lambda$  nodes. This factors the whole probability that the command corresponds to the language

$$p(\text{Correspondence} = \text{True}|\Lambda, \Gamma) = \prod_{\phi} p(\phi = \text{True}|\Lambda, \Gamma)$$

### Conversion from ESDCs to Grounding Graphs

Conversion from ESDCs to grounding graphs is straightforward. For every ESDC we first create a factor, a  $\phi$  node connected to it via a `phi`-labeled edge and a  $\gamma$  node connected to it via `top`-labeled edge. In the case of OBJECT ESDCs, if the ESDC contains a relation field then we use two factors with a  $\phi$  node for each, connected to a single  $\gamma$  node via `top`-labeled edges. One of the factors is for the figure of the object, and the other one is for the relation and the landmark. Then, for every figure or relation field of an ESDC we create a lambda node and connect it to the corresponding factor via an edge with a label `f` or `r`. For every landmark or landmark2 field of an ESDC we connect the factor of the ESDC with the  $\gamma$  node created for the corresponding landmark and connected with a `top`-labeled edge to its factor, to the fist ESDC via a edge with a label `1` or `12`.

## 2.1.2 Grounding model: Cost function and spatial features

### Factor potential definitions

The grounding model, the underlying probability distribution, is defined by the factor potentials  $\Psi$ . The potential function  $\Psi$  used in [55] for each factor is given by the log-linear function

$$\Psi(\text{factor}) = \exp(\vec{w} \cdot \vec{f}(\text{factor}))$$

where  $\vec{f}$  is a vector of features extracted from the factor and all of its observed neighbor nodes. All the features are binary or binarized as explained in [55].  $\vec{w}$  is a weight vector. We refer to  $\Psi$  not only as a potential function, but also as a *cost function*, because we can think of the negative logarithm of the probability in each potential as a cost that each grounding incurs.

### Learning of the factor potentials

For a fixed set of features, the potential function  $\Psi$  is uniquely defined through the weight vector  $\vec{w}$ . That means that learning of  $\Psi$  is equivalent to an optimization in the space of all vectors  $\vec{w}$ . The approach in [55] learns the potential cost function by finding the vector  $\vec{w}$  which maximizes the likelihood of the training data. To maximize the likelihood of the training data, [55] computes the gradient and use the Mallet toolkit [46] to optimize the parameters of the model via gradient descent with L-BFGS [19].

The data comes as a set of fully observed grounding graphs. Those graphs are generated from an annotated corpus of natural language commands parsed into ESDCs and annotated with groundings for each ESDC.

#### 2.1.3 Robot state

The robotic state is key for inferring the action sequence, since the possible actions are modeled as a part of the robotic state. The robotic state consists of a description of firstly the physical context as a list of objects and places and secondly a list of pairs of a possible action and a future state.

#### 2.1.4 Physical context

The physical context consists of a list of the objects and the places in the environment (Table 2.2). The spacial representation of each place and object consists of a prism which bases are parallel to the  $x$ - $y$  plane. Each prism is uniquely defined by a sequence of  $(x,y)$  coordinates representing the base and a pair of minimal and maximal  $z$  value.

Table 2.2: The physical context consists of a list of objects and list of places. Both objects and places are represented by a prism, but objects also have tags and a trajectory. The actual trajectories are omitted for the lack of space.

```

3 Objects:
object "washing, machine, pallet" trajectory length: 106
  prism base points (26.4, 34.9), (25.9, 35.7), (26.8, 36.2), (27.3, 35.4)
  top: 0.8, bottom: -6.0
object "flatbed, trailer" trajectory length: 1
  prism base points (38.4, 13.5), (37.4, 15.6), (51.4, 22.7), (52.5, 20.5)
  top: 2.6, bottom: -0.0
object "forklift" trajectory length: 201
  prism base points (24.2, 23.0), (25.7, 23.1), (25.6, 24.6), (24.1, 24.5)
  top: 2.0, bottom: 0.0
19 Places:
place
  prism base points (43.9, 17.1), (45.9, 17.1), (45.9, 19.1), (43.9, 19.1)
  top: 1.7, bottom: 1.4
place
  prism base points (38.3, 19.1), (40.3, 19.1), (40.3, 21.1), (38.3, 21.1)
  top: 0.5, bottom: 0.2
place
  prism base points (53.6, 61.4), (55.6, 61.4), (55.6, 63.4), (53.6, 63.4)
  top: 0.5, bottom: 0.2
place
  prism base points (58.3, 52.3), (60.3, 52.3), (60.3, 54.3), (58.3, 54.3)
  top: 0.5, bottom: 0.2
place
  prism base points (61.4, 47.2), (63.4, 47.2), (63.4, 49.2), (61.4, 49.2)
  top: 0.5, bottom: 0.2
place
  prism base points (64.2, 42.2), (66.2, 42.2), (66.2, 44.2), (64.2, 44.2)
  top: 0.5, bottom: 0.2
place
  prism base points (55.7, 57.0), (57.7, 57.0), (57.7, 59.0), (55.7, 59.0)
  top: 0.5, bottom: 0.2
place
  prism base points (22.6, 79.3), (24.6, 79.3), (24.6, 81.3), (22.6, 81.3)
  top: 0.5, bottom: 0.2
place
  prism base points (41.2, 19.4), (43.2, 19.4), (43.2, 21.4), (41.2, 21.4)
  top: 0.5, bottom: 0.2
place
  prism base points (24.2, 34.2), (26.2, 34.2), (26.2, 36.2), (24.2, 36.2)
  top: 0.5, bottom: 0.2
place
  prism base points (19.9, 32.4), (21.9, 32.4), (21.9, 34.4), (19.9, 34.4)
  top: 0.5, bottom: 0.2
place
  prism base points (72.1, 2.6), (74.1, 2.6), (74.1, 4.6), (72.1, 4.6)
  top: 0.5, bottom: 0.2
place
  prism base points (47.3, 64.2), (49.3, 64.2), (49.3, 66.2), (47.3, 66.2)
  top: 0.5, bottom: 0.2
place
  prism base points (49.4, 59.1), (51.4, 59.1), (51.4, 61.1), (49.4, 61.1)
  top: 0.5, bottom: 0.2
place
  prism base points (51.6, 55.1), (53.6, 55.1), (53.6, 57.1), (51.6, 57.1)
  top: 0.5, bottom: 0.2
place
  prism base points (54.1, 50.1), (56.1, 50.1), (56.1, 52.1), (54.1, 52.1)
  top: 0.5, bottom: 0.2
place
  prism base points (57.2, 44.9), (59.2, 44.9), (59.2, 46.9), (57.2, 46.9)
  top: 0.5, bottom: 0.2
place
  prism base points (60.3, 40.0), (62.3, 40.0), (62.3, 42.0), (60.3, 42.0)
  top: 0.5, bottom: 0.2
place
  prism base points (51.4, 66.4), (53.4, 66.4), (53.4, 68.4), (51.4, 68.4)
  top: 0.5, bottom: 0.2

```

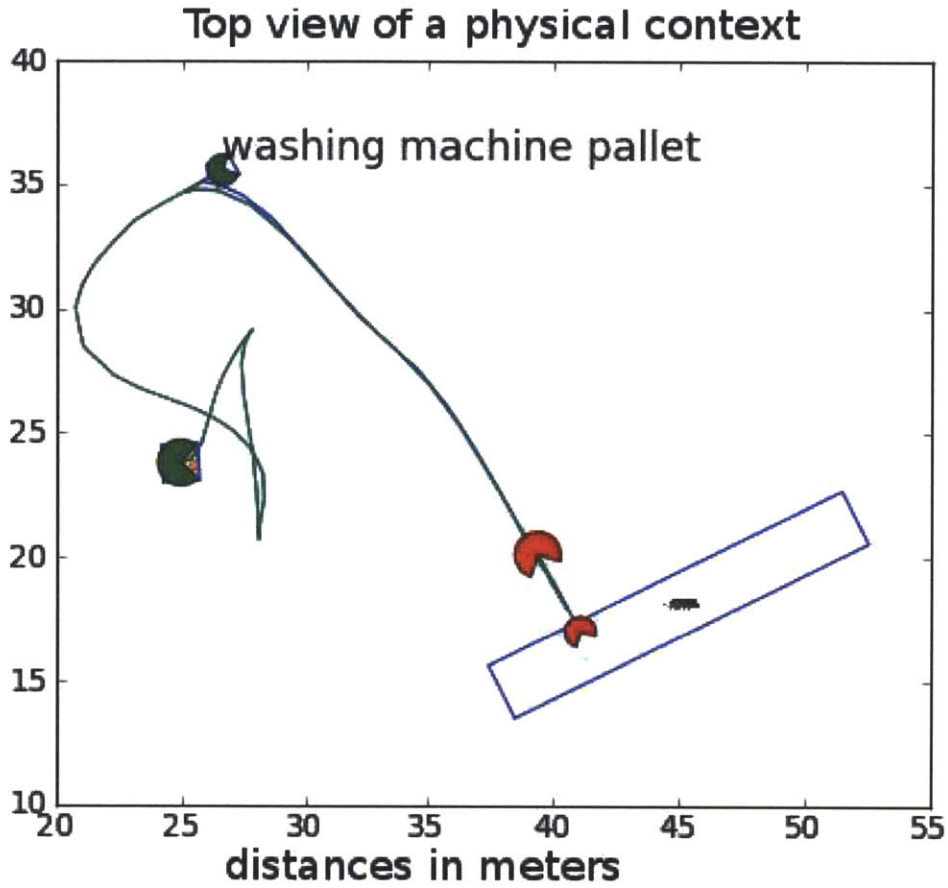


Figure 2-1: 2D rendering of the physical context from Table 2.2. We can observe the forklift trajectory in green and pallet trajectory in blue.

Each object also contains its trajectory through time as a sequence of timestamps in microseconds and a sequence of the same length of the position  $(x,y,z)$  and orientation  $\theta$  of the prism at the given timestamp. Objects also contain tags, where each tag is a string of letters. The physical context includes the robot as well as the other objects. The reference frame used is the fixed world frame. To provide an extra perspective on physical context in Figures 2-1 and 2-2 we show renderings of the same physical context as in Table 2.2 in respectively 2D and 3D.

### 2.1.5 Inference procedure

The inference procedure takes a grounding graph with observed  $\lambda$  variables and  $\phi$  variables set to True and performs an approximate inference by searching for the

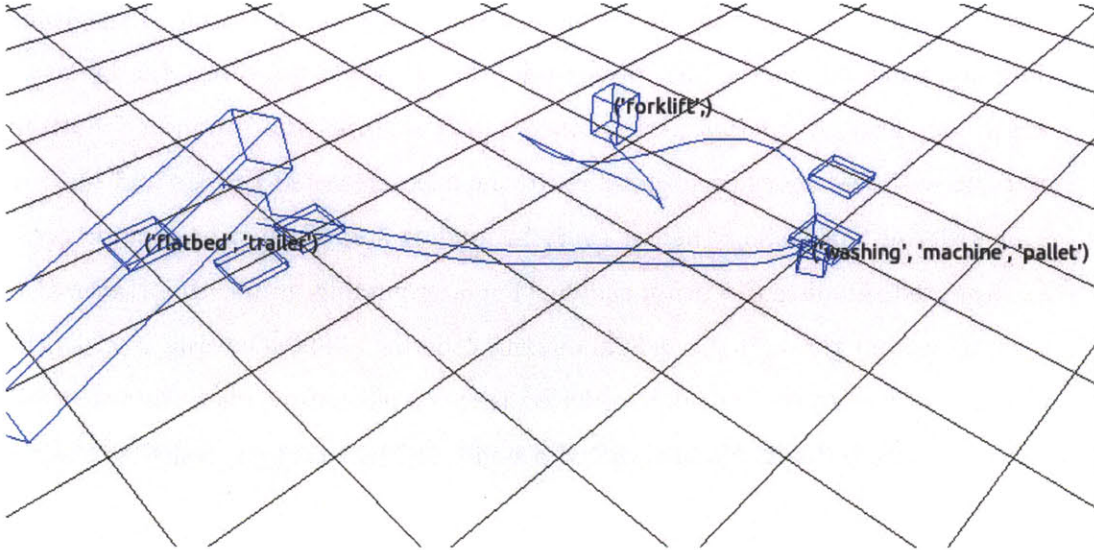


Figure 2-2: The same physical as in Figure 2-1 but from a 3D perspective. Note how the forklift, the pallet and the trailer are represented as prisms

assignments of the  $\gamma$  variables which maximize the correspondence probability

$$\arg \max_{\Gamma} \prod_{\phi} p(\phi = \text{True} | \Lambda, \Gamma),$$

which is equivalent to the minimum cost formulation

$$\arg \min_{\Gamma} \sum_{\phi} -\log p(\phi = \text{True} | \Lambda, \Gamma).$$

The inference is a beam search for the optimal assignment of groundings to the  $\gamma$  nodes of the grounding graph. The procedure traverses the  $\gamma$  nodes of the grounding graph in a certain order and tries to assign groundings for each traversed node. After each node, the inference only keeps the best few assignment sets so far (as given by the cost function) and discards the rest. Then, the next grounding assignments are considered in combination with the previous grounding assignment sets and the process repeats until groundings are assigned for all  $\gamma$  nodes. In the process the action sequence is found as the state advances for each EVENT ESDC.

The traversal ordering is important, because choosing which nodes to ground

earlier could have an effect on the final result of the beam search. If a grounding does not make any factor fully observable, then it would not direct the beam search which can lead to inferior performance. [55] performs the grounding bottom-up, starting with the nodes corresponding to the deepest nested ESDCs and moving up. This ordering guarantees that at every grounding step at least one factor potential is computed, guiding the beam search. For a grounding graph, this traversal order is implemented through the `DFSOnLeaving` ordering. `DFSOnLeaving` is a depth-first traversal of the graph, but node ordering is extracted not by when the traversal first visited a node but when it last visited a node. `DFSOnLeaving` is called at the  $\gamma$  node of the top level ESDC

---

**Algorithm 1** `DFSOnLeaving` Algorithm for ordering the nodes of a grounding graph

---

```

Procedure DFSOnLeaving
Require: A grounding graph  $g$ 
Require: Current node or factor in the graph  $c$ 
Require: A list of expanded nodes or factors  $E$ , initially empty
Require: A list of visited nodes or factors  $V$ , initially empty
  Add  $c$  to the list of expanded nodes  $E$ 
  for all neighbor node or factors  $n$  of  $c$  do
    if  $n \notin E$  then
       $V \leftarrow \text{DFSOnLeaving}(g, c, E, V)$ 
    end if
  end for
  Add  $c$  to the list of visited nodes  $V$ 
return  $V$  /* the list of visited nodes is the traversal ordering */

```

---

## 2.2 Natural Language Modeling and Processing

Natural language models can be abstractly divided into sentence-level models and multi-sentence models. N-grams [20] and Context-Free Grammars (CFGs) [32] are examples of commonly used language models. While CFGs are clearly sentence-based models, N-grams are sentence-agnostic. In practice such models are usually trained from large corpora of natural language such as the Penn Treebank [41]. The Stanford parser [35] is an example of an automatic parser, trained on the Penn Treebank which

returns CFG parses of the language.

When the language meaning is defined over multiple sentences the concept of coreference can be used to model the semantic connections across sentences. Linguistic coreference and coreference resolution have been studied extensively both from the linguistic side [32, 48, 34] and from computational linguistic side [22]. Some of the most successful examples of coreference resolution solvers, ARKref(based on [27]) and Reconcile [54] automatically extract coreferences from the text.

When following commands in a realistic environment, robots need to ground the meaning of the command phrases to the physical reality. The symbol grounding problem [28] is the link between computation which happens on abstract symbols such as numbers, strings or objects and the physical reality. In the case of giving natural language commands to a robot, the problem consists of mapping from the words to their meanings, which are a representation of the environment with direct correspondence. Multiple systems have inferred a robot action plan from natural language commands by modeling the language [57, 36, 30, 40, 24, 55]. The current work applies the method from [55] to a new robot, the PR2.

There is also a variety of work done on using grounded natural language as a modality for human-computer or human-robot interaction. [44] uses techniques from statistical machine translation to map from natural language directions to robot action sequences. [50] derives a finite-state machine language grammar from examples and uses features for mapping between words and objects in the world (which in their case are color rectangles on a computer screen). [26] also uses features for mapping between words of the language to objects in the environment and their relative position. It also includes a simple case of coreference — the use of the keyword “that” to refer to the object described in the previous sentence. [56] learns a mapping directly between sensory inputs and words in the language, and uses a fixed CFG to model the language. [37] and [39] use ontologies to represent language meanings. Computer vision techniques such as object detection in images ([25]) allow grounding of the meanings of certain words (such as “bicycle” or “door”) for which a trained object detector exists. [29] represents verb meanings as finite state machines and learns their

structures from a set of features of the environment. [52] allows robots to invent new words and communicate their meanings to other robots. The work presented in this thesis learns the grounding based on large set of features and our natural language model consists of a combination of a context-free grammar (ESDCs) and coreference.

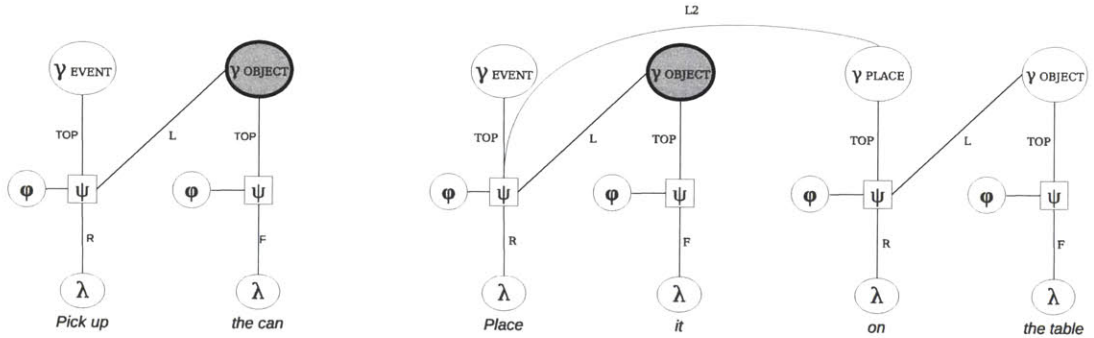
Discourse Representation Theories (DRT) [38, 47] are another way to model language semantics across sentence clauses. DRTs are multi-sentence language models and describe the representation the logical structure of a discourse, which could consist of a monologue as well as a dialog. DRTs represent entities as variables and all the relations between the entities as logical formulas deduced from the content of the discourse. To the best of our knowledge there are no real-world applications of DRTs on grounding natural language. While DRTs are state-of-the-art in theoretic language modeling, the work presented in this thesis aims to bring the practical robot capabilities closer to this theoretical understanding by using coreference to model semantic connections.

While DRTs are more expressive and can represent more semantics than sentence level models such as CFG, many practical natural language applications use the simpler, less expressive approach. This provides a opportunity for research for applications which bring some of the properties of DRTs into grounded natural language. Here, we can identify three such possibilities:

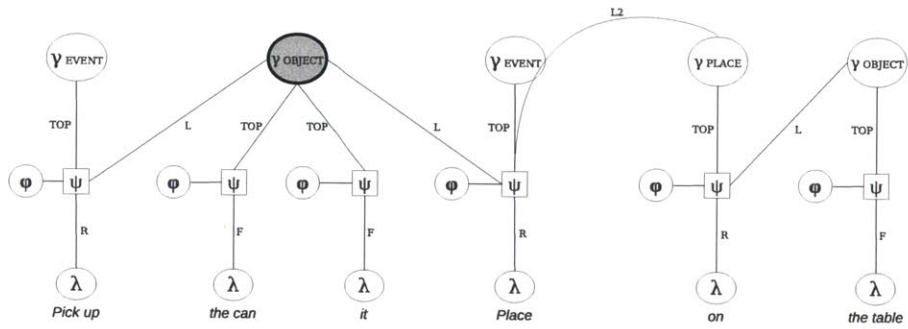
1. Extending the language models to handle cross-sentence relations (such as coreference information)
2. Representing composite semantics (via ontologies) — for example “Clean the table”
3. Using logical operators — for example “each”, “only if”, “the number of”

In this thesis we only address the first of the three, but future robotic systems might improve on multiple of these, and even find new ways to do so.





(a) Grounding graphs for the phrases “Pick up the can. Place it on the table.” *before* applying coreference. The coreferent nodes are highlighted in gray and with a darker boundary.



(b) Grounding graphs for the phrases “Pick up the can. Place it on the table.” *after* applying coreference. The node highlighted in gray and with a darker boundary is the result of merging the two coreferent nodes.

Figure 2-3: Merging coreferent  $\gamma$  nodes.

## 2.3 Coreference resolution as a part of $G^3$

In [53] our goal was to enable robots to follow commands consisting of multiple clauses and to participate in question-asking dialog. The strategy we used was to merge the grounding graphs through the coreferent  $\gamma$  variables (Figure 2-3). This way we constrain the groundings for the coreferent phrases to be the same. When we merge nodes, we also connect previously disconnected grounding graphs. Thus, the inference would not just find the right action sequence for each graph and then concatenate the action sequence but instead perform the inference jointly. Also, besides having a bigger and more interconnected grounding graph to perform inference on, this strategy does not change the grounding model of  $G^3$  allowing us to use the same grounding model in the cases with and without coreference.

To implement this merging strategy we have two needs arising. First, we need to guarantee that the graph merging preserves the qualities of the grounding graph. That means that there would be no new factor types and the potential function  $\Psi$  will be well defined on the merged graph.

Second, we need to adapt the inference procedure used in practice to support merged graphs. Merging graph nodes makes the graph structure more complex. In  $G^3$  without coreference all the resulting grounding graphs are trees. The inference assigns groundings to the grounding graph nodes in an ordering adapted from the `DFSOnLeaving` ordering. For that reason we create a new ordering which works on merged graphs, and preserves some of the good properties of the `DFSOnLeaving` ordering.

## Chapter 3

# Applying Generalized Grounding Graph ( $G^3$ ) to PR2

To apply  $G^3$  on the PR2, we need to take care of the three phases of robotic behavior — perception, planning and actuation. Our first steps for applying  $G^3$  on the PR2 is to allow perception and actuation, which we achieve by interfacing  $G^3$  with ROS, the software system of the PR2. To perceive the world we wrote wrappers around object recognition and positioning capabilities which work through ROS. These wrappers extract a robotic state which  $G^3$  can use for inference. To perform actuation we defined a set of atomic actions that the robot can perform in its environment through ROS and we wrote wrappers around a PR2 library called `pr2_pick_and_place_manager`.

For planning, we learn a grounding cost function for the PR2 (Section 2.1.2) from a corpus we collected through crowd-sourcing, utilizing the Amazon’s Mechanical Turk (AMT) platform. To take into account aspects the PR2 for which  $G^3$  was not originally designed we made modifications to allow it to infer with which hand the robot performs the actions and to predict what would be the trajectories of the actions. To predict the hands’ trajectories we create and use a database of recorded trajectories of the robot performing different actions and select trajectories via a heuristic.

In this chapter we first provide background on the RP2 and ROS, and then devote sections for the description the implementation of the perception, actuation and

planning phases, and also a section to describe how we evaluate the robot’s grounding and planning abilities.

## **3.1 PR2 and ROS overview**

This section describes the base software of the PR2 robot — ROS, the PR2’s sensors and actuators of interest and the software libraries that we utilize. We provide only concise high-level descriptions and refer to the official documentation online for more details.

### **3.1.1 ROS**

ROS (Robotic Operating System) is a collection of software libraries and tools for developing robotic software. ROS serves as a gluing layer for connecting sensors, actuators and processing software on a robot. ROS achieves this through message passing. Instead of having one big program to run on the robot, ROS allows robots to have multiple specialized processes (called nodes) which communicate with each other. ROS provides the communication infrastructure over standard TCP or UDP protocols. To allow processes to communicate using ROS, a central process called ROS Master [7] needs to be active. It allows processes to connect many-to-many (via topics) and one-to-one (via services) and it contains a parameter server which can hold some shared system parameters.

ROS can be installed and run on top of the three major computer platforms — Linux, Mac and Windows and also on multiple robot systems. Official support at the time of the writing of this thesis is only provided for Ubuntu Linux [15].

ROS nodes [9] are processes that perform computation and communicate via ROS. Nodes can interface with a sensor or an actuator, or perform computation. Any operating system process can be a ROS node. All information between ROS nodes travels in the form of ROS messages [8] via ROS topics [17] and ROS services [16]. Each ROS node can publish and subscribe to multiple ROS topics and participate in multiple ROS services. ROS messages are strongly typed and each message has a

Table 3.1: An example of a ROS message type definition —  
TabletopDetectionResult

```
# Contains all the information from one run of the tabletop detection node

# The information for the plane that has been detected
Table table

# The raw clusters detected in the scan
sensor_msgs/PointCloud[] clusters

# The list of models that have been detected
household_objects_database_msgs/DatabaseModelPose[] models

# For each cluster, the index of the model that was fit to that cluster
# or -1 if no fit was successful for that cluster
# keep in mind that multiple raw clusters can correspond to a single fit
int32[] cluster_model_indices

# Whether the detection has succeeded or failed
int32 NO_CLOUD_RECEIVED = 1
int32 NO_TABLE = 2
int32 OTHER_ERROR = 3
int32 SUCCESS = 4
```

defined type and structure (Table 3.1). Each ROS message is a tuple of data fields and constants, where fields can be either basic data types or other previously defined message types.

ROS topics work on a publish-subscribe mechanism — when a node publishes a message to a topic, every node subscribed to the topic receives the message. An example of a topic which the PR2 robot uses is `/robot_pose_ekf/odom_combined` on which the robot position is published as ROS message of the type `PoseWithCovarianceStamped`.

While topics are many-to-many communication media, ROS services are one-to-one. In every service the requester sends ROS message of a certain type to the replier, which responds with a ROS message of its own. The request message and the reply message need not be of the same type. An example of a ROS service is the tabletop object detection described in Section 3.1.3 in which the requester sends `TabletopDetectionRequest` message and the object detector replies

with `TabletopDetectionResult` message (Table 3.1)

ROS allows logging of topics into ROS Bags [2]. Each bag is a log of all the messages published on the recorded topics from the start of the recording until the end of the recording. ROS bags also contain timing information which shows at what time each certain message was published and can use that information to “replay” the log. In this thesis we refer to ROS bags simply as *logs*.

ROS has had multiple software versions. We performed the work in this thesis on ROS CTurtle.

### 3.1.2 PR2 sensors and actuators

[http://pr2support.willowgarage.com/wiki/PR2%20Manual/Chapter9#Sensor\\_Overview](http://pr2support.willowgarage.com/wiki/PR2%20Manual/Chapter9#Sensor_Overview) provides a full description of the PR2’s sensors. In this section, we briefly describe the sensors relevant for this thesis.

PR2s head has two degrees of freedom: turning left-right along the vertical axis and up-down along the robot’s left-to-right axis. On the head there are three cameras and one light projector. The light projector projects a patterned red light onto the environment which provides information about the depth of the environment. The result is a colored three dimensional point-cloud(Figure 3-1).

The PR2 has two arms — left and right. The two arms are symmetric to each other. Each arm has 7 Degrees of freedom (DOFs) of movement. The arms look similar to human arms, yet allow for non-human movements (for example — fully rotating the wrist around its axis). Each arm ends with a claw that can be opened and closed. It is with the claw that the PR2 grasps objects.

The PR2 has a wheeled base and a horizontal LIDAR above it. The LIDAR is helpful for estimating the robot’s position.

### 3.1.3 Software libraries

Here, we perform a high level overview of the software libraries in ROS relevant for the project. We describe `pr2_pick_and_place_manager`, which we used for object



Figure 3-1: The RP2 depth sensor has high accuracy and can perceive different types of objects and subjects, even humans. In this rendering, the author is perceived as a point cloud, next to a table in front of the robot.

detection and atomic actions, `gmapping` Simultaneous Localization And Mapping (SLAM) which we used to provide non-drifting position estimate, and the Gazebo simulator, which we used for recording videos and action trajectories of the robot.

#### `pr2_pick_and_place_manager`

We used a package in ROS called `pr2_pick_and_place_manager` as an API in the set of atomic actions we modeled for the PR2. It provides access to pickup, place and returning hand motions of the robot as function calls. It also provides access to object detection as a function call. We provide the signatures of those function calls and a high level explanation of how they work, but first we describe the household object database which PR2 uses to aid object detection.

**Household objects database** The household objects database is central in enabling the robot to fulfill object detection and object grasping. We do not call directly the database but the object detection uses it. It consists of items widely available for sale in IKEA, Target and common convenience stores. The database is provided by Willow Garage. The database stores a mesh representation for each object and a list of pre-computed grasp points. Each object also has a list of tags. The version of the database that we used consists of 170 objects, with 22 possible tags. The most common tags were `glass`, `bowl`, `bottle`, `cup`, `toy` and `can`.

**Detect objects** The signature of the object detection function call is `call_tabletop_detection()` and does not require any arguments. It returns a 2-tuple consisting of a list of detected objects and a table, if successful.

The object detector compares the detected objects against a database of known objects and if there is a match, the ID of the known objects is tagged to the recognized object. The result is a list of the detected objects, containing the 3D positions and orientations of the objects, their model ID, and the point cloud the robot perceives as them. Figure 3-2 shows a rendering of the object detection accuracy by drawing the mesh model of the detected object right over the point-cloud recognized. The format



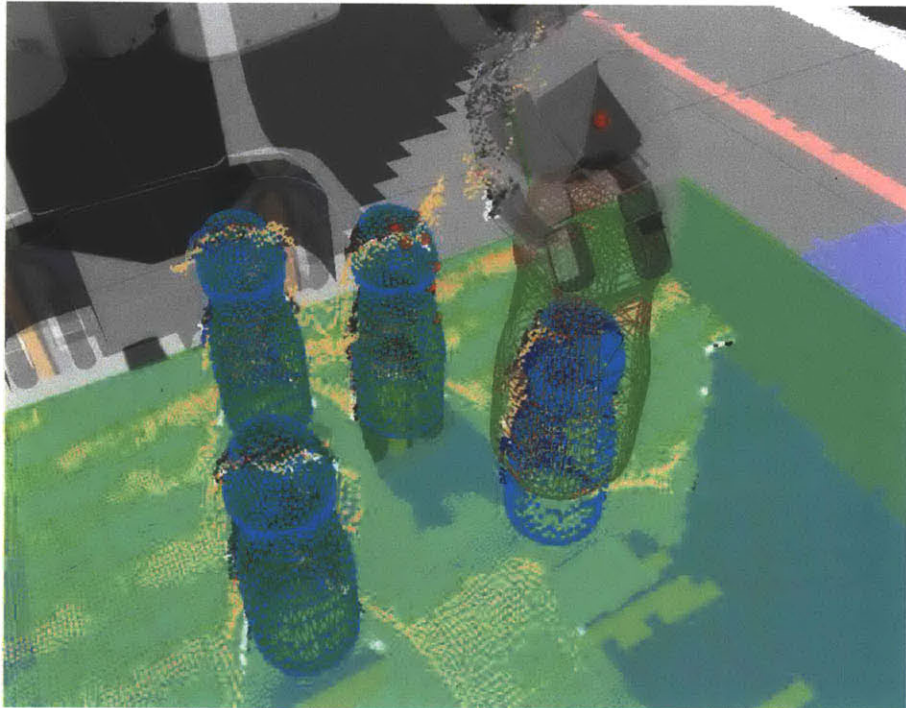


Figure 3-2: PR2 recognizes the objects on the table according to an object database. Each recognized object is drawn as a blue mesh, matching the underlying point-cloud for the object.

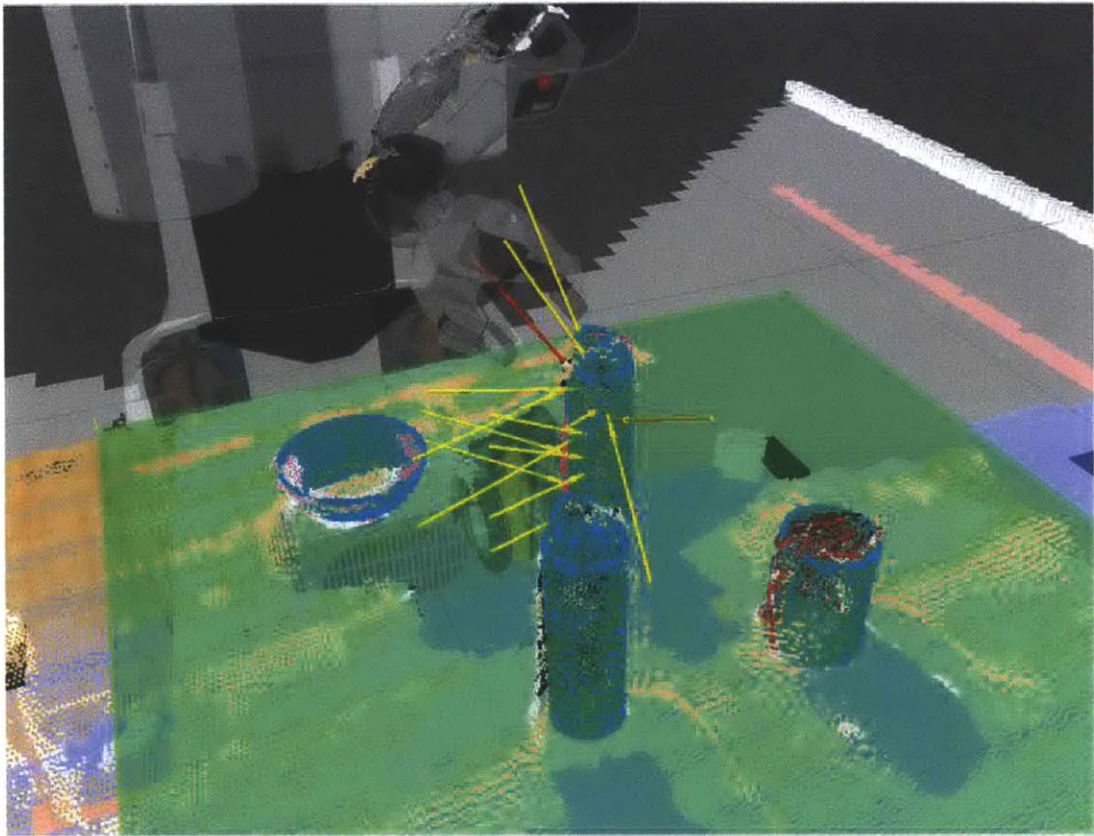


Figure 3-3: PR2 attempts to plan a pickup action using multiple pre-computed grasp poses for each object in the database. Each yellow arrow represents a grasp pose for which the robot could not find a suitable motion plan. The red arrow represents the selected grasp pose.

of the table contains a coordinate frame, the  $x$ - $y$  plane of which is the tabletop and a range in  $x$  and  $y$  defining the boundaries of the table. This representation assumes rectangular tables.

**Pickup object near point** The signature of the pickup function call is:

```
pick_up_object_near_point(point_stamped, whicharm)
```

It makes robot to pick up an object near a certain point in three-dimensional space with a certain arm. The robot first opens its claw. The robot then looks at this point and runs object detection. Then it selects the object that is closest to this point as

the object to pick. The robot then looks at the possible grasp positions for the object in the database and tries to plan a collision-free trajectory for the arm to reach the grasp position (Figure 3-3). Once it finds a satisfying trajectory the robot moves the arm to the pre-grasp position. Once the arm reaches the pre-grasp position the robot moves its wrist forward in the wrist frame of reference, surrounding the object. Then the robot closes its claw to grasp the object.

**Place object near point** To place an object that the robot holds on a certain spot on the table there are two function calls which need to be called in sequence:

```
set_place_area(place_rect_center, place_rect_dims)
```

```
put_down_object(whicharm)
```

The first call sets the center and the dimensions of the area on the table at which the object needs to be placed. The second call moves the arm and places the object down. The robot attempts to find a trajectory to move the held object above the place area through inverse kinematics (IK), trying to avoid collisions. Figure 3-4 shows the robot, holding an object and attempting to place it under the green arrow. Once the object is above its assigned place area the robot claw opens and the object falls on the table.

**Return hand to side** To return a hand to the side the function call is:

```
move_arm_to_side(whicharm)
```

It moves the joints of the arm to sets of predefined angles. This moves the robot arms to the side in a slightly bended configuration. This way the hands do not obstruct the robot's view at the table.

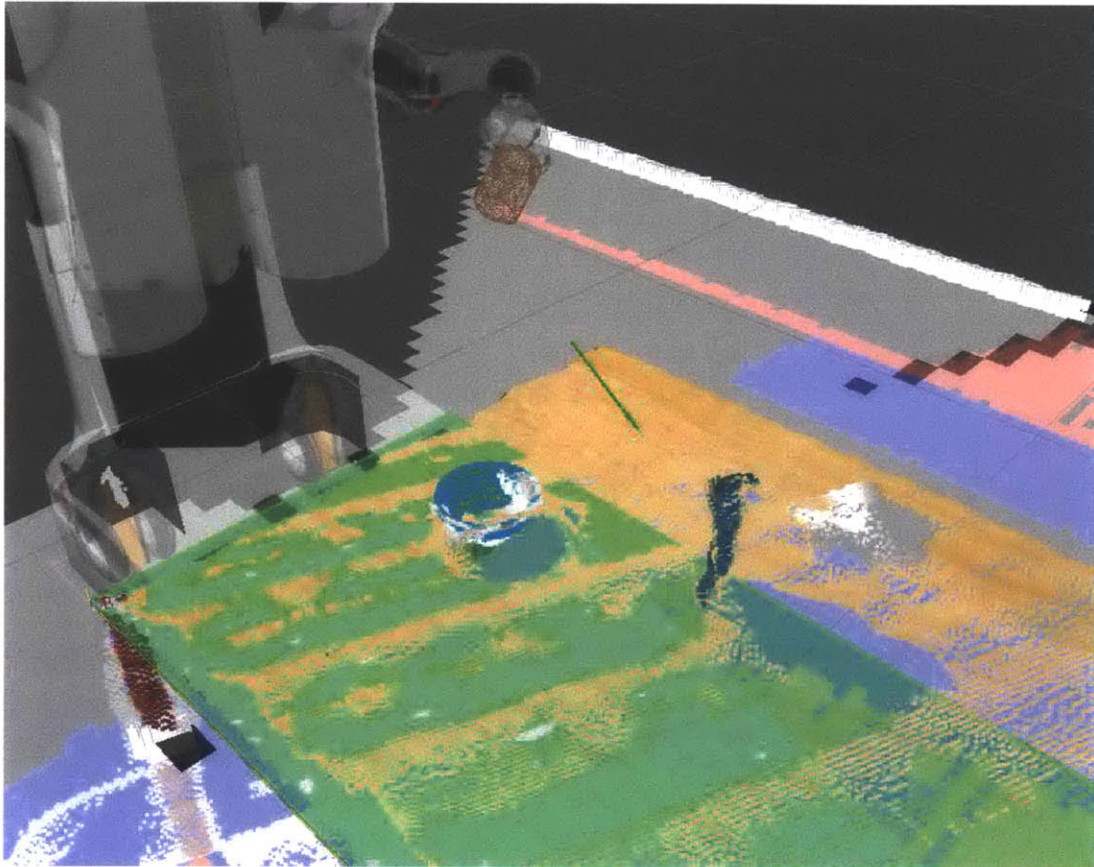


Figure 3-4: The PR2 places objects by moving its hand to a place pose and then opening its claw. The green arrow shows the pose selected, above the table.

## Positioning

We run `gmapping` SLAM on the robot in order to have accurate positioning for the base of the robot. `gmapping` provides much more accurate estimate of the robot position than the odometry itself. It publishes a transform between the robot's base frame and the global map frame. `gmapping` works by using the information from the base LIDAR as an input to a SLAM algorithm.

## Gazebo simulator

Gazebo [6] is a physics and rendering simulator. ROS provides a programming interface for spawning objects and reading the positions of all objects and robots in the environment. The PR2 robot with its sensors and actuators can also be spawned in simulation. Gazebo also allows spawning virtual cameras which we use to record videos of the robot in simulation and the positions of the robot hands and manipulated objects.

## 3.2 Perception

The format in which the PR2 robot perceives the environment through object detection is not the same as the format  $G^3$  uses to represent the robotic state. To extract a robotic state which for  $G^3$  we created a state extractor procedure which subscribes to robot and environment measurement updates and is able to produce an aggregated representation of the state. The state extractor takes information about the robot position in the global frame of reference and object detection information.

### 3.2.1 Robotic state

We modeled the robotic state for the PR2 to contain the physical context described at Section 2.1.4 but also to keep track of object relationships such as which object is in which hand, and which objects are on which table. Those are only used at inference stage to compute which possible actions the robot can do: with which arm it can pick

objects and which arm it can place objects, as well as at which positions it should attempt placing them.

The first relationship between objects that we model is which objects are held by which hand. Initially, the robot's hands are empty, but after the robot performs a pickup action we add the relationship between the robot hand doing the pickup and the picked up object. The second relationship is about places on a table. For every table we discretize its surface into a 5-by-5 grid and create place for each grid cell. The relationship is between the places and the table and says that the certain place is on the table. The PR2 will only attempt to place objects in places that are on a table.

We represent the robot in the physical context by three objects — the robot's body, and the robot's left and right hand. The physical context also includes any detected tables and objects and the places on the tables.

### 3.2.2 Robot position

The state extractor subscribes to the position of the robot in the global frame on ROS topic `/robot_pose_ekf/odom_combined`. The position is represented as a `PoseWithCovarianceStamped` ROS message containing an estimate of the  $(x, y, z)$  position and a quaternion orientation of the robot base. We use the knowledge of the robot position to situate the detected objects into the fixed global frame of reference.

The robot's proprioception gives us information about the position of its hands. We create a physical object representing the robot's body with a tag `robot` and two physical objects representing the robot's left and right hand. Each hand object has three tags — `robot`, `left/right` and `hand`. We use the position and orientation of the robot wrist rolls to represent the robot hand: `l_wrist_roll_link` for the left hand and `r_wrist_roll_link` for the right hand.

### 3.2.3 Perceiving objects

When the state extractor calls object detection, it returns a description of a table and a descriptions of the detected objects in the format described in Section 3.1.3, which is not the same as the format used in  $G^3$ 's inference. Therefore, we convert the detected table and objects on it into the physical object representation used by  $G^3$  in the physical context of the robotic state. The table description as returned by the object detector consists of a coordinate frame and a rectangle in the  $x$ - $y$  plane of it, representing the boundaries of the tabletop. We convert corners of the rectangle to the global coordinate frame and use their height to set the height of the table prism. For the objects, we use the point cloud returned by the object detector, transform it into the global frame of reference and take its bounding box in  $x$ ,  $y$  and  $z$ . We use the tags of the model ID in the database for tagging the created physical object if the object in the database, or just the tag `object` if the point cloud is not recognized as a part of the database.

However, the object detector is stateless; a call of the object detector at a later point in time does not contain any references to previously detected objects. This could cause a duplication problem if the robot detects the same object at different times and considers it to be two different objects. To avoid duplication we check if there is an object at approximately the same position and of the same type. If so, we assume that it is the same object as recognized earlier and we only update its position in the robot state but do not create a new object instance. Algorithm 2 is a detailed description of this process. In implementation we used a distance threshold of 0.2 meters.

## 3.3 Actuation

### 3.3.1 Atomic actions

We model the PR2 to use five different types of atomic actions in the robot state covering basic tabletop manipulation. Every plan obtained during inference is a se-

---

**Algorithm 2** Algorithm for incorporating object detection into the robotic state and avoiding duplicate objects or tables

---

**Require:** A distance threshold  $d$

**Require:** A previous or initial robot state  $S$

**Require:** 2-tuple (Table detection  $T_0$ , list of object detections  $O$ )

**if**  $T_0$  overlaps with any table  $T_S \in S$  **then**

$T_S \leftarrow T_0$

**for all** object  $o_i \in O$  **do**

**for all** object  $o_j$  on  $T_S$  **do**

**if**  $distance(o_i, o_j) \leq d$  **and**  $o_i.model = o_j.model$  **then**

$o_j \leftarrow o_i$

**break** to next  $o_i$

**end if**

**end for**

        Add  $o_i$  to  $S$  on table  $T_S$

**end for**

**else**

    Create a new table instance  $T_1 \in S$

$T_1 \leftarrow T_0$

**for all** object  $o_i \in O$  **do**

        Create a new object instance  $b$  in  $S$

        Set  $b$  on  $T_1$

**end for**

**end if**

---



quence of these actions. The five action types are called `JustPickup`, `JustPlace`, `ReturnHand`, `Pickup` and `Place`. The `Pickup` and `Place` actions are a combination of the `JustPickup` (and respectively `JustPlace`) action followed by `ReturnHand` action. For that reason here we describe the implementation of `JustPickup`, `JustPlace` and `ReturnHand` actions on top of the `pr2_pick_and_place_manager` library(Section 3.1.3).

#### `JustPickup`

The `JustPickup` action type takes a *object* in the physical context and a *hand* as arguments. It finds the centroid of the object in 3D and calls the `pick_up_object_near_point` function (Section 3.1.3) using this centroid.

During inference, the robot is allowed to perform a `JustPickup` action if the object is in reach of the arm and the arm does not hold another object.

#### `JustPlace`

The `JustPlace` action takes a *place* in the physical environment and a *hand* as arguments. It sets the place area center to be the center of the place and uses fixed place dimensions via the function `set_place_area`. Then it calls `put_down_object`(Section 3.1.3).

During inference, the robot is allowed to perform a `JustPlace` action if it is holding an object and the space it is trying to place the object at is free and reachable.

#### `ReturnHand`

The `ReturnHand` action takes a *hand* as an argument and directly calls the `move_arm_to_side` function (Section 3.1.3).

During inference, the robot is allowed to perform `ReturnHand` if and only if the previous action was a `JustPickup` or `JustPlace` action.

## 3.4 Planning

As planning consists of inferring the right plan for a given command in a given environment, to enable the PR2 to plan, we need to predict the effect of a plan and

to score the grounding of the plan to the command. We allow the robot to predict the effect of a plan consisting of an action sequence by predicting the trajectories that the robot’s parts and the objects in the environment will take. Then, we allow the robot to score different groundings by learning a grounding cost function from a corpus of natural language commands aligned with robotic actions and groundings.

### 3.4.1 Predicting action trajectories

To predict accurately what will happen when the robot attempts to pick or place an object we want the predicted trajectory to be as close as possible to what would really happen. The gold standard solution would be to have whatever action planner is there and to run it in simulation. However, simulation based on the robot perceptions might not be accurate, because the perception is not perfect and the action planner works in closed-loop fashion. Also, the simulation is slow, the action planner depends on the whole ROS system running, which would require running the simulation in a virtualized environment, making it even slower. We would require a lot of trajectory predictions when trying out different alternative actions, which makes using the same action planner impractical. Also, we cannot use a simplistic approach here because the robot hand trajectories are non-trivial in their positions (Figure 3-5).

Therefore, we take a simplified and computationally faster approach to allow the PR2 to predict the effect of its actions. We collect a database of robot action trajectories, relative to the robot’s base coordinate frame. Then, when the robot needs to infer an action we use a trajectory from this database to model what happens in the environment. If the action manipulates an object we also use the corresponding object trajectory. The database consists of a total of 180 trajectories — 90 hand trajectories and 90 corresponding object trajectories. Trajectories in the database also have tags about the action that generated them — this way we guarantee not to use trajectory generated by a different action, for example, using pickup trajectories for predicting place actions or vice versa (Figure 3-5).

We cannot just select any hand trajectory from the database for prediction — we want the trajectory which would be closest to what would actually happen. Since

we do not know what would actually happen, we employ a heuristic approach for selecting the trajectory from the database. In the database, for every path we record the initial position of the robot arm performing the action and the position furthest from the initial that the arm goes through. Then, during inference, for all robot actions we know the initial position of the robot arm, given by the position of the arm before the action. Then, for pick, place and return hand actions we can estimate the furthest point by the position of the object picked up or the place at which the object needs to be placed or by the default position of the arm to the side of the body.

Our heuristics consists of finding the trajectory in the database with most similar starting and furthest point. Since those are two different criteria for comparison, we combine them by summing the squared distances (Algorithm 3).

---

**Algorithm 3** Algorithm for selecting a action trajectory from a database

---

**Require:** A desired starting  $(x_0, y_0, z_0)$  position of the hand

**Require:** A desired farthest  $(x_f, y_f, z_f)$  position of the hand

**Require:** A database of trajectories  $D$

**for all** hand trajectories  $t_i$  in  $D$  **do**

$d_{\text{start}} \leftarrow$  3D distance between  $(x_0, y_0, z_0)$  and the path’s starting point

$d_{\text{far}} \leftarrow$  3D distance between  $(x_f, y_f, z_f)$  and the path’s farthest point

Combined distance  $d_i \leftarrow d_{\text{start}}^2 + d_{\text{far}}^2$

**end for**

**return**  $\arg \min_{t_i} d_i$  /\* the trajectory with closest overall start and far points \*/

---

Even if we find a trajectory which has close starting and furthest point to the ones we desire, the positions would be still slightly different which would introduce discontinuities in the trajectories. Such discontinuities are undesirable because they could lead to activating the wrong features, and would also create unrealistic predictions for the robot actions. Therefore, when we have selected a trajectory from the database, we perform an smooth coordinate transformation on it so that the starting and furthest point would match exactly the ones we desire. The coordinate transformation is defined as follows. Let our predicted trajectory has a start point  $P_0$  and far point  $P_1$ . Suppose also that the desired start and far points are  $D_0$  and  $D_1$ . Then

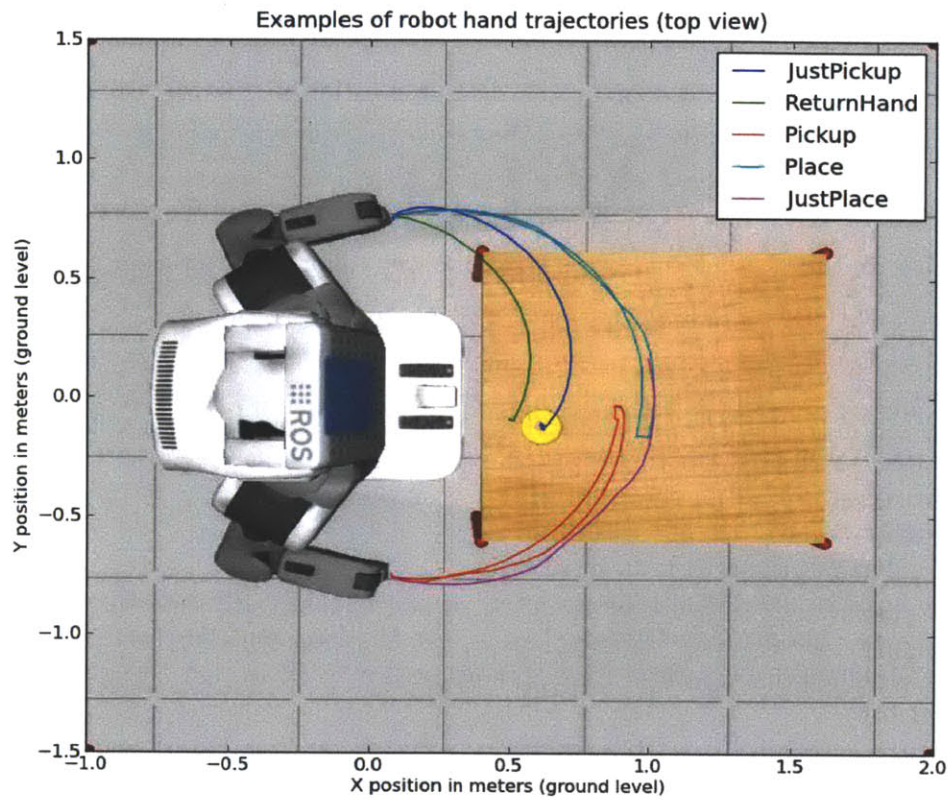


Figure 3-5: The robot hand trajectories can have complex structure and shape. Some trajectories of the left hand and objects picked/placed are drawn on the top, and of the right hand on the bottom. Each trajectory is of different action. The background is for illustration purpose only and is not the actual environment in which those actions were recorded.

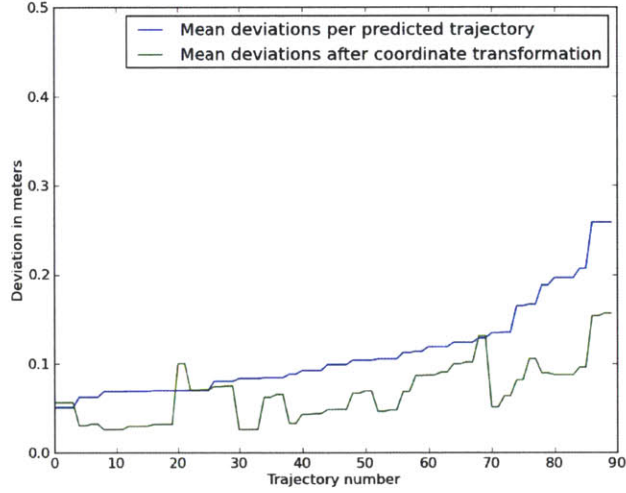


Figure 3-6: Mean deviations from predicted and the transformed trajectories other trajectory for each trajectory in the database (leave-one-out cross-validation).

for every point  $A$  in the trajectory we correspond a new point  $B$ , such that:

$$\vec{AB} = \frac{|AP_1|}{|AP_0| + |AP_1|} P_0 \vec{D}_0 + \frac{|AP_0|}{|AP_0| + |AP_1|} P_1 \vec{D}_1$$

It is easy to check that this correspondence sends  $P_0$  to  $D_0$  and  $P_1$  to  $D_1$ .

To evaluate how well the trajectory can predict motion trajectories we performed leave-one-out cross validation on the database. We consecutively took a trajectory from the database and selected a trajectory from the remaining ones according to Algorithm 3. We then computed the mean deviation between the the paths. To do so, we re-sampled each trajectory using 100 points and computed distances between corresponding points. We also computed mean deviations between the original path and the transformed predicted path. Figure 3-6 shows the resulting mean deviations before and after the smooth coordinate transformation. In 70/90 (77.8%) cases the mean deviation of the predicted path less than 13cm, and in 78/90 (86%) cases the mean deviation of the transformed path is less than 10cm.

We also present an illustration of how the trajectory database predicts action trajectories. In Figure 3-7 we show a trajectory we obtained by running the motion

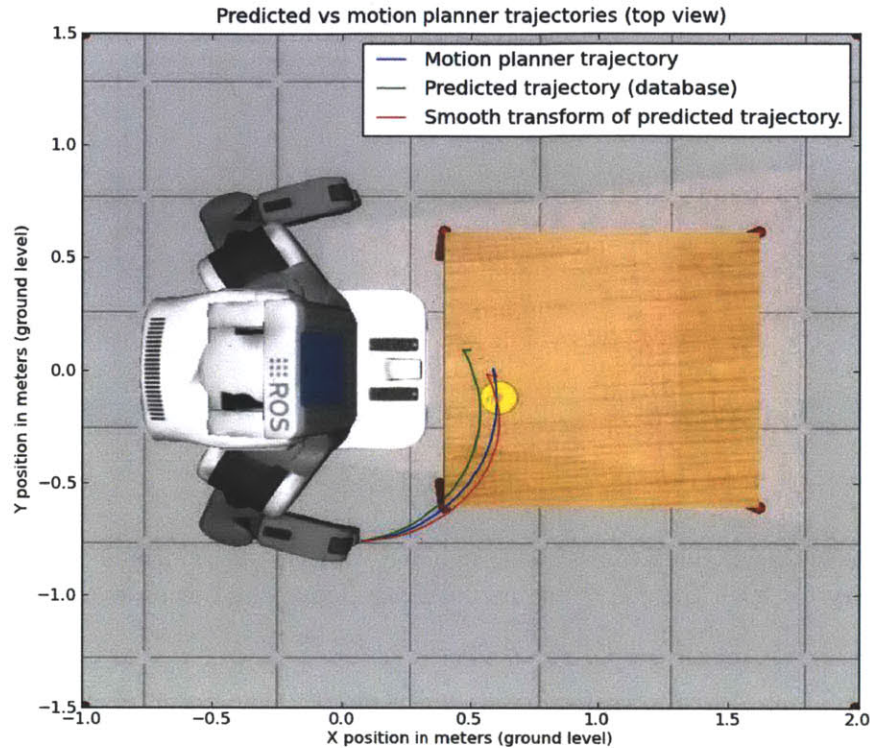


Figure 3-7: Comparison between true trajectory for JustPickup action from the motion planner, predicted trajectory from the database, and the transformed version of the predicted trajectory. Top view.

planner, the closest trajectory from the database according to the heuristic, and the trajectory obtained after performing smooth transformation.

### 3.4.2 Corpus collection

We recorded 7 scenarios of the PR2 robot performing a series of actions in simulation. Each scenario comprised of 2 to 4 actions. In each video we placed the robot next to a table and spawned 2 to 4 objects from the household objects database (Section 3.1.3). We used different colors to render the objects (black, white, red or yellow) and the position information from Gazebo about the object trajectories contained the color of the object in the object tags. For each scenario we recorded a video of the robot doing the actions and a log in simulation.

We then partitioned the videos and the logs into subsequences of actions and created a total of 42 shorter videos. We posted these videos on AMT and asked the subjects to write a command that they would use if they wanted the robot to perform the action in the video. We presented all the subjects with the same task description (Figure 3-11) for their human intelligence task (HIT), but they saw different videos. We posted each video multiple times and received several commands for each video. We paid \$0.25 per command, and the average hourly rate that the subjects received was \$11.84. A total of 28 subjects participated. We discarded non-grammatical commands and commands which did not have anything in common with the video. The total number of grammatically correct and relevant commands is 187.

For each of those commands we then manually annotated the ESDCs structures for the text (Table 2.1) and the groundings for each ESDC (Figure 3-8). To annotate the groundings we used the position information in the log associated with each video (Section 3.1.1). Figure 3-8 shows an annotation interface which we used for ease of annotation and to reduce the chance of annotation errors. This annotation interface allowed us to annotate EVENT ESDC figures with partial paths describing only the specific action and not the whole trajectory through the duration of the scenario (Figure 3-9).

The annotated groundings presented the ground truth positions of the objects. This way we created a pairing between the commands and the position information from the simulation.

In order to learn whether a correspondence variable  $\phi$  (Section 2.1.1) is `True` or `False` we also created negative annotations. They were the same number as the positive annotations. For each positive annotation, we first changed the correspondence variable from `True` to `False` and then selected random groundings from the environment, keeping the text unchanged. Then we did a manual check that all the groundings are indeed negative and changed them when necessary according to our best judgment.

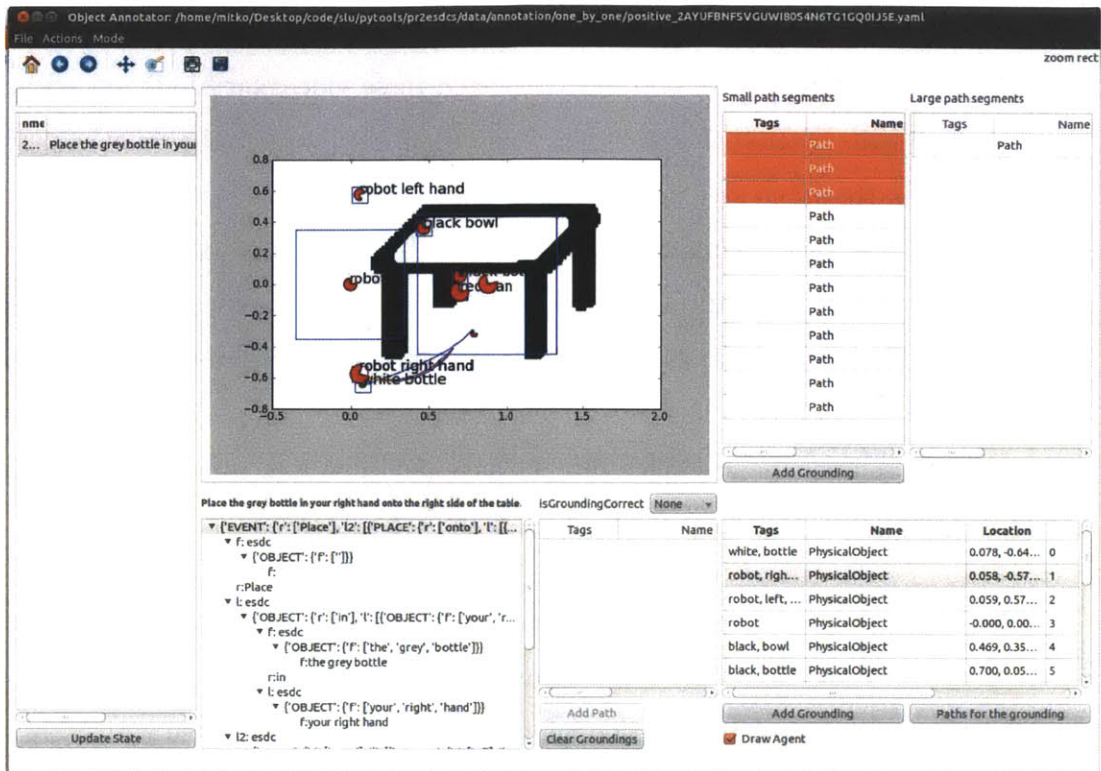


Figure 3-8: The annotation interface we used allowed us to select groundings for different ESDCs, which were then transformed to groundings in the grounding graph.

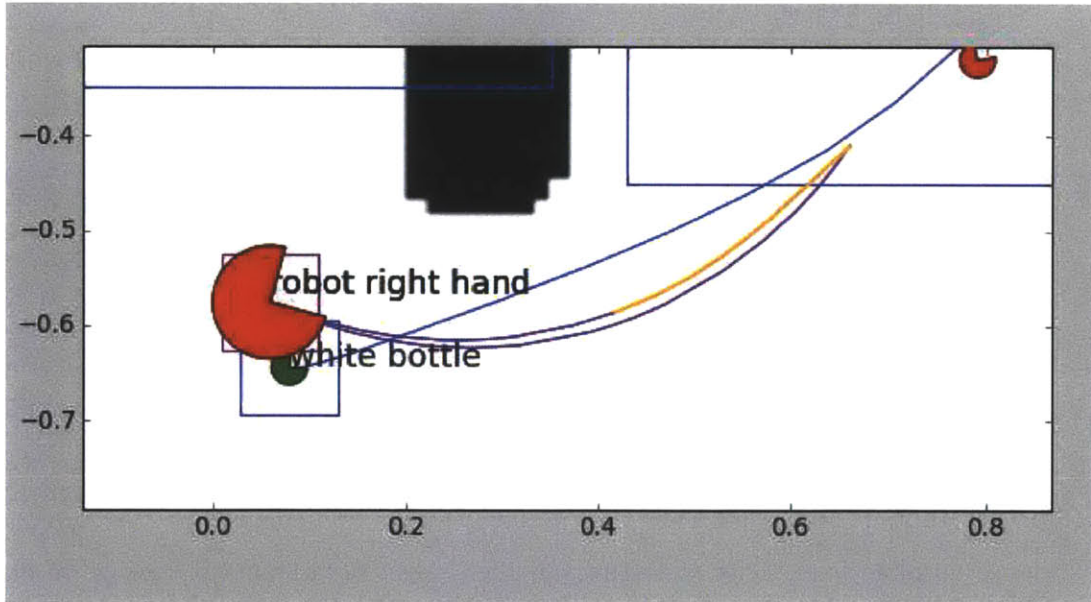


Figure 3-9: We annotated the groundings for the actions with the parts of hand trajectories corresponding *only* to the action performed. Here, in orange, we show one partial path to illustrate the annotation process.



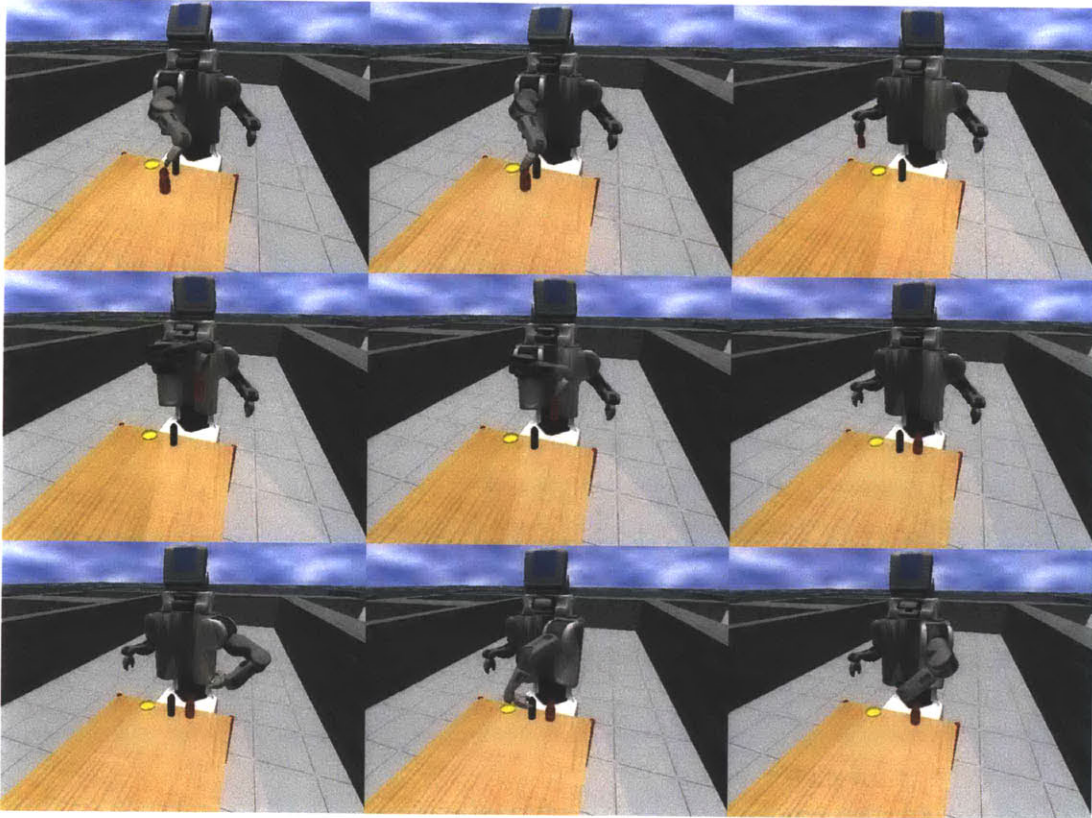
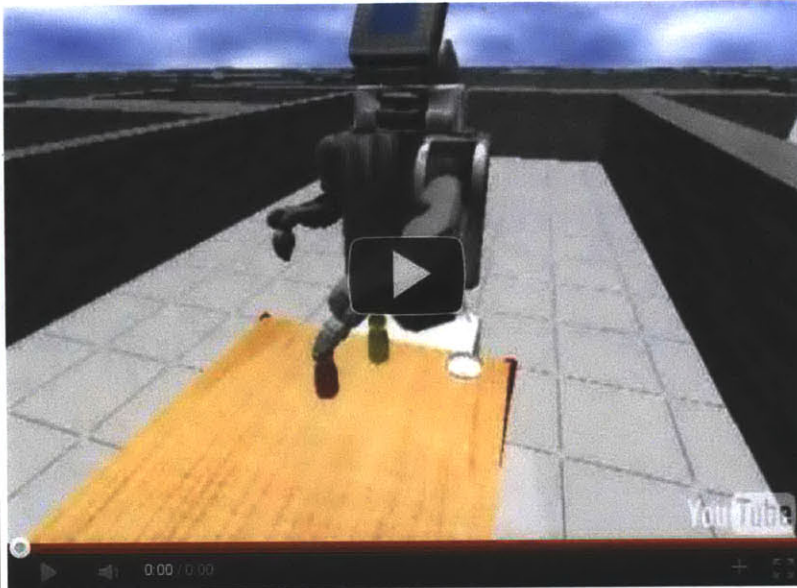


Figure 3-10: Videos can contain multiple robot actions. In this example, we have 9 evenly spaced through time screen-shots from a video, in which the robot picks a red object, places it on the table, and then picks up a black object. The order of the screenshots is left to right, and then top to bottom.

Before completing this HIT, you must agree to the [consent form](#), if you have not already done it.



We are trying to collect a diverse set of commands people might give to robot.

Please write a natural language instruction that you would speak to the robot shown, if you wanted to command them to carry out the action presented in this video.

For example, depending on the video, you might say something like, "Pick up the can".

Give a high-level command, but be specific enough that a human in the place of the robot would know what to do.

Play the video more than once and use the highest resolution possible if necessary.

Use correct grammar and punctuation.

**WRITE COMMAND HERE:**

(Optional) Please provide any further comments

Submit

Figure 3-11: The form we used to solicit robot commands clearly describes the task and the purpose. The subjects signed a consent form before the task. Then they watched the embedded video, wrote a command, and were paid \$0.25 per command.

### 3.4.3 Learning the grounding function

We use the same learning technique and features as described in [55] and in Section 2.1.2, but we also extend it by the learning and inference by learning the figure of each EVENT ESDC. The original framework only contains the agent as the event grounding, which works for robotic forklift because it is more or less a rigid body excluding the forks, which were not modeled in the physical context (as can be seen in Figures 2-2 and 2-1). However, the PR2 robot has two hands and performs manipulations with them. Its body remains stationary for tabletop manipulations, and would not be useful to learn any features on its trajectory, nor it would help during inference time. Therefore, we keep the robot body as the agent of each action, but model the hand used for each action as the figure of the EVENT ESDC.

At learning time we annotate the figure of each EVENT ESDC with the hand and corresponding trajectory (Figure 3-9). We also computed a new set of features based on the combination of the EVENT figure and the rest of the groundings for the EVENT. Those features are computed the same way as for the combination of the EVENT grounding and the rest of the grounding, just computed on the figure grounding instead. During inference time we infer which part of the robot body is the figure for each EVENT ESDC.

We split the corpus by scenarios into training and testing corpora. This resulted in 103 (55%) training commands and 84 (45%) testing commands. Since we split the corpus based on the scenarios, the resulting training and testing corpora each had an equal number of positive and negative examples. To evaluate whether the learned cost function can predict correspondence variables correctly, we used it to predict the correspondence variables  $\phi$  for each ESDC in each testing example (Table 3.2).

## 3.5 Evaluation technique

To evaluate how well the learned function predicts the correspondence variable  $\phi$  for a factor, based on observing all  $\gamma$  and  $\lambda$  nodes neighboring the factor, for every factor in the testing dataset. From the table we see that the f-score on the testing

Table 3.2: Performance on predicting the correspondence variables  $\phi$  in the testing dataset.

ESDC Type	Precision	Recall	F-score	Accuracy	Positive Examples	Negative examples
OBJECT	90.6%	81.4%	86.5%	85.8%	377	377
PLACE	86.4%	82.6%	84.8%	84.4%	46	46
PATH	96.2%	73.9%	85.5%	83.6%	69	69
EVENT	88.7%	79.1%	84.5%	83.6%	158	158

dataset is around 85% for all ESDC types. Comparing to [55] this has a higher f-score on PLACE, PATH and EVENT ESDCs and lower f-score on OBJECT ESDCs We observe higher precision than recall. This means that the cost function we learned is better at recognizing a negative example when it is given one, than recognizing positive examples.

Then for every video and command pair, in both the training and testing dataset, we perform full action plan inference using annotated ground truth ESDCs for the command and the initial state as perceived by the robot at the beginning of the video. The robot did not perceive the object color but we used the object color in the object tags during the learning phase so we manually added the color information to the initial robot states. We used the plans resulting from this inference as a basis of the evaluation.

For every resulting plan we first compare the inferred object groundings to the corresponding object grounding in the annotated corpus and we counted the cases in which the two object groundings had exactly the same tags. We observed 85.3% out of 404 pairs matching in the training corpus and 64.5% out of 288 matching on the testing corpus. Even though the learned cost function had relatively high success on inferring the correspondence variables  $\phi$  we see significant drop when inferring objects. This could be due to the fact that each factor potential is a conditional probability distribution, conditioned on the objects, but it could also be due to accumulation of grounding errors in the inference.

For a given command there could be multiple different sequences of robotic actions

which correctly follow the command. For example if the robot receives the command “Put the object in your left arm on the table,” the specific place on the table at which to put the object is not specified so the robot can put it anywhere on the table and be considered to have done the right thing. This plurality of correct responses make it hard to automatically evaluate the goodness of a selected action plan.

To address this, the forklift study in [55] performs evaluation by recording videos of the robot following the commands and posting the videos to AMT asking the subjects whether the commands make sense. For the PR2, we could not record videos for each inferred plan, because recording simulation videos in ROS is very slow and requires a human to physically run the simulation. Running it remotely in parallel on a server farm is not possible because the Gazebo simulator’s rendering engine does not run through a remote shell. For that reason we performed manual evaluation only on the plan that the robot infers. Our criterion was *Do the robot actions make sense for this command in this environment?*

To illustrate our criterion and judgment, we provide the following examples. If the robot is given a command “Pick up the red can” and on the table in front of it there are two red cans and one red bottle, then it is considered correct to pick up any of the cans but not the bottle. If the robot picked the can and then did something else such as placing the can back or picking another object with the other hand, then the robot actions as a whole were considered wrong. Also, if for example the robot was explicitly instructed to pickup an object with a certain hand (“Pick up the black bowl *with your right hand*”) then it was considered wrong if the robot used its other hand to pick up an object. If however the hand identification was ambiguous or not mentioned specific we didn’t held it against the robot if it used the other hand. For example, in “Pick up the black bowl *with right hand*” (no “your”) it is not clear from which perspective the hand is right. The camera in the video was facing the robot so left and right are reversed for the subject giving command and the robot.

Using this criterion we got 43/103 (41.7%) correct plans on the training corpus and 19/84 (22.6%) correct plans on the testing corpus. We also observe that if we consider the relationship between the plan cost and correctness and we plot the

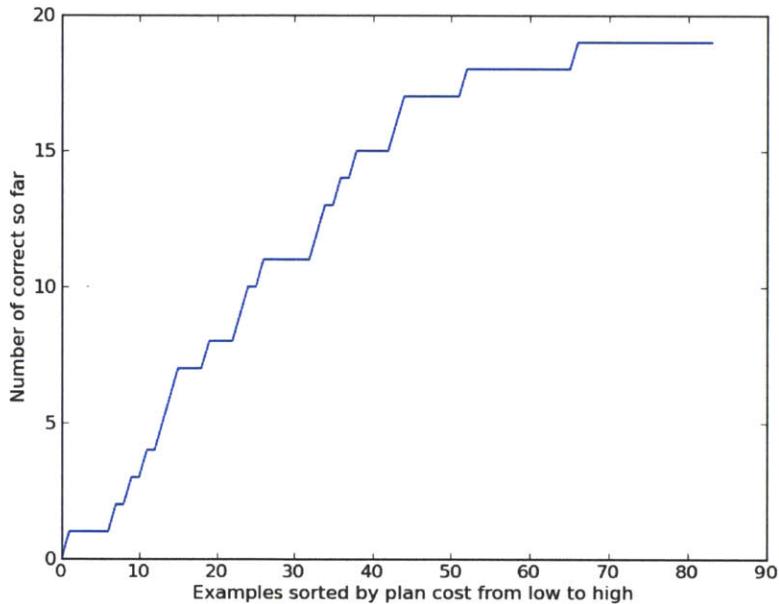


Figure 3-12: If we sort the inferred plans by their cost, the ones with lower cost are more likely to be correct.

number of correctly inferred plans among the first several lowest cost plans we get approximately concave graph (Figure 3-12) - the low cost plans are more likely to be correct. Among the 30 lowest cost plans we get 11 correct. This lower performance than the reported performance on a robotic forklift in [55]. There,  $G^3$  with correct ESDCs annotations achieves 63% on the 30 commands for which the cost of the inferred plan was lowest.

As inference depends on the learned cost function, the graph structure, the concrete environment and the representation of the robotic actions there could be multiple points of failure. The learned cost function relatively successfully predicts correspondence variables  $\phi$  between objects, places, paths and events in the PR2 environment and their description in the text (around 85%). However, this by itself doesn't guarantee high performance on the inference, because during inference the correspondence variables  $\phi$  are known, and the  $\gamma$  groundings are inferred. Thus, we see a drop in the performance (to around 64%-71%) when inferring the groundings for the object nodes. During inference, other effects could decrease overall performance — the grounding

inference ordering can affect how many and which groundings are inferred correctly, and inferring multiple different robotic actions need fail only once for the whole inference to be considered wrong. Sometimes the robot had trouble understanding commands which specified with which arm the action should be taken because the grounding for the hand phrase in the command, even if correct, doesn't constrain the actions available to the the robot. For example, for the command "Pick up a red can with your left hand." the robot might decide to pick up a red can with its right hand if the inferred trajectory scores better, or if the robot grounds a red can that is close to the right hand but unreachable for the left hand. There were several examples in which the robot inferred all the actions and groundings correctly, except for the hand, and those were considered wrong inferences by our manual evaluation.

### **3.6 Summary**

We applied the  $G^3$  framework on the PR2 robot, by modeling the robots perception and actuation, modeling the robot actions and learning a grounding model on a corpus that we collected and annotated. We achieved good learning performance, average object grounding performance and lower than previously reported on forklift performance on plan inference.





# Chapter 4

## Coreference resolution

Language is not just a sequence of sentences, but between the sentences there are also semantic connections, some of which can be captured through the linguistic technique of coreference resolution. Such semantic connections can be useful for following natural language commands which span multiple sentence clauses. To the best of our knowledge, previous work on grounding natural language commands has not included coreference resolution as a part of the language model except in some special cases of anaphoric coreference. In [53] we showed theoretically how the language model of  $G^3$  can support coreference resolution with minimal changes in the grounding graph data structure format. In this chapter we explore how this merging can be done in practice, the challenges and the performance of including coreference as a part of the language model for natural language grounding.

### 4.1 Graph merging algorithm

A *coreference resolution engine* takes a sentence with selected noun phrases and returns a partition of those noun phrases into coreferent groups. For example, if no noun phrases are coreferent every ESDC is in its own group. If all of them are coreferent then there is a single group. We use this partition to constrain the inference that if two noun phrases in the command are in the same partition group then their groundings in the environment must be the same. We impose this constraint by

merging the coreferent  $\gamma$  nodes in the grounding graphs(Algorithm 4). In this work we only apply coreference on leaf OBJECT ESDCs (Section 2.1.1). We prove that the Algorithm 4 merges the coreferent  $\gamma$  nodes in a way which does not introduce any new factors or factor types (Theorem 1). For that reason we can use the same cost function for grounding — there would not be any new factor types which means that the cost function would not be ill defined.

We want to note that the algorithm ensures that the factor types would remain unchanged by discarding coreferences when more than one of them are neighboring the same factor. Such problematic coreferences can appear if there is a reflective relation such as “the can by itself” or an equality relation such as “it is the bottle” and the coreference resolver in use correctly finds those, or if the coreference resolver is erroneous and discovers coreference between two nodes neighboring the same factor when in fact they are different entities. In our dataset and the coreference resolvers we used there were no such problematic coreferences.

---

**Algorithm 4** Algorithm for merging grounding graphs based on coreference

---

**Require:** A sequence of grounding graphs  $G$

**Require:** Coreference resolution engine  $C$

$T \leftarrow$  the text of all graphs  $\in G$ .

$N \leftarrow$  the leaf OBJECT noun phrases for all graphs  $\in G$ .

$P = C(T, N)$  /\* a partition of the noun-phrases according to  $C^*$ /\*

**for all** partitions  $p_i \in P$  **do**

**if** more than one noun phrases in  $p_i$  have nodes connected to the same factor  
    **then**

        Discard all but one of those noun phrases from the partition

        Create a separate single-phrase partition in  $\hat{P}$  for each discarded

**end if**

**end for**

**for all** partitions  $p_i \in P$  **do**

$Y \leftarrow$  the set  $\gamma$  nodes which  $\lambda$  node figures noun-phrases are  $\in p_i$

    Merge all  $\gamma$  nodes in  $Y$  into a single node

**end for**

**return** The union of all grounding graphs into a single grounding graph

---

**Theorem 1.** *After applying Algorithm 4 on a list of grounding graphs  $\{g_i\}_{i=1}^k$ , we obtain a merged graph  $g^*$  for which the following statements are true:*

1. *There is a bijection between the factors before the merging and after the merging. This bijection can be traced by the IDs of the factors of  $g^*$ , which are the same with the IDs of the factors of the input graphs  $\{g_i\}_{i=1}^k$ .*
  
2. *Every factor after in  $g^*$  the merge has the same number and type of edges incident as its corresponding factor with the same ID in  $\{g_i\}_{i=1}^k$  had before the merge.*

*Proof.* We prove both statements by induction on the merge steps. Both statements hold before the merge steps begin.

To prove statement 1, we note that every merge operation only changes IDs of nodes and the number of nodes. Therefore, the number of factors, and their IDs are not affected by the merge operation in Algorithm 4.

To prove statement 2, we note that at every merge operation, the number and edge types of the neighboring nodes for every factor remain the same. If no neighboring nodes of the same factor participate in the merge, then nothing changes in the neighborhood of the factor and statement 2 is true. If more than zero nodes participate in merging, then their number is exactly one  $\gamma$  node because Algorithm 4 specifically prevents multiple such nodes by discarding. After the merge this  $\gamma$  node is replaced by a new  $\gamma$  node, but it remains connected to the factor through an edge with the same label as before. □

Since the factors would have the same number and type of edges before and after the merging, we do not need to learn any new factor models in the cost function in order to handle commands consisting of multiple dialog acts. We also do not extend the data structure representation of the grounding graph — it remains the same, just bigger and more connected. The graph merging, however, could cause loops to appear in the merged grounding graph, while the original one-clause grounding graphs were guaranteed to be trees.

## 4.2 Node ordering algorithm

The previous implementation of the inference assumed that grounding graphs are trees, since each grounding graph came from a single clause, and performed the inference in the `DFSOnLeaving` (Section 2.1.5) ordering, which has the nice property that it does the inference bottom-up. It starts from the leaf factors and consecutively goes up towards the top level `EVENT` factors. However, when Algorithm 4 merges grounding graph nodes due to coreference, the resulting grounding graph may not be a tree any more, which means that the `DFSOnLeaving` ordering would not always produce such bottom-up ordering. Also, it is possible that the merged graph has multiple components and it would not be possible to traverse them with a single depth-first search. Even if we decide to traverse them in order we could have a situation in which the first and the third original graphs are connected, but the second one is not connected to them, and we will need some way to make sure that we infer the event described by the third graph after the event described in the second graph.

To overcome these challenges we describe a new ordering algorithm which aims at achieving the following objectives

1. Return the same ordering as a consecutive `DFSOnLeaving` when there are no coreferences.
2. Traverse the nodes in a “bottom-up” fashion
3. Traverse nodes about `EVENT` `ESDC` clauses mentioned earlier in the command at an earlier stage of the inference

For the ordering algorithm we need a special data structure — stack with history which we call `HStack`. This is a normal stack but it checks whether a given element has *ever* been pushed to the stack instance before. If so, the element is not added to the stack. The stack has three API calls: `stack.top()`, `stack.pop()` and `stack.push(element)`. `stack.top()` returns the top element of the stack without removing it from the stack, and `NULL` if the stack is empty. `stack.pop()` returns the top element of the stack and removes it from the stack. `stack.push(element)` adds

the element to the top of the stack if this element has never been inserted before, in which case it does nothing.

We also introduce the concept of a top-level node to mean any node in a given grounding graph which is only connected to factors via top-labeled edges. Typically those are EVENT ESDCs which describe an action for the robot.

---

**Algorithm 5** Algorithm for ordering the nodes of a merged grounding graph

---

**Require:** A merged grounding graph  $g$   
 $O \leftarrow$  empty list /\* the ordering \*/  
 $HStackS \leftarrow$  new HStack  
 $T \leftarrow$  the set of top level nodes of  $g$ , ordered in reversed order of their position in the text.  
**for** node  $n \in T$  **do**  
     $S.push(n)$   
**end for** /\* at this stage the first top-level nodes are at the top of the stack and the last are at the bottom \*/  
**while**  $S.top()$  **do**  
     $c \leftarrow S.top()$  /\* current node \*/  
    **if**  $c$  is  $\gamma$  node **then**  
        **for all** factors  $f$  in which  $c$  is the top node **do**  
            Add  $f$  to  $O$   
            Push the r, f, l and l2 nodes of  $f$  to  $S$  (only if they are  $\gamma$  nodes)  
        **end for**  
    **end if**  
    **if** we have not pushed new nodes in the stack since  $c$  **then**  
        Add  $c$  to  $O$   
         $S.pop()$  /\* remove  $c$  from the stack \*/  
    **end if**  
**end while**  
**return**  $O$

---

We present the new ordering in Algorithm 5 and the intuition behind it is that the ordering first traverses all factors reachable through their top-labeled edges from the first top level ESDC, then all factors reachable through their top-labeled edges from the second top level ESDC and so on. If there are no coreferences, this ordering would be equivalent of running DFSOnLeaving on each component of the merged grounding graph consecutively. This ordering has the property that it will traverse all the top level EVENT ESDCs by their order of appearance in the command.

We do not claim optimality properties of this ordering. For exact inference al-

gorithms such as elimination, which are parametrized by an ordering, finding the optimal ordering could be NP-hard [31]. The inference procedure used by  $G^3$  is approximate so it has no guarantees of finding the global optimum grounding assignment. We instead provide a description of the behavior of the ordering. The ordering is guaranteed to traverse the top-level EVENT ESDC nodes according their ordering in the text, because of the way we push them into the `HStack`. The ordering traverses the nodes in a bottom-up fashion because the it adds  $\gamma$  nodes corresponding to nested ESDCs to the stack after their parent ESDC. In the case of no coreference the ordering becomes the equivalent of `DFSOnLeaving` ordering, and the stack which is implemented implicitly in the `DFSOnLeaving` ordering by the recursive call becomes explicit as `HStack`.

## 4.3 Implementation

### 4.3.1 Coreference engines

Multiple coreference resolution libraries are available off-the-shelf. They are usually trained and tested on corpora of newspaper articles and achieve around 70% f-score there, but do not perform as well on a spatial manipulation task corpus such as the one we collected. We picked `Reconcile` [54] as the coreference resolution engine that we use in this project to illustrate the current practical abilities. We chose it because we can supply our own noun phrases (leaf `OBJECT` ESDCs), while other coreference engines such as `ARKref` use their own parser to extract noun-phrases, but also because we could trade-off between precision and recall through changing a parameter called `ClusterThreshold`.

In our evaluations we compare four different alternatives for coreference resolution which we call `Pass`, `ReconcileNPS`, `ReconcileNPSAnaphora`, and `GroundTruth`.

### 4.3.2 Pass coreference

This is an empty coreference resolution. It does not find any coreferences. It is equivalent of just  $G^3$  without any coreference resolution in the language model. The implementation consists of partitioning the noun phrases in a way that every noun phrase is in its own partition

### 4.3.3 ReconcileNPS coreference

This is Reconcile engine [54], with leaf OBJECT ESDCs as noun phrases(hence acronym NPS in ReconcileNPS). We picked the value of the `ClusterThreshold` to be maximize precision on the training corpus.

Before Reconcile performs coreference on a sentence it allows us to pre-define the set of noun-phrases in the sentence. It then assigns a score between 0 and 1 to every pair of noun phrases and clusters groups of noun-phrases in which the pairs have high enough score. There is not a unique good way to perform the clustering-a conservative estimate may only look create groups with high scores and would find a small number of coreferences, while a loose grouping may create groups with lower scores and would find more coreference. Reconcile uses a parameter called `ClusterThreshold` to trade-off between those behaviors. The expectations about the output, depending on the `ClusterThreshold`, are that with higher parameter values we will get a higher precision and lower recall.

We ran Reconcile with different values of the `ClusterThreshold` parameter on the annotated corpus and recorded precision and recall (Figure 4-1). In our application we care much more about the precision than the recall, because it is worse to introduce false coreference information than to just miss the chance to improve. For that reason we chose the value of the parameter to be 0.532, since we observed the highest precision of 95% there (Figure 4-1).

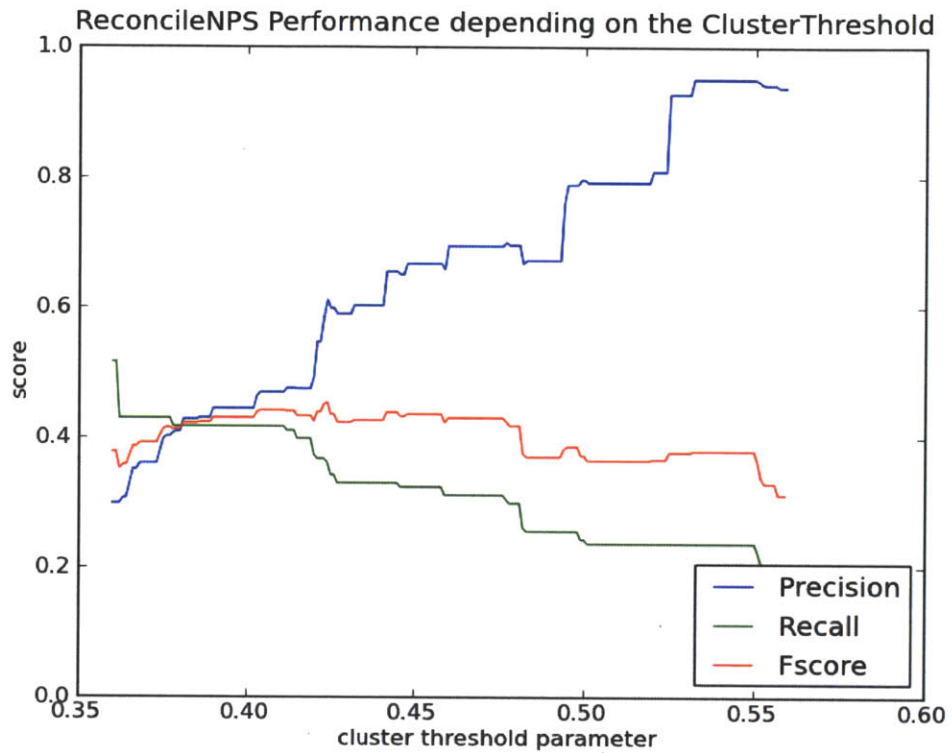


Figure 4-1: The precision of the ReconcileNPS coreference engine increases as we increase the value of the `ClusterThreshold` parameter. The highest precision value is 0.95, achieved at `ClusterThreshold` 0.532. We selected this parameter value solely based on the training corpus data.



#### 4.3.4 ReconcileNPSAnaphora coreference

This is the same as ReconcileNPS with heuristic filters post-processing of its result. The heuristics filter the result to allow only certain types of anaphora. We again picked the `ClusterThreshold` parameter to maximize precision on the training corpus. Performing grounding of an leaf OBJECT ESDC which text is an anaphoric mention (eg. “it”, “them”) is harder than performing grounding of an ESDC which text contains some physical description (eg. “the black bowl”), because in the first case the text itself carries no specific information. All the specific information of the object is in its coreferent phrases in the text. This indirection is a large source of errors in the grounding part of the inference, because if all the inference knows about a  $\gamma$  node comes from the word “it”, then there is no sufficient information to find the right grounding. For that reason we created a modified version of ReconcileNPS which aims at achieving high precision on anaphoric coreferences, which we called ReconcileNPSAnaphora.

We implemented it by applying filters to the output of ReconcileNPS. The filters consist of two sets of strings. The first set is a list of required phrases  $R$ . The second set is a list of forbidden phrases  $F$ . When ReconcileNPS returns a grouping of the noun phrases, for every grouping, if it contains at least one phrase in  $R$ , at least one phrase which is not in  $R$  and does not contain a phrase in  $F$ , then we keep the grouping. Else, we split the grouping into single non-coreferent noun phrases.

For the filters we used  $R = \{\text{it, then}\}$ , and  $F = \{\text{the table}\}$ . With those settings we ran Reconcile with different values of `ClusterThreshold` and selected the value of 0.424 because we found that at that value ReconcileNPSAnaphora has highest precision of 85% (Figure 4-2).

#### 4.3.5 GroundTruth coreference

This method uses the ground truth coreferences as seen in the annotated corpus.

The ground truth knowledge comes from the annotated corpus. If two leaf object ESDCs are grounded to the same object in the corpus then they are coreferent. The

ReconcileNPSAnaphora Performance depending on the ClusterThreshold

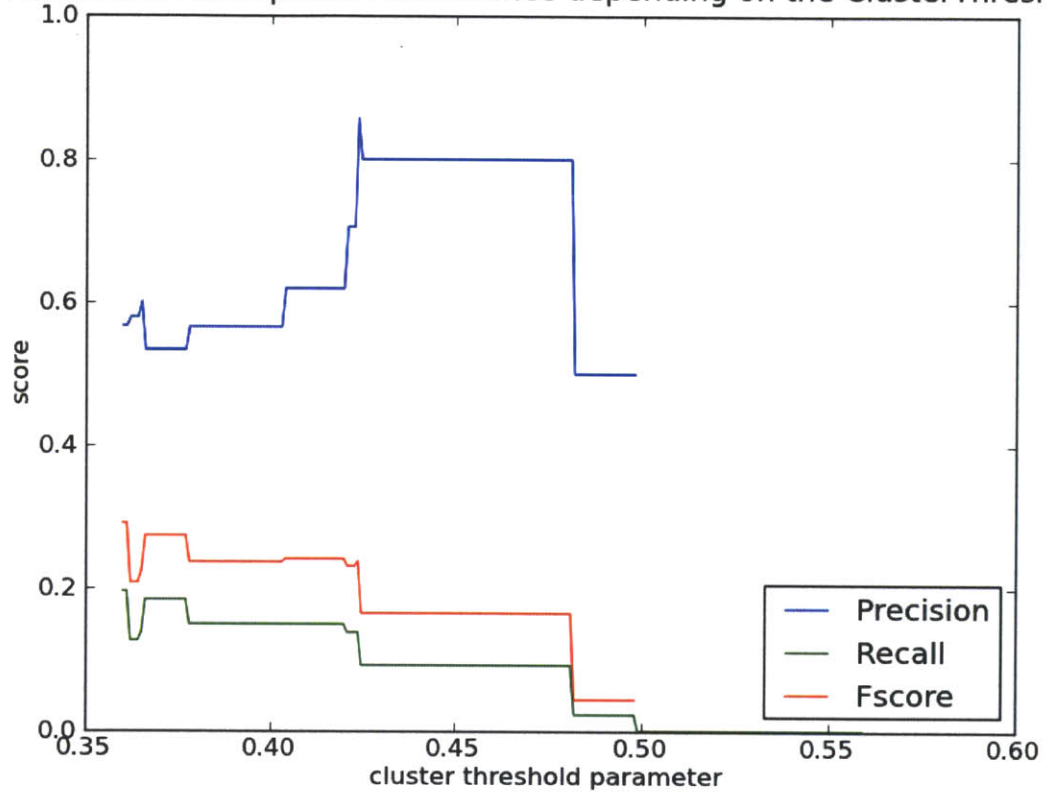


Figure 4-2: By comparing the precision and recall of the coreference on the training corpus for different values of the ClusterThreshold parameter, we selected the one with the highest precision. The highest precision value is 0.85, achieved at ClusterThreshold 0.424.

Table 4.1: Object grounding performance on the training dataset

Type of inference	All nodes		Merged nodes	
	Percent correct	Out of	Percent correct	Out of
Pass	85.3%	404	0.0%	0
ReconcileNPS	84.8%	370	90.0%	30
ReconcileNPSAnaphora	85.4%	392	100.0%	10
GroundTruth	90.0%	311	97.1%	70

Table 4.2: Object grounding performance on the test dataset

Type of inference	All nodes		Merged nodes	
	Percent correct	Out of	Percent correct	Out of
Pass	64.5%	288	0.0%	0
ReconcileNPS	64.3%	269	88.8%	18
ReconcileNPSAnaphora	66.4%	265	50.0%	14
GroundTruth	70.9%	220	68.8%	61

ground truth coreference partitions the noun-phrases according to this criterion. This way it serves to predict show us the best-case effect of coreference resolution.

## 4.4 Comparison of coreference resolvers

To evaluate our hypothesis that the use of coreference resolution information improves the grounding performance we compared the object groundings that  $G^3$  with coreference infers with the annotated true groundings (Tables 4.1 and 4.2). We use the same technique from Section 3.5 and report the percentage and number of times the inferred object grounding has the same tags as the annotated one. We observe higher performance in the case of ground truth coreference, and higher performance on the training dataset than on the testing dataset.

To evaluate the plans as a whole, we use manual evaluation to score the usefulness of coreference resolution as a means for improving plan inference performance. We use the same correctness criterion (*Do the robot actions make sense for this com-*

Table 4.3: Plan correctness performance of different coreference resolvers on the examples of the dataset which include coreference

Type of inference	Training (44 commands)	Testing (41 commands)
Pass	5 (11.3%)	4 (9.7%)
ReconcileNPS	5 (11.3%)	5 (12.1%)
ReconcileNPSAnaphora	7 (15.9%)	7 (17.0%)
GroundTruth	9 (20.4%)	14 (34.1%)

Table 4.4: Plan correctness performance of different coreference resolvers on the whole dataset

Type of inference	Training (103 commands)	Testing (84 commands)
Pass	43 (41.7%)	19 (22.6%)
ReconcileNPS	43 (41.7%)	20 (23.8%)
ReconcileNPSAnaphora	45 (43.6%)	22 (26.1%)
GroundTruth	47 (45.6%)	29 (34.5%)

*mand in this environment?*) as in Section 3.5, and we compare the four coreference resolution engines. We first look at those commands in the corpus in which there is coreference between phrases. Table 4.3 show inference performance on the manually evaluated robotic plans for the examples which include coreference in them. There were a total of 85 commands which contained coreference- 44 in the training corpus and 41 in the testing corpus. We observe higher performance of ground truth coreference than automated coreference, which in turn has higher or equal performance than no coreference on both the training and the testing dataset. Table 4.4 shows inference performance on the whole corpus, including both commands which do not have coreference in them as well as commands which do have coreference in them. Those results look similar in their qualitative characteristics because on the examples with no coreference all coreference engines performed the same adding 40 more correct inferences on the training dataset and 15 on the testing dataset.

The cost of a plan obtained after inference in  $G^3$  can be used as a confidence

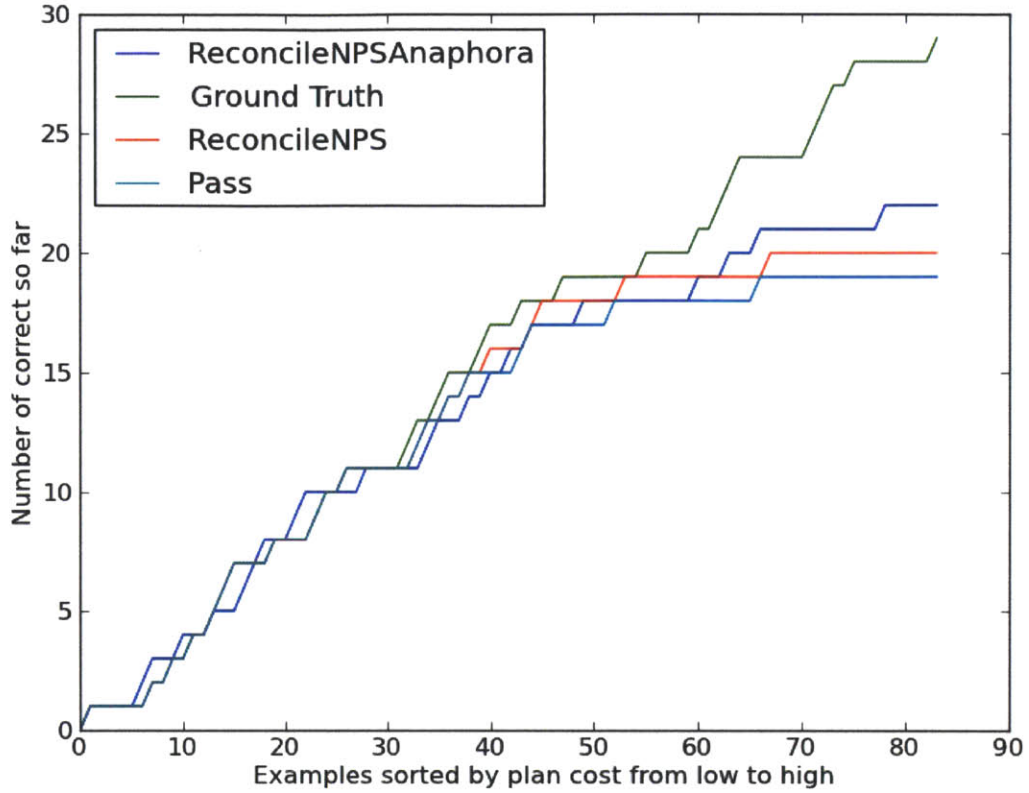


Figure 4-3: With ground truth coreference, the  $G^3$  infers correct actions even for higher-cost longer commands. The other coreference methods infer correct plans mostly for lower cost commands, which is visible in the flattening of their plots.

score. There in Figure 4-3 we show the number of correct testing examples sorted by the total cost of the plan found during inference. Without coreference resolution the performance flattens on the the examples with high cost. However, in many cases higher cost could mean just a longer command with more groundings, rather than just a less-confident command. By using coreference resolution we can handle some longer commands successfully which straightens the relation between inferred cost and correctness.

## 4.5 Summary

While overall inference was poor, we show that using coreference resolution information can improve performance. Typical examples in which coreference helped were situations in which the noun-phrase “it” was correctly resolved to corefer to the previous mention of the object. Without coreference, the phrase “it” would be grounded to the object in the environment with most prominent features in the cost function. This object is possibly not the right one in situations with more than one object in the environment. Overall, we showed that using coreference resolution as a part of the language model can improve the robot’s ability to ground the noun-phrases in the command to objects in the environment and to infer actions for manipulation of these objects.

Inference with ground truth coreference was considerably superior than the same with an automated coreference. We note the future need of a coreference engine with more solid performance on commands. However, the use of coreference, even if it is perfectly correct, would still not solve the outstanding difficulties of inferring all the correct groundings and mapping them to robotic actions. Future approaches will need to address those.

# Chapter 5

## Conclusion

The work presented in this thesis applies the  $G^3$  natural language understanding framework on a new robot, thus validating  $G^3$ 's ability to generalize over robotic platforms. We identified the components of the framework which needed to be adapted—the prediction of the robot action trajectories and the learning of event models. We adapted the framework according to these needs and successfully interfaced it with the new type of robotic environment. In the process we also collected an annotated corpus of language commands paired with robotic actions, which could be re-used to help bootstrap future research. Our evaluation showed lower performance on the PR2 than  $G^3$  previously achieved on a robotic forklift.

We also introduced the concept of coreference resolution from linguistic theory into robot's abilities to follow natural language commands. We showed theoretically and evaluated practically how to incorporate coreference resolution into the language model, and that it leads to better understanding of the command, and to better performance when attempting to follow the command. Our evaluation showed that using the true coreference information in the inference leads to better robotic actions than without coreference. Even though we used state-of-the-art automated coreference resolution engines, the robotic actions inferred using them only showed meager performance increase than without coreference.

We hope that the lessons learned in this process will help future robotic researchers to develop more portable natural language understanding abilities which can easily

be applied to different robots. We also hope that the gap between robot's natural language following abilities in practice and the theoretical understanding of will decrease and robots of the future will be able to understand all commands that humans would understand as well.



# Bibliography

- [1] Apple - iphone 4s - ask siri to help you get things done. <http://www.apple.com/iphone/features/siri.html>.
- [2] Bags - ros wiki. <http://www.ros.org/wiki/Bags>.
- [3] Beer me robot. <http://www.willowgarage.com/blog/2010/07/06/beer-me-robot>.
- [4] Call for proposals: Pr2 beta program. <http://www.willowgarage.com/pages/pr2-beta-program/cfp>.
- [5] Commanding robots using natural language. <http://www.willowgarage.com/blog/2012/01/18/commanding-robots-using-natural-language>.
- [6] gazebo - ros wiki. <http://www.ros.org/wiki/gazebo>.
- [7] Master - ros wiki. <http://www.ros.org/wiki/Master>.
- [8] Messages - ros wiki. <http://www.ros.org/wiki/Messages>.
- [9] Nodes - ros wiki. <http://www.ros.org/wiki/Nodes>.
- [10] Pr2 icra manipulation demo. [http://www.ros.org/wiki/icra\\_manipulation\\_demo](http://www.ros.org/wiki/icra_manipulation_demo).
- [11] Pr2 plugging itself. [http://www.ros.org/wiki/pr2\\_plugs](http://www.ros.org/wiki/pr2_plugs).
- [12] Pr2 robot - overview. <http://www.willowgarage.com/pages/pr2/overview>.
- [13] Pr2 solves the rubik's cube. [http://www.youtube.com/watch?v=S0d14\\_A\\_0YY](http://www.youtube.com/watch?v=S0d14_A_0YY).
- [14] The results are in: Pr2 beta program recipients! <http://www.willowgarage.com/blog/2010/05/04/pr2-beta-program-recipients>.
- [15] Ros/installation - ros wiki. <http://www.ros.org/wiki/ROS/Installation>.
- [16] Services - ros wiki. <http://www.ros.org/wiki/Services>.
- [17] Topics - ros wiki. <http://www.ros.org/wiki/Topics>.

- [18] Towels! (uc berkeley). <http://www.willowgarage.com/blog/2010/04/02/towels-uc-berkeley>.
- [19] Galen Andrew and Jianfeng Gao. Scalable training of L1-regularized log-linear models. In *Proc. Int'l Conf. on Machine Learning (ICML)*, 2007.
- [20] Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18:467–479, 1992.
- [21] Benjamin Cohen, Daniel Benamy, Anthony Cowley, William McMahan, and Joseph Romano. Poop scoop: Perception of offensive products and sensorized control of object pickup. *IROS, The PR2 Workshop*, 2001.
- [22] A. Culotta, M. Wick, R. Hall, and A. Mccallum. First-order probabilistic models for coreference resolution. In *In Proc. HLT-NAACL 2007*, 2007.
- [23] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.
- [24] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *ICRA*, pages 4163–4168, 2009.
- [25] Pedro Felzenszwalb, Ross Girshick, David Mccallester, and Deva Ramanan. Object detection with discriminatively trained part based models. Technical report, 2009.
- [26] Peter Gorniak and Deb Roy. Understanding complex visually referring utterances. In *Proceedings of the HLT-NAACL 2003 workshop on Learning word meaning from non-linguistic data - Volume 6*, HLT-NAACL-LWM '04, pages 14–21, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.
- [27] Aria Haghighi and Dan Klein. Simple coreference resolution with rich syntactic and semantic features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, pages 1152–1161, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
- [28] Stevan Harnad. The symbol grounding problem, 1990.
- [29] Daniel Hewlett. *A Framework for Recognizing and Executing Verb Phrases*. PhD thesis, The University of Arizona, 2011. Ph.D. thesis.
- [30] K. Hsiao, S. Tellex, S. Vosoughi, R. Kubat, and D. Roy. Object schemas for grounding language in a responsive robot. *Connection Science*, 20(4):253–276, 2008.

- [31] Michael I. Jordan. Learning in graphical models. 19(1):140–155, 2004.
- [32] Daniel Jurafsky, Chuck Wooters, Jonathan Segal, Andreas Stolcke, Eric Foslter, Gary Tajchman, , Nelson Morgan, and Nelson Morgan. Using a stochastic context-free grammar as a language model for speech recognition, 1995.
- [33] Leslie Pack Kaelbling and Toms Lozano-prez. Hierarchical task and motion planning in the now. <http://hdl.handle.net/1721.1/54780>, 2011.
- [34] Frank Keller and Ash Asudeh. *Constraints on Linguistic Coreference: Structural vs Pragmatic Factors*, pages 483–488. Lawrence Erlbaum, 2001.
- [35] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *IN PROCEEDINGS OF THE 41ST ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, pages 423–430, 2003.
- [36] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Proc. ACM/IEEE Int’l Conf. on Human-Robot Interaction (HRI)*, pages 259–266, 2010.
- [37] Lars Kunze, Tobias Roehm, and Michael Beetz. Towards semantic robot description languages. In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May, 9–13 2011.
- [38] A. Lascarides and N. Asher. Segmented discourse representation theory: Dynamic semantics with discourse structure. *COMPUTING MEANING*, 3, 2001.
- [39] S. Lemaignan, R. Ros, and R. Alami. Dialogue in situated environments: A symbolic approach to perspective-aware grounding, clarification and reasoning for robot. In *Robotics, Science and Systems, Grounding Human-Robot Dialog for Spatial Tasks workshop*, 2011.
- [40] M. MacMahon, B. Stankiewicz, and B. Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. In *AAAI*, pages 1475–1482, 2006.
- [41] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, 19:313–330, June 1993.
- [42] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *ICRA*, 2010.
- [43] Jennifer Barry Mario Bollini and Daniela Rus. Bakebot: Baking cookies with the pr2. *IROS, The PR2 Workshop*, 2001.
- [44] Cynthia Matuszek, Dieter Fox, and Karl Koscher. Following directions using statistical machine translation. In *HRI*, 2010.

- [45] Cynthia Matuszek, Brian Mayton, Roberto Aimi, Marc Peter Deisenroth, Liefeng Bo, Robert Chu, Mike Kung, Louis LeGrand, Joshua R. Smith, and Dieter Fox. Gambit: A robust chess-playing robotic system. In *Proc. of the IEEE International Conference on Robotics and Automation*, May 2011.
- [46] Andrew Kachites McCallum. MALLET: A machine learning for language toolkit. <http://mallet.cs.umass.edu>, 2002.
- [47] Tillmann Pross. *Grounded Discourse Representation Theory*. PhD thesis, Institut für maschinelle Sprachverarbeitung, 2010. Ph.D. thesis.
- [48] Marta Recasens and Marta Vila. On paraphrase and coreference. *Computational Linguistics*, 36(4):639–647, 2010.
- [49] L. Riano and T.M. McGinnity. Autonomous skills buildings and re-use of software in robotics. *IROS, The PR2 Workshop*, 2001.
- [50] Deb Roy. A trainable visually-grounded spoken language generation system.
- [51] M. Likhachev S. Gray, C. Clingerman and S. Chitta. Pr2: Opening spring-loaded doors. *IROS, The PR2 Workshop*, 2001.
- [52] Ruth Schulz, Arren J. Glover, Michael Milford, Gordon Wyeth, and Janet Wiles. Lingodroids: Studies in spatial cognition and language. In *ICRA*, pages 178–183, 2011.
- [53] Dimitar Simeonov, Stefanie Tellex, Thomas Kollar, and Nicholas Roy. Toward interpreting spatial language discourse with grounding graphs. In *2011 RSS Workshop on Grounding Human-Robot Dialog for Spatial Tasks*, 2011.
- [54] Veselin Stoyanov, Claire Cardie, Nathan Gilbert, Ellen Riloff, David Buttler, and David Hysom. Coreference resolution with reconcile. In *Proceedings of the ACL 2010 Conference Short Papers*, ACLShort '10, pages 156–161, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.
- [55] S. Tellex, T. Kollar, S. Dickerson, M.R. Walter, A. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proc. AAAI*, 2011.
- [56] P.L. Williams and R. Miikkulainen. Grounding language in descriptions of scenes. In *R. Sun and N. Miyake (Eds.), Proceedings of the 28th Annual Conference of the Cognitive Science Society*, 2006.
- [57] T. Winograd. *Procedures as a representation for data in a computer program for understanding natural language*. PhD thesis, MIT, 1970.