

Language-Based Bidirectional Human and Robot Interaction Learning for Mobile Service Robots

Vittorio Perera

CMU-CS-18-108

August 22, 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Manuela Veloso, Chair

Jaime Carbonell

Stephanie Rosenthal

Xiaoping Chen, University of Science and Technology of China

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

Copyright © 2018 Vittorio Perera

This research was sponsored by the Office of Naval Research under grant number N000141712899, Silicon Valley Community Foundation and the Future of Life Institute under grant number 2015-143894, and the National Science Foundation under grant numbers IIS-1012733, IIS-1218932, and IIS-1637927. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: human-robot interaction, natural language processing, spoken language, dialogue

*To my friends and family.
In particular, to Nonna Iole who taught me how to learn.*

Abstract

We believe that it is essential for robots that coexist with humans to be able to interact with their users seamlessly. This thesis advocates the use of language as a rich and natural interface for the interaction between robots and humans. We assume that a mobile service robot, such as the CoBot robot, is equipped with domain information about its environment and is able to perform tasks involving autonomous navigation to desired goal positions. The thesis provides the robot and the human with the ability to interact in natural language, introducing a novel bidirectional approach for the exchange of commands and information between a robot and its users.

In the human-to-robot direction of interaction, we assume that users provide a high-level specification of what the robot should do. This thesis enables a mobile service robot to understand (1) requests to perform tasks, and (2) questions about the robot experience as stored in its log files. Our approach introduces a dialogue-based learning of groundings of natural language expressions to robot actions and operations. These groundings are learned into knowledge bases that the robot can access.

In the robot-to-human interaction direction, this thesis enables a robot to match the detail of the explanations it provides to the user's request. Moreover, we introduce an approach that enables a robot to pro-actively report, in language, on the outcome of a task after executing it. The robot contextualizes information about the task execution by comparing it with its past experience.

In a nutshell, this thesis contributes a novel, language-based, bidirectional interaction approach for mobile service robots, where robots learn to understand and execute commands and queries from users, and take the initiative to offer information, in language, to users about their experience. So, the language exchange can be initiated by the robots, as well as by the humans.

We evaluate the work both on the actual CoBot robots, and on constructed simulated and crowd-sourced data.

Acknowledgments

This thesis would not have been possible without the help of many. First and foremost, I would like to acknowledge my advisor, Manuela Veloso, for her guidance over the five years I have worked toward my PhD; in our weekly meetings, we discussed research and much more, including career and ambitious long-term goals. I would also like to thank the members of this thesis committee: Jaime Carbonell, Stephanie Rosenthal, and Xiaoping Chen.

This work on interaction with mobile service robots would not exist without the CoBot robots. I owe a great debt of gratitude to everyone who made the CoBot robots a reality, including: Mike Licitra, who physically built the robots; Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal, who laid the foundation for the complete navigation, task execution, and symbiotic autonomy of the robots; and, finally, everyone else who worked on and contributed to the robots.

During the years I spent at CMU, the CORAL meetings have been a constant source of inspiration. I would like to recognize the help of all the students who participated in these meetings: Joydeep Biswas, Brian Coltin, Richard Wang, Max Korein, Juan Pablo Mendoza, Philip Cooksey, Devin Schwab, Rui Silva, Kim Baraka, Steven Klee, Anahita Mohseni-Kabir, Ishani Chatterjee, Rongye Shi, Nicholay Topin, Ashwin Khadke, Travers Rhodes, and Arpit Agarwal (plus all the visiting students).

I would never have considered pursuing a PhD, if it not for my initial work with Daniele Nardi, and Thomas Kollar, who were great mentors; I am thankful for the opportunity I had to meet and work with them.

Last but by no means least, a special thanks to my family and my girlfriend Martina for their unwavering support.

Contents

1	Introduction	1
1.1	Thesis Question	2
1.2	Approach	3
1.3	Contributions	4
1.4	Reading Guide to the Thesis	6
2	CoBot as a Service Robot	9
2.1	An Example Scenario	9
2.2	CoBot Tasks	11
2.3	CoBot Navigation	13
2.4	CoBot-Human Interaction	15
2.5	CoBot Logging	17
2.6	Summary	20
3	Dialogue-Based Learning of Groundings from Spoken Commands to Task Execution	21
3.1	KnowDial	22
3.1.1	High-Level Joint-Probabilistic Model	23
3.1.2	Frame-Semantic Parser	24
3.1.3	Knowledge Base	25
3.1.4	Grounding Model	26
3.1.5	Querying the Web	27
3.1.6	Dialogue Manager	27
3.2	Experimental Evaluation	29
3.2.1	Learning Location Groundings	29
3.2.2	Learning Object Groundings	32
3.3	Running Example	34
3.3.1	Accessing and Updating the Knowledge Base from Dialogue	34
3.3.2	Accessing and Updating the Knowledge Base from the Web	36
3.4	Summary	38
4	Understanding and Executing Complex Commands	41
4.1	Complex Commands	42
4.2	Detecting Complex Commands	43

4.2.1	A Template-Based Algorithm	43
4.2.2	Experimental Evaluation	45
4.3	Dialogue	48
4.3.1	A Structure-Based Dialogue	48
4.3.2	A Rephrasing Dialogue	48
4.4	Execution	50
4.4.1	A Reordering Algorithm	50
4.4.2	Experimental Evaluation	51
4.5	Summary	52
5	Learning of Groundings from Users' Questions to Log Primitives Operations	53
5.1	Robot Logs	54
5.2	Log Primitive Operations	55
5.3	Question Understanding	56
5.3.1	Parsing Model	57
5.3.2	Grounding Model	57
5.4	Experimental Evaluation	59
5.5	Checkable Answers	61
5.6	Summary	62
6	Mapping Users Questions to Verbalization Levels of Detail	65
6.1	Route Verbalization	66
6.1.1	Environment Map and Route Plans	66
6.1.2	Verbalization Space Components	67
6.1.3	Variable Verbalization Algorithm	67
6.2	Dialogue with a Robot that Verbalizes Routes	69
6.2.1	Data Collection	70
6.3	Learning Dialogue Mappings	72
6.3.1	Demonstration on the CoBot Robots	74
6.4	Summary	75
7	Proactively Reporting on Task Execution through Comparison with Logged Experience	77
7.1	Task Time Expectations	78
7.2	Comparative Templates	80
7.3	Comparative Templates for Complex Commands	84
7.4	Summary	88
8	Deep Learning for Semantic Parsing	89
8.1	Semantic Representations	90
8.2	Deep Learning Models	92
8.3	Experimental Results	96
8.4	Summary	99

9	Related Work	101
9.1	Human-to-Robot	101
9.2	Robot-to-Human	104
9.3	Semantic Parsing	107
10	Conclusion and Future Work	109
10.1	Contributions	109
10.2	Future Work	110
A	Corpora	113
A.1	Task Corpus	113
A.2	Complex Commands Corpus	117
A.3	LPO's Corpus	121
A.4	Verbalization Corpus	124
A.5	Comparative Template Corpus	129
	Bibliography	137

List of Figures

1.1	The CoBot robots	1
1.2	KnoWDiaL approach	4
1.3	LPOs understanding approach	5
1.4	Verbalization space approach	5
1.5	Comparative template approach	6
2.1	An example scenario	10
2.2	CoBot graphical user interface	10
2.3	The GHC 8th floor vector map used by the CoBot robots	13
2.4	Navigation graph	14
2.5	The semantic map of the CoBot robots	15
2.6	CoBot tasks graphical user interface	16
2.7	CoBot website	16
2.8	Data structure for the <code>CobotSpeechRecognitionMsg</code> message	17
2.9	The ROS nodes running on the CoBot robots	18
3.1	KnoWDiaL interaction	22
3.2	Schematic overview of KnoWDiaL	23
3.3	Corpus and its annotations	24
3.4	Semantic frames	26
3.5	A dialogue example	28
3.6	Map used for KnoWDiaL experiments	30
3.8	Entropy for location references	32
3.9	KnoWDiaL experimental results	33
3.10	“Go to the small-size lab” example	35
3.11	Knowledge base updates	36
3.12	“Bring coffee to the lab” example	37
3.13	Locations used to query OpenEval	38
4.1	Template examples	44
4.2	Parse tree for the sentence “If the door is open go to the lab and to my office”	45
4.3	Corpus complexity levels	46
4.4	Complex commands examples	47
4.5	Correctly decomposed complex commands.	47
4.6	Correctly decomposed complex commands after dialogue	49

4.7	Reordering algorithm evaluation	52
5.1	LPOs examples	56
5.2	Parsed sentence example	57
5.3	Knowledge base example	58
5.4	Survey website used to crowd-source the LPO corpus	59
5.5	Grounding operations errors	60
5.6	Number of facts in the knowledge base	61
5.7	Path taken by the CoBot robot	62
6.1	CoBot planning example	68
6.2	Verbalization dialogue survey	71
6.3	Experimental results	73
6.4	Comaprison of unigram and bigram in the verbalization corpus	74
6.5	Demonstration of movements in the verbalization space	75
7.1	Half-Normal distribution	80
7.2	Computing task expectations	81
7.3	Comparing current time	82
7.4	Comparative corpus	82
7.5	Comparative templates	82
7.6	Comparison regions	83
7.7	Changing meaning of comparative templates	86
7.8	Comparative templates for complex commands	86
7.9	A complex command	87
7.10	Comparing the cost of execution traces	87
8.1	SLU representation	90
8.2	AMRL example sentence	91
8.3	AMRL representations for two complex sentences	91
8.4	Linearized annotation for AMRL	92
8.5	Topology of the baseline multi-task DNN model	93
8.6	LSTM unit	94
8.7	Multitask models	95
8.8	Action distribution in the corpus	98

List of Tables

2.1	CoBot tasks and their arguments	12
2.2	CoBot topics, their content and the level they are assigned to	19
6.1	Sample sentences from the corpus	70
6.2	Phrasing of survey instructions	72
8.1	Models and their results	98
8.2	Difference at ICER	99
8.3	Results compared to various baselines	100

Chapter 1

Introduction

We believe that it is essential for robots that coexist with humans to be able to interact with their users seamlessly. In this thesis, we advocate the use of language as a rich and natural interface for human-robot interaction. Natural language allows users that are not expert robot developers to interact with robots and understand them in a special way and simultaneously, offering a high degree of expressive power.

We focus on spoken interaction between users and a mobile service robot. Figure 1.1 shows the CoBot robots, the mobile service robots used to develop, implement, and evaluate the approach described in this thesis. We define a *mobile service robot* as a robot that autonomously and accurately navigates its given environment, and executes tasks requested by its users.



Figure 1.1: CoBot robots, the mobile service robots used throughout this thesis.

A mobile service robot can execute tasks that involve traveling between locations within its environment. Therefore, a large part of this thesis deals with language that refers to locations in the robot’s environment and the tasks it can execute. We do not tackle other common robotic tasks that often involve natural language, such as providing manipulation instructions or following navigation directions. More specifically, in this thesis, we examine how the robot understands sentences like “Can you take this package to the small size lab?” or “How many times did you go to the 7th floor conference room?” However, we do not tackle sentences like “Stack the red

box on top of the yellow one,” or “Go down the hallway past the third room and then take a right turn.” We focus on task-related language, rather than step-by-step instructions, to provide a direct novel interface to the services offered by the robots.

Although the approach this thesis describes was developed for the CoBot robots, a wheeled platform (Figure 1.1), we are neither concerned with whether the robot drives to a location or uses biped locomotion, nor with whether the robot is deployed in indoor or outdoor environments. The focus of this thesis is on mobile robots that offer their users services requiring to travel between locations within the environment. The same approach that we develop in this thesis, for the CoBot robots, is applicable to various mobile agents, a prime example of which are self-driving cars. In Chapter 3, we show how to enable a robot to understand commands, such as, “Go to Vittorio’s office.” Similarly, we could enable a car to understand commands such as “Go to Vittorio’s place.” The only difference is that, for the CoBot robots “Vittorio’s office” must map to Office 7004, while for a self-driving car “Vittorio’s place” must map to the address 5825 Fifth Avenue. To wit, in both cases, the agent must learn how to map a natural language expression to a location on the navigation map, of the Gates-Hillman Center and the city of Pittsburgh respectively.

The thesis introduces a mode of bidirectional interaction between a robot and its users. The first direction is from user to robot; in this direction, a user asks the robot to perform tasks or makes inquiries about the robot’s autonomous execution. The second direction is from robot to user; in this direction, the robot describes its navigation experience, pro-actively reports on the executed task and provides explanations about the choices made. We observe that the internal state of an autonomous robot is often concealed from users. Our decision to enable a second direction of interaction, from robot to human, aims to make the robot more transparent to its users.

The project of enabling a bidirectional interaction between mobile service robots and their users consists, at its core, of bridging two representations. The user’s representation, expressed using natural language, and the complex numerical representation used by the robot, programmatically developed for a mobile service robot using maps, sensor reading, scheduling plans and more. This thesis explores and contributes, in the human-to-robot interaction, how to map and ground natural language to robot internals, while in the robot-to-human direction, how to render as to *verbalize* these robot internals into natural language that matches the users questions.

1.1 Thesis Question

This thesis tries to address the following question:

How can we enable an effective bidirectional human and robot language-based interaction where: humans ask robots to perform tasks and about the robot’s autonomous experience, and robots understand user task requests and report to humans on tasks executed?

We argue that, for robots to coexist with humans, a **bidirectional** interaction is desirable. In the human-to-robot direction, this interaction provides a simple yet powerful interface to the robot, in the form of requests expressed using natural language. In the robot-to-human direction,

the interaction provides the ability to offer more information and increase the robot transparency.

This thesis focuses on interactions between users and a specific type of robot, mobile service robots. Given the nature of these robots, interactions consistently revolve around the **tasks** the robot can offer.

Finally, the bidirectional interaction between user and mobile service robot is realized by the user **asking** the robot to perform tasks, and the robot understanding these requests and **reporting** to the user. In particular, the robot can report when the user **asks** about its previous experience, but it can also pro-actively **report** once it finishes executing a task.

1.2 Approach

Enabling a bidirectional interaction between mobile service robots and their users involves, at its core, bridging user and robot representations. In the human-to-robot direction, we must create a bridge between natural language and the internals of a robot. We assume that the internals of the robot are represented by symbols the robot can act upon directly (e.g., actions to perform and location on a map). Our approach relies on the use of a Learnable Knowledge Base. This Knowledge Base stores mappings between natural language expressions and symbols that the robot uses (i.e., its internals) to represent them. These mappings are stored in the form of binary predicates. The values of a predicate are, respectively, a natural language expression (e.g., “conference room”) and a robot symbol (e.g., GHCF7101). The name of the predicate defines the type of symbol is being stored (e.g., a robot action or a location).

We do not provide, a priori, the Knowledge Base facts. Instead, our approach relies on a dialogue that enables the robot to autonomously learn new facts to store in the Knowledge Base, by interacting with its users. The dialogue is a key component of our approach, as it allows the robot to learn facts that map natural language into symbols it can understand, and drives the interaction between the robot and users toward actions that the robot can perform.

Enabling a bidirectional interaction in the robot-to-human direction requires a bridge from the internals of the robot to natural language. To do this, we must translate symbols, such as locations on a map or actions the robot executes into natural language. Our approach relies on contextualizing the information provided to the user. As an example, we consider the robot position, stored by the robot as coordinates (x, y) on a map. The robot could simply report these coordinates to the user. To figure out which point corresponds to the coordinates provided, the user needs: 1) to have a map of the environment, corresponding to the one the robot uses; 2) to know where the origin is on the map; and 3) to know the measuring units used by the robot. Instead, our approach translates the coordinates used by the robot into an expression, such as “hallway 7100.” Similarly, we should consider the robot’s report on the time it took to execute a task. Rather than citing the task duration (e.g., 42 seconds) our approach uses expressions like, “it took me 42 seconds, as much as usual.” This expression contextualizes the robot symbol (e.g., the length of the task) by comparing it with the usual duration of the same task.

Moreover, we assume that different users might ask for different levels of detail when requesting explanations from the robot. Therefore, the language generated by the robot needs to meet the user’s level of request. To do so we rely on the Verbalization Space [65], which describes various dimensions of the language the robot can use. To generate language meeting the

user requests, we enable our robot to learn a model mapping user requests to diverse points in the Verbalization Space.

1.3 Contributions

The key contributions of this thesis, organized by the direction of interaction, are the following:

Human-to-Robot Interaction

KnoWDiaL is an approach that lets the robot learn task-relevant environmental knowledge from human-robot dialogue and access to the web. KnoWDiaL introduces our learned Knowledge Base and a dialogue approach that enables the CoBot robot to store mappings from natural language expressions to its internals (i.e., tasks and their arguments). Using KnoWDiaL, the CoBot robots are able to understand commands to perform a given task. Figure 1.2 summarizes the approach introduced by KnowDiaL.

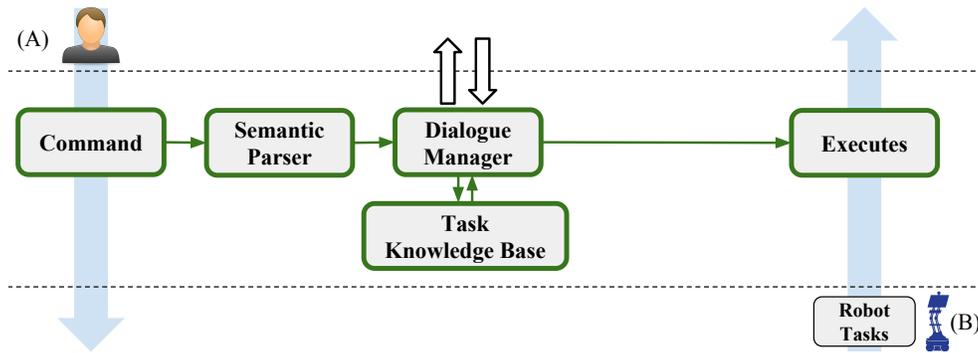


Figure 1.2: Interaction from (A) the user to (B) the robot. The KnoWDiaL approach enables the robot to understand simple commands to perform a task, and to learn language groundings into the task Knowledge Base.

A template-based algorithm for understanding and executing complex sentences that involve multiple commands. Our approach to understanding complex language requests, that involves conjunctions, disjunctions and conditionals, consists of breaking a request into multiple simple sentences (possibly recursively) and processing each of them separately. This approach allows the robot to search for a plan that satisfies the user request and improves the time needed to execute it.

Log Primitive Operations (LPOs) that extend the capabilities of the robot beyond simply executing tasks, to answering questions about the robot’s past experience as operations in the logged experience. LPO’s mark a change of paradigm in the use of logs, which are no longer a mere debugging tool, but can also be processed by the robot for data to help autonomously answer users’ questions. To ground user questions to LPO’s, our approach uses techniques similar to the ones developed to understand simple commands. Figure 1.3 shows our approach.

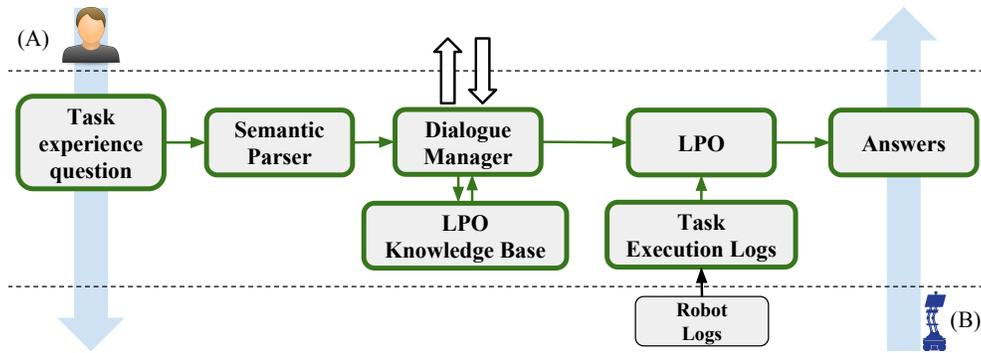


Figure 1.3: Interaction from (A) the user to (B) the robot. Log Primitive Operations enable robots to answer questions about their past task experience. Through the Dialogue Manager, the robot learns groundings from the user questions into the LPO Knowledge Base.

Robot-to-Human Interaction

A crowd-sourced on-line study that allows the robot to correctly map the requests from users to describe the path taken to a point in the Verbalization Space. The Verbalization Space, first introduced in [65], describes variations in the language used by a robot to describe the route it followed. In our study, we show how the verbalization offered by the CoBot robot corresponds with the expectations of the users and that, through dialogue, the robot can vary its verbalization and satisfy subsequent requests to change the description provided. Figure 1.4 shows our approach to map requests from the user to a point in the Verbalization Space.

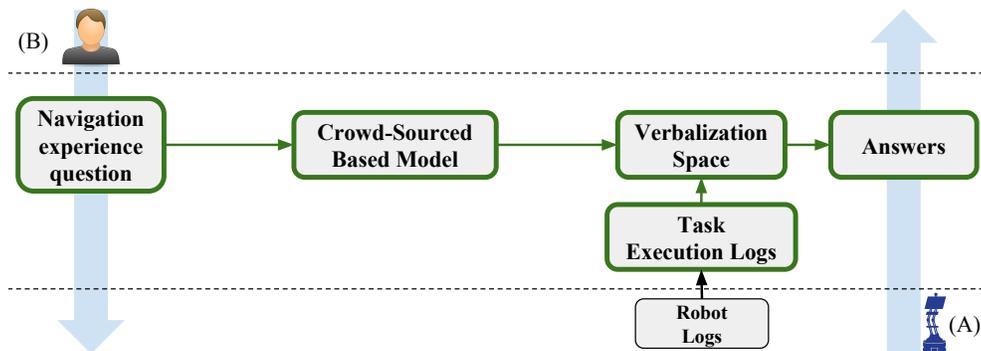


Figure 1.4: Interaction from (A) the robot to (B) the user. The model learned through an on-line study allows the robot to map requests from users to points in the Verbalization Space.

Comparative Templates that allow a robot to pro-actively report on task execution. Our goal is to enable robot to provide a short, yet informative, summary of the task execution to the user, in terms of its relationship with the robot’s past experience. To accomplish this we enable our CoBot robot to report on the time taken to execute a task. We contextualize the information provided to the user by comparing the time taken to execute the current task with the time expectation, derived from the robot logs. Figure 1.5 shows this last contribution.

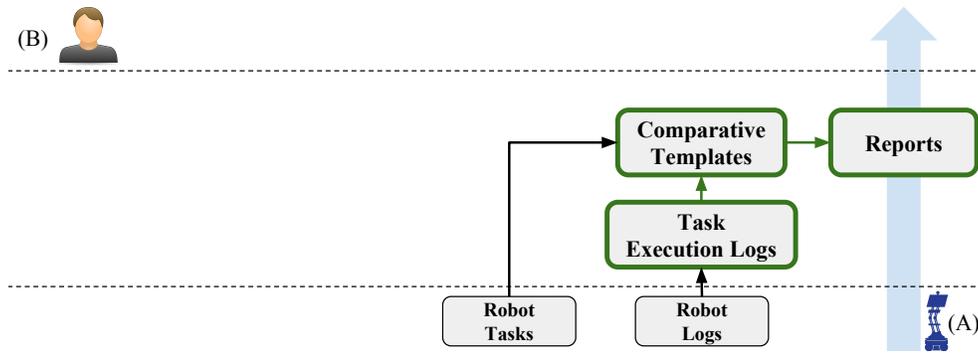


Figure 1.5: Interaction from (A) the robot to (B) the user. Using Comparative Templates a robot pro-actively reports on a task it executes, by comparing its performance with its logged past experience.

1.4 Reading Guide to the Thesis

The following outline summarizes each chapter of this thesis.

Chapter 2 – CoBot as a Service Robot introduces the CoBot robots. This thesis introduces an approach to bidirectional, language-based interaction that is general, but was originally developed for and implemented in the CoBot robots. In this chapter, we present an overview of the CoBot robots. This thesis builds upon some of the CoBot capabilities (e.g., localization, navigation and logging), so Chapter 2 focuses on the specific components of the robot that enable these capabilities.

Chapter 3 – Dialogue-Based Learning of Groundings from Spoken Commands to Task Execution introduces KnowDiaL, which enables the robot to learn task-relevant knowledge from dialogue with the user. Using a Learnable Knowledge Base, the robot is able to store and reuse mappings from natural language expressions to symbols it can act on (e.g., a location on the map, a task to execute).

Chapter 4 – Understanding and Executing Complex Commands introduces a template-based algorithm for understanding and executing complex sentences that involve multiple commands. Requests from users can involve multiple tasks for the robot to execute. Our approach breaks a complex sentence into multiple simple sentences that the robot processes individually. This allows the robot to understand the user’s request and, consequently, to search for a plan to execute the request optimally, from the robot’s point of view.

Chapter 5 – Learning of Groundings from Users Questions to Log Primitives Operations introduces a novel use for the logs of a mobile service robot. Typically, the log of a robot is used for debugging purposes only. In this chapter, we enable a robot to autonomously answer questions about its past experience. To achieve this, we introduce Log Primitive Operations which enable a robot to search its logs to answer such questions.

Chapter 6 – Mapping Users’ Questions to Verbalization Levels of Detail introduces our approach to enabling the CoBot robots to provide descriptions of the route they follow. We rely on the Verbalization Space [65] that characterizes variations in the language a robot

can use to describe the route it follows. Through a crowd-sourced on-line study, we learn a model that enables the robot to provide verbalizations that match the users' requests.

Chapter 7 – Pro-actively Report on Task Execution Through Comparison with Logged Experience introduces Comparative Templates. Comparative Templates allow a robot to pro-actively report on its task execution. To enable the robot to do this, in a short yet informative way, we focus on the robot reporting the time a task has taken. Our approach contextualizes the information the robot provides to the user, by comparing it to the time the same task typically takes, based on the logged experience.

Chapter 8 – Deep Learning for Semantic Parsing presents more detailed work on semantic parsing. Chapter 3 and Chapter 5 both use semantic parsing to enable the CoBot robots to understand the user' requests. In this chapter, we show how to use a deep learning approach to enable semantic parsing.

Chapter 9 – Related Work reviews the literature related to the approach presented in this thesis. In particular we focus on human-to-robot and robot-to-human interaction. We also review relevant work on semantic mapping, which is at the core of our approach to understanding user requests.

Chapter 10 – Conclusion and Future Work concludes this thesis with a summary of its contributions, and presents expected directions for future related research.

Chapters 3 through 8 present the contributions of the thesis. Each of these chapters is prefaced by an example of the language discussed in the chapter.

Chapter 2

CoBot as a Service Robot

This thesis introduces a novel approach to enabling bidirectional communication between users and robots. Users can request robots to perform tasks and ask about the robots' past experiences. Robots can report on the results of tasks executed. We believe that the approach we developed is general, but the algorithms we introduce have been designed, implemented and tested on the CoBot robots. For this reason, in this chapter, we present more information on the CoBot robots. In Section 2.1, we present an example scenario that allows us to introduce the main components of the CoBot robots: the tasks they can execute (Section 2.2), their navigation and localization (Section 2.3), the interactions they can have with users (Section 2.4) and, finally, their software architecture and logging capabilities (Section 2.5).

2.1 An Example Scenario

We consider the scenario shown in Figure 2.1: a user says to the CoBot robot “*CoBot, can you go to Manuela’s office?*” In this section we analyze each step of this interaction. When deployed, the CoBot robots display a graphical user interface (GUI) on their on-board laptop. Figure 2.2 shows the CoBot robots’ GUI. The user-robot interaction starts when the user pushes the “Speak” button on the GUI. When the button is pressed, the user can start talking to the robot. At the same time, the robot starts recording the audio input and, when the user finishes speaking, the robot uses a cloud-based automated speech recognition (ASR) service to transcribe the input sentence. The transcription returned by the ASR is not always accurate; in this example, the robot receives the following string “call but can you go to manuela’s office” where CoBot has been incorrectly transcribed as call but. Nonetheless, the robot correctly matches the string to an action it can perform (i.e., GoTo) and a location on its map (i.e., F8002).

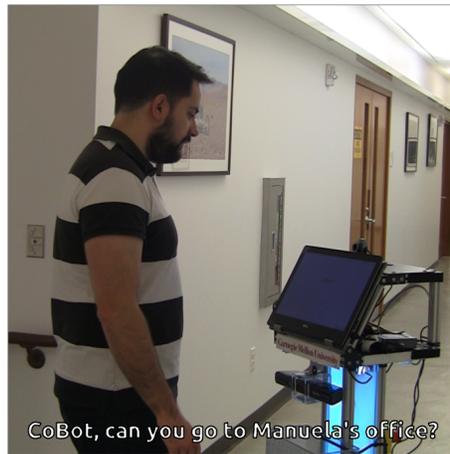


Figure 2.1: The example scenario considered in this section. A user asks the robot, “CoBot, can you go to Manuela’s office?”



Figure 2.2: The graphical user interface (GUI) displayed by the CoBot robots’ on-board laptop.

To execute this action, the task $\text{GoTo}(F8002)$ must be scheduled on the robot with which the user is interacting. Multiple CoBot robots service The Gates-Hillman Center, task requests are handled by a central scheduler. In this example scenario, the robot interacting with the user sends a special request to the central scheduler. This special request ensures that the scheduler assigns the task to the robot that sent it. Once the task has been scheduled on the current robot, the execution begins. To carry out the task $\text{GoTo}(F8002)$, the robot drives to a specific point in the building.

To drive to its destination, the robot first retrieves the (x, y, θ) coordinates of location F8002. Next, the robot needs to plan its route through the building. The robot computes its path to the destination using a Navigation Graph [7]. The robot then follows the path computed, updating

its position on the map using Episodic non-Markov localization [9]. As the robot drives to its destination, it may find obstacles on its path (e.g., people talking in the hallways). When the robot finds an obstacle on its path, and cannot go around it for lack of space in the corridor to go, the robot stops and, to reach its destination, requests passage, saying, “Please excuse me.” Upon arrival at its destination, the robot announces that it has completed its task and waits for a user to confirm the execution.

While the robot interacts with its users, drives to its destination and, more generally, runs, it records a log of its execution. The log records several kinds of information, including the position of the robot on the map, the task being executed, and the command received.

Going through this simple scenario, we mentioned multiple core elements of the CoBot robots:

1. The ability to understand spoken commands;
2. The capacity to schedule and execute tasks;
3. The algorithm used to localize and navigate;
4. The interaction the robots have while executing tasks; and
5. The logging process being executed while the robot runs.

CoBot’s ability to understand spoken commands is the focus of Chapter 3, so, in the remainder of this chapter, we provide more details on the remaining elements mentioned in our example scenario.

2.2 CoBot Tasks

In our example scenario, the sentence spoken by the user is matched with the task `GoTo(F8002)`. The CoBot robots offer multiple tasks to their users, but before describing each of them, we must define a task in this context. A task refers to a function that the robot can execute. We use the term “function” because, like a programming function, each task takes a fixed number of arguments as its input. Like programming functions, we represent a task as `Task(*args)`. The arguments are specific to each task, and each argument has a specific type. As an example, consider the task `GoTo(F8002)`. This task takes a single argument of *location* type. More generally, the CoBot robots’ tasks allow for two types of arguments: *location* and *string*.

The CoBot robots offer their users four tasks:

- **GoTo** requires a single argument: a *location*, the robot’s destination. To execute this task, the robot drives from its current position to the specified destination. The scenario described earlier, is an example of this task in the form of `GoTo(F8002)`.
- **PickUpAndDelivery** requires three arguments: the object to be found in terms of a *string*, the source as a *location* where the object can be found, and the destination as another *location* where the object must be delivered. To execute a `PickUpAndDelivery` task, (`Pu&Delivery`, for short), the robot drives to the source location, asks for the object to be put in its basket, and then drives to its destination to deliver the object. An example of this task is `Pu&Delivery(“AI book”, F8002, F3201)` indicating that the robot should go to F8002, ask for an “AI book”, and deliver it to F3201.

- **Escort** requires two arguments: a person’s name and a destination. The person name is a *string*, but the destination, once again, is a *location*. To execute this task, the robot drives to the elevator on its destination floor and waits for the person to be escorted. The person’s name is displayed on the screen, asking the user to press a button when ready to go. When the button is pressed, the robot asks the person to follow it and drives to its destination. An example of this task is `Escort(“Stephanie”, F8002)` indicating that the robot should wait for Stephanie and escort her to location F8002.
- **MessageDelivery** requires three arguments: the message to be delivered, the person sending the message (i.e., the sender) and the destination where the message needs to be delivered. The message and the sender are *strings*, but the destination is a *location*. To execute a `MessageDelivery` task, the robot first drives to its destination where, upon arrival, it announces that it has a message from the sender. Last, the robot reads the message aloud. An example of this task would be `MessageDelivery(“I am writing my thesis”, “Vittorio”, F8002)` which means that the robot should deliver a message from Vittorio to location F8002 saying, “I am writing my thesis.”

It is worth noting that, in terms of execution, the `Escort` and `MessageDelivery` tasks can be modeled as `Pu&Delivery` tasks. For `Escort` tasks, the person acts as the object and the source can be computed as a function of the destination (the elevator closest to it). For `MessageDelivery` tasks, the message acts as the object and the need for a source is shortcut. Table 2.1 recaps the tasks that the CoBot robots can execute, their arguments and, for each argument, its type. Finally, although the tasks we described are specific to the CoBot robots, our approach is more general, and only requires that the tasks the agents execute be expressed in terms of specific arguments.

Task	Arguments	(Type)
<code>GoTo</code>	destination	(location)
<code>Pu&Delivery</code>	object	(string)
	source	(location)
	destination	(location)
<code>Escort</code>	person	(string)
	destination	(location)
<code>MessageDelivery</code>	message	(string)
	person	(string)
	destination	(location)

Table 2.1: CoBot tasks and their arguments.

In the initial example, we have also seen how, to be executed, tasks must be scheduled on a robot. Besides asking the robots to execute tasks using spoken language, users can also request tasks via a web interface. A centralized scheduler [22] collects these requests and assigns them to one of the currently available robots. The scheduler assigns tasks to specific robots while maximizing the total number of tasks executed. When a CoBot robot directly receives a spoken command, a special message is sent to the scheduler, specifying the task, as well as the name of the robot that received the command. When the central scheduler receives such a message, it

schedules the task on the same robot that sends it. Doing this ensures that the robot that receives a command is the same one assigned to execute it. This choice is motivated by our assumption that, from the user’s point of view, it would feel less natural to ask one robot to execute a task and having another robot execute it.

2.3 CoBot Navigation

In the example scenario described in Section 2.1, we have shown how, once a task has been scheduled, the robot can start executing it. Each of the tasks that the CoBot robots can execute involves driving to one or more locations. To be able to reach its destination, the robots localize in the environment using Episodic non-Markov localization [9]. Detailing the specifics of this algorithm is beyond the scope of this document. On the other hand, we have already mentioned multiple elements (i.e., the Navigation Graph, the *location* used as type for a task argument) that the CoBot robot uses when navigating. Here, we provide an overview of how the CoBot robots localize in and navigate the environment.

The CoBot robot stores the information needed to localize, navigate, and plan its tasks in, respectively, the Vector Map, the Navigation Graph, and the Semantic Map.

The **Vector Map** is a representation of the layout of the building stored in vectorial form; that is, each constituent segment of the map is stored, using a pair of 2D points described by their (x, y) coordinates (i.e., a vector). Figure 2.3a shows the Vector Map, as stored by the robot in a text file. Figure 2.3b shows, instead, a plotting of the vectors stored in the Vector Map file. The robot uses this Vector Map to localize correctly. To simplify, we can say that the robot uses the reading from its sensors (lidar and Kinect) to find planar surfaces, matches these planar surfaces to walls described in the Vector Map and continuously updates its position [9].

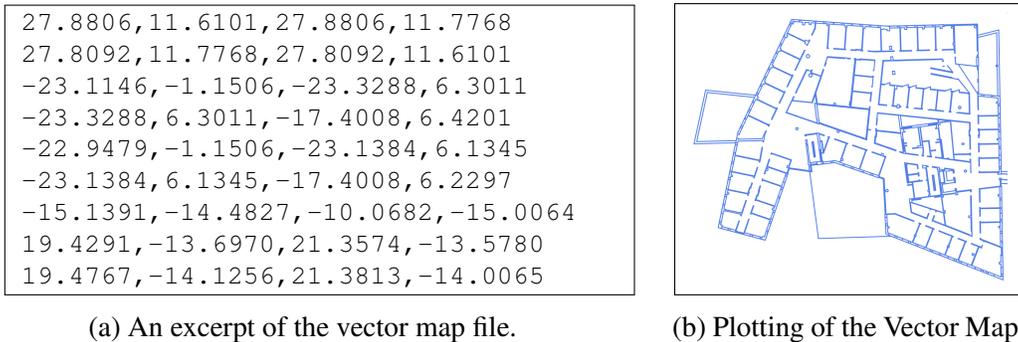


Figure 2.3: The GHC 8th floor Vector Map used by the CoBot robots.

The **Navigation Graph** stores the information the robot needs to move around the environment described by the Vector Map. The vertexes of the Navigation Graph are points on the map, identified by their (x, y) coordinates, and edges are straight lines connecting them. The navigation graph is stored by the robot as a binary file. Figure 2.4 shows the plotting of the Navigation

Graph, with green vertexes and pink edges, overlaid on the Vector Map. As mentioned above, the robots use the Navigation Graph when they must physically move to a given destination. To reach a location (x, y) , the robot first identifies and drives to the closest point on the Navigation Graph, and then, the robot computes a path, via the graph, to the point closest to its destination. Finally, the robot drives, in a straight line, from the point on the Navigation Graph to its desired position.

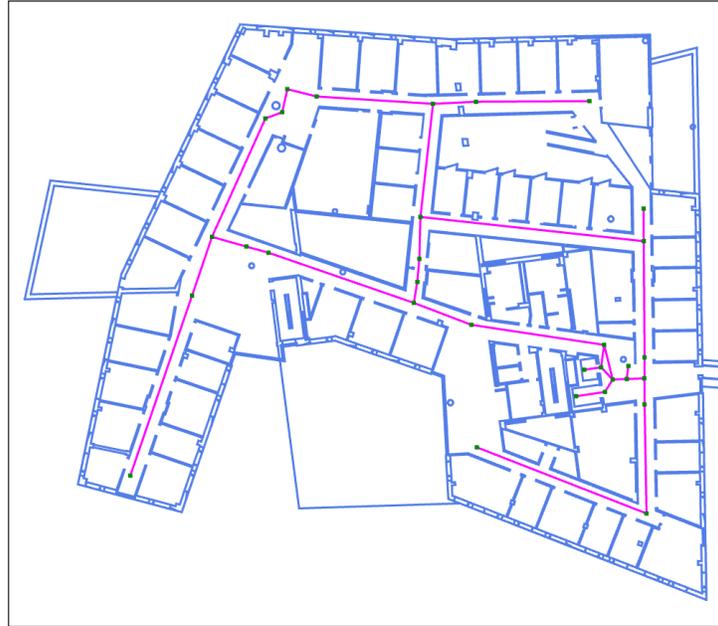


Figure 2.4: Navigation Graph.

The **Semantic Map** provides the information that the CoBot robots use to plan and execute their tasks. The Semantic Map is composed of two parts: a list of locations and a graph spanning them. Figure 2.5a shows the list of locations as stored by the robots. Each entry in the list of location records the location type (i.e., one out of the following six types: Office, Printer, Stairs, Bathroom, Elevator or Kitchen), a location index, corresponding to the room number, (e.g., F8001, F8002, F8010), and the coordinates (x, y, θ) , where θ defines the orientation the robot should face when it stops at the room location. Figure 2.5b shows the graph of the semantic map, as stored by the robots. Each edge in the graph is recorded as the index of the vertexes that it is connecting together with an edge type (either Hallway or Elevator). Finally, Figure 2.5c shows the Semantic Map location list and its graph, overlaid on the Vector Map.

In summary, the CoBot robots use a Vector Map to localize, a Navigation Graph to navigate around the building, and a Semantic Map to plan their tasks. This thesis focuses on physical agents, offering services that require travel between locations. Therefore we are not interested in the Vector Map, The Navigation Graph or the Semantic map per se, but rather in the abstract functionality they offer (i.e., localization, navigation and task execution).

```

Office,F8001,19.9039,-45.6455,-1.84065
Office,F8002,19.0738,-44.472,1.23874
Other,O818,17.6506,-10.3775,0
Stair,S81,7.51533,-4.00557,0
Bathroom,B8M,6.70925,-6.50267,0
Elevator,E81,11.1528,2.45,1.67706
Kitchen,K81,11.9178,-10.8102,0

```

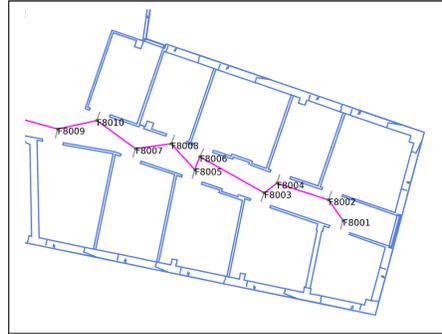
```

Hallway-F8001-F8002
Hallway-F8002-F8004
Hallway-F8004-F8003
Hallway-F8003-F8006
Hallway-F8006-F8005
Hallway-F8005-F8008
Hallway-F8008-F8007

```

(a) The vertexes of the semantic map.

(b) The edges of the semantic map.



(c) Plotting of the Semantic Map.

Figure 2.5: The semantic map of the CoBot robots.

2.4 CoBot-Human Interaction

In the scenario described in Section 2.1, we can observe two types of interaction between a CoBot robot and its users: first, there is a spoken interaction, where the user asks the robot to perform a task; second, there is a request for help, where the robot asks bystanders to move out when its path is blocked. Spoken interaction is the focus of this work and will be described, in more detail, in the remaining chapters. Requesting help, together with interaction via a website or on-board GUI, is another possible interaction between the CoBot robots and their users, detailed in this section.

The CoBot robots request for help is part of a wider approach, Symbiotic Autonomy [64], in which the robot performs tasks for humans and requests help to overcome its limitations and complete tasks successfully. An example of such a relationship is presented in Section 2.1 where, when confronting the blocked path, the robot asks the user to move out of the way by saying, “Please excuse me.” A second example of this relationship is displayed when the robot needs to travel between floors. As the CoBot robots cannot interact with the elevator directly, they drive up to the front of the elevator and ask bystanders, aloud, to press the elevator button for them. Sometimes, there are no bystanders when the robot arrives at the elevator, or the robot’s path is blocked, not by people, but by actual obstacles. When such instances occur, the robot first asks for help by saying it out loud and, if five minutes pass without help, the robot sends a request for help to a mailing list [8]. Symbiotic autonomy is a key component of the CoBot robots and we embrace it, in our approach, to enable bidirectional interaction between human and robot. In particular, we will see in Chapter 3 and Chapter 5, that when the robot is not able to fully understand the user’s request, it enters into a dialogue with the user itself. The goal of

the dialogue is to overcome the limited understanding of the input sentence and to recover its full meaning.

In the example scenario from Section 2.1, we have shown how the user starts the interaction by pressing the “Speak” button on the GUI, shown in Figure 2.2. A second button, labeled “Schedule Task,” can be used to request a task using a GUI rather than spoken language [83]. When this button is pressed, the screen shown in Figure 2.6 is presented to users. By filling in each field of the form, the user can schedule a task for the robot to execute. It is worth noting that the fields in the form correspond to the arguments of the tasks that we described in Section 2.2.

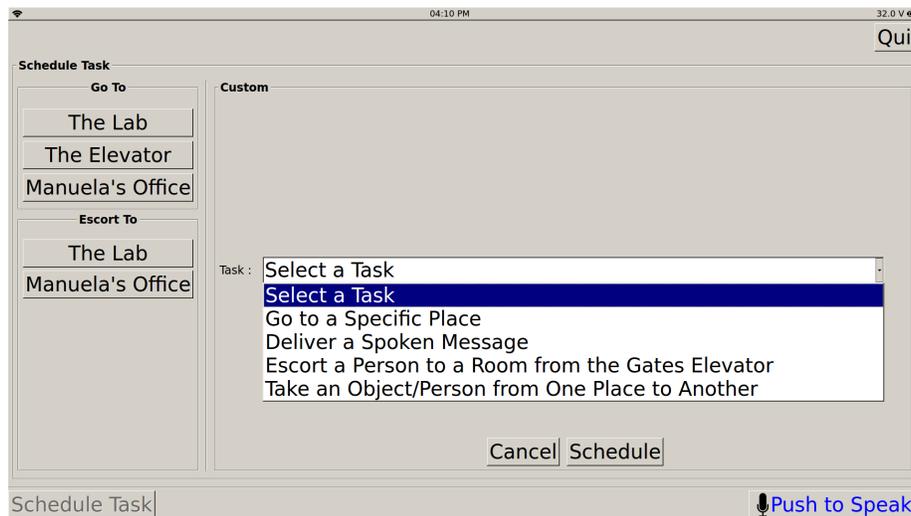


Figure 2.6: CoBot tasks GUI.

Finally, users can interact with the robot (i.e., schedule tasks) using a website. The website, shown in Figure 2.7, is designed similarly to the on board GUI. It consists of a form that users can fill in, with each field corresponding to one of the task arguments.



(a) Book a robot (b) Confirm (c) View booking

Figure 2.7: Several views of CoBot website. Pictures taken from [83].

In conclusion, we have shown how users and the CoBot robots can interact in multiple ways. Users can request tasks via an on-board GUI or website, and robots can ask for help to accomplish tasks. This thesis introduces a new paradigm into the interaction between robots and their users.

Now, both parties can use natural language; users can request that the robot execute tasks and ask questions about the history of tasks executed, and robots can describe their navigation experience and pro-actively report on the tasks they execute.

2.5 CoBot Logging

In the last part of the scenario described in Section 2.1, we mentioned that, during each run, the robot records logs of its execution. To detail the operation of the CoBot robots' logging, we must take a step back to observe the design of their software architecture.

The development of the CoBot robots is developed based on ROS¹ (Robotic Operating System), a meta-operating system that provides hardware abstraction, low-level device control, the implementation of commonly-used functionality, message-passing between processes and package management. ROS makes it possible to use a modular architecture to design the robot code. Each module, called a node, implements specific functionality, such as hardware drivers, localization, navigation, graphical user interfaces or a task planner.

ROS allows for two forms of inter-node communication: services and topics. Services are one-to-one interfaces for sending dedicated commands from one node to another. Services are the ROS equivalent of remote procedure calls (RPC), where the node providing a specific service acts as the server and the node requesting it as the client. Topics are data streams, which may have multiple publishers and multiple subscribers. Each topic is characterized by a single message type, and the message type defines the data structure of the messages exchanged on the topic. For example, the message `CobotSpeechRecognitionMsg` (Figure 2.8) is composed of utterances, a list of strings representing the possible transcription of the speech recorded, and confidences, a list of floats representing the confidence of the ASR for each of the transcriptions.

<code>string[]</code>	<code>utterances</code>
<code>float32[]</code>	<code>confidences</code>

(a)

<code>utterances:</code>	[<code>"go to the small size lav",</code>	
		<code>" go 2 small sized lab",</code>	
		<code>" goto the small size lab"]</code>	
<code>confidences:</code>	[<code>0.85, 0.425, 0.2125</code>	<code>]</code>

(b)

Figure 2.8: (a) Data structure for the `CobotSpeechRecognitionMsg` message. (b) An instance of the `CobotSpeechRecognitionMsg` message.

When a CoBot robot is deployed, multiple nodes run at the same time, exchanging messages on several topics. Figure 2.9 shows a graph where each vertex represents a ROS node, and each edge is a topic used by the connected vertexes to communicate. The messages exchanged by each node can contain all sorts of information.

¹www.ros.org

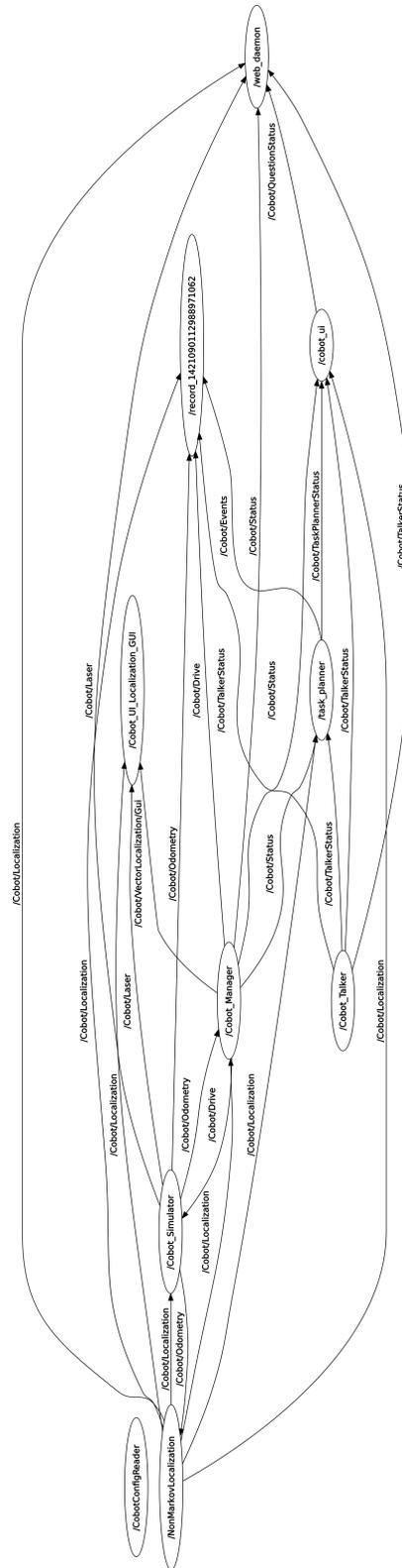


Figure 2.9: The ROS nodes running on the CoBot robots, connected by the ROS topics they publish and subscribe to.

Rather than describing each topic, here we divide them into three levels and present examples of the kinds of information that each level provides. The three levels are as follows:

- **Execution Level:** The lowest level contains information such as the robot’s position (x, y) on the map, together with its orientation, the current voltage of the batteries, the linear and angular velocities, sensors readings (e.g., lidar and Kinect) and information about the navigation, such as the current destination and whether the current path is blocked.
- **Task Level:** At this level, we find all the information regarding the tasks that the robot can execute, including the current task being executed, the time since the task was started, the estimated time to completion and the list of the tasks already scheduled.
- **Human-Robot Interaction Level:** Finally, at this level, we find the information related to interactions with humans, such as events recorded by the GUI (e.g., pressing a button), results of speech recognition, the number of humans detected, open doors detected and the questions asked by the robot.

Table 2.2 shows a few examples of the messages, their content, and the levels where they belong.

Topic	Content	Level
CoBot/Drive	x-velocity, y-velocity, y-velocity, ...	Execution
CoBot/Localization	x, y, angle, angleUncertainty, locationUncertainty, map , ...	Execution
CoBot/TaskPlannerStatus	currentTask, currentSubTask, currentNavigationSubTask, timeBlocked, taskDuration, subTaskDuration, navigationSubTaskDuration, navigationTimeRemaining, timeToDeadline ...	Execution, Task
CoBot/QuestionStatus	question, multiple-choice, click-image, ...	Task, HRI
CoBot/DoorDetector	door-X, door-Y, door-status	HRI
CoBot/SpeechRecognition	utterances, confidences	HRI

Table 2.2: CoBot topics, their content and the level they are assigned to.

Now that we have described the software architecture of the CoBot robots and categorized the topics used by their nodes, we can detail the logging process. A tool called ROSBag² provides ROS with native logging capabilities. ROSBag records the messages being exchanged on each of the topics, in the order that they are published. We regard these recordings as the logs of the robots. By nature, these log files are sequential and, to access the information recorded, we must either go through them in their entirety or pre-process them and store the information in another format. In the rest of this document, we refer to files saved by ROSBag as the robot logs, or .bag files (from the extension the tool uses when saving files). Depending on the task for which the CoBot robots are being deployed, we can set ROSBag to record different sets of messages. As such, not all the logs contain the same information. For example, we seldom record messages related to sensory messages (e.g., reading from lidar and Kinect sensors), as they produce a very

²<http://wiki.ros.org/rosbag>

large amount of data. On the other hand, even in a typical deployment, data is produced at rates that range from 15 to 33 KB/sec [8].

In this section, we have presented the software architecture and logging system of the CoBot robots. In Chapter 6 and Chapter 7, we will see how the robot's ability to record logs of its execution is a key requirement for the applicability of our approach. However, every agent is not required to record logs in the format of .bag files; the robot must only be able to search these logs for information to answer questions and report to its users.

2.6 Summary

In this chapter, we have run through an example scenario, where a user asks the robot to “go to Manuela’s office,” and the robot executes the corresponding task. This example scenario allowed us to introduce the main functionalities of the CoBot robots. The CoBot robots are able to execute tasks, navigate and localize in their environment, interact with their via various interfaces (e.g., an on-board GUI and a website) and record logs of their execution. These capabilities do not represent contributions of this thesis, but rather building blocks available at the start of the work on this thesis that we have used as a foundation.

Chapter 3

Dialogue-Based Learning of Groundings from Spoken Commands to Task Execution

Human: “Go to the kitchen.”

We have shown how the CoBot robots are able to map a sentence like “CoBot can you go to Manuela’s office?” to a task like GoTo(F8001). In this chapter we detail how this mapping happens.

Speech-based interaction holds the promise of enabling robots to become both flexible and intuitive to use. In particular, for a mobile service robot like the CoBot robots, speech-based interaction has to deal with tasks involving people, locations and objects in the environment. If the user says “Go to Manuela’s office” or to “Get me a coffee” the mobile robot needs to infer the type of action it should take, the corresponding location parameters and the mentioned object.

If we place no restrictions on speech, interpreting and executing a command becomes a challenging problem for several reasons. First, the robot may not have the knowledge necessary to execute the command in this particular environment. In the above examples, the robot must know where “Manuela” or “a coffee” are located in the building, and it should understand the type of action a user asks for when using phrases like “get me” or “go to.” Second, performing robust speech recognition is a challenging problem in itself. Often speech recognition results in multiple interpretation strings, some of which might be a partially correct translation, but others can be less intelligible. Finally, speech-based interaction with untrained users requires the understanding of a wide variety of ways to refer to the same location, object or action.

To bridge the semantic gap between the robot and human representations, and to enable a robot to map users’ sentences to tasks, we introduce **KnoWDiaL**[42, 58]. KnoWDiaL is an approach that allows robots to **L**earn task-relevant environmental **K**nowledge from human-robot **D**ialogue and access to the **W**eb. Figure 3.1 shows an example of the interactions that KnoWDiaL enables.

KnoWDiaL contains five primary components: A frame-semantic parser, a probabilistic grounding model, a Knowledge Base, a Web-based predicate evaluator and a dialogue manager. Once a user provides a spoken command, our frame-semantic parser maps the entire list of speech to text candidates to pre-defined frames containing slots for phrases referring to action

	<p>Dialogue Example</p> <p>Human: Get me a coffee. CoBot: According to OpenEval this object is most likely to be found in location “kitchen”. Is that correct? Human: Yes.</p>	<p>Learned Facts</p> <hr/> <p>objectGroundsTo(“coffee”, 7602) actionGroundsTo(“get”, Pu&Delivery)</p>
(a)	(b)	(c)

Figure 3.1: Example of an interaction between a user and mobile service robot CoBot, with KnoWDiaL. **(a)** Speech-based verbal interaction; **(b)** action and object inferred from spoken command and access to the web with OpenEval for object location inference; **(c)** Learned knowledge base on “F7602” being the room number from the robot semantic map of the location “kitchen.”

types and slots for phrases referring to action parameters. These frames are modeled after the task the robot can execute, shown in Table 2.1. Next, using the Knowledge Base, the probabilistic grounding model maps this set of frames to referents. In our system, referents are either known action types (robot tasks) or room numbers (the locations from the robot’s semantic map), which we assume are known. In case required information is missing, the dialogue manager component attempts to fill missing fields via dialogue or via Web searches. In the event that it attempts a Web search, it generates a query to OpenEval, which is a Web-based predicate evaluator that is able to evaluate the validity of predicates by extracting information from unstructured Web pages [67, 68]. When the action type and required fields are set, the dialogue manager asks for confirmation, executes the task and updates the Knowledge Base.

The rest of this chapter is organized as follows: Section 3.1 presents the complete KnoWDiaL system with its five main components. Next, in Section 3.2, we presents our empirical evaluations (the controlled experiments). Then, in Section 3.3, we run through several examples of dialogue interaction with KnoWDiaL implemented on CoBot. Finally, in Section 3.4, we draw our conclusions on KnoWDiaL as a whole.

3.1 KnoWDiaL

The overall structure of KnoWDiaL is shown in Figure 3.2. The input from the user – a spoken command – is recorded and processed by a third-party speech recognition engine¹. The output of KnoWDial is either a task for the robot to execute or a question for the user to answer. In case the robot needs to ask a question, we use a third-party text-to-speech engine² to allow the robot to say the question out loud.

The first component of KnoWDiaL is a frame-semantic parser. This parser labels uses each of

¹On the CoBot robots we use Google ASR Cloud services, <https://cloud.google.com/speech/>

²On the CoBot robots we use Espeak, <http://espeak.sourceforge.net/>

the speech-to-text candidates returned by the ASR and stores them in predefined frame elements, such as action references, locations, objects or people.

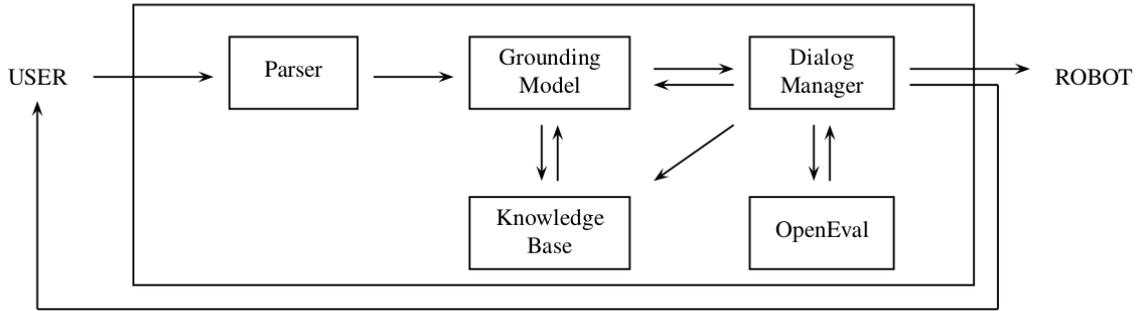


Figure 3.2: Schematic overview of KnoWDiaL with its five components.

The second component of KnoWDiaL is a Knowledge Base storing groundings of commands encountered in previous dialogues. A grounding is a probabilistic mapping of a specific frame element obtained from the frame-semantic parser to locations in the building or tasks the robot can perform.

The Grounding Model, the third component of KnoWDiaL, uses the information stored in the Knowledge Base to infer the correct action to take when a command is received. Sometimes not all of the parameters required to ground a spoken command are available in the Knowledge Base; when this happens, the Grounding Model resorts to OpenEval, the fourth component of KnoWDiaL. OpenEval is able to extract information from the World Wide Web to fill missing parameters of the Grounding Model.

In case a Web search does not provide enough information, the fifth component of KnoWDiaL, the Dialogue Manager, engages in dialogue with the user, and explicitly asks for the missing parameters. The Dialogue Manager also decides when to ask a follow-up question and when to ask for confirmation. When a command is successfully grounded, the Dialogue Manager schedules the task in the CoBot planning system and updates the KnoWDiaL Knowledge Base.

3.1.1 High-Level Joint-Probabilistic Model

Before describing the five components of KnoWDiaL in detail, we first formally introduce our high-level model. We formalize the problem of understanding natural language commands as inference in a joint probabilistic model over the groundings Γ , a parse P and speech S , given access to a Knowledge Base K and OpenEval O . Our goal is to find the grounding that maximizes the joint probability, as expressed by the following equation:

$$\arg \max_{\Gamma} p(\Gamma, P, S | K, O) \quad (3.1)$$

This joint model factors into three main probability distributions: A model of speech, a parsing model and a grounding model. Formally:

$$p(\Gamma, P, S | K, O) = p(\Gamma | P, K, O) \times p(P | S) \times p(S) \quad (3.2)$$

In our system, the probability of the speech model $p(S)$ is given by a third-party speech-to-text engine. The factors for parsing and grounding, respectively $\times p(P|S)$ and $p(\Gamma|P, K, O)$ will be derived in the upcoming sections.

3.1.2 Frame-Semantic Parser

The speech recognizer returns a set $\mathcal{S} = [S_1, \dots, S_n]$ of speech-to-text candidates and a confidence C_{S_i} for each of them. The first step of KnowDiaL is to parse each of these sentences. To train our parser we collected a corpus of approximately 150 commands (see Appendix A.1) from people within our group, read these out loud for the speech recognizer and annotated the resulting speech to text candidates by hand. Figure 3.3 shows a small sample of the annotations used. The labels we used were: *action*, *toLocation*, *fromLocation*, *toPerson*, *fromPerson*, *robot*, *objectHere* and *objectElse*. Words we labeled as *action* were the words used to refer to the tasks the robot can execute (e.g., “go,” “bring” or “please deliver”). Locations, both *toLocation* and *fromLocation*, include expressions like “classroom” or “printer room.” People, labeled as *toPerson* or *fromPerson*, include expressions like “Tom” or “receptionist.” Examples of object references are “cookies” or “tablet pc” and are labeled as *objectHere* or *objectElse*. With *robot* we labeled parts of the command that refer to the robot itself. Words that were supposed to be ignored were labeled with the additional label *none*.

Commands in Corpus and Their Annotation

- [Go]_{action} to the [bridge]_{toLocation} [CoBot]_{robot}
 - Could [you]_{robot} [take]_{action} a [screwdriver]_{objectElse} to the [lab]_{toLocation}
 - [Go]_{action} to [Dana’s]_{toPerson} [office]_{toLocation}
 - Please [bring]_{action} a [pencil]_{objectElse} from the [lab]_{fromLocation} to the [meeting room]_{toLocation}
 - [Return]_{action} these [documents]_{objectHere} to [Diane]_{toPerson}
 - Please [get]_{action} [me]_{toPerson} some [coffee]_{objectElse} from the [kitchen]_{fromLocation}
-

Figure 3.3: A corpus of approximately 150 go to location and transport object commands is annotated by hand. Separate labels are used to distinguish whether an object can be found at the current location or elsewhere. After learning, KnowDiaL is able to properly recognize the action type of these commands and extract the required parameters.

Generally speaking, labeling tasks often require a much bigger training set [71], but 150 commands proved to be enough to train our parser. We identify three main reasons for this: First, for each command we get multiple, slightly different, speech-to-text candidates (typically 5 to 10), resulting in an increase in the effective size of the corpus. Second, our set of labels is relatively small. Third, the language used to give commands to our robot is limited by the tasks the robot is able to execute, transporting objects and going to a location.

If $l_i \in \{action, fromLocation, toLocation, fromPerson, \dots\}$ is the label of the i -th word in a speech-to-text candidate S , and this candidate contains N words s_i , then the parsing model is

represented as a function of pre-learned weights w and observed features:

$$p(P|S) \triangleq p(l_1 \dots l_N | s_1 \dots s_K) \quad (3.3)$$

$$= \frac{1}{Z(S)} \exp \left(\sum_i^N w \cdot \phi(l_i, s_{i-1}, s_i, s_{i+1}) \right) \quad (3.4)$$

where $Z(S)$ is a normalization factor and ϕ is a function producing binary features based on the part-of-speech tags of the current, next, and previous words, as well as the current, next, and previous words themselves. The weights w for each combination of a feature with a label were learned from the corpus mentioned before. We learned them as a Conditional Random Field (CRF) and used gradient descent (LBFGS) as our method to optimize.

After labeling all of the words in each of the speech interpretations in \mathcal{S} , we want to extract a frame from them. To do so for each $S \in \mathcal{S}$, we greedily group together words with the same label. The output of the frame-semantic parser is therefore a set of parses $\mathcal{P} = [P_1, \dots, P_n]$, one for each of the speech interpretations in \mathcal{S} . Each of the parses P_i consists of labeled chunks along with an overall confidence score C_{P_i} .

3.1.3 Knowledge Base

In the Knowledge Base of KnowDial, facts are stored by using five predicates: *actionGroundsTo*, *personGroundsTo*, *locationGroundsTo*, *objectGroundsTo* and *locationWebFitness*. Four of them, the *GroundTo* predicates, are used to store previously user-confirmed groundings of labeled chunks obtained from the semantic-frame parser. The fifth, *locationWebFitness*, is used when querying OpenEval. The rest of this section describes each of the predicates in detail.

The predicate *actionGroundsTo* stores mappings between references to actions and the corresponding tasks for the robot. KnowDial enables the CoBot robots to execute two tasks, *GoTo* and *Pu&Delivery*. Whenever the robot receives a task request, the chunk labeled as action is grounded to one of these two tasks. Examples of this type of predicate are *actionGroundsTo*(‘take’, *Pu&Delivery*) and *actionGroundsTo*(‘get to’, *GoTo*).

The two predicates *personGroundsTo* and *locationGroundsTo* have very similar functions, as they both map expressions referring to people or locations to places the robot can navigate to. As we shown in Chapter 2, the CoBot robots use a semantic map to execute their tasks. In the semantic map each room in the building is labeled with a four digit number. KnowDial uses these labels as groundings for both of the predicates, as in *locationGroundsTo*(‘small-size lab’, 7412) or *personGroundsTo*(‘Alex’, 7004). The function of these two predicates is similar, but they convey slightly different information: *LocationGroundsTo* saves the way people refer to specific rooms, and *personGroundsTo* is intended to store information about where the robot is likely to find a specific person.

The fourth predicate storing information about grounding is *objectGroundsTo*. Similarly to the *personGroundsTo*, this predicate stores information about where the robot is likely to find a specific object. As for the two previous predicates, objects are grounded to room numbers, for example: *objectGroundsTo*(“screwdriver”, 7412) or *objectGroundsTo*(“marker”, 7002).

A number is attached to each of the four grounding predicates in the Knowledge Base to keep track of how many times an expression, e , has been mapped to a grounding, γ . From now on, we

will refer to this number by using a dotted notation, such as $locationGroundsTo(e, \gamma).count$ or simply as $count$; the updates of this value are explained through detailed examples in Chapter 3.3.

Finally, the last predicate used in KnowDiaL is $locationWebFitness$. The Knowledge Base contains one instance of this predicate for each $locationGroundsTo$ element. The goal of this predicate is to store how useful each expression referring to a location is, when querying the Web using OpenEval (more details in Section 3.1.5). To keep track of this information, we assigned a score between 0 and 1 to each expression. Examples of this predicate are $locationWebFitness("the\ elevator", 0.890825)$ and $locationWebFitness("elevated", 0.375724)$.

3.1.4 Grounding Model

In KnowDiaL, all of the tasks the robot can execute are represented by semantic frames. A semantic frame is composed by an action, a , invoking the frame, and a set of arguments, e ; therefore, $grounding$ a spoken command corresponds to identifying the correct frame and retrieving all of its argument. Figure 3.4 shows two examples of semantic frames and their arguments; these frames match the tasks initially introduced in Table 2.1.

Frame: <i>GoTo</i> - Parameters: destination	Frame: <i>Pu&Delivery</i> - Parameters: object, source, destination
---	--

Figure 3.4: Semantic frames representing two of the tasks our CoBot robots are able to execute.

We make the assumption that the action, a and the arguments e of a frame can be grounded separately. The chunks returned by the frame-semantic parser correspond either to an action, a or to one of the arguments, e . Therefore, to compute $p(\Gamma|P, K, O)$ in Equation 3.2, we first need to find the most likely grounding γ for the action and then for each of its arguments. The general formula used to compute the likelihood of a grounding is the following:

$$p(\gamma_*|F; \mathbf{K}) = \frac{\sum_i C_{S_i} \cdot C_{P_i} \cdot groundsTo(chunk_i, \gamma_*) \cdot count}{\sum_{i,j} C_{S_i} \cdot C_{P_i} \cdot groundsTo(chunk_i, \gamma_j) \cdot count} \quad (3.5)$$

where γ_* is a specific grounding, C_{S_i} and C_{P_i} are respectively the confidence of the speech recognizer and of the parser, i ranges over the set of parses \mathcal{P} , j ranges over all of the various groundings for the frame element being considered (*i.e.*, the action a or one of the parameter e), and $groundsTo$ is one of the predicates in the Knowledge Base: $actionGroundsTo$, $locationGroundsTo$, $personGroundsTo$ or $objectGroundsTo$. The $chunks$ used to compute Equation 3.5 are the ones matching the frame element currently being grounded. For instance, if we are trying to ground the action, only the chunks labeled as *action* are considered.

Section 3.3 explains with detailed examples how Equation 3.5 is used to infer the correct grounding for a command.

3.1.5 Querying the Web

One of the core features of KnoWDiaL is the ability to autonomously access the Web *to ground* the location to which the robot needs to navigate based on a request that does not explicitly mention a known location. So far we have provided this ability for the robot to determine the location of objects. For example, if a user requests “Please, bring me coffee,” the robot may not know the location of the object “coffee.” KnoWDiaL accesses the Web by using OpenEval [67, 68] to determine the possible location(s) of objects corresponding to its map of the building. The detailed description of OpenEval technology is beyond the scope of this thesis, but we review the general approach of OpenEval by focusing on its interaction with KnoWDiaL.

OpenEval [66] is an information-processing approach capable of evaluating the confidence on the truth of any proposition by using the information on the open World Wide Web. Propositions are stated as multi-argument predicate instances. KnoWDiaL uses two predicates when invoking OpenEval to determine the possible object location and the appropriateness of those locations: *locationHasObject* and *locationWebFitness*. KnoWDiaL fully autonomously forms the queries, as propositions (instantiated predicates) to OpenEval based on its parsing of a user request. An example of a proposition generated by KnoWDiaL to query OpenEval is *location-HasObject(kitchen, coffee)*, to which OpenEval could return a confidence of 80%, meaning that it computed a confidence of 80% on the truth of the proposition: that *kitchen* is a *location* that has the *object coffee*.

In terms of the two predicates of KnoWDiaL, the hope is that OpenEval returns high confidence on valid pairs of objects and locations, as well as on appropriate locations themselves. For example, OpenEval returns a high and low confidence, respectively, on queries about locations “kitchen” and “kitten,” where the latter could have been incorrectly generated by a speech interpreter. KnoWDiaL uses the confidence values returned by OpenEval in two core ways: To decide whether to ask for further information from the user, and to update its Knowledge Base on grounding information. We provide details on the Knowledge Base updates in Chapter 3.4.

3.1.6 Dialogue Manager

Our Dialogue Manager uses each of the modules described in the previous sections to interpret commands and come up with a single complete grounding. The Dialogue Manager takes the speech-to-text candidates \mathcal{S} as input and tries to ground the command received to one of the tasks the robot can execute and all of its parameters by using the Knowledge Base. If some of the groundings cannot be retrieved from the Knowledge Base, the Dialogue Manager tries to fill in the missing fields either by asking specific questions or querying the Web via OpenEval. Once all of the groundings have been retrieved, the Dialogue Manager asks for confirmation, updates the Knowledge Base and schedules the task. Algorithm 1 shows all of the steps described.

Figure 3.5 shows a typical dialogue for a user who asks CoBot to deliver a pencil to the meeting room. The speech recognizer returns three text-to-speech candidates (Figure 3.5b). These are parsed (Figure 3.5c) and then grounded by using the Knowledge Base. Given the current Knowledge Base, not shown here, KnoWDiaL is able to ground the action required with the command *Pu&Delivery*, that the object to be delivered is *pencil* and that it should be delivered to room 7502. The information missing to completely ground the command is where the object can be

Algorithm 1 dialogue_manager(S)

$F \leftarrow \text{parse_and_frame}(S)$
 $\Gamma \leftarrow \text{ground}(F)$
 $\Gamma^* \leftarrow \text{fill_missing_fields}(\Gamma)$
ask_confirmation(Γ^*)
update_knowledge_base(Γ^* , F)
schedule_task(Γ^*)

found (Figure 3.5d); to retrieve it, KnowDial queries the Web and, after receiving confirmation from the user (Figure 3.5e), executes the task.

USER: “Go deliver a pencil to the meeting room.”

(a) Spoken Command.

$s_1 =$ “go deliver a pencil to the meeting rooms”

$s_2 =$ “good liver pen still in the meeting room”

$s_3 =$ “go to live fo r pencil to the meaning room”

(b) Speech to Text Candidates.

$f_1 =$ [go deliver]_{act} [pencil]_{objElse}
[meeting rooms]_{toLoc}

$f_2 =$ [good liver]_{act} [pen still]_{objHere},
[meeting room]_{toLoc}

$f_3 =$ [go]_{act} [live]_{toPer}
[pencil meaning room]_{toLoc}

(c) Parsing and Framing.

$\Gamma = [a = \text{Pu\&Delivery}, e_{\text{source}} = \text{NULL},$
 $e_{\text{destination}} = 7502, e_{\text{obj}} = \text{pencil}]$

(d) Action / Parameter Grounding

CoBot: “You want me to deliver the object pencil, is that correct?”

USER: “Yes”

CoBot: “I am querying the web to determine where the object pencil is located.”

Location-references with high count and validity score are extracted from the Knowledge Base: “kitchen,” “office,” and “lab.” OpenEval is evaluated for these expressions.

$\Gamma^* = [a = \text{Pu\&Delivery}, e_{\text{source}} = 7602,$
 $e_{\text{destination}} = 7502, e_{\text{obj}} = \text{pencil}]$

CoBot: “I have found that object *pencil* is most likely to be found in location *office*. I am going to get object *pencil* from location *office* and will deliver it to *room 7502*, is that correct?”

USER: “Yes”

(e) Dialogue

Figure 3.5: A dialogue example. Given the command the robot is able to ground, using its Knowledge Base, the action, and two of the frame parameters (“*destination*” and “*object*”). The third parameter needed to completely ground the command (“*source*”) is grounded using OpenEval.

Additionally, to make KnowDial easier to use, we added a small set of keywords to the

Dialogue Manager. If the words “cancel” or “stop” are detected, the current dialogue is canceled and the user can give a new command to the robot. If the words “wrong action” are recognized, KnoWDiaL asks explicitly for the task it needs to perform and then resumes its normal execution. The Dialogue Manager also recognizes keywords such as “me” and “this” to handle commands involving the current location as one of the action parameters (e.g., “Bring me a cookie” or “Bring this paper to the lab”). In this case, a temporary location *here* is created to ground the location and, during execution, is converted to the room nearest to the position where the robot received the command.

3.2 Experimental Evaluation

The performance of KnoWDiaL is determined by the performance of each of its five components. Therefore, in this section we describe two experiments, gradually increasing the number of subsystems involved. In the first experiment, we consider command referring only to *GoTo* tasks. The goal is to measure the ability of the Semantic Parser and the Grounding Model to extract the correct frame and execute the task as required. We measure the performance of these two components in terms of the number of required interactions (i.e., the number of questions our participants need to answer), and we compare the ways people refer to diverse types of locations. In the second experiment, we evaluate the Dialogue Manager and OpenEval with commands involving both *GoTo* and *Pu&Delivery* tasks.

3.2.1 Learning Location Groundings

In our first experiment, we asked nine people to command the robot for about ten minutes, sending it to six locations on its map (in simulation). The subjects ranged between the age of 21 and 54 and included both native and non-native English speakers, which made the task more challenging for the robot. Although the task itself was fixed, people could use language that was natural to them. To prevent priming our participants with location references, we used a printed map of the building that CoBot operates in. Six locations were marked on this map and annotated with a room number, as in Figure 3.6. The aim of this experiment was to test the ability of KnoWDiaL to learn referring expressions for various locations through dialogue alone. In order to properly assess this ability we started our experiment with an empty Knowledge Base. After each person interacted with the robot, the knowledge was aggregated and used as a starting point for the following participants.

We compared the results of KnoWDiaL with two baseline. The first baseline, called the *Naïve Baseline*, enables the robot to execute the task without learning any semantic information about the environment. When receiving a command, a robot using this baseline enters a fixed dialogue. The dialogue consists of two questions the first explicitly asking which task the robot should execute, and the second asking for the room number of the destination. Although it is less natural than the proposed approach because the person must explicitly define the room number and action, only two questions are required before the robot can execute the task. The second baseline proposed, called *Semantic Baseline*, tries to execute the assigned task while learning semantic knowledge about the environment. Using this second baseline, the robot first asks for

the task to be execute and then for the destination. In contrast with the Naïve Baseline, the Semantic Baseline does not explicitly ask for the room number of the destination; therefore, if the user does not use a four-digit room number to express the destination, the robot asks a third question to retrieve it.

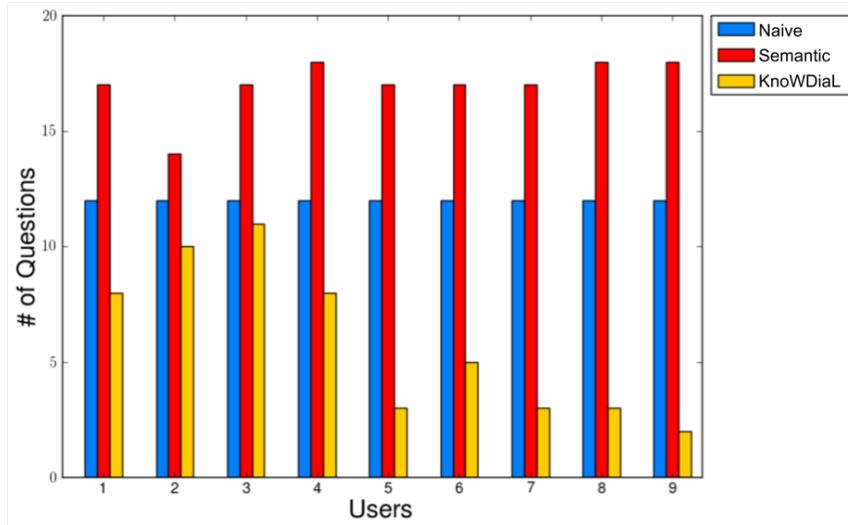


Figure 3.7: Comparison between KnoWDiaL and the two proposed baselines.

A total of 54 “go to” location commands were given by our subjects. These contributed to a Knowledge Base with 177 predicates, grounding either a location reference or phrases referring to a person. Figure 3.7 shows the results of this experiment. On the horizontal axis are the nine people who interacted with the robot and on the vertical axis are the number of questions asked during each session. The KnoWDiaL approach always performs better than both baselines, moreover the number of questions asked shows a decreasing trend. We foresee the number of questions asked to decrease even further as the robot accumulates more facts in its Knowledge base, but we do not expect the number of questions to go to zero. This is because of the long-tail distribution of words in the English language. In most cases, users phrase a command using words and expressions that dominate the long tail distribution, and the robot quickly learn these expressions. Occasionally, the users will phrase their request using unusual expressions (from the tail of the distribution), and the robot will ask questions to ground them. We can observe this trend in the interaction the robot had with the users involved in the experiment. In the interactions with the first four users the robot asked questions to ground both the action and the destination. When User 5 started giving commands to the robot, the Knowledge Base already stored the most common ways to refer to the task (e.g. *go to*, *bring me*, *take me*). Because KnoWDiaL only had to ground the destination, we observe a drop in the number of questions asked after User 4. Interestingly, to understand the request from User 6, the robot had to ask more questions than with the previous user. This is because the user phrased their requests using peculiar expressions the robot had not encountered in its interactions with previous users. For instance, User 6 referred to room 7107 as “the Hogwarts stairs,” an expression that was not repeated by any other user.

Finally, we use entropy to evaluate whether different people refer to a location by using very different expressions. When calculating statistical entropy over the set of referring expressions

for a specific location, we find the lowest entropy to be 2.8 for the location “the elevator.” The highest entropy 3.3 was found for “the meeting room” and “the atrium.” For the latter location, people were using references like “the open area,” or “the place with the large windows.” On average, the entropy for expressions referring to location was 3.0. Figure 3.8 shows the Knowledge Base count corresponding to each referring expression for the lowest and highest entropy location. Because speech-to-text is not always perfect, “the atrium” was often translated into “the 8 gym.” Our dialogue system does not see a difference between the two translations and will learn to understand commands involving an inaccurate speech-to-text translation just as quickly as ones involving the right translations. As long as speech-to-text is consistent, it does not have to be perfect to allow our dialogue system to learn.

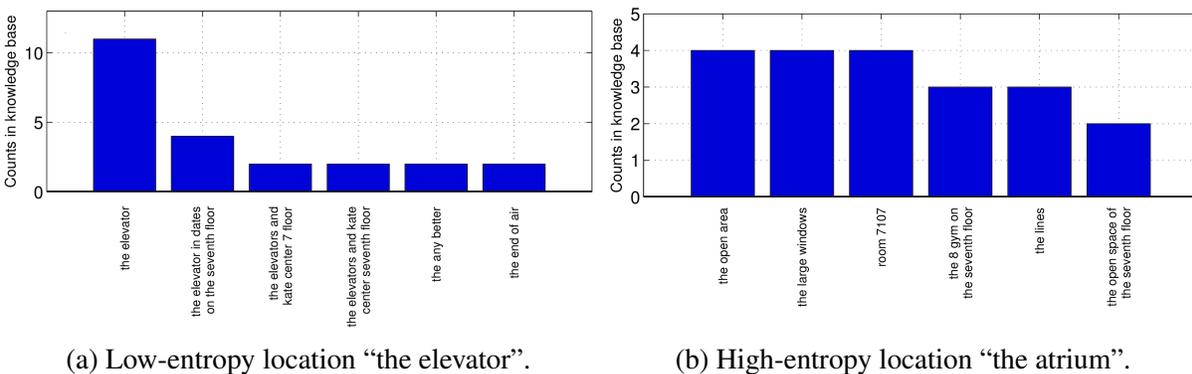


Figure 3.8: The six location references that were most frequently encountered at two locations.

3.2.2 Learning Object Groundings

Our second experiment involved ten people, with six of them being non-native English speakers. Our aim here is to test the dialogue system as a whole, including the grounding of objects. Once again, participants were provided with a map that had seven marked locations, annotated with room numbers. Unlike our initial experiment, not all of the participants were familiar with the building this time; therefore, we also provided a suggested expression for each of the marked locations on the map. Participants were free to use this suggested expression or any other synonym or another expression that they found to be natural. We showed to each participant a sheet with pictures of 40 objects that our robot would be able to transport. We chose pictures instead of a list of words in order to prevent priming our participants with a way to refer to specific objects.

Each of the participants were asked to command the robot through its speech interface for about 15 minutes. The participants were free to choose whether they would ask the robot to transfer one of the objects, or to simply send it to a specific location. A *Pu&Delivery* command could involve asking the robot to deliver an object provided by the user to any of the locations on the map or asking it to deliver an object that first needs to be collected at some other place. In the latter case, the *source* could either be explicitly provided in the command (“bring me a cookie from the kitchen”) or not be specified by the user (“bring me a cookie”). If the from-location is not explicitly provided, the robot has to come up with a reasonable place to look for the object.

It could do so either by doing inference over the *objectGroundsTo* predicates in its Knowledge Base (implicit grounding of the from-location) or by using the expression for the object to query OpenEval.

The baseline that we are comparing our system to is a dialogue manager that simply asks which action it should take, followed by a question for each of the parameters needed to execute this action. In the case of a “transport object” command, as shown in Figure 3.4, three parameters are necessary: (1) the source of the object (*i.e.*, the location where it can be found), (2) its destination and, as the robot should ask someone to put the object in its basket at the source, (3) a reference to the object itself. Therefore, the baseline for a “transport object” command is four questions, and the baseline for “go to” location commands is two.

We started the experiment with an entirely empty Knowledge Base. After a total of 91 speech-commands, our system had asked only 67% of the number of questions the baseline system would have asked. Our system posed more questions than the baseline system would have done in only in 12% of the commands (the worst-case was three additional questions).

To explain this result in more detail, we take a closer look at each of the elements that we are grounding. Most of the learning with respect to grounding action types takes place during the commands provided by the first three people in the experiment (Figure 3.9a). Apparently their way of invoking an action, generalizes easily. In the remainder of the experiment, starting at Command 33, the wrong action type was recognized only six times.

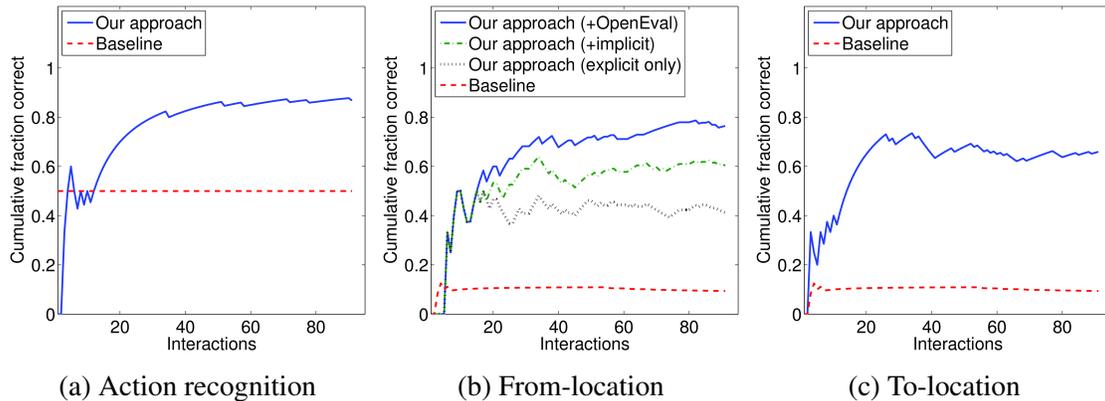


Figure 3.9: Grounding performance during an experiment involving 91 “go to” locations and “transport object” tasks. The baseline for these graphs consists of the percentage we would be able to guess correctly by picking a random action type, or a random location from our Knowledge Base. A from-location can be found in three ways: By grounding a location reference provided by the user (explicit grounding) by grounding the object reference to a location (implicit grounding) or by using OpenEval.

Roughly two thirds of the commands in this experiment were “transport object” commands; the others were “go to” location commands. In the case of a “transport object” command, our subjects chose not to provide the from-location 31 times; and out of these the expression for the object could not be grounded directly from the Knowledge Base in 19 cases. OpenEval was able to come up with a correct object location 11 times (Figure 3.9b). This was done either

by returning a single correct location (2 times), by asking the user to choose out of two high-probability locations (7 times), or by offering three high-probability locations (2 times). As shown in the graph, it takes some commands before OpenEval becomes useful because first some location references with web fitness scores need to be in the Knowledge Base.

At the end of the experiment, 41% of the from-locations were found by grounding a location reference provided by the user. Taking into account that the user did not always explicitly provide this parameter, 75% of the provided from-location references are grounded correctly, slightly better than what has been achieved when grounding the to-location (Figure 3.9c).

Average entropy calculated over the referring expressions for each of the locations in the Knowledge Base was 3.0, which is equal to what we obtained in our first experiment. Therefore, we conclude the suggested location reference did not lead to a smaller spread in referring expressions that participants were using for locations.

Measured over the entire experiment, 72% of the object expressions in “transport object” commands were extracted correctly. In our system extracting the object reference from a speech-to-text candidate does not require a Knowledge Base, so this percentage remained constant throughout the experiment.

3.3 Running Example

We now illustrate the complete KnoWDial with two examples to show the details of the accesses and computations underlying the updates to the Knowledge Base. In the two examples, the new groundings come, respectively, from the dialogue with the user and from access to the Web.

3.3.1 Accessing and Updating the Knowledge Base from Dialogue

When receiving a spoken command, in this example the sentence “Go to the small-size lab,” the first step is to process the audio input and get a set of multiple transcriptions from the ASR. The speech recognizer used on the CoBot robots returns an ordered set of interpretations, $\mathcal{S} = [S_1, \dots, S_n]$, but only provides a confidence score, C_{S_1} , for the first one. Because KnoWDial expect each transcription to have a confidence score we compute it using the following formula:

$$C_{S_i} = \max(C_{S_1} - \alpha \cdot C_{S_1} \cdot i, \alpha \cdot C_{S_1})$$

where i is the rank of each interpretation and α is a discount factor. We use this formula to ensure that each transcription has an associated confidence. Other formulas could be used to compute the confidence of each transcription but, as long as the score computed reflects the ordering returned by the ASR, the final result of the grounding process would not change. Figure 3.10a shows all of the transcriptions obtained from the Automated Speech Recognition (ASR) along with the confidence scores computed.

Next, each transcription is parsed. The result of this step is shown in Figure 3.10b and consists of a set of parses $\mathcal{P} = [P_1, \dots, P_n]$ where each parse P_i contains a set of labeled chunks along with a confidence score, C_{P_i} for the whole sentence.

Because the goal is to completely fill a semantic frame representing one of the tasks the robot can perform, we first need to identify the frame invoked by the command received that

is to *ground* the action of the command. To do so, we query the Knowledge Base for all the *labelsGroundTo* predicates whose first argument matches any of the label sequence in \mathcal{P} and for all the *actionGroundsTo* predicate whose first argument matches any of the chunks labeled as *action* in $[P_1, \dots, P_n]$. This query returns a set of j possible groundings γ , in this example [*GoTo*, *Pu&Delivery*]. To select the correct grounding, we use Equation 3.5 for the *actionGroundsTo* predicate and compute the probability for each of the j groundings returned. We select the grounding with the highest value as correct, which in this example, is the task *GoTo* with a probability of 0.994.

Go to the small size lav	0.85
go 2 small sized lab	0.425
goto the small size lab	0.2125
get the small sized love	0.2125

(a) Speech recognition results

[Go to] _{action} [the small size lav] _{toLocation}	0.8
[go 2] _{action} [small sized lab] _{toLocation}	0.1
[goto] _{action} [the small size lab] _{toLocation}	0.3
[get] _{action} [the small sized love] _{objectHere}	0.7

(b) Parses

actionGroundsTo('go to', GoTo)	5.0
actionGroundsTo('goto', GoTo)	2.3
actionGroundsTo('goto', Pu&Delivery)	0.3
actionGroundsTo('get', Pu&Delivery)	2.15
locationGroundsTo('the small size lab', 7412)	7.9

(c) Initial Knowledge Base

Figure 3.10: Multiple transcriptions (a) and parse (b) for the command “Go to the small-size lab.” (c) Shows the initial Knowledge Base with the count for each predicate.

Once the action has been grounded, the corresponding semantic frame shows which parameters are needed. For the *GoTo* frame, the only parameter needed is the destination. To ground the destination we query the KB for all the *locationGroundsTo* predicates, whose first argument matches the chunks labeled as *toLocation*. Similarly to what happened for the action, j possible groundings are returned, and for each of them, we compute its probability by using Equation 3.5 for the *locationGroundsTo* predicate. The one with the highest probability is then selected as the final grounding; in our example, room 7412 is selected with probability 1, as it is the only *locationGroundsTo* predicate available in the Knowledge Base (shown in Figure 3.10c).

At this point, the semantic frame representing the command received has been completely filled. Before executing the corresponding task, KnoWDial engages in a short dialogue with the user, and if everything is confirmed, lets the robot execute the task.

Finally, while the robot is executing the task KnowDiaL updates its KB. For each of the chunks of all of the parses in \mathcal{P} , the count of the corresponding predicate is increased by $C_{S_i} \cdot C_{P_i}$; in particular, the chunks labeled as *action* increase the count of the *actionGroundsTo* predicate, and the chunks labeled as *toLocation* increase the count of the *locationGroundsTo* predicate. If, for any of the predicates, an instance is not already present in the Knowledge Base a new one is simply added. Figure 3.11 shows the updated Knowledge Base after the command has been executed.

actionGroundsTo('go to', GoTo)	5.68
actionGroundsTo('goto', GoTo)	2.36375
actionGroundsTo('go 2', GoTo)	0.0425
actionGroundsTo('goto', Pu&Delivery)	0.3
actionGroundsTo('get', Pu&Delivery)	2.15
locationGroundsTo('the small size lab', 7412)	7.96375
locationGroundsTo('small sized lab', 7412)	0.0425
locationGroundsTo('the small size lav', 7412)	0.68

Figure 3.11: Update KB, highlighted in blue are the predicates that have been added or updated.

3.3.2 Accessing and Updating the Knowledge Base from the Web

For this second example, we will consider the command “*Bring coffee to the lab.*” Similar to the previous example, the audio input is processed by the ASR, and its output is parsed. The result, a set of speech interpretations \mathcal{S} and a set of parses \mathcal{P} , are shown together with the initial Knowledge Base in Figure 3.12.

Again, the first step is to ground the action of the command, that is, to identify the corresponding semantic frame. To do so, we query the Knowledge Base for *actionGroundsTo* predicates and then use Equation 3.5 to compute the most likely action corresponding to the command received. Given the KB for this second example, the only matching action is also the only action in the KB and is therefore selected with a probability of 1.

Having grounded the action, meaning the semantic frame corresponding to the command has been identified, the next step is to ground all of the parameters of the frame. For the *Pu&Delivery* frame, we need three parameters: the object, the location where it can be found, and the location where it has to be delivered. First, we check for the object. That is, we see if we can find a chunk labeled as *objectHere* or *objectElse* in any of the \mathcal{P}_i parse. KnowDiaL simply selects as the object the first chunk whose combined speech and parse confidence is greater than a given threshold, $C_i \cdot C_{P_i} \geq \tau$ with $\tau = 0.15$. In our example the chunk selected to represent the object is “*coffee*”.

Next, we need to figure out where the object can be found. To do so, we first check if the command explicitly mentions it, and we see if, in any of the parses in \mathcal{P} , we can find a chunk labeled as *fromLocation*. If this is the case for each *fromLocation* chunk i we query the KB for a

matching *locationGroundsTo* predicate. This operation returns a set of j possible groundings γ and, again, to compute the more likely we use Equation 3.5.

If the location the object has to be retrieved from is not explicitly mentioned, we query the KB for all of the *objectGroundsTo* predicates, whose first argument matches any of the *objectElse* chunks and compute the more likely grounding applying Equation 3.5 to the *objectGroundsTo* predicate.

Briggs coffee to the lav	0.85
bring the cofee to the lab	0.425
Rings coffe the lab	0.2125

(a) Speech recognition results

[Briggs] _{action} [coffee] _{objectElse} [to the lav] _{toLocation}	0.23
[bring] _{action} [the cofee] _{objectElse} [to the lab] _{toLocation}	0.88
[Rings] _{action} [coffe] _{objectElse} [the lab] _{toLocation}	0.18

(b) Parses

actionGroundsTo('bring', Pu&Delivery)	2.1
actionGroundsTo('rings', Pu&Delivery)	0.3
locationGroundsTo('to the lab', 7412)	4.3
locationGroundsTo('kitchen', 7602)	1.62
locationGroundsTo('kitcheen', 7602)	0.34
locationGroundsTo('kitchenette', 7602)	1.35
locationGroundsTo('office', 7004)	2.8
locationWebFitness('office', 0.98)	
locationWebFitness('kitchen', 0.92)	
locationWebFitness('kitcheen', 0.34)	
locationWebFitness('kitchenette', 0.93)	
locationWebFitness('to the lab', 0.88)	

(c) Initial Knowledge Base

Figure 3.12: Multiple transcriptions (a) and parse (b) for the command “Bring coffee to the lab.” (c) Shows the initial Knowledge Base with the count for each predicate.

Unfortunately, in our example there is no chunk labeled *fromLocation* in \mathcal{P} and no *objectGroundsTo* predicate in the Knowledge Base. When this happens, to figure out where we can find the object, we resort to OpenEval. To query OpenEval, we need a specific object, which we have already identified as “the coffee” and a set of possible locations, \mathcal{L} . To build the set \mathcal{L} , we first query the Knowledge Base for all of the *locationGroundsTo* predicates and add their first argument to \mathcal{L} . Next, we make a second query to the KB and filter out all of the elements in \mathcal{L} having a *locationWebFitness* score below a threshold of 0.9. Finally, we make sure that all of the elements left in \mathcal{L} refer to different physical locations by checking the groundings of the

locationGroundsTo and selecting, among those referring to the same room number, the reference with the highest count. This whole process, for the Knowledge Base of our example, is shown in Figure 3.13.

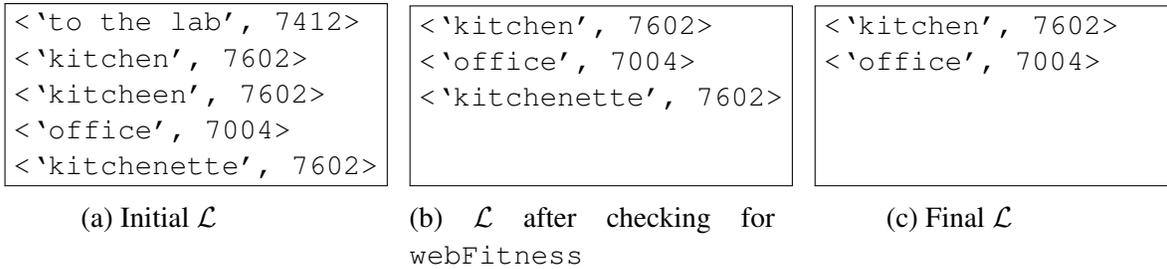


Figure 3.13: The set of possible locations used to query OpenEval.

Querying OpenEval returns a score, C_O , for finding “*the coffee*” in each of the locations in \mathcal{L} ; KnowDiaL then asks the user to select the correct location between those with a score above 0.8. In our example, out of the two locations in \mathcal{L} , only *kitchen* is above the threshold and the user is simply asked to confirm if it is the correct location.

Finally, we need to find out where the object needs to be delivered. To do so, we look in all of the parses in \mathcal{P} if there are chunks labeled as *toLocation*. If we find at least one such chunk, we query the KB for all of the matching *locationGroundsTo* predicates and compute the more likely grounding by using Equation (3.5). In our example, we find multiple chunks labeled as *toLocation* but only one *locationGroundsTo* predicate matching them. Therefore, the grounding 7412 is selected with a score of 1. In general, if there is no *toLocation* chunk or the grounding cannot be retrieved from the KB, KnowDiaL engages in a short dialogue with the user asking for the location explicitly.

Now that the semantic frame has been completely filled, the robot can execute the corresponding task. As this happens, KnowDiaL updates its KB; as in the previous example, for each chunk, the corresponding predicate is increased by the combined score of the speech recognizer and the parse. Moreover, a new predicate, *objectGroundsTo* is added to the KB with the object used to query OpenEval, “*the coffee*”, as a first argument, the grounding associated to the location with the highest C_O score as a second argument and the product of the three confidences $C_O \cdot C_{S_i} \cdot C_{P_i}$ as *count*.

3.4 Summary

In this chapter, we have presented KnowDiaL, an approach for a robot to use and learn task-relevant knowledge from human-robot dialogue and access to the World Wide Web. We have introduced the underlying joint probabilistic model consisting of a speech model, a parsing model, and a grounding model. We focus on two of the tasks the CoBot robots can execute. These tasks involve actions, locations, and objects. Our model is used in a dialogue system to learn the correct interpretations of referring expressions that the robot was not familiar with beforehand. Commands involving various actions, locations and people can be dealt with by adding new

facts to the Knowledge Base and by searching the Web for general knowledge. We presented experiments showing that the number of questions that the robot asked in order to understand a command decreases as it interacts with more people, and that our KnoWDiaL approach outperforms a non-learning baseline system. Finally, we detailed two running examples to demonstrate the use of KnoWDiaL in the CoBot robots.

Chapter 4

Understanding and Executing Complex Commands

Human: *“Bring me some coffee or tea.”*

We have shown how KnowDiaL enables our CoBot robots to understand spoken commands. We argued that language enables robots to become both flexible and intuitive to use, but we have also seen that KnowDiaL enables the robots to understand only simple commands, such as “Please, take this book to the lab.” On the other hand, natural language enables more elaborate specification of requests. The user may ask a robot to perform a set or sequence of tasks, give options to the robot or ask it to perform a task only if certain conditions are met. We view such examples of elaborate natural language as complex commands. In this chapter, to handle these complexities, we introduce a template-based approach that is able to break a complex command into its atomic components and connectors [55].

Due to the complexity of natural language, the approach we introduce is, inevitably, not always able to correctly resolve a complex command in its atomic components. Therefore, we also designed two dialogue systems that allow the user to refine and correct the extracted command structure, guided by the robot. Moreover, when executing complex commands, the robot can make choices and find the optimal sequence of tasks to execute. By rearranging the order in which each atomic task is executed while enforcing the constraints imposed by the structure of the sentence, we can substantially reduce the distance that the robot travels to execute complex commands.

In the rest of this chapter, we first formalize our definition of a complex command, and then we introduce our template-based algorithm to break a complex command into atomic components and connectors. Next, we evaluate our approach on a corpus of 100 complex commands. Then, we show two dialogue models for recovering possible errors introduced by the algorithm. Finally, we present our reordering algorithm that can improve the robot’s execution of complex commands.

4.1 Complex Commands

We have shown how we represent the tasks a robot can execute with semantic frames. For each task of the robot, we defined a separate frame with its own set of frame elements. Semantic frames are also used to represent the commands given to the robot. When the command refers to a single frame and each frame element is uniquely instantiated, we call the command an *atomic command*. “Please robot, go to the lab.” is an example of an atomic command. This command refers to a single frame, *GoTo*, and its only frame element, *Destination*, is instantiated as “the lab”; therefore, we consider this command an atomic command. When a command is not atomic, we call it a *complex command*. We identify four types of complexity that can arise in a command, which we introduce below.

Set of tasks: The user may ask the robot to perform a set of tasks, for which the command refers to multiple frames.

Example 1. “Go to the lab and bring these papers to my office.”

With this command, the user is asking the robot to perform two tasks: *GoTo* and *Deliver*.

Disjunctive task elements: The command might refer to a single frame but some of the frame elements are not univocally instantiated.

Example 2. “Bring me some coffee or tea.”

This command refers to the *Delivery* frame, but the *Object* can be instantiated either as “tea” or “coffee.”

Explicit sequence of tasks: The user may ask the robot to perform an ordered sequence of tasks. Users can refer to a sequence of tasks explicitly in their command.

Example 3. “Go to the lab and then to my office.”

Conditional sequence of tasks: The user may use conditionals to ask the robot to perform an ordered sequence of tasks.

Example 4. “Bring me some coffee if it’s freshly brewed.”

Based on the assumption that we have a frame to represent the action of checking whether coffee is freshly brewed, this command refers to a sequence of two tasks, in which the second might or might not be executed depending on the outcome of the first one.

Our goal is to represent a complex command with a set of atomic commands. To preserve the original meaning of the command, we use four operators to connect the atomic commands extracted from the original command. The operators are **AND**, **OR**, **THEN** and **IF**. Each of these operators corresponds to one of the types of complexity just introduced.

- **AND** is used for commands referring to a set of tasks; therefore the command in Example 1 can be rewritten as the following:

[Go to the lab] **AND** [Bring these papers to my office]

- **OR** is used when an element of the frame can be instantiated in multiple ways. Example 2 can be rewritten as the following:

[Bring me some coffee] **OR** [Bring me some tea]

- **THEN** orders tasks into a sequence. Accordingly, Example 3 can be rewritten as the following:

[Go to the lab] **OR** [Go to my office]

- **IF** is used for the sequence of tasks involving conditionals. Example 4 can be rewritten as the following:

[Coffee is freshly brewed] **IF** [Bring me some coffee]

For the **IF** operator, as shown in the last example, the condition is always moved to the beginning of the sequence, as the robot needs to check it before proceeding.

Finally, sentences can have varying degrees of complexity. In order to measure the complexity of each command, we use the number of atomic commands that the command contains. Accordingly, an atomic command has a complexity level of 1, and all the examples given in this section have a complexity level of 2.

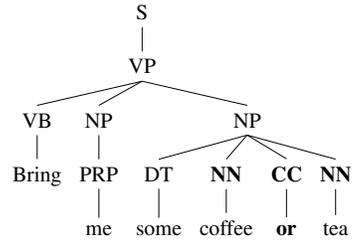
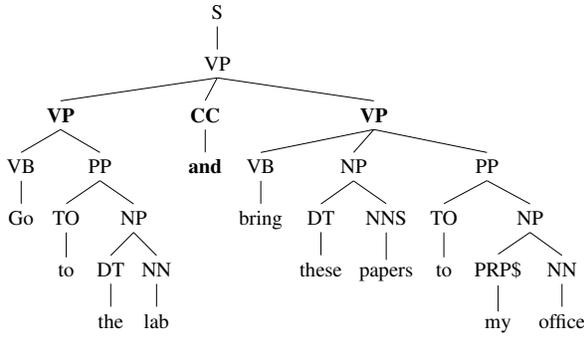
4.2 Detecting Complex Commands

So far, we have identified the types of complexity that a command can present. Next, we present a template-based algorithm to break complex commands into their atomic components, and then we show the results on a corpus of 100 commands.

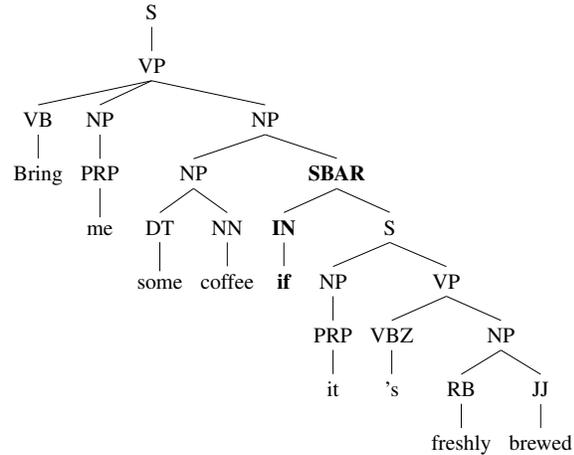
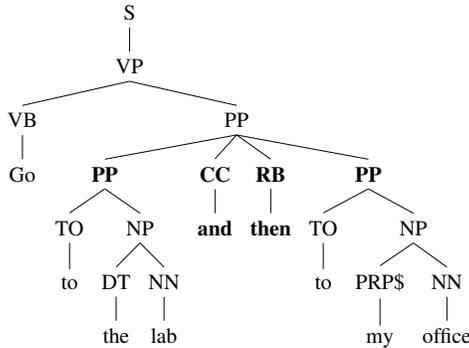
4.2.1 A Template-Based Algorithm

To detect complex commands and break them down into their atomic components, we leverage the syntactic structures of the sentences. We define a *template* as a specific structure in the parse tree of the command. We identify one or more templates for each of the defined operators. Each *template* defines not only the structure associated with a specific operator but also the rules for breaking the complex command into its components. Figures 4.1a, 4.1b, 4.1c and 4.1d show the syntactic parses for the four examples presented in Section 4.1. In each of the parse trees shown, we highlighted in boldface the templates used to break down the complex commands into atomic commands.

Our approach is to, first, parse the received command and then to inspect the parse tree for the defined templates. In the given examples, each complex command is composed of only two atomic commands. However, this is not always the case. Therefore, we break down a command into simpler components and then recursively check each component until we obtain atomic commands. Algorithm 2 details the approach we devised. The `DECOMPOSE` function implements our approach. The function takes a command s as its input and parses it into a tree p similar to the ones shown in Figure 4.1 (line 2). Next, our approach iterates on each node of the parse tree going top-to-bottom, left-to-right (line 3). For each of the nodes in the parse tree, our approach checks if the node matches one of the templates t we defined (line 4). If this is the case, the algorithm applies the rules defined by the template t to break the sentence in simpler components (line 5).



(a) [Go to the lab] **AND** [Bring these papers to my office] (b) [Bring me some coffee] **OR** [Bring me some tea]



(c) [Go to the lab] **THEN** [Go to my office] (d) [Coffee is freshly brewed] **IF** [Bring me some coffee]

Figure 4.1: Example parse trees and the corresponding textual parenthetical representations. The templates used are shown in boldface.

Algorithm 2

```

1: function DECOMPOSE(s)
2:   p = parse(s)
3:   for node in p do
4:     if node == t then
5:       LH,O,RH, = BREAK_SENTENCE(s, t)
6:       L = DECOMPOSE(LH)
7:       R = DECOMPOSE(RH)
8:       return [L O R]
9:     end if
10:  end for
11:  return s
12: end function

```

The `BREAK_SENTENCE` function takes the command s and the matching template t as its inputs, applies the rule specified by the template to decompose the input command, and returns the triplet (LH, O, RH) . LH and RH are two simpler commands, respectively, left-hand and right-hand commands; O is the operator matching the template t . Finally the `DECOMPOSE` function is called recursively on the two simpler extracted commands (line 6 and 7).

We now consider a complex command, such as “If the door is open go to the lab and to my office,” and we look at how the `DECOMPOSE` function operates on it. Figure 4.2 shows the parse tree of the sentence. Once the sentence is parsed, the function starts searching the parse tree. The root node of the tree matches the template for the **IF** operator, and the sentence breaks the sentence into “the door is open” as the LH command and “go to the lab and to my office” as the RH command. Next, the function is called recursively. The LH command is a simple command, so the function returns the sentence as it is. For the RH command the function—after parsing—finds a second template matching the **AND** template. The function breaks down the sentence, and the new LH and RH commands are “go to the lab” and “go to my office,” respectively. These are returned to the initial call of the function, which can now end and returns the following:

[the door is open] **IF** [go to the lab **AND** go to my office]

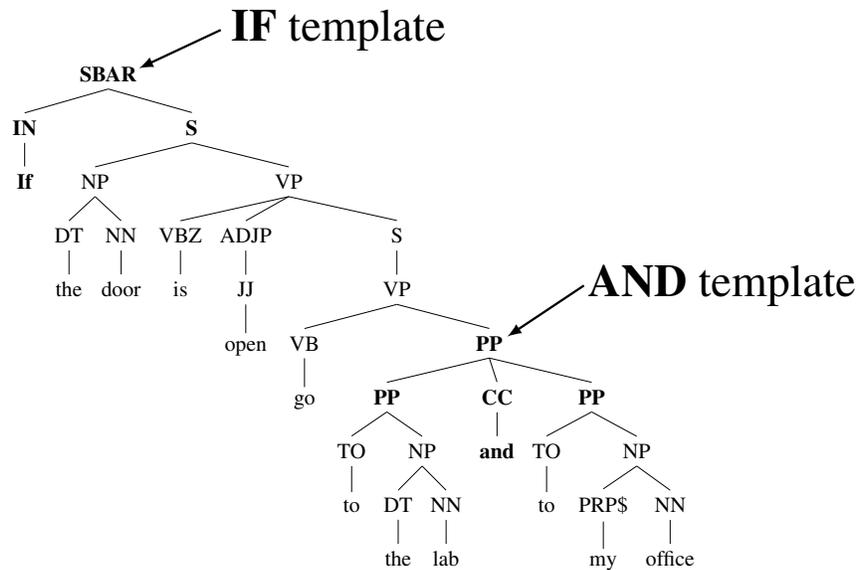


Figure 4.2: Parse tree for the sentence “If the door is open go to the lab and to my office,” with in boldface, the two templates found by the algorithm for the **IF** and **AND** operators.

4.2.2 Experimental Evaluation

We gathered a corpus of 100 complex commands by asking 10 users to each give 10 commands. The users, who were graduate students at our institution, had varied backgrounds, ranging from math and statistics to computer science and robotics. The exact instructions given to the users were the following:

We are asking you a list of 10 commands for CoBot robots. The commands can contain conjunctions (e.g., “Go to the lab and then to Jane’s office”), disjunctions (e.g., “Can you bring me coffee or tea?”) and conditionals (e.g., “If Jane is in her office tell her I’ll be late”). A single sentence can be as complex as you want. For instance, you can have conjunctions inside of a conditional (e.g., “If Jane is in her office and she’s not in a meeting tell her I’m on my way”). Although the sentences can be as complex as you want, we are looking for sentences that you would realistically give to the robot (both in length and content).

Figure 4.3 shows the number of commands for each level of complexity, and Figure 4.4 shows—for each complexity level—one example of the sentences contained in the corpus. The whole corpus we gathered is shown in Appendix A.2. Whereas most of the commands have a complexity level between 1 and 3, people also use more complex instructions and, occasionally, long and convoluted sentences.

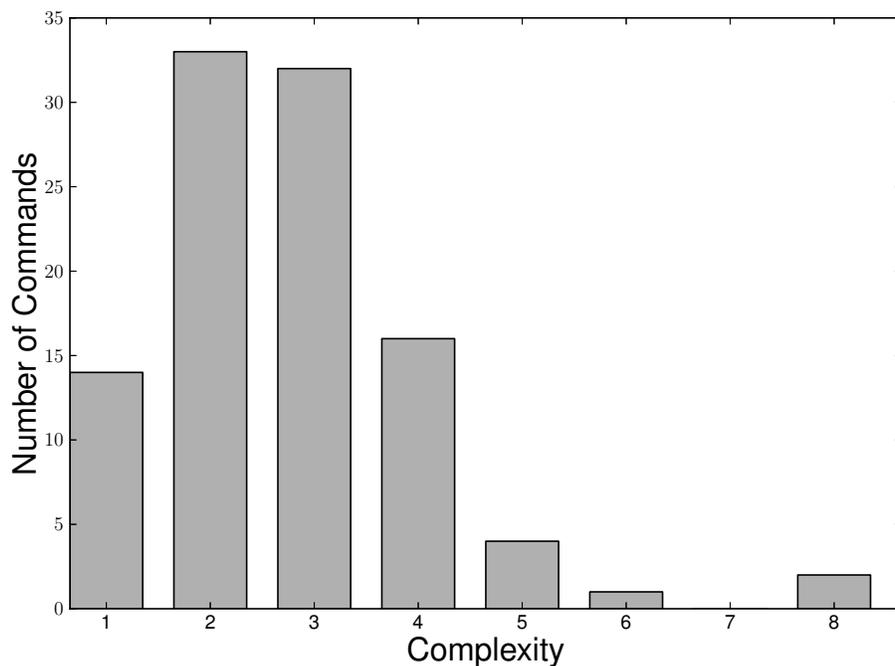


Figure 4.3: Number of commands per complexity level.

To measure the accuracy of our approach, we manually broke each command in its atomic component and compared the extracted structure with the result returned by our algorithm. The overall accuracy of the algorithm was 72%. Figure 4.5 shows the percentage of matching commands for each complexity level.

As expected, our approach does not have any problems with the atomic commands. Out of 14 commands only one was not understood correctly. The only sentence not correctly understood was the following: “The 3rd floor lab has taken our gaffer tape and screw driver. Please bring it to the 7th floor lab.” In this sentence, the algorithm incorrectly recognizes the template for an **AND** operator, even if the user is asking only for one of the tools.

For the commands of complexity levels 2 and 3, the algorithm is able to correctly decompose 76.9% of the complex commands. As the complexity increases, the accuracy decreases, but

1	Bring some coffee to my office please.
2	If you have no tasks scheduled, go explore the 5th floor.
3	Go to the supply room and, if you find a stapler, bring it to me.
4	Please go to the lab and if the door is closed go to John’s office, and ask him to send me the memory stick.
5	I need you to first bring me a cup of tea, or a bottle of water, or a soda, and then go to Chris office and ask her to order more bottled water.
6	If Jane is in her office, ask her when she wants to go to lunch, go to Chris office and tell him her reply, then come back here and tell Jane’s reply to me.
8	If Christina is in her office, pick up a package from her, deliver it to Jane, then go to the lab and say that the package has been delivered. Otherwise, go to the lab and say that the package has not been delivered.

Figure 4.4: Examples of commands in the corpus for each complexity level.

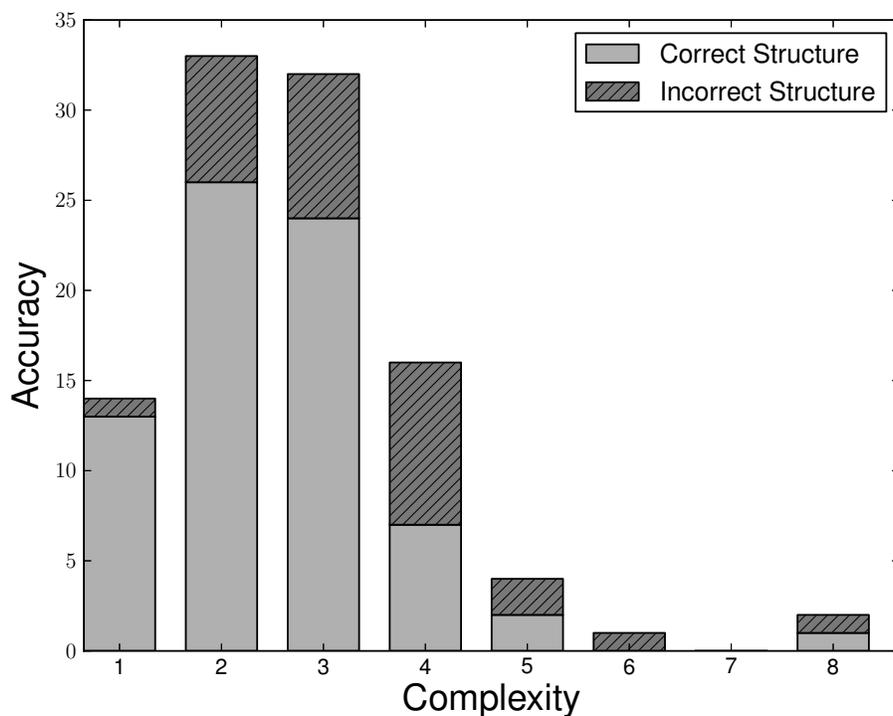


Figure 4.5: Complex commands correctly decomposed for each complexity level.

the corresponding number of commands decreases as well. The main reason that we identify for the more complex commands’ lower accuracy is the lack of appropriate templates. As an example, consider the following sentence: “If the person in Office X is there, could you escort me to his/her office if you’re not too busy?” This sentence can be represented as [[person in office X is there] **AND** [not too busy]] **IF** [escort me to his/her office]. Our templates allow the conditions of an **IF** operator to be at the beginning or at the end of the command but not in both locations. Therefore, our current set of templates was not able to properly break down this type of command, although our templates successfully covered a wide range of other commands.

Another possible source of ambiguity is the `PickUpAndDelivery` task. Users might ask the robot “Go to the lab, pick up my notebook and bring it back here.” If we only look at the syntactic structure of this sentence we might infer that the user is asking the robot to perform multiple tasks (due to the conjunction between two verb phrases) but, in reality, they are asking for a single task. To address this situations we could implement a hierarchy in the templates to define an order in which to apply them. Instead, our approach is to combine the template-based automation with dialogue to recover from possible errors and resolve ambiguity.

4.3 Dialogue

As we have shown in the evaluation section above, after processing our corpus using Algorithm 2, we are still left with a few complex commands that we are not able to understand correctly. Nonetheless, in order for the CoBot robot to execute a complex command correctly, it needs to recover the correct structure representing it. In this section, we introduce two models for a dialogue that allow the robot to correctly recover the structure of a complex command.

4.3.1 A Structure-Based Dialogue

The first dialogue model that we introduce aims at recovering the structure of a complex command. When receiving a command, the robot executes Algorithm 2 and offers the user the extracted structure as its corresponding textual parenthetical representation (see Figures 4.1a-4.1d). If the offered structure correctly represents the command broken into its atomic components, the user can confirm the command, and the robot will start executing it. Otherwise, the robot enters into a dialogue with the user to get the correct structure. Algorithm 3 shows the details of the structure-based dialogue.

First, the algorithm checks whether the command is simple or complex. If the command is a simple command, the robot asks for confirmation of it and then executes it. If the command is a complex command, the robot needs to recover all of the command’s components. In the dialogue, the robot first asks for the operator and then for LH and RH commands. Because both the LH and the RH commands can be complex commands the dialogue will recur for each of them.

Algorithm 3 shows the overall procedure for the structure-based dialogue. The `ASK_COMPLEX` function asks the user whether the command is complex and saves the answer in the Boolean variable `cmp1`. If `cmp1` is false, the robot, using the function `ASK_SIMPLE`, asks for a simple command. If `cmp1` is true, the robot enters the recursive step of the dialogue and asks for a connector and two simpler commands. This dialogue generates, in the worst case, a total of $2n - 1$ questions to recover a command of complexity level n .

4.3.2 A Rephrasing Dialogue

In our second dialogue model, the robot, rather than guiding the user through the steps needed to recover the structure of a complex command, asks the user to rephrase his or her request. We call this model Rephrasing Dialogue.

Algorithm 3 Structure-Based Dialogue

```
1: function ST_DIALOGUE()  
2:   cml = ASK_COMPLEX()  
3:   if cml then:  
4:     o = ASK_OPERATOR()  
5:     rh = ST_DIALOGUE()  
6:     lh = ST_DIALOGUE()  
7:     return [rh, o, lh]  
8:   else  
9:     return ASK_SIMPLE()  
10:  end if  
11: end function
```

Similar to the structure-based dialogue, the rephrasing dialogue asks the user for confirmation of the correctness of the structure extracted, using a textual parenthetical notation. If the user classifies the structure as incorrect, the dialogue algorithm will request that the users rephrase the command, using one of the known templates and giving short examples of the four complex command operators.

Utilizing this rephrasing dialogue approach, we rephrased the sentences that had an incorrect structure. The accuracy improved from 72% to 88%. Figure 4.6 shows the results for each complexity level.

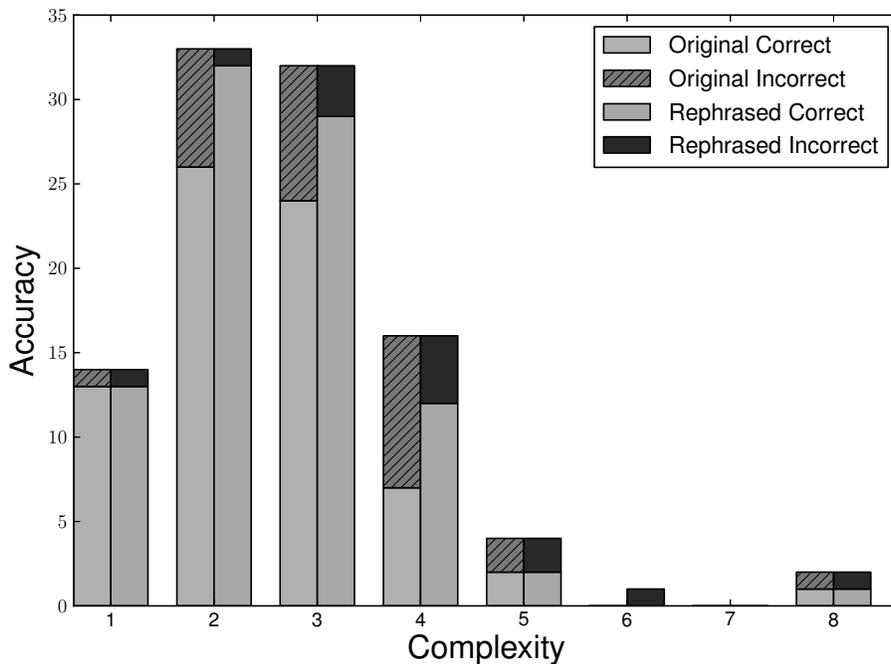


Figure 4.6: Complex commands that were correctly decomposed for the original command and the rephrased one.

As we can see, using a rephrasing dialogue improves the accuracy of the commands. The commands of complexity one do not benefit from the rephrasing dialogue since the only error encountered is not due to a lack of appropriate templates, but rather to the ambiguity of the `PickUpAndDelivery` task. For commands of medium complexity (levels 2 to 4), asking users to rephrase their requests using known templates strongly improves the ability of the robot to recover the correct structure of the commands while, for more complex commands (levels 5 to 8), the rephrasing dialogue does not appear to help. If the rephrasing dialogue is not able to extract the correct structure, a structure-based dialogue can follow it to ensure that all the commands are correctly understood.

4.4 Execution

Before being able to execute a complex command, the robot needs to ground the command. We introduced our grounding model for atomic commands in Chapter 3.1.4. For a complex command, we ground each of its atomic components independently. In the rest of this chapter, we assume that the correct structure representing a complex command has been extracted and that each atomic command has been grounded. Because the structure of the complex command has been recovered and each of its atomic components has been grounded, the next step is to execute the received command. Here, we present and evaluate an algorithm to compute an optimal plan for executing all the atomic commands in a complex command.

4.4.1 A Reordering Algorithm

Once all the atomic commands have been grounded, the robot can start executing them. A naïve approach would be to simply execute each task in the order given in the initial command. We assume that the robot has a measure of the cost of executing each single task. For the CoBot robot, we can use the distance that the robot needs to travel to evaluate its cost. As we showed in Chapter 2.3, one can easily compute this distance by using the robot Navigation Map. Our goal is to find the optimal plan that satisfies the constraints expressed in the original complex command and to minimize the overall execution cost.

The idea is to leverage the structure extracted from a complex command. Each of the four operators we use to describe the structure of a complex command allows for various optimizations or specifies a constraint. For each operator, we generate these sequences of commands:

- **AND** operators originally refer to a set of commands. Therefore, we generate a sequence for each permutation of the connected commands.
- **OR** operators give multiple options to the robot. Accordingly, we generate a sequence for each connected command, and each sequence contains only one of the commands.
- **THEN** operators constrain in the order in which tasks should be executed.
- **IF** operators, similar to **THEN** operators, express a constraint, but the LH part of the command is executed only if the conditions expressed in the RH part are met.

The algorithm that we present generates all the possible sequences of tasks that satisfy the request expressed by the complex command, evaluates the cost of each of sequence and executes the optimal sequence. The algorithm takes a command C as its input and starts with an empty list of task sequences. If C is an atomic command, the corresponding task is returned. Otherwise, the algorithm considers the RH and LH sides separately, generates all the possible sequences for both of them, combines them according to the operator O , and adds them to the list of possible sequences. Algorithm 4 shows our approach.

Algorithm 4

```

1: function CREATE_SEQUENCE(C)
2:   if IS_ATOMIC(C) then:
3:     return C
4:   else
5:     LH, O, RH  $\leftarrow$  C
6:     L = CREATE_SEQUENCE(LH)
7:     R = CREATE_SEQUENCE(RH)
8:     result = []
9:     for all l in L do
10:      for all r in R do
11:        result.append(COMBINE(l, r, O))
12:      end for
13:    end for
14:    return result
15:   end if
16: end function

```

4.4.2 Experimental Evaluation

To test our reordering algorithm, we generated three sets of random commands. The commands generated to evaluate our Reordering Algorithm are already grounded and are not English sentences. The first set of commands contains only **AND** operators, the second contains only **OR** operators, and the third contains **IF** or **THEN** operators as well as, for more complex commands, **AND** and **OR** operators. The first two sets are composed of commands of increasing complexity, with their number of atomic tasks ranging from one to five, whereas the third set contains commands whose complexity levels range from 2 to 10. Each set contains 50 commands, 10 for each complexity level.

Our approach is compared to a naïve base-line that—for **AND**, **THEN**, and **IF**—executes the tasks in the order given and that, for the **OR** operator, randomly picks one of the alternatives. To measure the cost of each sequence of tasks, we used the travel distance of the robot. We measured the improvement of our algorithm over the baseline as the ratio of the two distances. In measuring the cost of the execution, we assumed, for both the baseline and our approach, that the conditions of the commands with **IF** operators are always met.

Figure 4.7a shows the result for the **AND** set. As is expected for commands with level 1 complexity, the baseline and our approach achieve the same result. As the complexity increases, our reordering algorithm consistently improves, and for a command with level 5 complexity, we get a travel distance 1.68 times shorter than that of the baseline.

Figure 4.7b shows the result for the **OR** set. Again, for commands with level 1 complexity, the baseline and the reordering algorithm have the same result. As the complexity level increases, our approach starts improving compared to the baseline. The improvement is non-monotonically increasing due to the nature of the baseline. Because the baseline chooses the task to execute randomly, the improvement cannot be constant.

Finally, Figure 4.7c shows the result for complex commands containing all four operators. For this set, we start with commands with level 2 complexity (that is, a sequence of two tasks). Our approach consistently improves compared to the baseline.

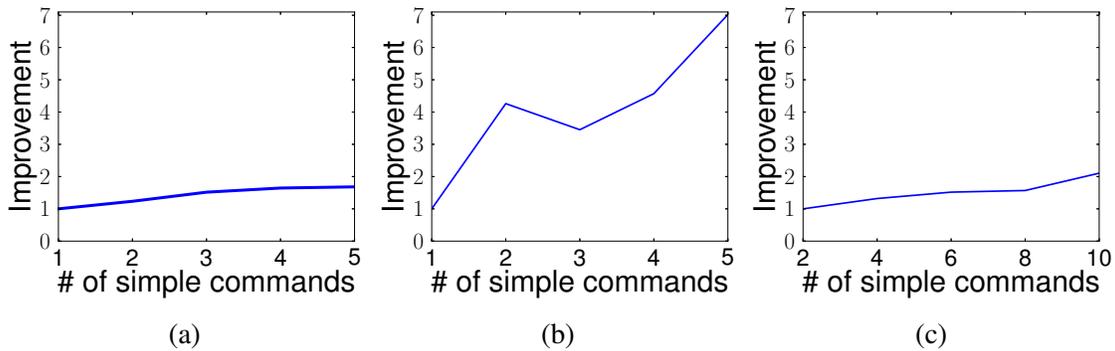


Figure 4.7: Comparison between the results of reordering the commands and executing them in the given order, measured as the ratio of the traveled distances. Commands include: (a) only **AND** operators, (b) only **OR** operators and (c) any of the operators.

4.5 Summary

In this chapter, we presented a novel approach to understanding and executing complex commands for service robot task requests. Our approach breaks down complex commands into their atomic components. To present our approach, we first identified four types of complexity. Next, we designed a template-based algorithm that is able to break down a complex command into its atomic components. The experiments show that the algorithm is able to correctly reduce a complex command 72% of the time. To further recover the correct structure of a complex command, we introduced two dialogue approaches. Finally, we presented a reordering algorithm that is able to find the optimal plan for executing a complex command that shows substantial improvement over a naïve baseline.

Chapter 5

Learning of Groundings from Users' Questions to Log Primitives Operations

Human:“*On average, how long do GoTo tasks take?*”

Robot:“I performed a total of 5 tasks matching your request, their average length is 105 seconds. The longest task took 218 seconds while the shortest took 4 seconds. The total time spent executing tasks matching your request is 525 seconds.

The capabilities of service robots have been steadily increasing. Our CoBot robots have traveled autonomously for more than 1000 kilometers [8], the Keija robot was deployed in a mall as a robotic guide [19], and the STRANDS project has developed robots aimed for long-term deployment in everyday environments [36]. On the other hand, it is still unclear how robots will fit into people's everyday lives and how the interaction between users and robots is going to shape up. One crucial aspect is related to the issue of trust between users and robots.

In our deployment of the CoBot robots, we observed how the internal state of the robot is often hidden to the users [5]. When the CoBot robots are executing a task, bystanders cannot really tell where the robots are going or what task they are executing. Ideally, users should be able to ask the robot what it is doing, why a particular choice was made or why a particular action was taken. In this chapter, we take steps in this direction by enabling users to ask questions about the past autonomous experiences of the robot. For the CoBot robots, we are going to focus on questions about the time and distance traveled during task execution, but we believe that the approach introduced is general.

Our first contribution is a novel use of *log files*. Typically, when available, developers use these files for debugging purposes. In this chapter, we use the logs recorded by our robots as the source of information. The CoBot robots can search the logs to autonomously answer when their users ask questions using natural language. For the robot to automatically retrieve information from the logs files, we define Log Primitives Operations (LPOs)[56, 57]. Using LPOs, we extend the ability of our robots to not only execute tasks, but also perform operation on the log files that they record.

Our second contribution is that we frame the problem of question understanding as grounding input sentences into LPOs. Similar to what we have done in Chapter 3, we define a joint probabilistic model over LPOs, the parse of a sentence, and a learned Knowledge Base. Once again, the Knowledge Base is designed to store and reuse mappings from natural language expressions to queries that the robot can perform on the logs. To evaluate our approach to understanding questions, we crowd-sourced a corpus of 133 sentences. Our results show that, using our approach, the robot is able to learn the meaning of the questions asked.

Finally, our third contribution is that we introduce the concept of *checkable answers*. To provide the user with meaningful answers, we introduce checkable answers whose veracity can quickly be verified by the users.

The rest of this chapter is organized as follows. First, we review the structure of the log files—initially introduced in Section 2.5—recorded by the CoBot robots. Then, we present the structure of the LPOs that our robot can autonomously perform on its logs. Next, we introduce our model for query understanding and present our experimental results. Finally we introduce the concept of *checkable answers* with comprehensive examples.

5.1 Robot Logs

Many systems come with logging capabilities. These capabilities are designed to allow developers to find and fix unwanted behaviors (i.e., bugs). In general, for a mobile service robot, a log file might include sensory data (e.g., readings from cameras or depth sensors), navigation information (e.g., odometry readings or estimated positions), scheduling information (e.g., the action being performed or the time until a deadline) or all of the above. Although these log files were initially developed for debugging purposes, we introduce a novel use for the log files recorded by a robot. The robots use the recorded logs as their memory and use the information stored in the logs to answer questions about their past experiences.

As we have said, the CoBot robots are developed using ROS [63], and their code is designed in a modular fashion; each module can publish messages on a topic or subscribe to it to receive messages. Our logging system, native to ROS, records every message exchanged by the running modules and saves them in a log file. When the robot is running, messages are exchanged over more than 50 topics at 60 hertz. The information exchanged on these topics ranges from low-level micro-controller information, to scheduling data, to encoder readings, to GUI event information.

A detailed description of the messages and information recorded in our log file is presented in [8]. Here, we recall that we can categorize the messages recorded at three levels: the **Execution Level** (this is the lowest level which contains information about the physical state of the robot), the **Task Level** (this level records all the information regarding the tasks that the robot can execute), and the **Human-Robot Interaction Level** (in which messages record information related to the interactions with humans).

Because our goal is to enable a robot to answer questions about the duration of the task and distance traveled while executing tasks, we are going to focus on messages in the execution and task levels. Specifically, we focus on two topics. The first one is `/Localization`, and it belongs to the execution level. This topic records the position (x, y, z, Θ) of the robot (z indicates the floor of the building on which the robot is currently located). Using the published

messages on this topic, we reconstruct the duration of the travel, the distance that the robot travels while executing a specific task and the path that robot took. The second topic we consider is `/TaskPlannerStatus`. This topic records information about the task being performed including: the *semantic frame* representing the task, the duration of the task execution, and the expected remaining time for the current task.

Finally, although it is not crucial to our contribution, it is worth noticing that log files recorded by the robot are sequential by nature. To quickly search through the logs and answer the users' questions, we use an intermediate representation in which each task and its relevant information (task type, task arguments, duration of travel and distance traveled) are indexed by the task's starting time.

5.2 Log Primitive Operations

To answer questions, our robots need to retrieve the relevant information from the logs. To accomplish this, we designed LPOs that the robot can autonomously perform on log files. An LPO comprises an *operation* and a set of *filters*. The *operation* defines the computation that the robot performs on the records, which are selected from the logs using the *filters*. Each record in the log files contains several fields (e.g., the position of the robot or the task being executed). Here, we define four quantitative operations that the robot can perform on the logs. A quantitative operation operates on one or more numerical fields of the records being considered. The operations that we define are the following.

MAX returns the largest value for the field being considered in the record specified by the filters.

MIN returns the smallest value for the field being considered in the record specified by the filters.

AVG returns the average value for the field being considered in the record specified by the filters.

SUM returns the total value for the field being considered in the record specified by the filters.

We also defined three additional non-quantitative operations. These operations are performed on the record(s) matching the LPO's filter and do not need to operate on any numerical field. The operations are the following.

CHECK returns "true" if the logs have at least one record matching all the filters specified by the query; otherwise, it returns "false."

COUNT returns the number of records matching the filters specified in the LPO.

SELECT returns all the records matching the filters specified in the LPO.

Filters are used to select the record(s) relevant to the query. We identify three types of filters. First, we define *task-based* filters. A user might ask about the time spent executing a specific type of task or about the distance traveled while going to a specific location. We allow for five task-related filters: *taskID*, *destination*, *source*, *object* and *person*. These five filters match the argument of the structure of the semantic frames that we use to represent the tasks that the CoBot robots can execute.

LPOs performed on the logs refer to the past experiences of the robot; therefore, we define

a second type of filter to select the window of time relevant to the LPO. We define this type of filter as a *time-based* filter. A time-based filter is typically characterized by a starting and ending time.

Finally, the third type of filter, a *quantity-based* filter, is used to select which field should be considered when applying the quantitative operations. We will focus on understanding questions about the duration of travel and the distance that service robots travel during their deployment; hence, the quantity-based filter is used to specify whether a question refers to time or distance.

In the next section, we show how we map users’ questions to the LPOs that the robot can autonomously perform on the logs. Figure 5.1 shows, for each of the quantitative operations that we defined, an example of an input sentence and the corresponding LPO that we aim to extract.

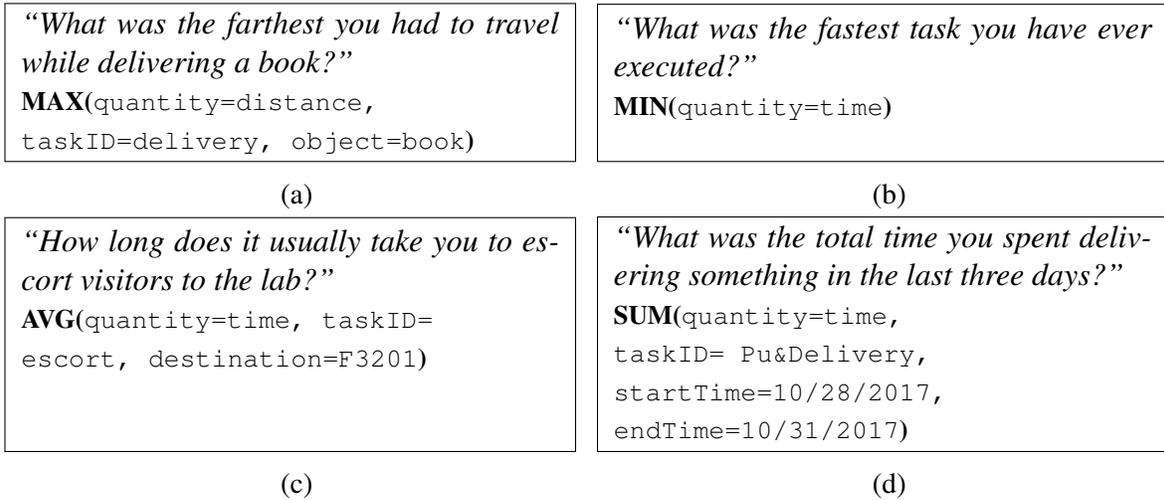


Figure 5.1: Examples of input sentences and the corresponding queries to be extracted. Each sentence implies a different set of filters to be used.

5.3 Question Understanding

In the previous section, we introduced the LPOs that the robot can autonomously perform on its log files. To enable a mobile robot to answer questions about the duration of its travel and the distance it has traveled, we frame the problem of understanding an input sentence as finding the best matching LPO to perform on the log files. This approach closely follows the one we introduced in Chapter 3 for understanding spoken commands. Formally we define a joint probabilistic model over the parse of a sentence (Ψ), the possible LPOs (\mathcal{L}) and a Knowledge Base (\mathcal{KB}). We aim to find the LPO, \mathcal{L}^* , that maximizes the joint probability, which is:

$$\arg \max_{\mathcal{L}} p(\mathcal{L}, \Psi | \mathcal{KB}) \quad (5.1)$$

Assuming that the parser is conditionally independent from the Knowledge Base, we can rewrite our joint model as the following:

$$p(\mathcal{L}, \Psi | \mathcal{KB}) = p(\mathcal{L} | \mathcal{KB}) p(\Psi) \quad (5.2)$$

We refer to the two factors of this model as the *Grounding Model* and the *Parsing Model*, and we detail them in the next two sections.

5.3.1 Parsing Model

To parse questions from users, we adopt a shallow semantic parser. Each word is first labeled using one of the following labels: *Operation*, *Quantity*, *TaskID*, *Destination*, *Source*, *Object*, *Person* or *Time*. We denote this set of labels as L . These eight labels match the structure of an LPO that the robot can perform on the logs and allow us to retrieve the part of the sentence that refers to the operation or one of the filters. A special label *None* is used to label words that can be disregarded. Once each word in a sentence has been labeled, we group contiguous words in the same label. Figure 5.2 shows an example of a parsed sentence.

“What was the [shortest]_{Operation} [time]_{Quantity} it took you to complete an [escort]_{TaskID} task in the [last three days]_{Time}?”

Figure 5.2: An example of a parsed sentence. Each word that is not between square brackets was labeled as *None*.

We model the parsing problem as a function of pre-learned weights w and observed features ϕ . Given a sentence S of length N , to obtain a parse, Ψ we need to label each word s_i as l_i , where $l_i \in L$. Formally we want to compute the following:

$$\begin{aligned}
 p(\Psi) &= p(l_1, \dots, l_N | s_1, \dots, s_N) \\
 &= \frac{1}{Z} \exp\left(\sum_1^N w \cdot \phi(l_i, s_{i-1}, s_i, s_{i+1})\right)
 \end{aligned}
 \tag{5.3}$$

where Z is a normalization constant to ensure that the distribution $p(\Psi)$ sums to 1. We obtained this model using a CRF where ϕ is a function producing binary features based on the part-of-speech tags of the current, next, and previous words, as well as the current, next, and previous words themselves.

5.3.2 Grounding Model

Using the parsing model, we are able to extract from a sentence the structure of the LPO that the robot needs to perform on the logs. The semantic parser identifies, for each chunk of the sentence, whether the chunk refers to the operation to be performed or one of the filters or whether we can disregard it. Users can refer to the same operation in multiple ways. As an example, consider a user asking the robot to perform an AVG operation. The user might ask “What is the usual time?” or “What is the typical time?” Therefore, to understand a sentence fully, we need to map words to symbols that the robot can process; that is, we need to ground the sentence.

The possible groundings for the *Operation* label are the four operations that we defined for the logs. For our robot, the *Quantity* label can be grounded to either time or space. The *Destination* and *Source* labels are grounded to office number in the building. The *Object* and *Person* labels

do not require grounding, as we can directly search the logs for matching strings. Finally, we need to ground the chunks labeled *Time* to an explicit start and end date.

To save and reuse the mapping from natural language expression to groundings, we designed a Knowledge Base. Our Knowledge Base is a collection of binary predicates in which the first argument is the natural language expression and the second is its grounding. We use four predicates that closely match the label used by our semantic parser: *operationGrounding*, *quantityGrounding*, *taskGrounding*, and *locationGrounding*. Figure 5.3 shows an example of the knowledge base.

OperationGrounding("farthest", MAX)
OperationGrounding("longest", MAX)
TaskGrounding("delivering", Pu&Deliver)
QuantityGrounding("far", SPACE)
LocationGrounding("Manuela's office", F8002)

Figure 5.3: An example of the Knowledge Base and the predicates it stores.

To each predicate in the Knowledge Base we attach a confidence score measuring how often a natural language expression e has been grounded to a specific grounding γ ; we use $C_{e,\gamma}$ to refer to the confidence score of a specific predicate.

We use the confidence score attached to each predicate in the Knowledge Base to compute $p(\mathcal{L}|\mathcal{KB})$. As we have shown, an LPO \mathcal{L} is composed of an operation O and a set of filters f . Therefore, we approximate the probability of a query as the following:

$$p(\mathcal{L}|\mathcal{KB}) = p(O|\mathcal{KB}) \prod_{i=1}^i p(f_i|\mathcal{KB})$$

Each of the terms in this product can be computed directly from the confidence scores stored in the Knowledge Base. To compute the probability of a specific grounding γ^* , whether this is for the operation or for one of the filters, we use the following formula:

$$p(\gamma^*|\mathcal{KB}) = \frac{C_{\gamma^*,e}}{\sum_{\gamma} C_{\gamma,e}}$$

When our robot receives a question, it first parses it to extract the structure of the LPO. Next, for each of the chunks extracted, it searches the Knowledge Base for matching predicates and computes the most likely grounding. When a natural language expression e cannot be grounded using the Knowledge Base, the robot enters a dialogue, asks the user to explicitly provide the grounding and then updates its Knowledge Base.

Finally, to ground the expressions referring to time-related filters, we use SUTime [16], an external library that recognizes and normalizes time expressions. Our Knowledge Base is able to learn static mapping from natural language expressions to groundings, but time expressions often require to be functionally grounded; that is, we also need to take into account the current time. As an example, consider the expression "in the last three days;" we cannot ground this expression to fixed start and end dates, as they continuously change.

5.4 Experimental Evaluation

To evaluate our approach, we crowd-sourced a corpus of 140 sentences from 20 different users through an Amazon Mechanical Turk¹ survey (see Appendix A.3 for the full corpus). We asked each user to provide questions asking robots about the time it spent and distance it traveled while executing tasks. First, we introduced the users to the robots’ capabilities (i.e., the tasks they perform, and the arguments of each task). Next, we illustrated the type of information the robots record in their log files, using a tabular format. Finally, we instructed the users to provide questions the robots could answer based only on the information presented in the logs. Figure 5.4a shows the instructions provided to the users and the tabular example of the information in the logs.

When filling the survey, we presented the user with a randomly-created table showing information from the logs, and asked them to provide a question requesting the robot to perform one of the operation we defined for LPOs; the page the users had to fill is shown in Figure 5.4b. This process was repeated 7 times, one for each of the LPO’s operation.

Task	Destination	Source	Object/Person	Distance	Start Time	Total Time
Escort	GHC7412	N/A	Ashwin	143 meters	Tuesday, August 14, 2018, 07:01	100.1 seconds
GoTo	GHC8002	N/A	N/A	223 meters	Tuesday, August 14, 2018, 12:47	156.1 seconds
Deliver	GHC2109	GHC2107	coffee	456 meters	Thursday, August 16, 2018, 05:02	319.2 seconds
Deliver	GHC7412	GHC7703	book	167 meters	Thursday, August 16, 2018, 09:08	116.9 seconds

While the robot is autonomously moving, users do not follow the robot, so they don't know about the robot experience history. However such knowledge is recorded in the logs. This survey aims at understanding how users would ask questions *using natural language* about the robot experience, such as tasks, time, locations, duration, etc. as recorded.

You will be asked to provide questions about the robot experience. The robot should be able to answer the questions based only on the information in the log tables (e.g., you can ask how long a task took but not how many people the robot saw while executing the task, as such information is not considered in the logs for this survey.)

(a)

Task	Destination	Source	Object/Person	Distance	Start Time	Total Time
Deliver	GHC7004	GHC7511	mail	203 meters	Thursday, August 16, 2018, 04:30	142.1 seconds
Deliver	GHC7412	GHC7511	coffee	239 meters	Saturday, August 18, 2018, 04:02	167.3 seconds
Deliver	GHC3201	GHC7511	mail	422 meters	Saturday, August 18, 2018, 05:24	295.4 seconds
Deliver	GHC8002	GHC7511	candies	69 meters	Saturday, August 18, 2018, 08:24	48.3 seconds

Provide a question asking to check if the robot has executed any of tasks shown in the table

Next

(b)

Figure 5.4: Survey website used to crowd-source the LPO corpus. (a) The instructions provided to the users. (b) The page requesting users to provide a question to the robot.

Out of the 140 sentences that the users provided, we had to discard 7 because they were either non-grammatical or could not be matched to any of the operations we defined. Therefore, in our experiments, we use 133 sentences.

We hand-label each sentence twice. First, we label each word in the sentence with labels for the parsing model. Second, we label each sentence with the corresponding LPO to be performed on the logs. After training our CRF, we evaluate the accuracy of the semantic parser using leave-one-out cross validation. We iterate on our corpus by leaving out each of the 20 users that took

¹www.mturk.com

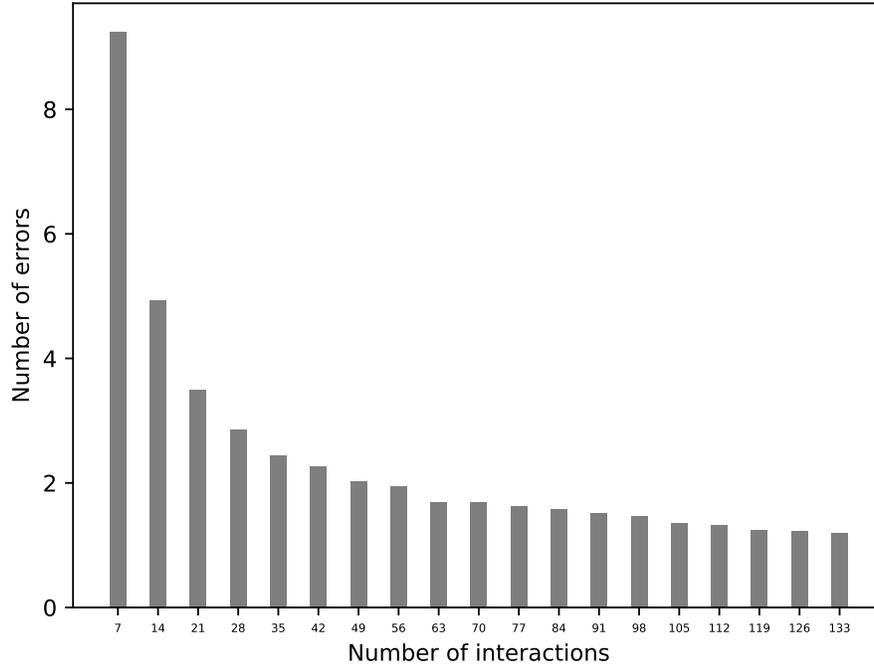


Figure 5.5: The number of errors made by the robot while grounding questions.

part in the crowd-sourcing survey; the parser achieves an average F_1 score of 0.84.

To evaluate the grounding model, we first look at the error our robot makes in grounding input sentences. We consider an error as having occurred both when 1) the robot cannot infer the grounding using the knowledge base and has to enter a dialogue and 2) the grounding inferred does not match our annotation. In Figure 5.5, each bin represents seven interactions: that is, seven sentences that the robot received and grounded. We start with an empty knowledge base; therefore, the robot is initially asking for the grounding of each chunk identified by the parser. As the Knowledge Base grows, the number of errors made quickly decreases. By the end of our experiment, we can observe that the robot makes fewer than two errors; that is, it is able to understand 5 out of 7 sentences without needing to ask questions or making any mistake. It is worth noticing that, for this kind of experiment, the order in which the sentences are processed can have a big impact. To smooth out the possible effect of specific sequences of sentences, we computed the results presented in Figure 5.5 as the average of 1000 randomized runs.

We also analyze the size of the Knowledge Base as the robot process more and more sentences. Figure 5.6 shows the number of facts (i.e., different predicates) stored in the Knowledge Base. We smooth out the possible effect of specific sequences of sentences by plotting the average number of facts stored in the Knowledge Base over 1000 randomized runs. Initially, we expected this plot to flatten out after the first few interactions. Instead, we observe that during the first interactions, facts are quickly added to the Knowledge Base (i.e., the plot shows a high slope); as time progresses, the rate at which facts are added decreases but never goes stops. By inspecting the Knowledge Base after the experiment, we observed that few predicates have a very high count and that many of the remaining predicates were only used once. This behavior

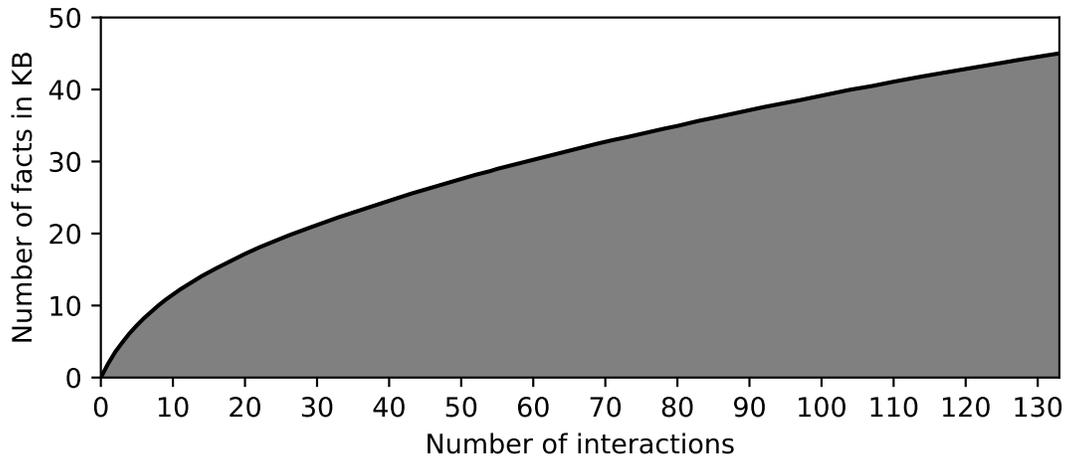


Figure 5.6: The number of facts stored in the Knowledge Base after each interaction.

mirrors the long-tail distribution typically found in language models, in which a few expressions are used very commonly and are quickly learned; however, from time to time, we still encounter new ways to refer to operations or filters.

5.5 Checkable Answers

So far, we have discussed how to enable a robot to understand questions. A user might ask “How much time did your shortest task take?” The robot parses and grounds the sentence, searches in its log files, and comes up with the answer “42 seconds.” Although simply reporting the time (or distance) answers the question, it hardly provides enough information for the user to verify whether the robot is truthful or not. Therefore, we introduce *Checkable Answers* which allow users to use knowledge of the environment or simple math to check the answers that the robots provide.

Checkable Answer: We define a Checkable Answer for quantitative operations as an answer that provides both the final value resulting from the operation and a procedure for verifying the consistency of such an answer.

For each of the four quantitative operations, we design a template that the robot can use to generate a Checkable Answer. These templates use information that the robot can extract from the log files by using a different operations or the map that it uses for navigation. The templates that we designed are the following.

MAX/MIN: The robot reports on each segment of the path taken while executing the task matching the filters in the query. We extract the path taken by using the messages on the `/Localization` topic. We segment the path every time the robot turns and refer to each turning point through the closest office to it. Figure 5.7 shows, in green, the path taken by the robot for the following checkable answer:

Q: *What was the shortest task you ever executed?*

A: *I executed my shortest tasks on the 2nd of April 2013; it took me 42 seconds. I started*



Figure 5.7: The path taken by the robot where we identify two segments.

from office O713 at 1:41:01pm, passed by office O711 at 1:41:16, and I arrived at office O706 at 1:41:43pm

AVG: The robot reports on the total number of tasks matching the filters in the query, their average, the longest and shortest task (computed using the MIN and MAX operators) and the total time/distance traveled (computed using the SUM operator).

Q: *On average, how long do GoTo tasks take?*

A: *I performed a total of 5 tasks matching your request, their average length is 105 seconds. The longest task took 218 seconds while the shortest took 4 seconds. The total time spent executing tasks matching your request is 525 seconds.*

SUM: The robot reports on the total number of tasks matching the filters in the query, the longest and shortest task (computed using the MIN and MAX operators) and the average time/distance traveled (computed using the AVG operator).

Q: *What is the total time you spent escorting someone?*

A: *I performed a total of 14 tasks matching your request. The total time was 2163 seconds. The longest task took 407 seconds, while the shortest took 4 seconds. The average time spent executing tasks matching your request is 145.50 seconds.*

5.6 Summary

In this chapter, with the goal of making service robots more transparent to their users, we enabled them to answer questions about the duration of the task and distance traveled during the task. To accomplish this goal, we first contribute a novel use of the log files that our robots record. Rather than limiting the logs to a debugging tool, we devise them as the memory of the robots, which

allows the robots to answer questions.

The second contribution of the chapter is its probabilistic approach to understanding questions. We introduce the LPOs, defined in term of operations and filters, which our robots can autonomously perform on the log files. We define a probabilistic model over the parse of questions, possible LPOs and a Knowledge Base to enable the CoBot robots to map questions to LPOs. Next, we evaluate our approach on a corpus of 133 sentences, showing how the robots are quickly able to learn the meanings of the questions asked.

Finally, the last contribution of this chapter is our use of *checkable answers*, in which answers to user questions are provided in terms of multiple log operations. This provides additional context and lets the user quickly verify the answer received.

Chapter 6

Mapping Users Questions to Verbalization Levels of Detail

Robot:“I started from office 7717. I went by office 7416 and took 28 seconds. I went through corridor 7400 and took 42 seconds. I went by the open area 730 and took 28 seconds. I reached office 7501.”

The CoBot robots have autonomously navigated than 1000 kilometers [82]. Due to the success of the autonomous algorithms, they consistently move in our environments and persistently perform tasks for us without any supervision. With robots performing more autonomous behaviors without human intervention, we do not know much about their paths and experiences when they arrive at their destinations without delving into their extensive log files. In the previous chapter, we have shown how we can enable a robot to answer questions about its past experiences. In this chapter, we introduce a new challenge: how to have robots respond to queries, in natural language, about their autonomous choices, including their routes taken and experienced. We are interested in ways for robots to *verbalize* (an analogy to visualization) their experiences via natural language.

We notice that different people in the environment may be interested in different specific information, for specific parts of the robot’s experience, at different levels of detail, and at different times. A one-size-fits-all verbalization, therefore, will not satisfy all users. For example, as robotics researchers interested in debugging our robots’ behaviors, we often would like our robot to recount its entire path in detail. On the other hand, an office worker may only want a robot to identify why it arrived late. These variances in preferences are echoed in prior literature in which autonomous systems explain their behavior [10, 23, 77].

In prior work [65], the *verbalization space* has been introduced to capture the fact that descriptions of the robot experience are not unique and can vary greatly in a space of various dimensions. The Verbalization Space is characterized by three dimensions: abstraction, specificity and locality. Each dimension has different levels associated with it. The verbalization algorithm introduced in [65] leverages the underlying geometric map of an environment a robot uses for route planning and semantic map annotations to generate several explanations as a function of

the desired preference within the verbalization space.

In this chapter, we first present a summary of this prior work, including an example verbalization for the CoBot robots in the Gates-Hillman Center. Then, we address the fact that people will want to request diverse types of verbalizations and, as the robot verbalizes its route experiences, they may want to revise their requests through dialogue. We present a crowd-sourced on-line study in which participants were told to request types of information represented in our verbalization space. We then provide the robot’s verbalization response, asking the participants to write a new request to change the type of information in the presented verbalization. Using the verbalization requests collected from the study, we learn a mapping from the participant-defined language to the parameters of the verbalization space. We show that the accuracy of the learned language model increases with the number of participants in our study, indicating that, although the vocabulary was diverse, it also converged to a manageable set of keywords with a reasonable participant sample size (100 participants). Finally, we demonstrate human-robot dialogue that is enabled by our verbalization algorithm and by our learned verbalization space language classifier [59].

6.1 Route Verbalization

Verbalization is defined as the process by which an autonomous robot converts its experience into language. The variations in possible explanations for the same robot experience are represented in the *verbalization space* (VS). Each region in the verbalization space represents a different way to generate explanations to describe a robot’s experience by providing different information, as preferred by the user. Specifically, given an annotated map of the environment, a route plan through the environment and a point in our verbalization space, the Variable Verbalization Algorithm [65] generates a set of sentences describing the robot’s experience following the route plan. We summarize each of these aspects in turn and then provide example verbalizations for the CoBot robots.

6.1.1 Environment Map and Route Plans

As we have shown in Section 2.3, the CoBot robots maintain an environment map with semantic annotations representing a high-level landmark of interest. We define the map $M = \langle P, E \rangle$ as a set of points $p = (x, y, m) \in P$ representing unique (x, y) locations for each floor map m , and the edges $e = \langle p_1, p_2, d, t \rangle \in E$ that connect two points p_1, p_2 taking time t to traverse distance d .

The map is annotated with semantic *landmarks* represented as room numbers (e.g., 7412, 3201) and room type (office, kitchen, bathroom, elevator, stairs, other). The map is also annotated with a list of points as *corridors*, which typically contain offices (e.g., “7400 corridor” contains (office 7401, office 7402, ...)), and *bridges* as hallways between offices (e.g., “7th floor bridge” contains (other 71, other 72, ...)).

Using this map, a route planner produces route plans as trajectories through our map. The route plan is composed of a starting point S , a finishing point F , an ordered list of intermediate waypoints $W \subset P$ and a subset of edges in E that connect S to F through W . Our route planner

annotates route plans with *turning points* (e.g., [6]) to indicate the locations where the robot turns after moving straight for some time.

6.1.2 Verbalization Space Components

For any given route plan, many different verbalization summaries can be generated. The space of possible verbalizations is formalized as the *verbalization space* consisting of a set of axes or parameters along which the variability in the explanations is created. For the purpose of describing the path of the CoBot, our VS contains three orthogonal parameters with respect to the environment map and route plan: abstraction, locality, and specificity. These parameters are well documented in the research, though they are not exhaustive ([10, 23, 77]).

Abstraction A : Our abstraction parameter represents the vocabulary or corpus used in the text generation. In the most concrete form (Level 1), we generate explanations in terms of the robot’s world representation, directly using points (x, y, m) in the path. Our Level 2 derives angles, traversal time and distances from the points used in Level 1. Level 3 abstracts the angles and distances into right/left turns and straight segments. Finally, at the highest level of abstraction, Level 4 contains location information in terms of landmarks, corridors and bridges from our annotated map.

Locality L : Locality describes the segment(s) of the route plan in which the user is interested. In the most general case, users are interested in the plan through the entire Global Environment. They may only be interested in a particular Region defined as a subset of points in our map (e.g., the 8th floor or Building 2) or only interested in the details around a Location (e.g., 8th floor kitchen or office 4002).

Specificity S : Specificity indicates the number of concepts or details to discuss in the text. We reason about three levels of specificity: the General Picture, the Summary, and the Detailed Narrative. The General Picture contains a short description, only specifying the start and end points or landmarks, the total distance covered and the time taken. The Summary contains more information regarding the path than General Picture does, and the Detailed Narrative contains a complete description of the route plan in the desired locality, including a sentence between every pair of turning points in the route plan.

6.1.3 Variable Verbalization Algorithm

Given the route plan, the verbalization preference in terms of (A, L, S) and the environment map, the Variable Verbalization (VV) algorithm translates the robot’s route plan into plain English (pseudocode in Algorithm 5). We demonstrate the algorithm with an example CoBot route plan from starting point “office 3201” to finishing point “office 7416” as shown in Figure 6.1. In this example, the user preference is (Level 4, Global Environment, Detailed Narrative).

Algorithm 5 Variable Verbalization Algorithm

Input: $path, verb_pref, map$ **Output:** $narrative$

```
//The verbalization space preferences
1:  $(a, l, s) \leftarrow verb\_pref$ 
   //Choose which abstraction vocabulary to use
2:  $corpus \leftarrow \text{ChooseAbstractionCorpus}(a)$ 
   //Annotate the path with relevant map landmarks
3:  $annotated\_path \leftarrow \text{AnnotatePath}(path, map, a)$ 
   //Subset the path based on preferred locality
4:  $subset\_path \leftarrow \text{SubsetPath}(annotated\_path, l)$ 
   //Divide the path into segments, one per utterance
5:  $path\_segments \leftarrow \text{SegmentPath}(subset\_path, s)$ 
   //Generate utterances for each segment
6:  $utterances \leftarrow \text{NarratePath}(path\_segments, corpus, a, s)$ 
   //Combine utterances into full narrative
7:  $narrative \leftarrow \text{FormSentences}(utterances)$ 
```

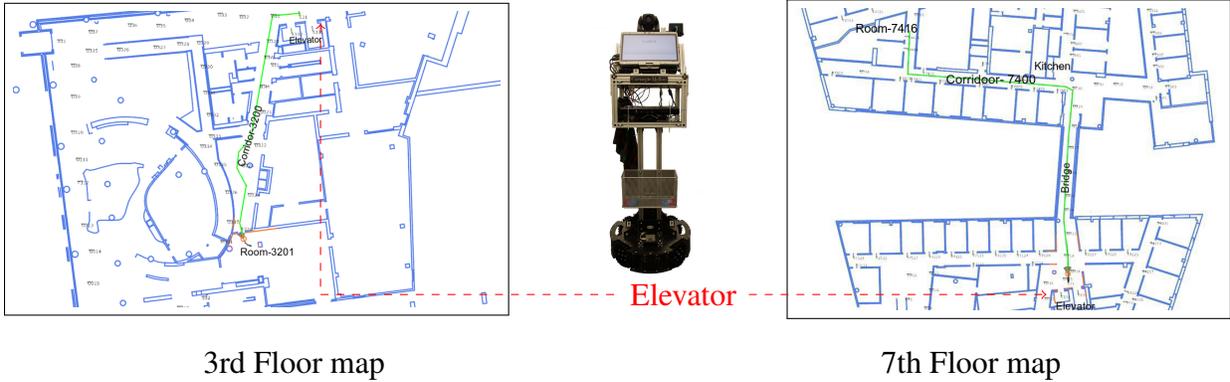


Figure 6.1: Example of our mobile robot’s planning through our buildings. Building walls are blue, the path is green, the elevator that connects the floors is shown in red; shown in black text are our annotations of the important landmarks.

The VV algorithm first uses abstraction preference a to choose which corpus (points, distances or landmarks) to use when generating utterances (Line 2). Because the abstraction preference in the example is Level 4, the VV algorithm chooses a corpus of landmarks, bridges and corridors from the annotated map. The VV algorithm then annotates the route plan by labeling the points along the straight trajectories by their corridor or bridge names and the route plan turning points based on the nearest room name.

Once the path is annotated with relevant locations, the algorithm then extracts the subset of the path that is designated as relevant by the locality preference l (Line 4). In this case, the locality is Global Environment and the algorithm uses the entire path as the subset. With the subset path, the VV algorithm then determines the important segments in the path to narrate with respect to the specificity preference s (Line 5). For Detailed Narratives, our algorithm uses edges

between all turning points, resulting in descriptions of the corridors, bridges and landmarks, and the start and finish points:

{s1: Office 3201, s2: Corridor 3200, s3: Elevator,
s4: 7th Floor Bridge, s5: 7th Floor Kitchen,
s6: Corridor 7400, s7: Office 7416}

The VV algorithm then uses segment descriptions and phrase templates to compose the verbalization into English utterances (Line 6). Each utterance template consists of a noun N , a verb V , and a route plan segment description D to allow the robot to consistently describe the starting and finishing points, corridors, bridges and landmarks, as well as the time it took to traverse the path segments. The templates could also be varied, for example, to prevent repetition by replacing the verbs with a synonym (e.g.,[81]). The following are the templates used on the CoBot robots for the Level 4 abstractions. We note next to the D whether the type of landmark is specific (e.g., the template must be filled in by a corridor, bridge, etc.), and we note with a slash that the choice of verb is random.

- “[I]_N [visited/passed]_V the [---]_{D:room}”
- “[I]_N [took]_V the elevator and went to the [---]_{D:floor}”
- “[I]_N [went through/took]_V the [---]_{D:corridor/bridge}”
- “[I]_N [started from]_V the [---]_{D:start}”
- “[I]_N [reached]_V [---]_{D:finish}”

The template utterances are joined together using “then,” but could, for example, be kept as separate sentences as well. Using the templates filled in with the corresponding verbs and segment descriptions, the VV algorithm generates the following verbalization (Line 7):

I started from office 3201, I went through the 3200 corridor, then I took the elevator and went to the 7th floor, then I took the 7th floor bridge, then I passed the 7th floor kitchen, then I went through the 7400 corridor, then I reached office 7416.

6.2 Dialogue with a Robot that Verbalizes Routes

The VV algorithm takes as its input the users’ preference (a, l, s) for the verbalization they will receive. We would like users to engage in a dialogue with the robot to express their verbalization preferences. In this section, we introduce a method for mapping users’ dialogue onto a Verbalization Space preference. As an example, consider the following command: “Please, tell me exactly what you did along your whole path to get here.” Because this sentence refers to the whole path, we would like the robot to use the Global Environment Locality. The level of Specificity should be a Detailed Narrative, as the user asks the robot to report exactly what it did. Finally, although nothing directly refers to it, we assume that a high level of Abstraction would be appropriate.

The ability of a robot to infer the correct levels of Abstraction, Specificity, and Locality should not be limited to one-time interactions. Once users ask for and receive their route verbalizations, they could be interested in refining the description the robot provides. If we continue the above example, after the robot offers a detailed description of its path, the user could say, “OK robot, now tell me only what happened near the elevator.” The user to be provided a second

summary of the task executed. The robot should generate this second summary by using the same values for Abstraction and Specificity as in the previous example, except with a Locality focus on the elevator’s region. Therefore, our learned mapping of users’ dialogue to Verbalization Space preference should also allow users to refine their previous preferences dynamically during the dialogue.

6.2.1 Data Collection

To enable a robot to infer the users’ initial Verbalization Space preferences correctly and to move in the Verbalization Space to refine the preferences, we gathered a corpus of 2400 commands from a total of 100 participants through an Amazon Mechanical Turk survey. Each participant was asked 12 times to request information about our robot’s paths and then to refine their request for different information. A small sample of the sentences in the corpus is shown in Table 6.1.

Please give me a summary of statistics regarding the time that you took in each segment.
Can you tell me about your path just before, during, and after you went on the elevator?
How did you get here?
Can you eliminate the time and office numbers please
What is the easiest way you have to explain how you came to my office today?
Robot, can you please elaborate on your path further and give me a little more detail?

Table 6.1: Sample sentences from the corpus.

After they gave their consent to participate in the survey, the users were given instructions on how to complete the survey. These instructions included 1) a short description of the robot’s capabilities (i.e., execute task for users and navigate autonomously in the environment) and 2) the context of the interaction with the robot. In particular, we asked the users to imagine that the robot had just arrived at their office and that they were interested in knowing how it got there. For each time the robot arrived at their office, the participants were given

- a free-response text field to enter a sentence requesting a particular type of summary of the robot’s path,
- an example of the summary the robot could provide, and finally
- a second free-response text field to enter a new way to query the robot assuming their interest changed.

This process was repeated 12 times for various parts of our Verbalization Space. Figure 6.2 shows the first page of the survey.

Instructions

- 1) The robot arrives at your office. Ask a question about the robot's path according to the requirements we provide. The robot understands all English sentences.
- 2) Read carefully the robot answer.
- 3) You change your mind about the information you'd like. Ask a second question to change the robot answer.

1. How would you ask the robot to thoroughly recount its path?

2. The robot provides the following summary: "I started from office 7717 on floor no.7. I went by office 7416 on floor no.7 and took 28 seconds. I went through corridor-7400 on floor no.7 and took 42 seconds. I went by hallway 729 on floor no.7 and took 42 seconds. I went through bridge-07 on floor no.7 and took 42 seconds. I went by hallway 723 on floor no.7 and took 28 seconds. I went by hallway 713 on floor no.7 and took 42 seconds. I went through corridor-7100 on floor no.7 and took 28 seconds. I went by office 7115 on floor no.7 and took 28 seconds. I reached office 7115 on floor no.7."

You now want the robot to give you a briefer version of this summary. How would you ask for it?

Next

Figure 6.2: The survey used to gather our corpus. The instructions above the two text fields read, "How would you ask the robot to thoroughly recount its path" and "You now want the robot to give you a briefer version of this summary. How would you ask for it?"

We note that the instructions to our survey did not mention the concept of verbalization and did not introduce any of the three dimensions of the verbalization space. This was done on purpose to avoid priming the users to use specific ways to query the robot. On the other hand, we wanted to make sure the sentences in our corpus would cover the whole verbalization space. So, when asking for the initial sentence on each page, we phrased our request in a way that would refer to a specific point on one of the axes of the Verbalization Space. As an example, in Figure 6.2, we ask for a sentence matching a point with Detailed Narrative Specificity; therefore, we ask, "How would you ask the robot to thoroughly recount its path?" The second sentence we requested on each page refers to a point on the same axis but with the opposite value. In Figure 6.2, we look for a sentence matching a point with General Picture specificity, and we ask the user, "You now want the robot to give you a briefer version of this summary. How would you ask for it?" In the first six pages of the survey, we asked for an initial sentence matching a point for each possible dimension (Abstraction/Specificity/Locality) at extreme values. The same questions were asked a second time in the remaining six pages of the survey. Table 6.2 shows the phrasing for each dimension/value pair.

Abstraction	High	“How would you ask the robot for an easy to read recount of its path?”
	Low	“How would you ask the robot for a recount of its path in terms of what the robot computes?”
Specificity	High	“How would you ask the robot to thoroughly recount its path?”
	Low	“ How would you ask the robot to briefly recount its path?”
Locality	High	“How would you ask the robot to focus its recounting of the path near the elevator?”
	Low	“How would you ask the robot to recount each part of its entire path?”

Table 6.2: Phrasing of survey instructions.

6.3 Learning Dialogue Mappings

We frame the problem of mapping user dialogue to the Verbalization Space dimensions of Abstraction, Specificity and Locality as a problem of text classification. In particular, we consider six possible labels corresponding to two levels, high or low extremes, for each of the three axes of the verbalization space. The corpus gathered from the Mechanical Turk survey was minimally edited to remove minor typos (e.g., “pleaes” instead of “please”) and automatically labeled. The automatic labeling of the corpus was possible because the ground truth was derived directly from the structure of the survey itself.

To perform the classification, we tried several combinations of features and algorithms. Here, we report only the most successful ones. The features considered for our classification task are unigrams, both in their surface and lemmatized forms, bigrams, and word frequency vectors. We also considered two algorithms, a Naive Bayes Classifier and a Linear Regression. The results are shown in Figure 6.3. On the x-axis, we show the number of participants, randomly selected from the pool of 100 survey takers, used to train the model. On the y-axis, we show the average accuracy among 10 leave-one-out cross-validation tests. As the number of participants increases, all of the approaches improve in performance. This is to be expected because the size of the corpus increase proportionally, but this also suggests that once a robot is deployed and is able to gather more sentences asking to verbalize a path, the accuracy of the classification will further improve.

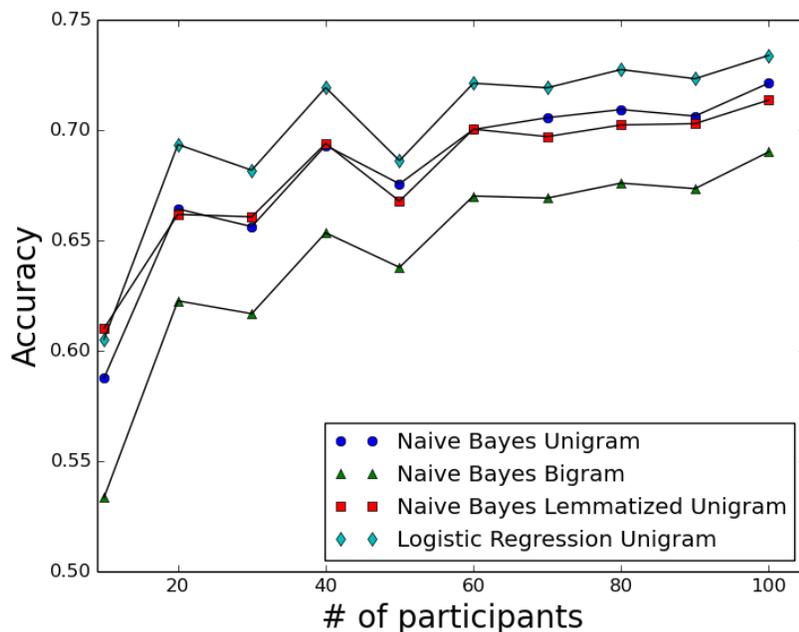


Figure 6.3: Experimental results. The x-axis shows the number of users used to train and test the model, and the Y-axis shows the accuracy achieved.

When trained on the whole corpus, Logistic Regression achieves the best results with 73.37% accuracy. The accuracy with the Naive Bayes Classifier is 72.11%, 71.35%, and 69% when trained, respectively, using unigrams, lemmatized unigrams, and bigrams. It is interesting to note that the Bayes Classifier and Linear Regression perform similarly, as each data point differs by less than 2%. We can also observe how lemmatizing the unigrams does not appear to have a strong effect on the classifier accuracy. Finally, it is worth noting that using bigrams negatively affects the classification accuracy. Although bigrams encode more information than unigrams, they also naturally produce a more sparse representation of the sentence. We believe that this, coupled with the size of our corpus, leads to lower accuracy rates. We can verify this explanation by analyzing the corpus used to train our classifiers. The corpus is composed of 950 distinct unigrams (i.e., words) but by a much larger number of bigrams, 4619. The fact that we have a much larger number of bigrams points in the direction of a more sparse representation but, to check this hypothesis, we can also look at how common the features—respectively unigrams and bigrams—are in each representation. In Figure 6.4 we classify the features in 5 bins based on how often each feature appears in the corpus: VLOW (< 10 occurrences), LOW (10 – 50 occurrences), MED (50 – 200 occurrences), HIGH (200 – 1000 occurrences), and VHIGH (> 1000 occurrences). When using bigrams a large number of features fall in the VLOW and LOW bins; therefore we have many features that only occur seldom. Moreover, compared to unigrams, bigrams have a smaller number of features in both the MED and HIGH bins, and none in the VHIGH bin; therefore, using bigrams, also results in fewer commonly-occurring features.

All the models were trained on the whole corpus, and, therefore, their use is twofold. First, we can use them to infer the correct point in the Verbalization Space when a user initially queries

the robot. Second, we can use them to dynamically update these values during the dialogue. Because our best model achieves 73.37% accuracy, the robot is not always going to provide the correct level of information when verbalizing its task. This should not be considered a problem because using the same models enables users to iteratively refine their queries through dialogue.

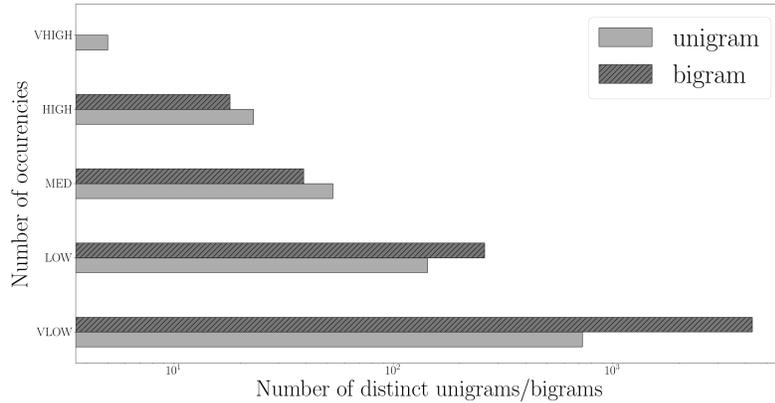


Figure 6.4: Comparison of unigram and bigram as features in the verbalization corpus.

6.3.1 Demonstration on the CoBot Robots

Finally, we demonstrate how we used the learned dialogue mapping to guide the verbalization process on the CoBot robots. In this example, we trained the Naive Bayes Classifier using only unigrams as features to predict the labels of the sentences. We start with the initial sentence given by the user:

HUMAN: Robot, please tell me exactly how you got here.

Our classifier finds the most likely label for the Detailed Narrative sentence; this corresponds to point 1 in Figure 6.5. Accordingly, the robot describes its task:

ROBOT: I started from office 7717. I went by office 7416 and took 28 seconds. I went through corridor 7400 and took 42 seconds. I went by the open area 730 and took 28 seconds. I reached office 7501.

HUMAN: OK, now only tell me about the open area.

Once again, the robot can classify the input sentence. The resulting label is Locality Location-specific. We can, therefore, update the parameters of the verbalization algorithm by increasing the locality value to produce a new summary. This corresponds to the transition from 1 to 2 in Figure 6.5.

ROBOT: Near open area 730. I started from office 7407 on floor 7 and went through corridor-7400.

HUMAN: Can you rephrase this in terms of what you computed?

This final command is classified as Level 1 Abstraction. The parameters of the algorithm are updated, as shown in the transition from 2 to 3 in Figure 6.5, and the robot produces a third summary of its task.

ROBOT: Near O730. Start point is (-1.36, 31.22) on floor GHC7, end point is (10.79, 30.20) on floor GHC7. Start point is (10.79, 30.20) on floor no.GHC7, end point is (16.0, 30.63) on floor no.GHC7.

This short demonstration shows how the verbalization algorithm, coupled with text classification, allows the robot to understand natural language commands asking it to verbalize its task. Moreover, the user is able to specify what type of information the robot should provide and, through interactive dialogue, further refine this information without delving into execution logs.

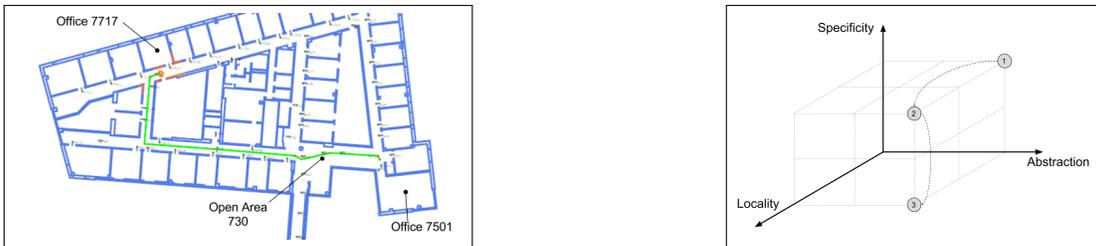


Figure 6.5: Demonstration of movements in the verbalization space. On the left is the path the robot will verbalize. On the right are three points in the verbalization space.

6.4 Summary

A significant challenge with autonomous mobile robots is understanding what they do when no human is around. In this chapter, we introduced verbalization as the process of converting sensory data into natural language to describe a robot’s experiences. We reviewed the verbalization space representing the various dimensions along which verbalizations can be varied and the algorithm for automatically generating them through our CoBot robot. Then, we presented our study of how users can request other verbalizations through dialogue. Using 2400 utterances collected from the study, we demonstrated that it is possible to learn a language model that maps user dialogue to our verbalization space. With greater than 70% accuracy, a robot that uses this model can predict what verbalization a user expects and can refine the prediction further through continued dialogue. Finally, we demonstrated this ability with example verbalizations for the CoBot’s route experiences.

Chapter 7

Proactively Reporting on Task Execution through Comparison with Logged Experience

Robot:“It took me slightly longer than usual to get here.’’

We have shown how, through Verbalization, the robot can provide summaries of the tasks it executes. Verbalization allows our robots to report on the task being executed when users ask about it. In this chapter, we focus on how the robots can pro-actively offer information without being asked. Moreover, although Verbalization only considers the current task being executed, in this chapter we enable the robot to contextualize the current execution using its previous experiences. To do so, this chapter contributes:

1. an approach to compare the execution of a task with the history of previously executed tasks as stored in the robot logs, and
2. an approach to translate this comparison into language to enable a robot-to-human interaction.

To enable a robot to pro-actively report on the tasks it executes, we focus on the time the robot took to execute them. Time is a good indicator of how a task was executed. An execution time longer than expected suggests that the robot encountered something unanticipated during the task execution (e.g., a person standing in the way that caused the robot to stop). For the user to know if the execution time is longer than expected, the robot contextualizes the information and compares it with the usual time a given task takes. We compute the expected time for a task using the information stored in the robot logs. Concretely, in this chapter, we demonstrate how we enabled the robot to report on the time taken to execute tasks and, rather than simply mentioning the length of the task (e.g., “75 seconds”), how we enabled the robot to use comparative expressions such as, “It took me three times longer than usual to get here.”

Before delving into more details of our approach, we need to make one key observation on how time is perceived and compared. When we compare time lengths, our perception is affected by the time difference and the time magnitude. Moreover, our perception of how much

longer something took is highly non linear. This observation becomes straightforward if we look at a concrete example. If we are waiting for a subway train, we might expect to have a train stop by every 5 minutes. If we end up waiting for 20 minutes, we might say that the train took considerably longer to arrive than expected. However, if we are flying across the country and our flight takes 15 minutes longer than the estimated 5 hours, we probably consider this delay negligible. In both these situations, the time difference is the same, but our perception is different. Now let us consider two more scenarios. In the first, we are waiting in line at a grocery store register. We have one person left in front of us, so we expect to be done with our grocery shopping in the next 5 minutes. The person in front of us forgot to get milk, makes a run for it and delays us for an extra 5 minutes. In total, it took us twice longer than expected to pay for our groceries, but this is probably not a big deal. In the second scenario, due to an accident on the beltway, it takes us 1 hour to get home instead of the usual 30 minutes. Surely, this is perceived as a much bigger increase even if, like in the grocery shopping scenario, our expected time has been doubled. In summary, our perception of how much longer than expected an event takes is affected by the length of the event and the increase in time incurred. The perception of how much longer an event takes also affects the language used to describe it. If we perceive a small difference in time, we use milder language (e.g., “it took a little longer” or “it was a bit late”), but if the difference is perceived as large, we use stronger expressions (e.g., “it took forever” or “it was way too long”). Our goal is for the robot to exhibit a similar behavior, where the strength of the expressions used to proactively report matches the difference between the expected time and the actual time for a task execution.

We can summarize our approach to enable the robot to pro-actively report in three steps:

1. Extract a model of the time a task takes from the log the robot records during its runs. This model provides a measure of the expected time for the task.
2. Enable the robot to compare the time a task takes with its expected time.
3. Translate the relationship between the time the task takes and the expected time into natural language and report to the user.

In Section 7.1, we describe the first two steps of our approach, and in Section 7.2, we introduce Comparative Templates to translate the relationship between the current and expected times into natural language. Comparative Templates are not only limited to proactively reporting on task execution, but can be used more generally to motivate any choice the robot makes that is based on a quantitative metric. In Section 7.3, we apply Comparative Templates to explain the robot’s choice when selecting a specific execution trace out of the ones matching complex commands.

7.1 Task Time Expectations

Our first goal is to find a way to describe how long the robot expects a task to take. The robot travels at a fixed speed; therefore, we can estimate the time needed to execute a task by measuring the distance the robot needs to travel and by using the robot’s speed to compute the time. In Section 2.3, we have shown how the robot uses the Navigation Map described to plan its path in the building. Using the information stored in the Navigation Map, we can easily determine the distance traveled for any task. Once we have computed the distance the robot will travel,

deriving the time is straightforward.

Using the distance and the robot’s speed to estimate the time a task takes does not account for the interaction the robot has with its users while executing tasks. When the robot travels along the hallways of the building, it may come across people. When this happens the robot stops and asks to go through (see Section 2.4). The time the robot spends asking people to move cannot be estimated based on the distance it travels. Similarly, when the robot travels across floors [64], it might need to wait shorter or longer times for the elevator. Once again, we cannot account for the time spent waiting for the elevator from the distance the robot travels when executing the task.

Instead, we choose to use a probabilistic model to represent the time the robot will take to execute a task. In particular, we fit a statistical distribution to the data stored in the robot logs. To do so, it is worth reasoning on what we would expect the data to look like. Because the quantity we are estimating is the result of a physical process (traveling in the building), it should have a lower bound. In particular, the time is never going to be less than the distance to be traveled at the robot’s speed. Such a lower bound represents the ideal and most common outcome of a task. Occasionally, a task could take longer due to the aforementioned interactions between the robot and its users; in a very unlucky situation, a task could take effectively forever (i.e., the robot is stuck for too long and runs out of battery power). Therefore, we choose to use a half-Normal distribution to represent the robot tasks:

$$f(x; \sigma) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}} e^{-\frac{x^2}{2\sigma^2}} \quad \text{with } x > 0$$

Figure 7.1 shows a half-Normal distribution with $\mu = 0, \sigma = 1$ next to a Normal distribution with the same parameters.

Similarly to a Normal distribution, we can describe a half-Normal distribution in terms of its mean and standard deviation. To find these parameters for a specific task, we can process past instances of the same task as captured by the logs, and use Maximum Likelihood Estimation. Although the mean of a task will be close to the expected time computed using the distance traveled, the standard deviation can be affected by many factors such as the length of the task (a longer task will probably have a larger σ), the floor the task requires visiting (tasks on a less busy floor will have a smaller σ) and whether or not the robot needs to take the elevator (taking the elevator will likely induce higher variance).

We show the process of extracting the expected time for a task using synthetic data generated by the CoBot robot simulator. We simulated 1000 runs for the GoTo (F8010) task with the robot starting from office F8001. Figure 7.2a shows the sequence of times the robot took to execute the task, Figure 7.2b shows a histogram of the times taken and Figure 7.2c shows the half-Normal distribution we fit using Maximum Likelihood Estimation. For this task we find the half-Normal distribution is represented by $\mu = 17.2250$ and $\sigma = 8.7067$.

The second goal is to enable the robot to compare the time a task took with its expected time, and to do so, we introduce the *Comparative Factor*. Given T , the time the robot took to execute a task, and μ and σ , the mean and standard deviation, respectively, extracted from the logs for the same task, we can always write:

$$T = \mu + \mathcal{C}\sigma$$

where \mathcal{C} is the Comparative Factor. The Comparative Factor expresses how close (or far) the current time T is to the expected time. If we consider again the task GoTo (F8010) for a current

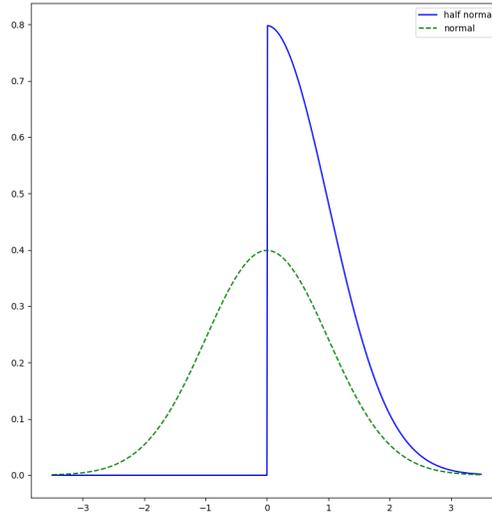


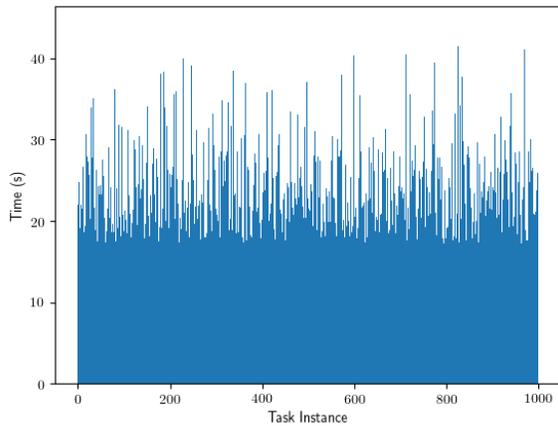
Figure 7.1: Half-Normal distribution compared with a Normal distribution. Mean and variance are, respectively, 0 and 1 for both the distributions.

time $T = 42$, the Comparative Factor \mathcal{C} is 2.8455. Figure 7.3 shows the time $T = 42$ with respect to the fitted half-Normal Distribution.

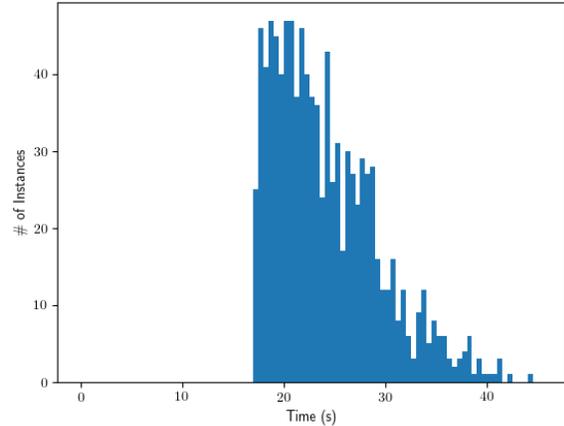
7.2 Comparative Templates

We have shown how, when the robot executes a task, it can use the previous execution to fit a half-Normal distribution and use the Comparative Factor as a measure to relate the current time with its expectations. Knowing that $\mathcal{C} = 2.8455$ is not very informative for a user, the next step in our approach is to translate Comparative Factor into natural language for our robot-to-human interaction.

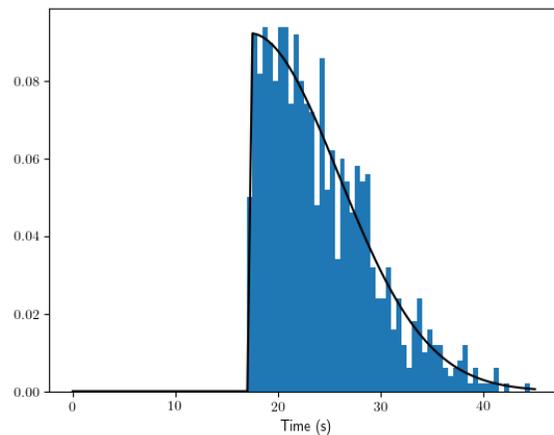
To learn the language used to compare time, we performed a crowd-sourcing study. A total of 20 users contributed to the study. Each participant was asked to provide 10 sentences comparing commuting times, resulting in a corpus of 200 sentences. At the beginning of the study, we randomly assigned each participant an initial commute time T of either 30, 45 or 60 minutes. We asked the participants to consider this time T as their usual commute time. Then, for five times, we assigned a new commute time X and asked the participants to report on how long their commute took. We explicitly asked the participants to report on this new commute time X by using language comparing X with T , the usual time initially assigned them. The participants provided two sentences for each new time we asked them to report on. The five new times X were randomly selected in five fixed intervals. These intervals are $[T - 10, T)$, $[T, T + 5)$, $[T + 5, T + 15)$, $[T + 15, T + 25)$ and $[3T, 4T)$. When recording the participants' sentences we also recorded the time and the intervals that prompted them. By doing so, we could divide the



(a) The data as computed by the CoBot robot simulator.



(b) A histogram of the data.



(c) The half-Normal distribution estimated from the data.

Figure 7.2: Computing task expectations for task GoTo (F8010) .

corpus into five sets of sentences, each referring to times that took progressively longer than the expected time.

Figure 7.4 shows an excerpt of the corpus we collected through the user study described. Interestingly, some of the sentences used simple expressions such as “*a bit less*” or “*later*” to compare times. On the other hand, some of the sentences in the corpus used expressions that described the mathematical relation between the times being compared (e.g., “*five minutes more,*” “*three times as much*”). To enable our robot to use both kinds of sentences, we introduce *Comparative Templates*. A Comparative Template is defined as a sentence and a set of functions. The sentence has zero or more functional slots, and the Comparative Template provides one function for each of the slots. Comparative Templates are instantiated by the robot to derive expressions comparing times. As an example, we consider the expression “*five minutes more.*” We can derive this sentence from the following Comparative Template “ $(T - \mu)$ more.” This Comparative Template has one functional slot, and the corresponding function is represented

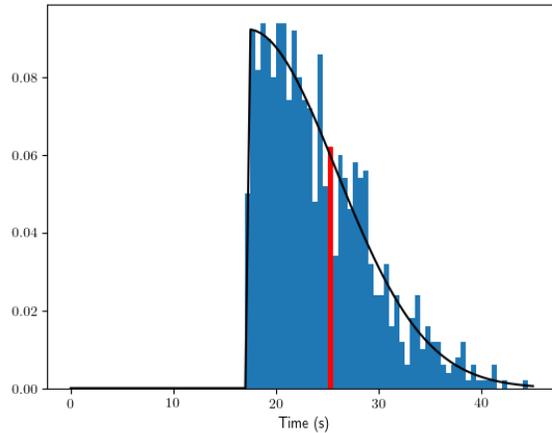


Figure 7.3: The current time being considered $T = 42$ highlighted in the histogram.

within the parentheses.

We derived the Comparative Templates starting from the sentences in the corpus. Figure 7.5 shows more examples of expressions found in the corpus and the corresponding Comparative Templates we extracted (see Appendix A.5 for the exhaustive list of Comparative Templates extracted).

It took me a bit less than normal time to get to work today.
 My commute was exactly the same today as it is on an average day.
 I got to work later than I expected.
 It took me almost 15 minutes longer than usual to get here today.
 Today's commute was absolutely horrible...3 freaking hours!
 It took me over three times as long as normal to get here today!

Figure 7.4: An excerpt of the corpus collected comparing commute times.

It took me almost 15 minutes longer than usual to get here today.
 It took me over three times as long as normal to get here today!
 Today my commute was pretty nice - I got to work 5 minutes earlier than usual.

(a) Sentences from the crowd-sourced corpus.

It took me almost $(T - \mu)$ longer than usual to get here.
 It took me over $(T\% \mu)$ times as long as normal to get here.
 I arrived here $(\mu - T)$ earlier then usual.

(b) Comparative Templates.

Figure 7.5: The Comparative Templates extracted from the crowd-sourced corpus.

We use Comparative Template to translate the Comparative Factor \mathcal{C} into natural language. To do so, we need to have the robot select an appropriate template given the value of \mathcal{C} . The corpus we crowd-sourced is divided into five sets; each set is composed of sentences that refer to

times that take progressively longer. We assign to each Comparative Template an index i from 0 to 4 based on which set it is derived from (with 0 being the set referring to the shortest times).

Next, we define *Comparison Regions* on a half-Normal distribution. Comparison Regions are intervals on the distribution. The boundaries of the interval are defined in terms of the distribution mean plus some multiple of its standard deviation (e.g., $[\mu + \sigma, \mu + 2\sigma]$). We represent each boundary of a Comparison Region using only the multiplicative factor of σ ; that is, we represent the interval $[\mu + \sigma, \mu + 2\sigma]$ as $[1, 2]$. We determine five Comparison Regions: $[-\frac{1}{4}, 0)$, $[0, \frac{1}{2})$, $[\frac{1}{2}, 1)$, $[1, 2)$, $[2, \infty)$ and we index them from 0 to 4. Figure 7.6 shows the five Comparison Region we determined.

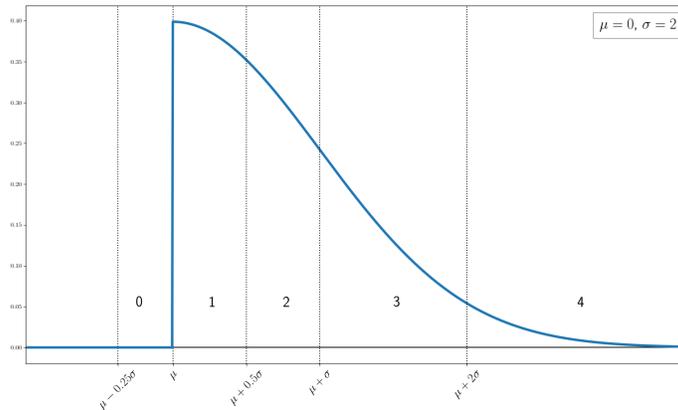


Figure 7.6: Comparison Regions on a half-Normal distribution.

When the robot executes a task in order to report to the user, it randomly selects from the set of Comparative Templates indexed as i , such that:

$$\mathcal{C} \in \mathcal{CR}_i$$

where \mathcal{C} is the Comparative Factor and \mathcal{CR}_i is the i^{th} Comparison Region. Once the Comparative Template has been selected, the robot instantiates it and reports to the user.

Going back to the task GoTo(F8010), we compute $\mu = 17.2250$ and $\sigma = 8.7067$. We assume that in the current execution of this task the robot takes $T = 42$ seconds and therefore $\mathcal{C} = 2.8455$. To pick an appropriate template, the robot selects the set indexed by $i = 4$ as $\mathcal{C} \in [2, \infty)$. In particular, the robot randomly picks the following template to report to the user:

"It was absolutely horrible, (T) to get here!"

After instantiating the template with the current time $T = 42$, the robot reports to the user by saying:

"It was absolutely horrible, 42 seconds to get here!"

7.3 Comparative Templates for Complex Commands

Human:“*Can you take the mail from the 3rd floor lab to Manuela’s office or go to Vittorio’s office and to the 7th floor conference room*”

Robot:“First I am going to F7004, then I am going to F7101. I will complete this task a little earlier than any of the alternatives.”

In Chapter 4, we have shown how the robot is able to understand complex commands. Our approach to understanding and executing complex commands is based on breaking them down into atomic components. Once the atomic components of a complex command have been grounded, the robot computes all the execution traces matching the complex command and selects the shortest one. As we have shown, Comparative Templates express relationship between time length; therefore, we should be able to use them to motivate the robot’s choice in selecting one trace over the alternatives.

So far, we have shown how we can use Comparative Templates to compare the time the robot takes to execute a task with the expected time. We further use them to compare execution traces and the times these would take. Before describing our approach, it is worth pointing out three key differences between comparing a task with its expectation and comparing execution traces:

1. Comparative Templates, when used to pro-actively report on a task, only compare two elements, the execution time T and the expectation μ . Instead, when dealing with execution traces, we might have more than two elements to compare (i.e., from a complex command, we could derive three or more execution traces).
2. Comparative Templates rely on having a statistical description (μ and σ) of the single task executed to select the right template set to use. Instead, execution traces are often composed of multiple tasks.
3. Comparative Templates typically convey expressions that refer to times that took as much as or longer than expected. Instead, when choosing one execution trace over the other, we are interested in expressions referring to a time shorter than the alternatives.

In the rest of this section, we will address the three key differences listed above, present our approach to motivate the choice of execution traces with Comparative Templates and run through an example.

The first difference is that when comparing execution traces, we might have more than two elements to compare. To address this issue, we only compare the two shortest execution traces but report on all of them. From a complex command, we obtain N execution traces. We will refer to the set of traces as $\{e_1, e_2, \dots, e_N\}$. We assume e_i and e_j are the shortest and the second shortest traces, respectively. We refer to the time estimated for each execution trace as $\{t_1, t_2, \dots, t_N\}$. The time for e_i , t_i , is shorter than the time for e_j , t_j , by some amount Δ , that is:

$$t_i = t_j + \Delta$$

Because e_i and e_j are the two shortest traces it must hold that:

$$\forall k \neq i, j \quad t_i \leq t_k + \Delta$$

Therefore, any relationship holding between t_i and t_j must, at least, hold between t_i and any of the other trace times. This allows us to compare t_i and t_j but still report on all the traces. As an example, consider t_1 being 5 minutes shorter than t_2 ; the robot is going to report using the following formula:

I am going to execute e_1 . It will take at least
5 minutes less than any of the alternatives

The second difference is that to select which Comparative Template to use, we rely on having a statistical distribution of the task. In particular, we use μ and σ to select the right set of Comparative Templates to sample from. Execution traces can be composed of multiple tasks; hence, we will need to derive a model (μ and σ) for the whole trace. Moreover, even in the simplest case when we have two execution traces each composed by a single task, we need to account for the mean and standard deviation of both tasks. To derive a model for the whole trace, we use a naïve approach. If an execution trace e is composed by a sequence of N tasks $\mathcal{T}_1, \dots, \mathcal{T}_N$ we first derive the model of each task from the log, as we have shown in Chapter 7.1. This returns the mean and standard deviation for each of the tasks ($\{\mu_1, \sigma_1\}, \dots, \{\mu_N, \sigma_N\}$). We define the mean and standard deviation of the whole execution trace as the sum of the mean and standard deviation of each task, formally:

$$\mu_e = \sum_{i=1}^N \mu_i \quad (7.1)$$

$$\sigma_e = \sum_{i=1}^N \sigma_i \quad (7.2)$$

The reason we call this a naïve approach is that the model of each task is derived by fitting a half-Normal distribution, but summing half-Normal distributions does not result in a half-Normal distribution.

Once we derive a model for the two execution traces we need to compare, we then need to select the appropriate set of Comparative Template to sample from. If the two traces e_1 and e_2 are modeled by $\{\mu_{e_1}, \sigma_{e_1}\}$ and $\{\mu_{e_2}, \sigma_{e_2}\}$, we use μ_{e_1}, σ_{e_1} to partition the half-Normal distribution, as shown in Figure 7.6, and μ_{e_2} to select the correct set. The reason for this choice is that, in our use of Comparative Templates, σ is a measure of how far we are from our expectations. Consequently, we are interested in measuring how far the second trace is from the one we selected to execute.

The third difference we need to address is the fact that Comparative Templates refer to actions that took longer than expected, however, for complex commands, we are interested in referring to times that took less than the alternatives. A Comparative Template as a whole is an expression conveying how much longer a task took compared with expectations. Most of the templates rely on a single word (typically the comparative adjective) to express the fact that the time was longer, but the rest of the template is a measure of the difference between the current time and expected time. By changing the comparative adjective to its opposite, we go from an expression referring to a longer time to one referring to shorter times. As an example, consider the template “It took me almost $(T - \mu)$ longer than usual to get here.” We transform this into, “It took me almost

$(T - \mu)$ ” shorter than usual to get here.” Figure 7.7 shows more examples of how, by switching a single word, we can change the meaning of Comparative Templates.

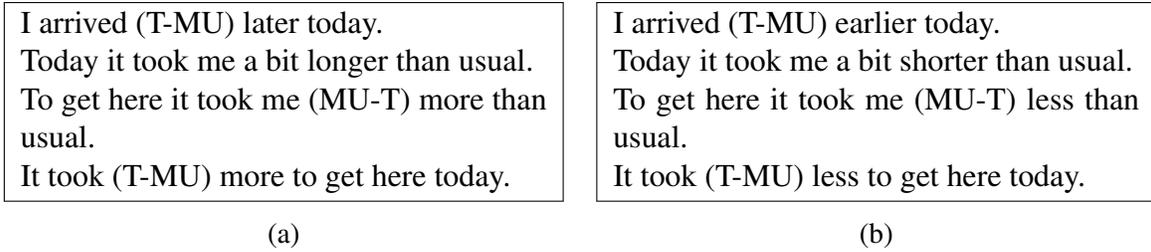


Figure 7.7: (a) The Comparative Templates extracted from the corpus and (b) their modified version to be used when comparing execution traces.

Changing the comparative adjective is not the only change we need to make to Comparative Templates before being able to use them to compare execution traces. Comparative Templates use the past tense to report after a task has been executed. We need to change the verb tense to the future because comparing execution traces happens before the execution. As we have already shown, we compare only the two shortest execution traces, but we want to report on all of them; hence, we need to make sure the templates refer to “any of the alternatives.” Finally, rather than comparing an expectation μ and the actual time T , we will compare two expected times, μ_{e_1} and μ_{e_2} . Therefore, we need to change the functional part of the templates accordingly. Figure 7.8 shows the final templates we use to compare execution traces.

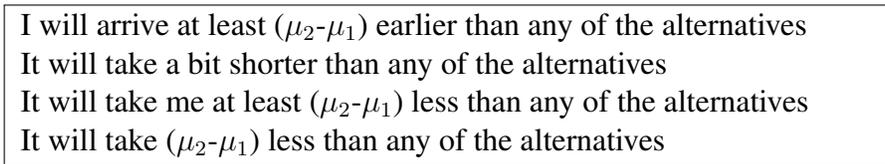


Figure 7.8: Comparative Templates used to compare execution traces and motivate Complex Command choices.

So far, we have detailed how we can adapt Comparative Templates to compare execution traces. Next, we present a detailed example to show how the CoBot robots use Comparative Templates when they receive complex commands. We assume that the robot is on the second floor of the Gates-Hillman Center in front of room 2109 and receives the following command: “Can you take the mail from the 3rd floor lab to Manuela’s office or go to Vittorio’s office and to the 7th floor conference room?” Using the approach described in Chapter 4, the robot first breaks down the command into its atomic components and then grounds each of the simple commands; these two steps are shown in Figure 7.9.

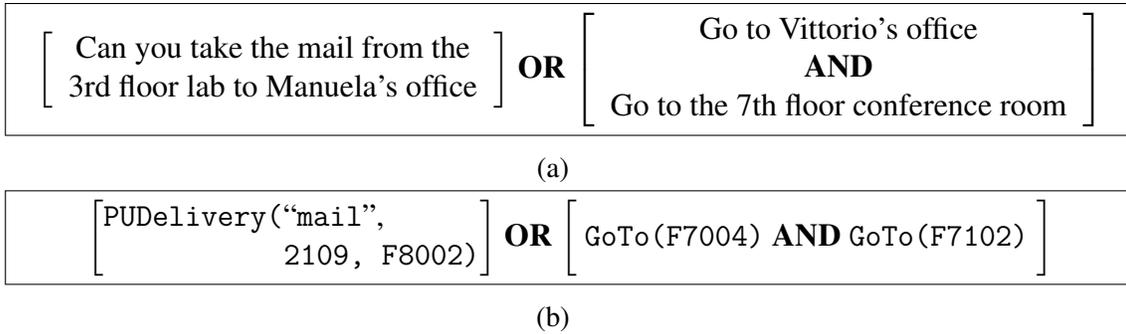


Figure 7.9: (a) The simple commands extracted from the sentence: “Can you take the mail from the 3rd floor lab to Manuela’s office or go to Vittorio’s office and to the 7th floor conference room?” (b) The task obtained after grounding the simple commands.

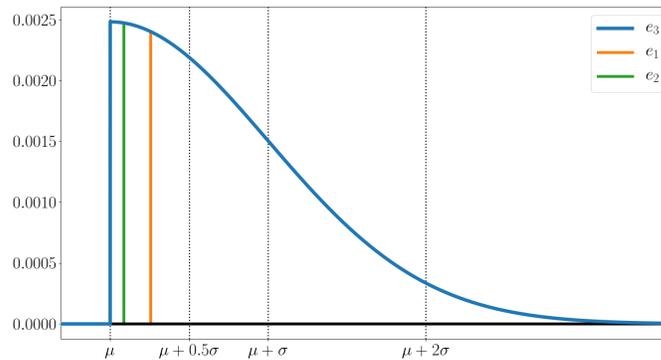


Figure 7.10: Comparing the estimated cost of the execution traces corresponding to the command, “Can you take the mail from the 3rd floor lab to Manuela’s office or go to Vittorio’s office and to the 7th floor conference room?”

Once the commands have been grounded, we obtain three tasks \mathcal{T}_1 , \mathcal{T}_2 , and \mathcal{T}_3 , respectively: PUDelivery(“mail”,2109, F8002), GoTo(F7004) and GoTo(F7102). The next step is to compute the execution traces corresponding to the complex command. In this case, we obtain three possible traces: $e_1 = [\mathcal{T}_1]$, $e_2 = [\mathcal{T}_2, \mathcal{T}_3]$, and $e_3 = [\mathcal{T}_3, \mathcal{T}_2]$. Next, the robot needs to evaluate the cost of each trace. Recall that to estimate the cost of a trace, we first compute the expected time for the tasks and, for e_2 and e_3 , add them. The resulting values for the three traces are, respectively, $(\mu_1 = 723.2, \sigma_1 = 358.3)$, $(\mu_2 = 677.3, \sigma_2 = 329.3)$, and $(\mu_3 = 650.3, \sigma_3 = 321.3)$. The robot selects the cheapest, e_3 , to be executed and then reports to the user on the choice made. To report, the robot first needs to select a suitable Comparative Template. Figure 7.10 shows where e_1 and e_2 fall with respect to the cost estimated for e_3 . To select the correct template, the robot only considers μ_1 , which falls in the interval $[\mu; \mu + \frac{1}{2}\sigma]$. Accordingly, the robot selects one of the comparative templates indexed by 1, in this case, the sentence “I will complete this task a little earlier than“. Finally, the robot reports to the user saying:

“First I am going to F7101, then I am going to F7004. I will complete this task a little earlier than any of the alternatives.”

7.4 Summary

In this chapter, we have shown how we can enable the CoBot robot to proactively report on its task execution. Our goal is to have the robot report in a concise yet informative way. To do so, our approach leverages Comparative Templates. Comparative Templates capture the natural language expression used to compare lengths of time. To select the correct Comparative Template to be used, we relied on a statistical description of the expected length of a task. We presented a running example of how the CoBot robots use Comparative Templates to report. Finally, we showed how to adapt Comparative Templates to enable our robots to motivate their choice when they receive a complex command and need to select one execution trace over the others.

Chapter 8

Deep Learning for Semantic Parsing

Human: *“play ray of light by madonna.”*

Semantic parsing is the process of mapping natural language into a formal representation of its meaning. In Chapter 3 and Chapter 5, we used a CRF to extract semantic frames to represent the meaning of, respectively, a request to execute tasks and questions about a robot’s past experience. In this chapter, we introduce two representation, Spoken Language Understanding (SLU) and Alexa Meaning Representation Language (AMRL), which an intelligent assistant (i.e., Alexa) uses to represent the meanings of users’ request [60]. We present a deep learning approach to enable the semantic parsing of AMRL. Our approach leverages transfer learning from SLU, for which a large labeled corpus is available (~ 2.8 million sentences), to the AMRL, for which a smaller dataset is available ($\sim 350,000$ sentences).

The work we present in this chapter directly connects to the thesis’s main contribution on bidirectional human-robot interaction in several ways:

- First, the problem of an intelligent assistant understanding users’ requests closely resembles the one of CoBot understanding user commands (from Chapter 3). Similarly, after being given the user’s request, an intelligent assistant needs to determine which action to take and to identify all the arguments it the selected action.
- Second, we do not have access to a large corpora of labeled sentences in the main domain of this thesis (i.e., mobile service robots). The approach that we introduce to transfer knowledge among various representation of a sentence’s meaning can, however, be applied to readily available corpora and can thus improve the accuracy of the semantic parser that the CoBot robots use.
- Third, the semantic parsers used that the CoBot robots use were trained using small corpora of sentences that were collected from inside our lab or through crowd-sourcing studies. Here, we have the opportunity to work with large corpora of sentences that were collected directly from users.

In this chapter, we first introduce the two representations that the intelligent assistant uses: SLU and AMRL. Then, we present the deep learning models that we introduce to perform semantic parsing. Finally, we show the results from our experimental evaluation.

8.1 Semantic Representations

The SLU uses a factored approach to represent a sentence’s meaning. Users’ utterances are first categorized into domains, which are the general categories for a request (for “play ray of light by madonna” MusicApp is the domain). Next, SLU determines the intent of the sentence, or the action that the sentence requires (for the example sentence, the intent is ListenMedia). Finally, SLU identifies each of the slots in the sentence. Slots are mentions within the utterance; the example sentence here has two slots “ray of light” (the song to be played) and “madonna” (the singer) Figure 8.1 shows the SLU representation for the example sentence.

	<i>“play ray of light by madonna”</i>
Domain:	MusicApp
Intent:	ListenMediaIntent
Slots:	play [ray of light] _{Song} by [madonna] _{Singer}

Figure 8.1: Example of the spoken language understanding (SLU) representation.

The use of this factored representation enables an intelligent agent to understand users’ requests. However, it presents several limitations:

- As the abilities of the intelligent assistant increase, surface forms can overlap across domains. Consider the following two sentences: “Alexa, order me an echo dot” and “Alexa, order me a taxi.” Although the two sentences are very similar, they clearly relate to different general categories (i.e., domains). The first can be categorized in the shopping domain, but the second is in the transportation domain.
- Cross-domain utterances cannot be correctly represented using a factored approach. A user interested in watching a hockey game might ask, “find me a restaurant near the sharks game”. Using the SLU representation, this sentence can be assigned to either the local or the sports search domains, but in reality, it belongs to both.
- Complex sentences cannot be easily supported. As an example, consider the sentence, “play hunger games and turn the lights down to 3”. This utterance is composed of two requests, but the factored representation can only assign it to a single domain.

The AMRL was designed to overcome these limitations [44]. The Alexa Meaning Representation Language is a compositional, graph-based semantic representation that comprises five primary components: actions, roles, types, properties and verb operators. Below, we describe each of these components with reference to the sentence “play ray of light by madonna” (Figure 8.2).

- **Actions** define the core functionality used for spoken language understanding. In the example sentence, the action is PlaybackAction and it is used on a MusicRecording.
- **Roles** define how actions operate on entities (i.e., concepts or things). In the example sentence the object’s role defines the entity on which the action operates (i.e., a MusicRecording).

- **Types** apply to each entity mention in a request; they are hierarchical. The example sentence has two types: MusicRecording (referring to “ray of light”) and Musician (referring to “madonna”).
- **Properties** define the relationships between variables of a given type. In the example sentence, the byArtist property ties together two types: the MusicRecording “ray of light” and the Musician “madonna”. In this case, the property defines who wrote the song.
- **Operators** represent complex logical or spatial relationships. In the example sentence, no operator is needed to represent the meaning of the sentence.

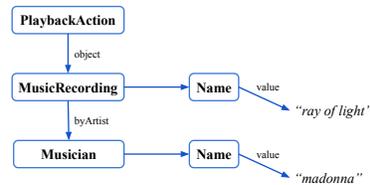
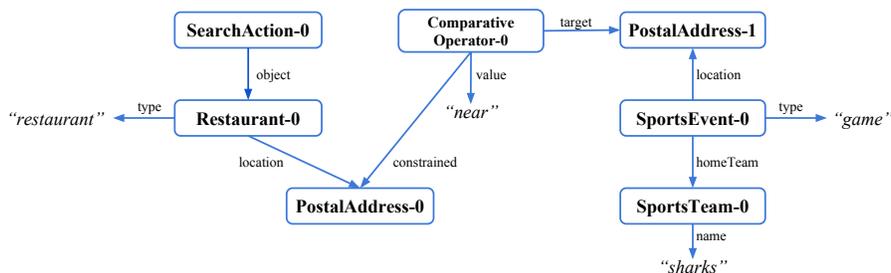


Figure 8.2: Meaning representation of the sentence “play ray of light by madonna” using AMRL.

Figure 8.3 shows the AMRL representation for two other sentences; these require the use of operators.



(a) “find restaurants near the sharks game.”



(b) “play charlie brown and turn the volume up to 10”

Figure 8.3: AMRL representations of (a) a sentences with cross-domain references, and (b) a sentences requiring multiple actions. Both types are handled using operators.

The annotations for sentences represented using SLU are factored using three components. The first and second components categorize the whole sentence at the domain and intent levels, respectively. The third component labels each of the words in the sentence and determines the span of each of the slots. Therefore, the annotations for SLU closely match the representation shown in Figure 8.1.

The annotations for sentences represented using AMRL are also factored into three components: Properties, Types and Intents.

- The Property component stores a linearized version of the ARML graph. To linearize this graph, we start at root, invert the properties to avoid cycles and two-headed graphs, and recursively descend to the leaf.
- The Types component stores information about the entities’ types, including the entity mention’s leaf property (“type,” “value” or “name”) and its type.
- The Intent component includes the action, its roles (e.g., “object”) and each of the role’s type.

Figure 8.4 shows an example of the annotations for the AMRL. Using this type of annotation allows us to formulate the parsing of the AMRL as a sequential prediction problem.

<i>“play ray of light by madonna”</i>	
Properties:	play [ray of light] _{object.name} by [madonna] _{object.byArtist.name}
Types:	play [ray of light] _{name@MusicRecording} by [madonna] _{name@Musician}
Intents:	PlaybackAction<object@MusicRecording>

Figure 8.4: Linearized annotation for AMRL.

Unlike SLU, AMRL forgoes the domain-factored approach, which allows AMRL to overcome SLU’s two limitations (the similar surface forms across domains and the cross-domain utterance). Moreover, to handle complex sentences, AMRL introduces operators. AMRL and SLU also have some similarities. The intent expressed in the SLU representation often matches or can be directly mapped to the action of AMRL. Moreover, the span of the SLU slots typically matches the span of AMRL properties and types. In the next section, we show how we can leverage these similarities to enable transfer learning from SLU to AMRL.

8.2 Deep Learning Models

The models we devise to learn a semantic parser perform multitask learning. Using multitask learning, these models jointly learn each task. For SLU Domain, Intent and Slots predictions are learned jointly, for AMRL Properties, Types, and Intents are learned jointly. Unlike models that are trained separately for each of the tasks, multitask models allow for the exploitation of commonalities between tasks [15], as in the overlapping spans between AMRL types and properties.

Our baseline model is a deep bidirectional LSTM neural network that is trained using multi-task learning. It has four primary LSTM layers, each of which predicts a different component of the parse. The first layer provides a shared word embedding [21], and the remaining three LSTM layers are connected to an affine transform and a softmax layer to provide the output for each of the three tasks (prediction of properties, type and intent). IOB-tagging (inside, outside and beginning) is used to denote the span of Properties and Types. Residual connections connect the first layer with the third and fourth ones. Figure 8.5 shows the topology of this model. The input

of this baseline model is a one-hot vector in which each word in the sentence is represented as a vector the size of the vocabulary; all values are 0 except for the index representing the word. The first layer of the baseline model performs the type prediction, the next layer makes the property prediction, and the final layer classifies the intent.

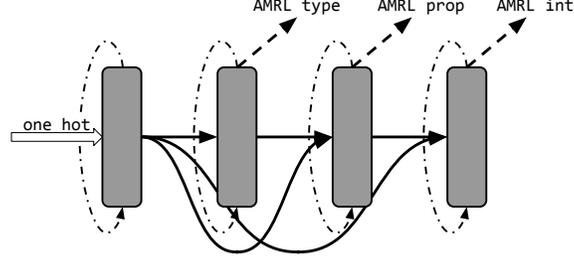


Figure 8.5: Topology of the baseline multitask DNN model. Each component above is a bidirectional LSTM layer. The final layer is a softmax, and the dashed line represents the recurrent connection. The “type” and “prop” components perform sequential prediction, and the “int” component categorizes the user’s intent.

LSTMs

We leverage LSTMs for both sequential prediction and classification. The input, x_t , of each LSTM unit is obtained by combining the output of previous units through an affine transformation:

$$x_t = W^{(x)}w_t + b^{(x)} \quad (8.1)$$

where W is a vector of learned weights and b is a bias term.

Figure 8.6 shows the structure of an LSTM unit. This type of unit produces a hidden representation for each token in the utterance (i.e., each word in the sentence) by encoding the token as a function of its input, x_t ; the encoding of the previous token, h_{t-1} ; and the previous memory cell, c_{t-1} . Each token is encoded by set of five parameter vectors: an input gate, i_t ; a forget gate, f_t ; an output gate, o_t ; a memory cell, c_t ; and a hidden state h_t . These vectors are combined as follows to produce the representation, h_t , at the current token:

$$\begin{aligned} i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b^{(i)}) \\ f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b^{(f)}) \\ o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b^{(o)}) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \sigma(W^{(c)}x_t + U^{(c)}h_{t-1} + b^{(c)}) \\ h_t &= o_t \odot \sigma(c_t) \end{aligned}$$

where σ is the sigmoid function, \odot is element-wise multiplication, W and U are parameter matrices, and b is the bias vector. We denote the forward and backward hidden representations as \overrightarrow{h}_t and \overleftarrow{h}_t , respectively.

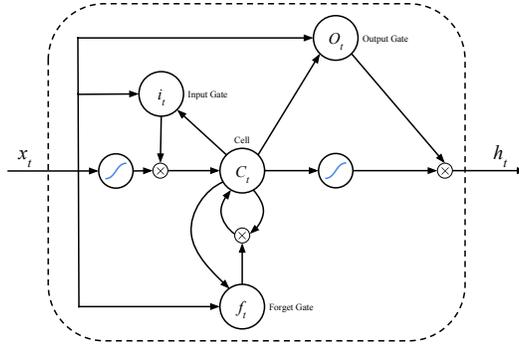


Figure 8.6: Structure of an LSTM unit.

For the output layers, we form a prediction by feeding the concatenated hidden representations for the token that we wish to label, $[\vec{h}_t; \overleftarrow{h}_t]$, into another affine transform, followed by a softmax to obtain the posterior probabilities for the set of labels:

$$y_t = \text{softmax}(W^{(y)}[\vec{h}_t; \overleftarrow{h}_t] + b^{(y)})$$

We apply dropout after each LSTM layer. Each block, as in Figure 8.5, includes forward and backward LSTM. The output layers have affine and softmax components. For classification tasks (e.g., to predict the intent), only the final states of the forward and backward LSTM are used.

Models Topologies

In this section, we explore various topologies (i.e., models) of Deep Neural Network to enable the learning of a semantic parser for AMRL. In particular, we leverage the similarities between the AMRL and SLU prediction problems (e.g., overlap between SLU slots and AMRL types).

SLU slots are named entities, so they overlap significantly with the AMRL types. To leverage this commonality, the first two topologies leverage SLU type embeddings as the initial layer in the model. This layer can be either serial or parallel to the AMRL model. When it is serial, a learned slot embedding is used to predict the later stages of AMRL (properties, types and intents). When it is parallel, only a common word embedding layer is shared. The parallel topology (+SLU Slots) is shown in Figure 8.7a, while the serial one (+SLU Slots Pipe) is shown in Figure 8.7b.

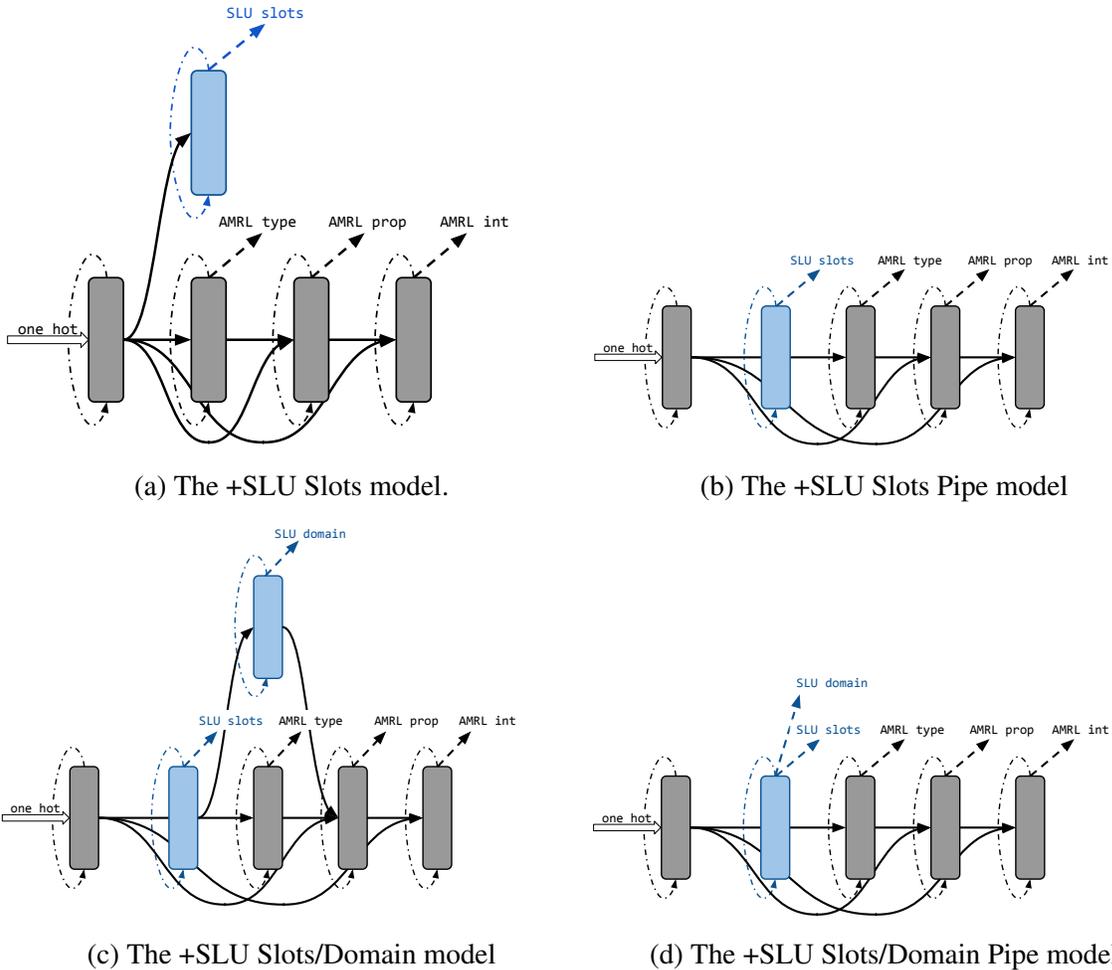


Figure 8.7: Topologies for the multitask models used for training AMRL models alongside the slot and domain embeddings from the existing SLU system. Highlighted in blue are the bidirectional LSTM layers that are not in the baseline model.

The remaining two topologies additionally leverage information from the SLU domains. Because we anticipate that the domains will have biggest effect on the choice of properties, we add a separate domain prediction stage and connect this to the property prediction stage. The domain layer is trained as a task in parallel, and it feeds into the property layer. The rationale behind this choice is that, although domain prediction and property prediction are very different tasks, the domain, when correctly classified, can restrict the kinds of properties that we expect (e.g., if a sentence is classified in the *Music* domain we expect properties such as *byArtist* but not these such as *weatherCondition*). A shared embedding space is still learned as input to the type model. This model (+SLU Slots/Domain) is shown in Figure 8.7c. The final model (+SLU Slots/Domain Pipe) has a common LSTM embedding layer for the SLU slots and domains. The hypothesis is that a single LSTM layer might be enough to encode all the information from the SLU representation. Figure 8.7d shows the topology for this network. Intent embeddings from the SLU problem are also added to initial models, but these do not result in significant performance gains.

We also explore adding word-embedding vectors and gazetteers (lists of entity mentions) to the input of the baseline. Word embeddings are provided as 300 dimensional pre-trained word2vec embeddings that have been trained on the Google News corpus of 100 billion utterances [52] and that are incorporated as additional input to the one-hot encoding of each word. We use gazetteers from the Alexa ontology (which is behind AMRL) in a similar way to the word embeddings (i.e., as an additional input per word). For example, a Musician gazetteer contains a list of musician names like “Sting.” These features are indicators that are set to 1 if the current word or word sequence appears in the gazetteer, or 0 otherwise. This model uses the same topology as the baseline (shown in Figure 8.5).

Decoding and Optimization

To decode the output of the DNN models and improve the accuracy of the results, we prepare a beam-search decoder to leverage two primary constraints. The first constraint is to ensure that the sequence of labels obey the IOB constraints. These constraints include:

1. that an $I-$ tag follows either another $I-$ tag or a $B-$ tag and
2. that a $B-$ tag follows only an $O-$ or an $I-$ tag.

The second constraint ensures that the final property label (e.g., the name, type or value) matches the initial one. For example, in Figure 8.4, if “ray of light” was predicted to be “type@Music-Recording” but its property was predicted to be “object.name,” then this would be an invalid transition.

The beam search limits the property candidates to those that are above the minimum probability. The value of the lower-bound probability determines the aggressiveness of the pruning. The search is performed on the set created by combining the property candidates with all the possible matching types.

We optimize the parameters of the networks in two ways. In the first, we fine-tune the pre-trained embeddings for the domain and slots. In the second, we perform joint training, for which the SLU embedding layers are learned at the same time as the other embeddings. Each model is trained until it fully converges on the training set; this typically takes around 60 epochs. We use a fixed learning rate of 0.0005 and an L2 penalty of 1×10^{-8} , as well as a batch size of 128 sentences each. The output of each bidirectional LSTM layer is a vector of size 256 that is created by concatenating the hidden representations of the forward and backward LSTMs (each of size 128). To prevent overfitting to our training set, we take two measures. First, we connect the output of each bidirectional LSTM layer to a drop-out component with a retention rate of 80%. The output of the drop-out component is then used as the input for the following layers. Second, we use weighted cross-entropy loss functions with a weight of λ when joint training (see below). We also evaluate our accuracy using the development set to show the generalization performance.

8.3 Experimental Results

To validate the approach in these experiments we use two datasets: (1) a large corpus collected for SLU and (2) a smaller corpus of linearized AMRL parses. The SLU corpus is composed of

~2.8 million unique sentences. These sentences span on the order of 20 domains, 200 intents and 200 slots. The vocabulary of this dataset amounts to ~150,000 distinct words. The representation for this corpus is shown in Figure 8.1. The AMRL corpus is significantly smaller than the SLU corpus and contains only around 350,000 unique sentences in the linearized representation. This data consist of intents, types and properties. Figure 8.4 shows an example of an annotation for the same sentence shown in Figure 8.2. The AMRL corpus spans more than 60 linearized intents, 200 properties and 200 types. The total vocabulary of this corpus is around 42,000 words, one order of magnitude smaller then the SLU one. The development set and test set contain around 48,000 sentences annotated using AMRL. Accuracy is reported only for the AMRL test set.

Four metrics are considered to evaluate our models. The first two, $F1_{IC}$ and $F1_{SC}$, are F1 scores that are evaluated at intent and slot levels, respectively. $F1_{SC}$ is a strong metric, as it requires the spans and labels for both the property and the type tasks to be correct.

The Intent Classification Error Rate (ICER) is inversely correlated with $F1_{IC}$ and it measures the number of incorrect intents. ICER is not always sufficient, as there may be multiple intents in an utterance. The formula for ICER is:

$$\frac{\#incorrect(intents)}{\#total\ intents}$$

Finally, the Intent Recognition Error rate (IRER) is computed as:

$$\frac{\#incorrect(interpretation)}{\#total\ utterances}$$

We consider an interpretation to be incorrect if any of the slots or intents differs from the ground truth.

Table 8.1 shows the results for individual model architectures. The baseline model is trained using only the AMRL corpus. The vocabulary is pruned, and only words that appear twice or more are used. Every other word is mapped to an “unknown” token, resulting in a vocabulary of ~25,000 words. Based on these results, it appears that using SLU tasks provides a clear benefit when training AMRL models; our best model (+SLU Slots/Domain) outperforms the baseline across all the metrics—in some cases, by a considerable margin.

We also compare the fine-tuning pretraining embeddings to joint training (learning the SLU embedding layers at the same time as the other embeddings). For pretraining, we train a network to predict the SLU tasks; once converged, the AMRL-specific components (i.e., the last three LSTM layers of each model) are added and are then trained using a cross-entropy loss until they reach convergence. Joint training optimizes both the SLU and AMRL tasks at the same time. At each time-step, a random training instance is selected from one of the two corpora with probability ($p = |AMRL|/|SLU|$). Because the size of the SLU corpus is much bigger than that of the AMRL one, we need to prevent overfitting on the SLU tasks. To do so, we use weighted cross entropy loss functions for which the SLU tasks are down-weighted by a factor $\lambda = |AMRL|/(10 \times |SLU|)$.

The models that use joint training have slightly higher accuracy than the others, but there are no conclusive results. For the first two models, pretraining appears to result in higher accuracy, but for the remaining two, the opposite seems to hold true. One possible explanation for this behavior is that the use of cross entropy loss function is used in conjunction with the joint training

<i>Models</i>	$F1_{IC}$	$F1_{SC}$	$ICER$	$IRER$
Baseline	0.9383	0.8439	6.4464	25.7176
+SLU Slots	0.9416	0.8551	6.1254	24.2876
+SLU Slots (jt)	0.9389	0.8449	6.4587	25.6867
+SLU Slots pipe	0.9432	0.8602	5.8867	23.1312
+SLU Slots pipe (jt)	0.9390	0.8456	6.3311	25.3534
+SLU Slots/Domain	0.9431	0.8614	5.9649	23.0222
+SLU Slots/Domain (jt)	0.9435	0.8621	6.0204	23.1147
+SLU Slots/Domain pipe	0.9351	0.8232	6.8270	29.9418
+SLU Slots/Domain pipe (jt)	0.9400	0.8538	6.1789	24.4501

Table 8.1: Models and their results. Models marked as (jt) are trained using the joint training approach.

approach. In the experiments, we fix the weight for the loss function; however, additional hyperparameter tuning might improve the overall result.

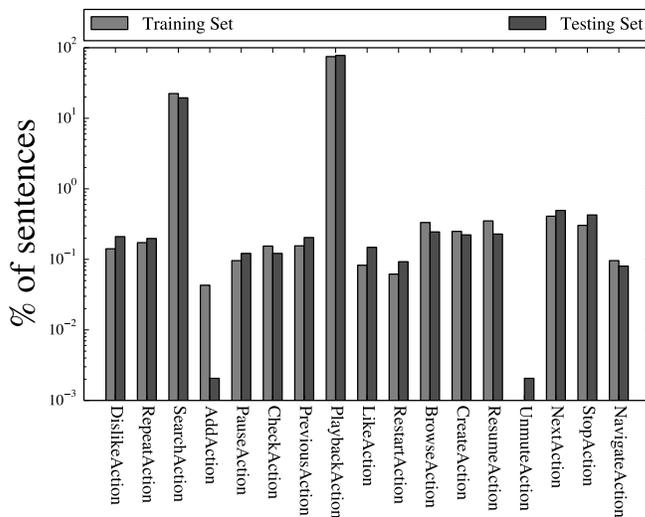


Figure 8.8: Action distribution of the training and testing sets for the AMRL corpus.

We also compare the accuracy of the baseline to that of the best performing model across various actions. Figure 8.8 shows that our dataset is skewed, with two of the actions (Playback and Search) covering more than 97% of the total training instances; the remaining 15 actions have an almost uniform number of sentences. Table 8.2 shows the gain in ICER for each action. The improvement is modest for the two most represented actions but is much more pronounced for the less represented ones. In general, we find that, when there is sufficient data available (i.e., for the Playback and Search actions), adding more information from the SLU task is not very helpful. On the other hand, when fewer training instances are available, the information that the SLU tasks provide is very valuable and strongly improves the results.

<i>Actions</i>	<i>Training instances</i>	Δ ICER
PlaybackAction	HIGH	-0.53
SearchAction	MED	-0.17
NextAction	LOW	-3.34
ResumeAction	LOW	-10.81
BrowseAction	LOW	+5.88
StopAction	LOW	-1.93
CreateAction	LOW	0
RepeatAction	VLOW	+3.13
PreviousAction	VLOW	-23.23
CheckAction	VLOW	-5.08
DislikeAction	VLOW	-1.96
PauseAction	VLOW	-11.86
NavigateAction	VLOW	-5.12
LikeAction	VLOW	-6.94
RestartAction	VLOW	0

Table 8.2: Δ at ICER. The HIGH bin contains more than 100,000 examples. The MED bin contains between 2,000 and 100,000 examples. The LOW bin contains between 800 and 2,000 examples. The VLOW bin contains between 100 and 800 examples. The actions are in decreasing ordered by the number of occurrences in the AMRL training set.

In Table 8.3, we evaluate the best models against those trained with word embeddings and gazetteers. The baseline model, which only has access to the AMRL training dataset, has the lowest accuracy. Adding gazetteer and word embeddings improves accuracy, but the best model is the one that transfers the learned representations from the SLU task (model from Table 8.1). Incorporating the additional constraints (e.g., IOB and final property) into the decoder results in the best model. For these experiments, we used a decoder with a beam of size 3 and a minimum probability of 10^{-7} . As expected, the decoder does not impact any of the intent metrics ($F1_{IC}$ and ICER).

8.4 Summary

In this chapter we introduce an approach Deep Neural Network-based approach to learn the semantic parsing for the AMRL. Our approach leverages the SLU representation, for which a large corpus of annotated sentences is available, to improve the semantic parser that was trained for the AMRL. Our experiments show that learning embeddings from related tasks (i.e., SLU predictions) can improve the accuracy of AMRL models. In particular, domain and slot embeddings help significantly improving accuracy by 3.56% in terms of IRER (full-parse accuracy). Moreover, a constrained decoder that leverages IOB and the type/property can further decrease IRER by 1% in absolute terms.

	$F1_{IC}$	$F1_{SC}$	$IRER$
baseline	0.93	0.84	25.71
baseline+emd+gaz	0.94	0.86	23.65
devised model	0.94	0.86	23.11
devised model+decoder	0.94	0.87	22.15

Table 8.3: Results compared to various baselines. The baseline model is the multitask model that is trained only on AMRL data. the baseline+emb+gaz model has word embedding and gazetteer features as inputs. Devised model is the best version without the word embeddings or gazetteer features. The devised model+decoder includes the results of the devised model after decoding (without gazetteers or word embeddings as input). Beam size is three and a floor probability of 10^{-7} is used.

Chapter 9

Related Work

Since SHRDLU [84], understanding natural language has been a long lasting goal for Artificial Intelligence researchers. The whole field of Natural Language Processing is dedicated to developing novel approaches for understanding, interpreting and manipulating human languages. In this chapter, we review the relevant literature—focusing on agents that interact with humans and, in particular, on agents (both real and simulated) that act in the physical world.

It is worth noting how, from a robotic point of view, a wide variety of hardware has been employed to study human-robot interaction through natural language. In this type of research, wheeled platform are, by far, the most common [12, 13, 18, 28, 35, 76]. Humanoid robots are also common choices [31, 76, 81]. Other, less common robotic platforms include quadrotors [38, 41], robotic arms [46] and automated forklifts [74].

Similarly, we find that the interactions between robots and humans can include a wide range of tasks. One of the more frequent goals is for robots to follow navigational instructions [2, 38, 41, 49, 50]. A closely related task is for robots to follow manipulation instructions (e.g., how to perform a pick and place task) [31, 76]. Robots that understand natural language are also often employed as guides in museums [12, 79] and malls [19]. Finally, the robots’ ability to understand language can be leveraged to develop personal robots [46] or robot companions [35, 72].

In the rest of this chapter, we analyze these related works in more detail. In Section 9.1, we look at human-to-robot interaction and focus on the various types of structures used to represent robot actions. In Section 9.2, we review the works on robot-to-human interaction. This direction of interaction, outside of conversational agents, has been less studied than its dual. Therefore, we extend our review to include computational storytelling and works pertaining to intelligibility or explanation of intelligent agents. Finally, in Section 9.3, we review some of the relevant work related to semantic parsing.

9.1 Human-to-Robot

In this section, we review the works on robot-to-human interaction. We identify three main approaches to mapping input sentences to robot actions: 1) representing the agent’s action with frame-like structures, 2) reducing the input sentence to a logical form that the robot can evaluate and execute upon, and 3) deriving a probabilistic model to map the sentence directly into the

robot's actions.

Frame-like Structures

Frame-like structures are a common approach to representing the robot's intended action based on an input sentence. In terms of Frame Semantics [29], frames both represent and generalize actions. Each frame provides a specific set of semantic roles that represent the various elements involved in the action. The use of Frame Semantics is general and not restricted to robot actions; FrameNet [3] is a semantic resource that is based on Fillmore's Frame Semantics; it has produced an extensive collection of frames as well as a large annotated corpus.

The BIRON robot [35], is intended as a robot companion for home-tour scenarios. BIRON parses input sentences using an LR(1)-parser and maps the input to frame-based XML-structures. If a structure is entirely instantiated, the robot executes the user's request. Otherwise, a dialogue manager tries to recover the missing information, using slot-filling policies to drive the interaction with the user.

In [49], the MARCO agent follows route instructions in virtual indoor environments. The structure used to represent the robot's actions is called *compound action specification*; this structure defines both what the robot has to do (i.e., it is an imperative model) and the condition under which the action should be taken. From an input sentence, MARCO first extracts the syntactical structure using a probabilistic, context free grammar. This structure (i.e., the sentence's syntactical parse tree) is then processed into a nested attribute-value matrix in which each word's meaning is resolved using WordNet [53] and then mapped to the compound action specification for execution.

Interaction Templates are used in [13] to map language to robot actions. These templates have two roles. First, they provide boilerplate instructions for the discourse subject; they also contain syntactic and semantic structures that the robot can use to map input sentences to actions. The focus of this work is on enabling the incremental processing of the user's input through the use of *discourse context*, which maintains the shared discourse history between the robot and the user.

RoboFrameNet [76] is directly inspired by FrameNet [3]. Its creators demonstrate the use of semantic frames to represent actions on two robots: an anthropomorphic robot (PR2) and a small wheeled robot (Turtlebot). In FrameNet a frame is intended as a scene being carried out, but in RoboFrameNet, a frame represents a plan for the robot to enact. When executing a user's request, the input is first parsed using a dependency parser. The extracted dependency graph is then processed using *lexical units*, which store the verb, the verb's corresponding frame and the set of mappings from grammatical relations to the frame elements. Using these lexical units, the input sentence is mapped to the robot's actions.

Logical Forms

A second, widely used approach is to reduce the input sentence to a logical form that can then be evaluated when answering queries or executing tasks. A wide range of logical forms can be used to meet this goal. In [11, 12], the Godot robot translates sentences into first-order logic statements using Discourse Representation Theory [39], which is implemented through Discourse Repre-

sentation Structures to allow for negation, disjunction, implication, and modal operators. Using these structures, the robot is able to answer questions about its state and about the environment by directly evaluating the logical formulas that it has extracted from the input sentence.

A similar approach is used in [46], in which the authors use description logic instead of first-order logic. The goal in this study is for the Jido robot to act as a helper in a moving scenario. To do so, the robot maintains an active knowledge base that provides a symbolic model of the world. This knowledge base is stored using OWL¹ Description Logic. The input sentence is first resolved using a custom-made, rule-based parser and then resolved into Resource Description Framework statements that can be evaluated against the robot's knowledge base.

The most common logic formalism for representing robot's actions is λ -calculus. In [2], a virtual agent follows navigation instructions in a simulated environment. The goal is to create a joint model for both context and semantics so as to enable robots to follow received instructions. Input sentences are parsed, using Combinatory Categorical Grammar, into λ -calculus statements. These statements, in turn, model the robot's actions as neo-Davidsonian events.

In [50], the goal is, once again, to follow navigational instructions in an indoor environment. The input sentences are mapped into Robot Control Language, a LISP-like formal language that is capable of representing the robot's operations in its environment. This mapping is realized using the Unification-Based Learner [45] that models both the syntax and the semantics (using λ -expressions) of the sentence.

The KeJia robot [17, 18], was first designed to compete in Robocup@Home [80], and was then deployed as a mall guide [19]; it also leverages λ -calculus. Its current goal is to provide route guidance and information services to mall customers. To do so, the authors use a Combinatory Categorical Grammar to map input sentences into λ -expressions that represent the semantics. These expressions are coupled with the robot's visual input and processed using a *spatial relations processing* module that lets the KeJia resolve expressions such as "the drink to the right of a food."

In [77] the Segbot robot understands users' navigation and delivery requests. These requests are, once again, mapped to a λ -expressions. The focus of this study is on enabling the robot to interactively learn the meanings of words through dialogues with users. The lexicon that the robot's semantic parser uses is initialized with a small seed of lexical entries (i.e., words-to-meaning pairs), and the robot builds on this lexicon through its interactions with users.

Finally, in [28], the authors combine λ -calculus with semantic annotations using temporal and dynamic logic. The goal is to enable robots to incrementally translate route instructions into formal logical descriptions. The robots could then use these descriptions to determine the achievability of a goal or to generate actions to reach its goal. The use of temporal and dynamic logic allows this formalism to specify conditions outside the agents' path or plans.

Probabilistic Model

Finally, we review works in which a probabilistic models are used to map human language to robots' actions. In [31], the goal is to remove the kinesthetic requirements of Programming by Demonstration. The problem of finding the right action to execute is framed as probabilistic

¹<https://www.w3.org/TR/owl2-primer/>

inference. In particular, the authors look for a generative model to provide natural language commands to the robot in a situated context. Therefore, the goal is to find the command $c^* \in C$ to maximize the joint probability distribution $p(C, S, L)$ over commands C , robot states S and language L .

In [41], the authors introduce Spatial Description Clauses (SDCs) to model the structure of language used in directions. Once again, the goal is to enable robots to follow navigation instructions. SDCs are composed of a figure, a verb and a spatial relationship; they are designed to capture the linguistic structures of directions. Each sentence that the robot receives as input by is modeled as one or more SDCs. Therefore, to follow its instructions, the robot needs to find the path, $p^* \in P$, to maximize the conditional joint probability distribution, $p(P, S|O)$, over the possible path, P ; the sequence of SDCs, S ; and the robot's observation, O .

The Spatial Description Clauses reviewed, were initially designed to provide semantic representations for instructions in a 2-dimensional environment. In subsequent work on SDCs [38], the same authors showed how they can be extended to the 3-dimensional case and demonstrate that their approach enables the control of an air vehicle via natural language.

Finally, the Generalized Grounding Graph (GGG) framework [74] builds upon SDC and further generalizes that concept. The previous that SDCs need to have a fixed, flat structure is not used in this study. Instead, this approach dynamically instantiates a probabilistic graphical model based on an input sentence. This model is thus able to preserve the command's hierarchical and compositional semantic structure.

Summary

The works that we review in this section take various approaches to mapping natural language to robots' actions but they share a particular focus on enabling robots to understand spatial language. Understanding such language is critical when a robot needs to follow directions, or when the user needs to provide step-by-step instructions. In this thesis, we assume that the robots are able to autonomously perform its tasks. Therefore, our attention is focused on enabling robots to understand high-level specifications of tasks for which spatial language is unlikely to appear.

9.2 Robot-to-Human

If we exclude conversational agents, the problem of robot-to-human interaction is less studied than its dual. Therefore, in this section, we review the relevant work in three related areas of research: 1) computational storytelling, 2) explainability of intelligent agents, and 3) robot-to-human interaction.

Computational Storytelling

In a survey of the field of computational storytelling [33] the author focuses agents' production of stories. The author also highlights that most systems' output is not narration in a natural language but rather a fabula (i.e., a conceptual description of what should be told). Moreover, the surveyed works are mostly interested in the creativity and unexpectedness of the generated stories. This is a fundamental difference with the problem we faced in this thesis. The output that

we aim to generate is not novel but instead is based on the data that robots collect in their logs. Therefore, rather than focusing on what these stories are going to tell, we are more interested in the actual storytelling process.

In [61], the authors are interested in Interactive Storytelling and are concerned with both scalability issues and real-time performance for a system that generates larger stories. The authors frame the problem of storytelling in terms of knowledge-based planning to show how using PDDL 3.0 [32], it is possible to encode narrative control knowledge as state trajectory constraints and to decompose planning problems using landmark planning.

This approach is further investigated in [62], in which the authors use the same approach to generate stories with multiple story-lines that advance at the same time through interwoven narrative. Relative to the previously described work, this study's focus on multiple subplots presents several challenges: how to distribute virtual characters across subplots, how long each subplot presentation should be and how to transition between subplots. This approach is more complex than having robots report on their execution. The robot is typically the only actor and a single story-line is enough to provide the information needed.

Finally, StatsMonkey [1] is a system for writing newspaper-style accounts of sporting events using Internet-gathered statistical data about the games. The statistical data are mostly in the form of play-by-play data or box-scores, and the output combines them using three types of narratives: extreme or rare events, thresholds crossing, and comparisons to expectations. These narratives are combined using *angles*, which are hand-designed rules that order the narratives by importance; this helps provide a big-picture view of the game. As in the case of a robot that has to report on the tasks it has executed, StatsMonkey needs to transform data into a description of an event occurred. The main difference between our approach and the one described in this work is that the input data for StatsMonkey are often already textual (e.g., play-by-play recounts) and do not require any grounding.

Explainability of Intelligent Agents

Much of the literature in the human-computer interaction research field is focused on intelligibility of intelligent systems (e.g., for context-aware systems [23]). Intelligibility is an agent's ability to explain its own behaviors and/or the reasoning for its actions. A lack of intelligibility can lead to failure due to lack of trust from the agent's users [54].

In [48], the authors present a study of over 200 participants. An agent that performs activity-recognition for an exercise (i.e., a wearable device that measures biometric information and classifies whether the wearer is exercising), can provide several types of explanations for its classification. The results of this controlled experiment show that automatically providing explanations for such a decision, and in particular, the reason behind it, can improve users' trust, satisfaction and acceptance.

The work presented in [43], takes this a step further by trying to address the question of how intelligent agents should explain themselves to users. The authors use a music recommender system to show how a user's mental model affects the user's ability to operate the system. In these experiments, the authors focus on the completeness and soundness of the agent's explanations. The results show that greater completeness in a system's explanation helps users to build more useful mental models. An explanation that is both complete and sound is the most successful,

but users perceive that a very sound explanation incurs in a higher cost.

Finally, [14] uses a Clinical Decision Support System to show how automatically generated explanations can increase the users' trust. A second interesting find is that *Confidence Explanations* (i.e., the system's reported confidence in its own diagnosis) do not seem to cause participants to increase their trust in or reliance on the system. On the other hand, *Why Explanations* (i.e., the facts that the system used in its reasoning when reaching the diagnosis) increased users' trust on the system—but not necessarily their reliance on it.

Robot-to-human Interaction

All the works that we reviewed in this section deal with virtual agents. In this last part, we focus on physical agents that interact with users. In [81], a pair of humanoid robots comment on a live game of robot soccer. The communication between these two humanoid commentators and the audience uses both speech and gestures. The robots summarize the game as it evolves by providing a *Game History* that only retains the most salient events. To create an entertaining but consistent commentary, the commentators' output is generated by combining two functions: first, a deterministic function that maps the events in the game history to a set of actions the robots can take; and second, a non-deterministic function that allows the robots to select one options from a generated set.

In [10], a NAO robot gives direction to visitors in a building. The robot generates a plan for reaching the visitor's destination by running A^* on an annotated map. The generated path, a sequence of segments describing the distance and direction of travel, is then translated into natural language and gestures. To avoid verbose directions, the system adopts a set of simplification rules (e.g., skipping insignificant paths, merging coincident paths, or short-circuiting longer explanations).

Finally, in [75], the authors present an approach that enables robots to recover from failures by communicating their need for specific help to a human partner. In the given setup, a robotic manipulator collaborates with a human to assemble furniture. This approach uses the Generalized Grounding Graph framework [74] and applies *inverse semantics* to generate requests by emulating humans' ability to interpret a request.

Summary

The works we review in this section come from three areas of research but they all relate to enabling robot-to-human interaction. We share with the computational storytelling field an interest in how agents tell stories—specifically, how they report on the tasks that they execute. The works from the human-computer interaction literature that pertain to the explainability of intelligent agents reveal that a robot must have the ability to explain its choices if it is to be accepted and trusted in everyday life. Finally, we review works on physical agents (i.e., robots) that are able to initiate interaction with users whether this is to provide commentary, to give instructions or to ask for help.

9.3 Semantic Parsing

A semantic parser is a function that maps an input sentence into a formal representation of that sentence’s meaning (i.e., its semantics). There are two forms of semantic parsing. The first, shallow semantic parsing (or semantic role labeling) was introduced in [29] and [34]. Semantic role labeling aims to identify the entities in the sentence and to labeling them with the roles that they play. We use this approach in our semantic parser in Chapter 3 and Chapter 5.

The second form of semantic parsing that we identify is deep semantic parsing (or as compositional semantic parsing); it is intended to produce a deeper semantic analysis than semantic role labeling can provide. Deep semantic parsing provides a representation of a sentence using predicate logic or another formal language that supports automated reasoning. Examples of deep semantic parsing include the works that rely on λ -calculus in Section 9.1 (e.g., [2, 18, 50]).

The function that maps a sentence to its meaning—the semantic parser—can be learned from data. This is not an easy task, and the method used is strictly connected to the formalism that is chosen to represent the semantics. For semantic role labeling, in which the input is a sequence of tokens (i.e., words) and the output is a sequence of labels, the typical approaches involve using Hidden Markov Models [78], Conditional Random Field [69]. In [27], the semantic parsing of causal relations is achieved using the Construction Grammar formalism (CxG). The aspects of each construction’s form are learned using a Conditional Random Field. Finally, Deep Neural Networks [25, 70] have recently become common tools for semantic parsing.

The methods for extracting a sentence’s the compositional semantics vary based on the representation. As we have shown if the representation uses λ -calculus, a common approach is to use combinatory categorial grammars (CCG) [73]. These grammars are composed of two main parts: the combinators, which are predefined, and the lexicon, which can be learned from the data. The approaches that have been devised to learn CCGs’ lexicon range from fully [85, 86] to weakly supervised [20, 47].

Another formalism that is commonly used to represent a sentence’s semantics is the Abstract Meaning Representation (AMR) [4, 26]. For example, the algorithm introduced in [30] represents an AMR parse as a graph. This approach identifies concepts using a semi-Markov model and then extracts relationships between these concepts by searching for the maximum spanning connected subgraph from an edge-labeled, directed graph that represents all possible relationships between the identified concepts.

Most of the approaches we review above, whether for semantic role labeling or for the extraction of compositional semantics, require labeled data in the form of paired token sequences (i.e., the words in a sentence) and their semantic parses. However, word-by-word annotations of large corpora are expensive. Therefore, efforts have been made to develop algorithms that allow for weaker forms of supervision. In [47], a semantic parser is learned from question-answer pairs. More recently, deep learning approaches for training a semantic parser with weak supervision have leveraged sequence-to-sequence[40] and sequence-to-tree[24] models. Finally, it is worth noting that semantic analysis can also be used for tasks, beyond semantic parsing, such as grounding topic models from documents into an entity taxonomy from a knowledge base [37].

Most of the approaches in this section require large amount of labeled data for training. This type of data is scarcely available in the robotics domain, however the lack of data for this domain,

coupled with the fact that we only need a limited set of labels, led us to use Conditional Random Fields and semantic role labeling in our approach to human-to-robot language understanding.

Chapter 10

Conclusion and Future Work

In this chapter, we first summarize the contributions of the thesis. We then discuss some possible avenues for future work and we conclude with a brief summary of this thesis.

10.1 Contributions

This thesis's key contributions are the following:

KnoWDiaL is our approach enabling a mobile service robot to understand task requests. KnoWDiaL relies on three main components: a semantic parser to extract the task and its argument from the input sentence, a Knowledge Base to store the groundings of the natural language expressions, and a dialogue manager to add and update the groundings that are stored in the Knowledge Base. KnoWDiaL uses dialogue to ask the user for the groundings of unknown natural language expressions. Our experiments show that the number of questions asked decreases as the robot stores more groundings in its Knowledge Base.

A template-based algorithm to understand complex commands. We define complex commands as sentences that refer to multiple tasks or those for which at least one of the task arguments can be instantiated in multiple ways. The algorithm that we introduce relies on templates: syntactic structures (to be found in the syntactic parse of the sentence) and corresponding procedures to break the sentence into simpler components. The algorithm recursively searches for templates and applies the matching procedure. The resulting simple commands are then grounded using KnoWDiaL. We introduce two dialogue that the robot can use to recover from possible errors of the template-based algorithm. Finally, we show how breaking a complex command into simpler components allows the robot to generate execution traces (i.e., distinct ways to achieve the same commands) while minimizing the cost of executing a complex command.

Log Primitives Operation to address users' questions about what the robot is currently executing or what it executed in the past. Log Primitive Operations expands the robot's capabilities not only to execute task but also to answer questions about the robot's previous experience. To do so, we introduce a novel use for the robot's logs, which are not only a tool for use in debugging

but also a source of working memory that the robot can query to answer users' questions. Log Primitive Operations are defined in terms of the operations that are to be performed, and a set of filters that reduce the number of records under consideration. The robot understands users' questions about its past experience by grounding each sentence into an LPO in the same way that KnoWDial grounds a sentence into a task to be executed.

A crowd-sourced on-line study to gather a corpus of 2,400 Verbalization requests. Using this corpus, we trained a model to predict points in the Verbalization Space. The robot uses the points that the model predicts to answer verbalization requests and generate descriptions that match the users' requested level of detail. Furthermore, the robot can use the same model to move within the Verbalization Space so as to enter into an iterative dialogue and satisfy subsequent requests to change the descriptions that it provides.

Comparative Templates to enable the CoBot robots to proactively report on task execution and to motivate their choices when executing complex commands. Using Comparative Templates, the robot can provide concise yet informative explanations using comparative language (e.g., "longer than usual"). To do so, the robots compare and contextualize the current tasks' execution with the expected times. The robots derive the expectation for any given task by querying the logs for past instances of the same task and building a probabilistic model.

10.2 Future Work

In this thesis, we addressed the challenges of enabling a bidirectional, language-based interaction between users and mobile service robots. We explored both directions of this interaction, from user to robot and from robot to user. There are multiple possible directions for future work but we believe two are the most interesting. The first involves enabling users to actively query and modify the robot's Knowledge Base. The second involves combining Log Primitive Operations in increasingly complex queries of the robot's logs.

Enabling the user to actively query and modify the robot's Knowledge Base. In Chapter 3 and Chapter 5, we enable the robot to understand the users' requests. Our approach uses a Knowledge Base to map natural language expressions into internal symbols such as map locations. The Knowledge Base is used to store information about the environment where the robot is deployed so that this information can be reused. An interesting question relates to what happens when the environment suddenly changes. Consider the following scenario: throughout a year of deployment, the robot has been repeatedly sent to the "small size lab," which is office F7412. Therefore, the Knowledge Base has accumulated a strong confidence in the predicate `locationGroundsTo("small size lab", F7412)`. During the summer, however, this lab moved from room 7412 to room 2109. In our current approach, it will take a long time for the robot to accumulate enough evidence to offer F20109 instead of F7412 as the new grounding for "small size lab." an alternate solution for this situation occurs is to enable the user to correct the information that the robot has stored. To do so, two main challenges need to be addressed. First, we need to identify which operations users can perform on the robot's Knowledge Base.

Second, the robot needs to learn a model so that it can map its interactions with users to these operations.

Combining Log Primitive Operations into more complex queries of the robot logs. In Chapter 5, we introduce Log Primitive Operations that enable a robot so that it can autonomously search its logs to answer questions about its past experiences. Now, consider a user asking a robot the questions, “Was this week busy?” To answer this question the robot could COUNT the number of tasks executed each week, average that figure (AVG) and compare with it the number of tasks executed in the last 7 days. If the number of tasks in the last week is greater than the average, the robot would answer positively; otherwise it would answer negatively. The operations that the robot has to perform (COUNT and AVG) are already defined in terms of LPOs, so the user could instruct the robot to combine these separate operations to answer questions that do not match a specific LPO. There are two main challenges in this type of interaction. First, the robot needs to be able to understand step-by-step instructions. Second, once instructed on the sequence of operations it is to perform, the robot must be able to store and reuse that sequence when prompted with the same request. To this end, the robot needs to learn to map between sentences and structures storing the sequence of operations it has to perform, such as Instruction Graphs [51].

Appendix A

Corpora

A.1 Task Corpus

The corpus used to train the semantic parser of Chapter 3.

Could you please bring me to 4470?

I have papers to be brought from Dr. Mercili's office and Professor Harper's office. Tell them you've come to bring them to me.

Can you get me a chocolate chip cookie from the 3rd floor cafe if they were baked today.

Please bring Mr. Clark from the elevator to room 6332.

Please take my homework and give it to my professor in the von Ahn awesome classroom.

Could you take this box to the kitchen?

Please go to 7008 and say hello from Diana.

Please bring up that whole tray of freshly baked cookies and get someone to place them on the counter in the kitchen.

Here's my laptop, bring it to the elevator.

Retrieve the signed contracts from rooms 3201, 7708, and the 7th floor conference room.

Go and buy a bottle of water and give it back to me.

CoBot, please get my print out from the printer room in 7128, and my yogurt, Stonefield, plain, from the refrigerator in the 7th floor kitchen next to the printer room. Thanks!

Please bring this envelope to Christine to be mailed.

Go get me a coffee from the kitchen and bring it to my office.

Can you get me a soda.

Pick up the paper drafts from Brian, Joydeep, Tom, Cetin, and Manuela and bring them to me.

Please bring me scissors.

Please take my dad to Rashid Auditorium.

Please find me a bottle of water.

Put these papers in the recycling bin.

Bring me a soda from the kitchen.

Bring me some paper towels.

Please bring me the cookies from the kitchen counter.

Tell Manuela, "Hello."

Return to your home.

Escort me to the 9th floor conference room.

Can you please return this envelop to Diane's office.

Please buy a blueberry yogurt, an apple, and a coffee from Entropy with my ID card.

Please take this folder back to Sophie in 7025.

Please say "congratulations Brian, the Kiltie Band performance was amazing!" to Brian Coltin in GHC7412.

Return to your home.

Go to the office and tell her I said hello.

Please bring me a tall latte from the 3rd floor cafe.

CoBot, bring me some water.

Find the nearest pencil and bring it back here.

Go get ice cream for all the people in the Gates and Hillman Center.

Tell Manuela that I'll be running a few minutes late.

CoBot, can you please find a bottle of water for me? It could be in Gayle's office or in the kitchen.

Get me some sugar from the canteen.

Wait outside the elevator for our guest, and bring him to the lab.

Pick up a bottle of water from Gayle and bring it here.

Please grab my tea from the cafe and then take my transcript from Mark Stehlik.

Could you bring a cup of hot water to me?

CoBot I have some treats you can take back to the lab.

CoBot, please take this memory stick to the CORAL lab.

Can you take this screwdriver to the lab?

Go get a coke for me.

Can you take these worksheets and bring them to room 5501.

I need you to get some flower pots from the 4th floor corridors for me.

Please take this folder back to Sophie in 7025.

Please deliver this letter to my advisor.

Please collect the time sheets from Johnson, John and Jon and bring them back to me.

Can you transport these envelopes to the CSD main office?

Please take this letter back to Diane's office.

Could you please bring me to the Rashid Auditorium?

Please take Nicole to 6105 GHC.

This box goes to Manuela, thank you.

Go to the printers and bring the printouts to my office.

Please bring Tim's ID card to him. He is in

Gates fourth floor 4407.

Carry 3 cups of coffee from room 6501 to room 5201.

Can you tell Professor Oak that he missed his lecture today?

Can you take this, this, and that from this, this, and that place.

Can you check if the 3rd floor cafe has freshly baked cookies.

Please take this envelope to 6105 GHC.

Can you get me a pencil?

Can you retrieve the files from my office.

Please bring this mail to Professor Kesden's office.

Could you deliver to Manuela this write up?

Please say hello to Manuela Veloso.

Please bring this USB key to Joydeep.

Can you return these documents to Diane.

Please bring me my printouts from the 9th floor printing room.

I am thirsty, bring me a cup of water.

Please help me send this book to JP.

Bring me a pen.

Can you go to the the kitchen and bring me a cup of coffee?

Take this box to the lab.

Please get me some coffee.

I need you to pass in rooms X, Y and Z to collect x,y and z in the respective room. Bring the objects here afterwards.

Please collect all the desks from all classrooms in the Gates and Hillman Center and bring them back to me.

Please go to Professor Kosbie's office and tell him "Carpe diem."

CoBot I have some treats you can take back to the lab.

Here is the paper for Tom, thanks!

Can you go back to the lab.

Pick up some water from room 7412 and bring it to room 7002.

Deliver the books at the lab to Manuela's office.

I need you to deliver this urgent document to

Brian.
Bring the box in your basket to Manuela.
Provide Robin with paper.
Can you please put this paper in the recycling bin.
CoBot, take this camera to Som.
Please bring me some water.
Please, bring me water
Can you bring me one slice of the free pizza?
Please bring me a screwdriver from the small size lab, and a LCD screen from the multi-robot lab in 3rd floor
Go to room 4405 and tell the people there that the lecture will be cancelled.
Bring Manuela a pencil.
Greet Eduardo at the elevators and bring him to my office.
Go find some envelopes for me.
Give to the Dean this letter.
Carry me to the 6th floor of the Gates and Hillman Center.
Can you retrieve the files from my office.
I need to get to the Rashid Auditorium.
CoBot, can you please tell Stephanie that I will work on the rebuttal now?
Can you please bring some water here?
Can you go to Gate 7021.
Please send a message to Mehdi to ask him if he'd like to meet now.
Can you go tell Christina I said hello she's in Gates 7008?
Take this book to the lab.
Go get our visitor from 6789 and bring him to my office 7505.
Deliver the mail to room Manuela in Gates 7001, Mehdi in Gates 7002, and Gayle in Gates 7008.
I need you to take a paper from Vittorio.
Fetch the coke from the 6th floor kitchen refrigerator. You will need to ask for assistance to open the door.
Please bring me some cookies from the ML Tea.
Could you please bring me the tickets for lunch

from Christina.
Send these documents to Manuela.
Could you get me a fork from the kitchen?
Please escort me to the kitchen after you finish bringing the cookies there.
Please tell Nicole, "meet me at the 7th floor elevator in 10 minutes."
Please fetch my printout.
Can you fetch my documents from Christine's office?
Please bring me to the nearest exit.
Please find my phone and bring it back to me.
Please tell her I said hello.
Please pick up a Coke from 6105 GHC, an envelope from 6107 GHC, and a bag of M&M's from 6203 GHC.
Please go to the third floor and tell the cafe workers to give you a tray of freshly baked cookies.
Find me a highlighter.
Take this package to Robin.
Deliver this cup at someone in the kitchen.
Please take this paper to Manuela.
Can you get some cookies for me from the cafe?
Please tell Johnson in Room 6441 that I will be gone this afternoon and won't be able to come to the meeting.
Please tell my friends Rich and Susie at the Reception area on the 5th floor to come to the 3rd floor Cafe.
Please send this folder to Johnson in room 5445.
Please welcome Rodney Brooks by the 5th floor elevator precisely at 2:00pm tomorrow and then escort him to my office.
Pick up mail from all the secretaries' offices and bring it to the main office.
Please tell my Concepts TA in the bottom helix classroom that I will be late.
Bring this package to room 7013.
Could you please find me a staple?
Please pick up the envelope from Theresa in 6105 GHC.

Go back to Diana.

Go to the conference room now.

Please find Vittorio and send him this tablet.

CoBot, please escort this person to Cetin Mericli's office on the 9th floor.

Please go to Professor Kosibie's office and tell him a lecture room is booked.

Can you fetch me a cup of tee?

Please get Tom a coke.

Please get milk from the refrigerator and cookies from the kitchen counter and bring them to me.

I would like espresso coffee.

Please say hello to Manuela Veloso.

A.2 Complex Commands Corpus

The corpus of complex commands used for Chapter 4. Each sentence is annotated with its complexity level.

- 1 Could you bring me the closest thing to coffee that you find within 20 minutes.
- 1 Make sure I am in my office before bringing me coffee.
- 1 Occasionally bring me coffee but never bring me tea.
- 1 CoBot please deliver this document to room B7021.
- 1 CoBot, could you please bring me something to eat.
- 1 CoBit, could you please escort to the lab the guest that is waiting in front of Manuela's office.
- 1 Could you please escort here the guest that is waiting in front of the elevator of the 5th floor.
- 1 CoBOT, you are bothering me, get out of here.
- 1 Escort this visitor to the nearest restroom.
- 1 Can you tell Manuela that I will be running 10 minutes late for our meeting at 2 pm.
- 1 The 3rd floor lab has taken our gaffer tape and screw driver, please bring it to the 7th floor lab.
- 1 I am in need of a staples, can you please ask around for staples?
- 1 Bring some coffee to my office, please.
- 1 Ask Richard to go to Vittorio's office.
- 2 Can you escort me to either the lab or Manuela's office, whichever is closer.
- 2 Unless we are in November, go to Office X everyday at 11 AM.
- 2 Please bring me tea and water for my visitor.
- 2 Go to Christina's office and ask her to get a new pizza for the CORAL meeting.
- 2 Could you please bring me a soda if Tazza d'oro is open.
- 2 Tell Vittorio to grab the ping pong ball and take him to the lounge.
- 2 If you have no tasks scheduled, go explore the 5th floor.
- 2 If the noise in the creative commons is above Y, tell everyone to be quiet and tell me they have been quieted.
- 2 If Manuela is in her office, ask her to meet me in the lab.
- 2 If there are visitors in the 3rd floor lab, escort them to the 7th floor lab.
- 2 Go to the lab and then to my office.
- 2 Go to my office or to the lab.
- 2 If my door is closed go straight to the lab.
- 2 Please bring me some coffee or tea.
- 2 Please bring this book to Steven or Brian.
- 2 Please go to the lab and bring back some candies.
- 2 Please tell Vittorio that I have given him a list of 10 commands and then bring me some candy.
- 2 Please deliver coffee and bagels for the 2 pm meeting.
- 2 Please escort this visitor to the 7th floor lab and then the 3rd floor lab.

- 2 Please tell JP that he should submit his 10 commands and then remind Kim afterwards.
- 2 Can you tell me if Vittorio is in his office.
- 2 If Vittorio is not in CMU, tell Manuela he's sick.
- 2 Go to Vittorio's office and tell him to move to other table, because I prefer his spot there.
- 2 Escort me to my new office, and introduce me to everyone there.
- 2 If my new office is in the same hallway as Manuela's office, tell her that.
- 2 Go to Richard's office and tell him your internet connection is much better now.
- 2 Get me coffee from the 9th floor kitchen if the good kind is there. Otherwise, get me tea.
- 2 If Manuela is in her office, come back here and tell me so.
- 2 Tell Manuela that I need her to come to the lab, unless she is already on her way.
- 2 Tell Joydeep that I need to borrow his pen and his notebook.
- 2 Ask Vittorio whether he has enough data for his experiments or he needs more.
- 2 Ask Vittorio if he wants to go to lunch with me or grab a coffee somewhere.
- 2 Bring my notebook to Vittorio and tell him to give it back to me when he is done.
- 2 Go to Vittorio's office and tell him that I will be late for the meeting.
- 2 Ask Manuela if she is thirsty and if she says yes bring her tea.
- 2 If Vittorio is not in a meeting, tell him that I would like to get coffee with him.
- 2 If outside is dark, tell me to stop working and go home.
- 3 Could you go near location X and search for coffee there only if you detect at least one human within one minute.
- 3 If the person in Office X is there, could you escort me to his/her office if you're not too busy.
- 3 Escort a visitor to my office, and then if she is thirsty, bring a bottle of water from the kitchen to my office.
- 3 If there is no one in the kitchen in the 7th floor, go to the kitchen in the sixth floor and bring me tea.
- 3 Please take this visitor to the lab, and then to Vittorio's office, and then to the third floor lab.
- 3 Go to the main office, ask if my package already arrived, and then take it to the lab.
- 3 Go to the lab and if the students are there tell them that they are late for the weekly meeting.
- 3 Go to Christina's office and if she is in there tell her that we are out of candy in the lab.
- 3 Go to the lab and if the students are in there, please escort them to Manuela's office.
- 3 If there is a box of papers outside Toms office, bring 30 pages to GHC 4215 and email me that the papers are there.
- 3 Get me a coffee and if they have raisin scones, a raisin scone.
- 3 Go to Tazzo or La Prima and get me a coffee.
- 3 Look for me in the lounge if I am not in the office or in the lab.
- 3 If my door is closed look for me in my office and then in the lab.
- 3 If Manuela's door is office or if you can hear her on the phone go back to the lab.

- 3 Go to the lab or, if the door is closed, go to the kitchen.
- 3 I am looking for either Joydeep, JP, or Danny. Please tell them I need to borrow their key to enter the 7th floor lab. I will be waiting in my office.
- 3 Bring the video camera and tripod to the 3rd floor lab, if I am not there, please try my office.
- 3 If no one is in the third floor lab, go to Manuela's office and tell her no one is working.
- 3 Go to GHC 4401. Stay there as long as there is a large crowd of people in the hallway, then go to the lab.
- 3 If Manuela is not in a meeting or in a meeting with a student, tell her the package has arrived.
- 3 If the visitor is at the elevator and he is not talking to someone, escort him to Manuela's office.
- 3 Go to the supply room and if you find a stapler, bring it to me.
- 3 Bring me a sandwich and a coffee if the coffee shop is open.
- 3 Go to Vittorio's office, bring him coffee and then escort him to Manuela's office.
- 3 Tell Manuela that Vittorio is looking for her and if she is not there go to Vittorio's office and tell Vittorio.
- 3 Ask Manuela if she wants coffee or tea and then bring her what she wants.
- 3 Escort me to Vittorio's office, if he is not there, escort me to Manuela's office.
- 4 I want you to escort me to Office X, but if I am not here within 10 minutes, deliver a spoken message to Office X saying that I will be coming later.
- 4 If it is Tuesday today, bring me coffee at 10 AM; otherwise bring me whatever you want at 11 AM.
- 4 When it is sunny bring me ice cream; in all other cases, do not bring me anything cold.
- 4 Please go to the lab, and if the door is closed, go to Joydeep's office, and ask him to send me the memory stick.
- 4 Take this visitor to Manuel's office, and if he is not there, take the visitor to the lab, or to Manuela's office.
- 4 Go to Manuela's office and if the door is open come back and say that Manuela is in her office.
- 4 Please go to to the kitchen or the coffee shop and, if you find coffee in either, bring it to me.
- 4 If someone enters the room, ask if they would like coffee, and if they say yes, get them coffee.
- 4 Escort these visitors to the creative commons, Manuelas office, and, if no one is in the 7th floor lab, the lab.
- 4 I need some candy from the 7th floor lab, can you bring them to my office at 2 pm, if I am not there, please try again at 3 pm.
- 4 Cobot, if Tiago is in his office, escort Manuel there, otherwise get him to the third floor lab.

- 4 Go to GHC 7723, then 7517, then 7415, then 7001. If you observe a human while doing so, say "Gates will be closed tomorrow from 7 to 8 a.m. due to a scheduled power outage".
- 4 Go to the lab if Joydeep is there, or to his office otherwise.
- 4 If Vittorio is not in a meeting, tell him to come to my office, then go to Manuela's office and tell her the same.
- 5 I need you to first bring me a cup of tea, or a bottle of water, or a soda, and then go to Christina's office and ask her to order more bottled water.
- 5 First go to the lab, and if there is noone there, go to Richard's office, and if he is there, bring me the book I ask him for.
- 5 Get a package from Manuela if she is in her office, then get a package from Christina if she is in her office, then deliver your packages to the lab.
- 5 If the door to the lab is open and there are two or more CoBots inside, deliver the message that the robots need to be deployed, then go to Manuela's office and tell her that her students are slacking off.
- 5 If Manuela is in her office, ask her when she wants to go to lunch, go to Vittorio's office and tell him her reply, then come back here and tell Manuela's reply to me.
- 6 Go to 7101 and 7102 and 7103 and 7104 or go to 6101 and 6102 and 6103 and 6104.
- 8 If Christina is in her office, pick up a package from her, deliver it to Manuela, then go to the lab and say that the package has been delivered. Otherwise, go to the lab and say that the package has not been delivered.

A.3 LPO's Corpus

The corpus used to train the semantic parser of Chapter 5.

How many times did you take the elevator in the past two weeks?
What's the fastest way to get to the 7th floor kitchen from here on a Wednesday afternoon?
How many people did you escort this month?
During the year of 2016, how many escort tasks to Manuela's office did you do?
How many people did you escort yesterday?
How many tasks did you perform on the seventh floor yesterday?
How many tasks did you execute in the last week?
How many people did you see yesterday in conference room 7007 ?
To which rooms did you escort people to?
How long did it take to escort John from the elevator to 8102 (Manuela's office) yesterday?
How many files did you deliver to GHC 8002 last Friday?
What were you doing at 11 AM yesterday?
On average, how long does it take to escort?
How many tasks have you successfully accomplished last month?
How many people blocked you on the way here?
What items did you picked up yesterday?
What rooms did you go to yesterday between 1pm and 2pm?
Where were you yesterday at 4pm?
Which kinds of task have been more requested during last week (or month)?
How long did it take you from 4405 to 8001?
How long on average do you execute a FindAndFetch task in the last 3 weeks?
Where were you just now before you came here?
Where did you start before you came here?
Have you been blocked by humans more frequently on the fifth floor or on the sixth floor?

How many objects did you delivered in the past month?
Why did it take so long for you to get here?
Are FindAndFetch tasks more frequent during the day or the afternoon?
How many times did you go to the 9th floor of GHC in 2014?
Which task do you perform the most?
Who requested the most tasks this week?
Which task did you complete with the highest average speed?
Did you escort somebody to room 8002 yesterday between 4pm and 5pm ?
Which object where you told to find and fetch the most?
For how long have you been blocked on the way to my room?
What is the fastest time so far that you have taken to recover from a blocked situation?
How many people do you escort every day?
At what times were you blocked in the past 24 hours?
How many task have you failed to accomplish last month?
When passed by the Kitchen on the 7th floor today, was anyone there?
Did you stay more than 3 hours in room 6020?
Have you ever been to room 1775?
Could you list all the objects you found until now?
When was the busiest (peak) hour for you, during a week?
What have you done yesterday afternoon?
On average, how many times do you get blocks in the last 10 FindAndFetch tasks?
How many people do you see right now?
Where are you going next?
What is the most popular time of day for FindAndFetch tasks?

Did you delivered anything yesterday between 10 and 11 a.m.?

What's the most frequent task you perform?

How long was the longest task you executed and why was it that long?

How long did it take on average to navigate from Manuela's office to the lab in November 2014?

How many people did you see on your way to the kitchen?

Did you have more tasks in the morning or afternoon yesterday?

Did you ever go to room 3210?

How long does it take to get a coffee, visit room 8002 and back ?

How many times were you interrupted when you were escorting?

How many times did you pass by room 8100 in the past week?

How many people did you serve coffee today morning?

Have you escorted anyone to 7101?

What is the total distance covered including all task for the last 24 hours?

How many objects have you picked up in a month?

What was the longest path you had to run on a FindAndFetch task? When was that, and what were you transporting?

Where do you go most of the times?

Where were you at 5pm last Monday?

How many times did you escort people?

How many tasks did you do since last week?

How many people have you met the day before yesterday around 3:00pm?

What is the probability that you encounter 2 humans within the first 50 meters (from starting point) in the last 100 tasks?

Where did you find the coffee?

Will you stop at 7101 before going to 7202?

What was the last place you escorted someone to?

Did you delivered something to the room 7002 this past week?

How many times have people asked you to bring coffee to them in the last month?

What is the percentage of objects you successfully found and fetched?

Were you blocked by a person last time you escorted a visitor to Manuela's office?

When was your last task?

Did you successfully escort someone to room 7412 at noon?

Which object results in more failures of fetch tasks?

Was the coffee machine near room 7017 blocked around 3 pm to 4 pm ?

Which floor do you mostly work on?

It took you a long time to fetch me a coffee, for how long did you wait until someone came and put the coffee in your basket?

How many people did you see on your way while you were escorting visitors last week from the 8th floor elevators to GHC 8002?

How long did the last FindAndFetch task take?

In what hour yesterday did you detected the most people?

How many different kinds of objects have you picked up?

What objects did you delivered today?

Did any object fell while you were trying to get it?

From where did you escort person X in the past week?

What is the most visited location on the map? how frequently did you ask for a human help? (in a week, month, etc.)

Have you picked up something for Manuela on Monday?

Please present the positions where you detect a human in last 5 FindAndFetch tasks.

Who are you bringing the coffee for?

How long did you wait before someone let you in the elevator?

What fraction of the time have you been idle in the past week?

How many humans have you detected yesterday?

Were there a lot of people on the bridge when you came here?

How many times did you get blocked for more than 1 minute in the past 30 days?

How many find-and-fetch tasks did you do in 2013?

When is your next task scheduled?

What is the most popular object to find for find and fetch tasks?

Do humans usually block you?

Did you see anybody at the desk by the elevator yesterday ?

Where all do you stay when you are idle?

Did you go to all the rooms on the 8th floor yesterday?

which was the office closest to you when you got blocked last time?

Did you every pick anyone at the 8th floor elevator?

What was the total time blocked for navigation for the previous week?

Why you failed to accomplish a task?

To whom do you usually deliver letters?

What was your most time consuming trajectory?

What was the last object you fetched?

Where are all the objects in the environment now?

How many (miles, kilometers) did you travel during a week?

Have you ever been in Coral Lab last week?

What is the average distance of the GoTo tasks in last 3 weeks?

Are you navigating alright? Anyone in your way?

Ask someone to give you coffee and bring it to my office

What is the greatest number of humans you have ever detected at once?

How long did you wait last time you were at an elevator hall?

Where did you drop off your last escorted person?

What percentage of time do you spend doing nothing?

Where can I find the closest coffee machine ?

Which position did you interrupted most number of time?

Did you go to Manuela's office in the past two hours?

How many days did you have to work after 5 pm last to last week?

What are all the deliver locations for FindAndFetch?

Have you failed to go to a place? Why?

What is the floor of the building where people block you the most?

When did you escort person X to room 1573?

How many tasks did you accomplished(failed) last month?

On your way to 7202 give this envelope to XXX in 7101.

In the morning, do you generally spend more time in the Gates side of the building or the Hillman side?

To where and when was your last Find-and-Fetch task?

How many humans did you encounter in your last task?

What was the average duration of finding the object in FindAndFetch?

Who is most helpful?

Did you take Dr. (Mr. , Ms.) So and So to Prof. ZZZ's Office (or to 7101)?

What was the longest time you have taken to complete a FindAndFetch task?

A.4 Verbalization Corpus

The first 240 sentences in the corpus of verbalization request used in Chapter 6. The whole corpus can be found at <https://bit.ly/2vJYV63>

Robot, give me a shorter version of the information you just gave me.
Can you give me a summary of what you compute?
Please recount your journey to my office without any mention of how long each step took you.
What path did you take when you traveled on floor no 7?
How did you get from where you started to here?
Robot, please simplify your information for me.
Robot, just tell me how you got here from the elevator; ignore everything before.
How should we travel to this current location.
Robot, give me all of the steps you took and the times. But after each step, hit enter to start a new line
Can you give me a more brief description of how you came to my office?
How did you get from the elevator to here?
How did you travel from the elevator to my current location.
Please tell me exactly how you got from your last stop to here.
Recount your path one minute before and one minute after the elevator.
Tell my briefly how you got to my office
Give me an easier answer for the last question I asked you
give a detailed summary for each path
Can you please provide me with your path and each place that you have visited?
Explain how you got here as simple as possible.
Give me the details of your trip beginning 1 minute before the elevator and ending 1 minute after the elevator.
Can you give a more thorough version of the path you took to get here?
Please recount path to my office, leave out any time stamps.
can you give me a brief version of the path you took?
Robot break down the path you took by telling me each time you had to change direction.
Please give me an easy to read and comprehend version of your path here.
Show only the steps taken starting near the elevator
Could you give me a more thorough summation of your path?
Can you sum up your journey in one sentence?
Robot, what route did you take to arrive at my office?
In layman's terms, describe how you got to my office.
Now could you tell me about the rest of your path, including near the elevator?
Give me the location descriptions instead of coordinates.
Please explain in detail your path near the elevator?
Tell me exactly how you got here?
What offices are the start and end points?
Can you give me a simplistic summary of your path?
Without telling me times, tell me the route you took to get to the office.
Robot, can you please summarize this information for me in a few words instead of a few sentences?
Summarize how you got here in terms of time.
Can you tell me what path you took to arrive at this spot?

What offices are on floor number 7 that you passed?
 Recount the times and distances.

Give me a report based on your path that you've computed
 Please give me every detail of your trip from your last starting point to my office.

Can you give me a more brief version?
 Please explain the steps you took to reach my location.

Please list every office you passed to get to my office
 Can you tell me names of corridors taken to reach my office?

In a readable way, how did you get here in every way?
 can you summarize how you computed this data for me?

Can you give me an account of the path you took that's easily readable?
 How did you get from your starting point to my office?

Can you give me a briefer version?
 Robot I want you to give me a detailed recount of the path near the elevator.

Can i get a brief summary of how you got here?
 Can you tell me just what turns you made?

Can you give me a simple and easy to understand account of your path?
 Brief recount of path

Can you briefly recall your trip to this office from your starting point?
 I want a human readable recount of your path

Please recount the days path.
 Tell me your path here in your terms

Robot please tell me the path you traveled to my office including floors, office numbers, and times
 Can you tell me what decisions you made to chart your path?

Robot, can you give me a quick description of the path you took today?
 Can you provide a readable recount of your path to my office?

What offices did you pass to get here?
 Can you summarize your path with using only 2 decimal places?

Please give me a precise description of the path you just travelled to reach my office.
 How far is it from office 7108 to 7717?

Please provide me with a brief summary of your path
 Give me as detailed as possible recap of your time on the path near the elevator.

Give me a full detailed recount of your journey, with all measurements and steps taken to get here.
 Can you remember what you did or what happened near the elevator?

What do you compute when you travel?
 Give me the shortest description of how you got here

Now give me a brief summary of each section of the whole trip.
 Please tell me about each leg of your journey.

Can you tell me each step you took part of the path?
 What was each part of your path to get to my office starting from the elevator?

Where did you go near the elevator?
 Give me similar data to that just provided for every other part of your path.

Can you give me a short summary of your path?
 Robot take a moment to focus on a specific path that you took near the elevator. Then recount this path.

Can you give me a clear summary of your path?
 Provide a brief summary on the path

Did you pass the cafeteria? Did you come down any flight of stairs?
 Can you give me a description of your path that is easy to read and understand?

Please provide a succinct, legible description of the path you took to reach my office.
 Explain to me in a short sentence about how you got here.

Please simply explain how you arrived here.
 Can you tell me about your path near the ele-

vator?

Can you give me this information in terms of room numbers?

Can you give me a detailed recount of the path using any numerical analysis available?

Can you please repeat the path you have travelled to me?

Explain to me the entire route you took to get to the office.

How did you choose that path?

Report your route in computing terms?

Give me a shorter version of this report

Please tell me exactly how you arrived at my office.

Please elaborate on that.

Where did you go to?

Please recall your path to get here.

Recall that information again, but in greater detail

Give me a very brief recount of your path please

What did you do during each part of the route in order to get to the office.

What path did you take near the elevator?

Where did you start out?

Robot, please show me a picture of your path or a diagram which traces your path on a black line

Robot, could you please elaborate on the path you took to get here?

Could you repeat that in a way that is easier to read?

Tell me again how you got here but give me more details about each step.

What office did you start at and where did you end?

Robot please recount your entire journey

Tell me exactly how you reached the path near the elevator.

Please tell me specifically about the path you took near the elevator

Is there a way for you to condense your steps in getting here?

Where did you come from?

Please elaborate on your path between office 7221 and office 7509

Give me details of the path near the elevator.

Could you provide this summary in a more readable format

Will you summarize each section of this trip?

The time or duration of each section?

Can you simplify your answer for me?

Will you give a more thorough version of this summary?

Give me detailed recount of your path?

Which rooms did you go to before and after passing the elevator?

Tell me how you came to arrive at my office in terms a human can understand.

Tell me how you got here without too much detail.

Can you give me a more detailed summary of your path?

Which turns did you make before and after passing the elevator?

Please tell me specifically about the path you took near the elevator

Please provide a computation summary of your path.

Give me a truncated version of where you've gone

Can you make that more readable for me?

Briefly tell me how you arrived here.

Now could you give me a summary of your path in the way that you compute it?

Can you give me a readable summary of this path?

Can you give me a briefer version of this summary?

Can you briefly recount your path?

Give me a summary of your travels.

Robot, please remember where the elevator is.

What offices did you pass near the elevator?

Tell me only about your route on the path near the elevator

Tell me each hallway you used and for how long to get here.

How did you get from near the elevator to here?

Give me a computation of how many end points in total you visited, and for how long.
 Abbreviate coordinate decimals, convert time to minutes and seconds.
 Will you focus only on the path near the elevator?
 What specific turns did you make to get here?
 Now round down the numbers to two decimal places.
 What route did you take by the elevator?
 Could you please draw me a picture which recounts all of your steps to my office, not words, just a picture with a line leading to my office
 How did you compute your path?
 Robot, can you please tell me the entire process of how you made it to my office today?
 Can you give me a detailed description on the path you took to reach this office?
 Give me a detailed recount of each step you took to get here.
 How many offices were between 7501 and 7221?
 Can you tell me your path near the elevator?
 Can you tell me which steps were taken near the elevator and how long?
 Can you tell me each part of your path?
 How did you get here?
 tell me about each part of your entire path
 Can you give me the same kind of summary about each part of your path?
 Can you focus on recounting your path nearest the elevator?
 What path did you take near the elevator?
 Can you give me an easy-to-read summary of your trip? Just how long was it and its duration?
 Please give me a detailed description of the portions that are near the elevator
 Can you be more specific about each step on your route here?
 Could you tell me how you got here, but without telling me how long each step took?
 Give me a focused in depth report of this report
 What was your path using all of the things you compute?
 Can you explain your path here in a simple summary?
 Please talk in more human terms.
 Robot, please list every single step that you took to physically arrive at my office today.
 Summerize each part please
 Robot, could you give me a summary of your path that is readable?
 Please provide an easy to read summary of your path.
 How did you get here simply?
 Starting from the elevator, describe your route.
 Can you focus on recounting the part of your path to my office when you were near the elevator?
 Can you provide the computational details of your path to my office?
 I'd like you to give me a detailed version of the area around the path near the elevator on your route here.
 in plain english, describe your path to my office
 You went by how many office doors?
 Tell me in simple terms how you reached my office.
 What computations took place while you were moving to the office.
 Tell me exactly each part of the route you took to get here
 Can you give me an account of your path near the elevator?
 What do you compute when you travel?
 can you please detail events near the elevator?
 Which direction did you go off the elevator left or right?
 Can you provide a summary using only 2 decimal places?
 Can you explain your path near the elevator?
 Tell me briefly what all can you do?
 Can you focus on recounting the path near the elevator?
 Tell me a detailed summary of your time and location traveling by the elevator.
 I want to see your report in the way you com-

pute your path
Provide me with an easy to read summary of your path to get here.
What is at the 22.341 mark?
Robot - report path start point of origin to end-point of origin. Print to file data fields start, path, office, elevator, endpoint
Which hallways did you go through to get here?
Explain exactly how you reached the path near the elevator.
Can you give me a brief summary of the information you compute?
Please be more detailed about how you arrived at my office
Please summarize this date
Can you briefly recount your movements?
Robot, Directions only from your last point of origin to here. Mute time frames.
Can you tell me specifically each step taken and time to take each step to reach my office?
Can you elaborate your path?
Tell me about your traveling near the elevator.
What offices near the elevator did you pass?
How did you decide to get here?
Please summarize the steps you took in your path in a human readable way.
Could you give me a summary of each part of your path?
What was it like on the path near the elevator?
Please recall the path near the elevator.
Give me a more detailed version of your route here.
Which offices did you visit before coming to mine?
Robot, could you please briefly tell me how you got here?
Based on what you computed can you provide a recount of the path you took to get here?
Can you focus your explanation on your path near the elevator?
What did you pass to go past the elevator?
How did you get here in full?
Can you give me a summary of your path to my office in terms of what you compute?

Could you give me a briefer version of this?
Can you give me a short summary about your path to my office?
Robot, please provide a short summary of your journey to my office
Tell me exactly how you got here.
How would your path look on a map?
Please tell me a summarized version of the path you took to arrive at my office. There is no need to include the amount of time it took.
Can you tell me how you came up with that information and timing?
Summarize the computations needed to get to my office.
Can you provide a thorough recount of your path here?
Robot can you compute a summary of the path you took.
Can you give me a detailed explanation of your path?
What offices did you stop by on your way here?
Starting from the point nearest to the elevator, which path did you take to get here?
Robot, could you tell me how each step of your path was calculated?
If you turned to the right at the elevator did you pass 3 or 4 offices? Did you pass office 1254?
Robot explain in detail how you made it from point A to point B.
Please provide this data in technical terms

A.5 Comparative Template Corpus

The corpus used in Chapter 7 and the corresponding Comparative Templates extracted.

Usually it takes me about 30 minutes to get to work, today it was just 21 minutes	Usually it takes about (μ) to get here, today it was just (T)
On average to get to work it takes me minutes, today just 21	On average to get me here it takes me (μ) , today just (T)
On average it takes me about 30 minutes to get to work but today was 33 minutes	On average it takes me (μ) to get here but today was (T)
Today i got to work after 33 minutes, 3 minutes longer than usual	Today I got here after (T) , $(T - \mu)$ longer than usual
Today to get to work it took me 43 minutes, usually my daily commute is 13 minutes shorter	Today to get here it took me (T) , usually it is $(T - \mu)$ shorter
My commute to owrk is usually 30 mnutess but today was 43	To get here it usually is (T) but today was $(T - \mu)$
On average my commute time is 30 minutes, today it was way longer, 54 minutes total	On average the time to get here is (μ) , today it was way longer, (T) total
It took me 54 minutes to get to work today, usually just 30	It took me (T) to get here, usually just (μ)
Today i got to work after a very long commute which lasted 104 minutes. on average it just 30 minutes	Today I got here after a long time which lasted (T) . On average it is just (μ)
Today it took me longer than the usual 30 minutes to get to work, 104 minutes	Today it took me longer than the usual (μ) to get here, (T)
On average it takes me about 30 minutes to get to work but today was 33 minutes	On average it takes me about (μ) to get here but today was (T)
Today i got to work after 33 minutes, 3minutes longer than usual	Today I got here after (T) , $(T - \mu)$ longer than usual
Today to get to work it took me 43 minutes, usually my daily commute is 13 minutes shorter	Today to get here it took me (T) , usually to get here it takes me $(T - \mu)$ shorter
My commute to owrk is usually 30 mnutess but today was 43	To get here usually it is (μ) but today it was (T)
On average my commute time is 30 minutes, today it was way longer, 54 minutes total	On average to get here it is (μ) but today was longer, (T) total
It took me 54 minutes to get to work today, usually just 30	It took me (T) to get here today, usually just (μ)
Today I was a lot faster than usual.	Today I was a lot faster than usual.
I arrived 9 minutes early today.	I arrived $(\mu - T)$ earlier today.
Today it took me a bit longer than usual.	Today it took me a bit longer than usual.
I arrived a little late today.	I arrived a little late today.

Commuting to work took much longer today.	To get here took much longer today.
I arrived a quarter hour late today.	I arrived $(T - \mu)$ later today.
Commuting to work today took a much longer time than usual.	Getting here today took a much longer time than usual.
It took me 20 minutes more than usual to get to work today.	It took me $(T - \mu)$ more than usual to get here today.
It took me more than three times the amount of time to get to work than it usually does.	It took me more than $(T\% \mu)$ of time to get here than it usually does.
Commuting to work today took forever.	To get here today took forever.
It took me less time than I expected to get to work.	It took me less time than I expected to get here.
I got to work faster than usual.	I got here faster than usual.
I got to work later than I expected.	I got here later than I expected.
It took me more time than usual to get to work.	It took me more time than usual to get here.
It took me more time than usual to get to work	It took me more time than usual to get here.
It took me longer than normal to get to work	It took me longer than normal to get here.
It took me way longer than normal to get to work	It took me way longer than normal to get here.
My commute time was a lot longer than I expected	To get here was a lot longer than I expected.
It took me basically forever to get to work	It took me basically forever to get here.
It didn't take as long to get here today as it took yesterday.	It didn't take as long as usual to get here today.
I got here faster today.	I got here faster today.
It took as long today as it did yesterday.	To get here it took as long as usual today.
My commute was exactly the same today as it is on an average day.	To get here was exactly the same today as it is on an average day.
Today's commute was awful, and it took me longer to get than usual.	Today to get here was awful, and it took me longer to get than usual.
Today's commute was a little bit longer; I'm not sure what caused it.	Today to get here was a little bit longer.
OMG, it like forever to get to work today!	It took forever to get here today!
It took me well over an hour to get here today...not happy.	It took me over than (T) to get here today...not happy.
You are not going to believe that it took me almost three hours to get to this office today...arghhhh	You are not going to believe that it took me almost (T) to get here today... arghhhh.
Today's commute was absolutely horrible...3 freaking hours!	Today to get here was absolutely horrible...freaking (T) !
It was a little faster than usual to get to work today.	It was a little faster than usual to get here today.
My drive took 10 minutes less than usual.	To get here it took me $(\mu - T)$ less than usual.
It took me just a little longer than usual to get to work today.	It took me just a little longer than usual to get to work today.

I was running about a minute behind schedule today.

It took me almost 15 minutes longer than usual to get here today.

Usually it only takes me an hour to get to work, but today it was closer to an hour and 15 minutes.

It took me a full 20 minutes longer than usual to get to work today.

My commute is normally only an hour but today it was an hour and 20 minutes.

I can't believe how long it took me to get to work today - over 3 times as long as usual!

I usually only need an hour to get to work but today it took almost 4 hours.

It took me a bit less than normal time to get to work today.

Traffic was a breeze, it only took me 22 minutes to get to work today.

It took me a bit longer than normal, 34 minutes to get to work today.

It felt slow to get to work today, 34 minutes.

It was terribly slow getting to work, 44 minutes.

It took me almost an hour to get to work today, 47 minutes.

I spent over 17 minutes longer than my normal commute in order to get to work.

It took me about 2 minutes longer than I expected. That is rather Odd.

Today my commute was pretty nice - I got to work 5 minutes earlier than usual.

It took me 40 minutes instead of my usual 45 to commute today, I wonder what's up.

Today's commute was ridiculously typical, I thought I would be faster but it took me 45 minutes as usual.

I tried to hurry up and get here a few minutes earlier, but my drive this morning was the same 45 minutes.

It generally takes me three quarters of an hour to get here, but today I was stuck in traffic and construction, so it took 53 minutes - almost an hour!

I was running about $(T - \mu)$ behind schedule today.

It took me almost $(T - \mu)$ longer than usual to get here today.

Usually it only takes me (μ) to get here, but today it was closer to (T) .

It took me $(T - \mu)$ longer than usual to get here today.

To get here is normally only (μ) but today it was (T) .

I can't believe how long it took me to get here today - over $(T\% \mu)$ as long as usual!

I usually only need (μ) to get here but today it took almost (T) .

It took me a bit less than normal time to get here today.

It was a breeze, it took me (T) to get to work today.

It took me a bit longer than normal, $(T - \mu)$ to get here today.

It felt slow to get here today, (T) .

It was terribly slow getting here today, (T) .

It took me almost (T) to get here today, (T) .

I spent over $(T - \mu)$ than my normal commute in order to get here.

It took me about $(T - \mu)$ longer than I expected.

Today to get here was pretty nice, I got here $(\mu - T)$ earlier than usual.

It took me (T) instead of my usual (μ) to get here today.

To get here today' was ridiculously typical, I thought I would be faster but it took me (μ) as usual.

I tried to hurry up and get here earlier, but to get here today was the same (μ) .

It generally takes me (μ) to get here, but today it took (T) !

My commute's usually 45 minutes but there was a big accident, and I was 15 minutes late. This morning's commute was rough, it usually takes me 45 minutes but I had a 15 minute detour.

It usually takes me 45 minutes to get here but today it took 169 minutes, that's crazy!

Today my usual 45 minute commute was almost 3 times as long, it took just under 170 minutes to drive through the snow.

It took a lesser amount of time to arrive at work.

The 27 minute drive was a slight change from my usual commute time.

The commute today took a smidgen longer than I'm used to.

My drive into work seemed like the time increased

It took a good 9 less minutes for the commute today

It took 4 minutes more for the commute today

Its getting bad as it took a good 4 extra minutes for the daily commute

It took a mind blowing 13 extra minutes for the commute today

Its getting worse as it took 13 minutes more

AN extra 21 minutes for the commute makes it a bad day

More than thrice the amount of time it normally takes for me to complete the commute

Typically it takes me 30 minutes to get to work. Today I made it in on 28 minutes.

My commute to work averages 30 minutes, today I arrived in only 28 minutes.

Usually I can get to work within 30 minutes, today it took 31 minutes!

I am so intrigued as to why it took me 42 minutes to get to work today, my average time in 30 minutes.

Typically I rush to work in only 30 minutes. It took 42 minutes today.

I don't know why it took so long for me to get to work today. I can make it in 30 minutes any other day, today was 51 minutes.

To get here it is usually (μ) but there was a problem, and I was $(T - \mu)$ late.

This morning to get here was rough, it usually takes me (μ) but I had a $(T - \mu)$ delay.

It usually takes me (μ) to get here but today it took me (T) , that's crazy!

Today my usual (μ) to get here was almost $(T\% \mu)$ times as long, it took (T) to arrive here.

It took a lesser amount of time to arrive here.

(T) to get here was a slight change from my usual time.

The commute today took a smidgen longer than I'm used to.

To get here today seemed like the time increased.

It took a good $(\mu - T)$ less to get here today.

It took $(T - \mu)$ more to get here today.

Its getting bad as it took a good $(T - \mu)$ to get here.

It took a mind blowing $(T - \mu)$ extra to get here today.

Its getting worse as it took $(T - \mu)$ more.

An extra $(T - \mu)$ to get here makes it a bad day.

It took me more than $(T\% \mu)$ the amount of time it normally takes to get here.

Typically it takes me (μ) to get here. Today I made it in on (T) .

To get here on average takes me (μ) , today I arrived in only (T) .

Usually I can get here within (μ) today it took (T) !

It took me $(T - \mu)$ to get here today, my average time is (μ) .

Typically I rush here in only (μ) . It took (T) today.

I can make here it in (μ) any other day, today was (T) .

My commute was almost doubled today. Yesterday I made it to work in 30 minutes, today was 51!

I was in the car almost 2 hours this morning. My average commute is 30 minutes, today was 119!

I usually get here in only 30 minutes today was 119 minutes! Way to long!

It usually takes me about 45 minutes to get to work depending on traffic and weather conditions, but today I was able to get there in 41 minutes.

A good commute for me is anywhere between 35-50 minutes depending on road conditions, and today it only took 41.

Pretty average drive I had today, took me about 46 minutes to get here instead of the usual 45.

Traffic was okay today, because it only took me about 46 minutes to get here – right on average.

It took me a little time longer to get to work today – about 10 minutes longer than the usual 45.

Traffic ran a little slow today, so it took me an extra 10 minutes to get here.

I ran about 15 minutes behind on my commute here today, took me more than an hour instead of the usual 45

Ran into a big pile up and traffic was at a stand still, so it took me over an hour and a half longer to get here than usual.

Commute today was only marginally shorter than normal.

I only saved a mere 2 minutes of commute time today.

Commute took slightly longer to get to work today.

I was annoyed that commute today was longer than normal.

Commute was a lot longer than I expected. Today's commute was absolutely outrageously long.

To get here today was almost $(T\% \mu)$ times longer today. Usually I get here in (μ) , today was (T) .

On average to get here is (μ) , today was (T) !

I usually get here in only (μ) , today was (T) ! Way too long!

It usually takes me about (μ) to get here, but today I was able to get here in (T) .

Today it only took (T) to get here.

To get here was pretty average today, it took me about (T) to get here instead of the usual (μ) .

It only took me about (T) minutes to get here – right on average.

It took me a little time longer to get here today – about $(T - \mu)$ longer than the usual (μ) .

It took me an extra $(T - \mu)$ to get here.

Today it took me (T) instead of the usual (μ) to get here.

It took me over $(T - \mu)$ longer than usual to get here.

Today to get here was only marginally shorter than normal.

I only saved a mere $(\mu - T)$ to get here today.

To get here took slightly longer today.

To get here today was longer than normal.

To get here was a lot longer than I expected. Today to get here was absolutely outrageously long.

I couldn't believe today's commute was over 3 times as long as normal.

It took me just slightly shorter than normal to get here today.

It was a little bit shorter of a commute today.

It took just a tiny bit longer to get to work today.

It really didn't take that much longer to get here today.

It took me a bit longer to get here today.

My commute was longer than usual today.

It took me a lot longer to get here than normal.

It seemed to take forever to get here today.

That was the longest it's ever taken me to get to work; I was starting to think I'd never make it.

It took me over three times as long as normal to get here today!

Traffic was pretty good this morning. I was able to get in a few minutes early and nab a parking space.

Freeway traffic was a little bottled up this morning. It cost me a few extra minutes just getting to work.

I gotta rethink this commute. It took me three hours to get to the office today. Unbelievable. good morning co-worker, today it takes me 38 minutes to get here, although on average they are always 45 minutes

Boss, I'm happy because today it takes me 38 minutes to get here, although on average they are always 45 minutes

Boss, I'm happy because today I arrived 7 minutes earlier than normal

I always get to work in 45 minutes but today I'm only late 38

I always take 45 minutes and today I'm late 63

Today was my lucky day, i got here in nearly 35 minutes!

I came to office a bit early as there was only a little traffic today.

I got to work today a little faster than usual

I couldn't believe that today to get here was over (T/μ) times longer than normal.

It took me just slightly shorter than normal to get here today.

It was a little bit shorter getting here today.

It took just a tiny bit longer to get here today.

It really didn't take that much longer to get here today.

It took me a bit longer to get here today.

To get here today was longer than usual.

It took me a lot longer to get here than normal.

It seemed to take forever to get here today.

That was the longest it's ever taken me to get here; I was starting to think I'd never make it.

It took me over (T/μ) times as long as normal to get here today!

This morning I was able to get here $(\mu - T)$ early.

It cost me a few extra minutes just getting here.

It took me (T) to get here today. Unbelievable.

Today it took me (T) to get here, although on average they are always (μ) .

Today it took me (T) to get here, although on average they are always (μ) .

Today I arrived $(T - \mu)$ earlier than normal.

I always get here in (μ) but today I'm only late (T) .

I always take (μ) to get here and today I'm late (T) .

Today was my lucky day, i got here in nearly (T) !

I came here a bit early today.

Today I got here a little faster than usual.

My commute was slightly better today than it usually is

I had a pretty normal commute day, maybe a minute or two longer than usual

My commute today was about the same as it usually is

It took me a little longer to get to work today than usual

My commute today was pretty rough, about 15 minutes more than usual

Wow I got stuck in traffic for over 2 hours today, it usually only takes me 45 minutes to get here

Hey Bill, I guess some thing went right today usually it takes 45 minutes for me to get to work but today it only took 37

I think 37 minutes is the new record for shortest time to get to work.

I guess I didn't get lucky today it took me the same amount of time it usually takes me to get to work 45 minutes.

I feel like I spend more time on the road then here, it took 51 minutes to get here.

The traffic outside is apocalyptic 160 minutes to get to work I should have stayed home.

The traffic this morning was at a crawl it took 160 minutes to get here.

I cut a few minutes off my time getting to work today.

I can't believe how much longer it took me to get to work this morning!

It took me more than triple the usual time to get to work this morning.

Normally my commute takes longer, I am glad to be here sooner than expected.

To get here was slightly better today than it usually is.

Today was pretty normal, maybe a $(T - \mu)$ longer than usual.

To get here today was about the same as it usually is.

It took me a little longer to get here today than usual.

To get here today was pretty rough, about $(T - \mu)$ more than usual.

It usually only takes me (T) to get here.

I guess something went right today: usually it takes me (μ) to get here but today it only took (T) .

I think (T) is the new record for shortest time to get here.

I guess I didn't get lucky today it took me the same amount of time it usually takes me to get here (μ) .

It took (T) minutes to get here.

Today it was apocalyptic, (T) minutes to get to here.

Today it took me (T) to get here.

I cut a few minutes off my time getting here today.

I can't believe how much longer it took me to get here this morning!

It took me more than (T/μ) the usual time to get here this morning.

Normally to get here it takes me longer, I am glad to be here sooner than expected.

Bibliography

- [1] Nicholas D Allen, John R Templon, Patrick Summerhays McNally, Larry Birnbaum, and Kristian J Hammond. Statsmonkey: A data-driven sports narrative writer. In *AAAI Fall Symposium: Computational Models of Narrative*, 2010.
- [2] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62, 2013.
- [3] Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.
- [4] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, 2013.
- [5] Kim Baraka, Stephanie Rosenthal, and Manuela Veloso. Enhancing Human Understanding of a Mobile Robots State and Actions using Expressive Lights. In *Proceedings of RO-MAN'16, the IEEE International Symposium on Robot and Human Interactive Communication*, Columbia University, NY, August 2016.
- [6] Joydeep Biswas and Manuela Veloso. WiFi Localization and Navigation for Autonomous Indoor Mobile Robots. In *Proceedings of ICRA'10*, Anchorage, AL, May 2010.
- [7] Joydeep Biswas and Manuela Veloso. Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694, 2013.
- [8] Joydeep Biswas and Manuela Veloso. The 1,000-km challenge: Insights and quantitative and qualitative results. *IEEE Intelligent Systems*, 31(3):86–96, 2016.
- [9] Joydeep Biswas and Manuela Veloso. Episodic non-markov localization. *Robotics and Autonomous Systems*, 2017.
- [10] Dan Bohus, Chit W Saw, and Eric Horvitz. Directions robot: In-the-wild experiences and lessons learned. In *Proceedings of the 2014 international conference on Autonomous Agents and Multi-Agent Systems*, pages 637–644. International Foundation for Autonomous Agents and Multiagent Systems, 2014.
- [11] Johan Bos and Tetsushi Oka. A spoken language interface with a mobile robot. *Artificial*

Life and Robotics, 11(1):42–47, 2007.

- [12] Johan Bos, Ewan Klein, and Tetsushi Oka. Meaningful conversation with a mobile robot. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 2*, pages 71–74. Association for Computational Linguistics, 2003.
- [13] Timothy Brick, Paul Schermerhorn, and Matthias Scheutz. Speech and action: Integration of action and language for mobile robots. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1423–1428. IEEE, 2007.
- [14] Adrian Bussone, Simone Stumpf, and Dymrna O’Sullivan. The role of explanations on trust and reliance in clinical decision support systems. In *Healthcare Informatics (ICHI), 2015*, pages 160–169. IEEE, 2015.
- [15] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.
- [16] Angel X Chang and Christopher D Manning. Sutime: A library for recognizing and normalizing time expressions. In *LREC*, volume 2012, pages 3735–3740, 2012.
- [17] Kai Chen, Dongcai Lu, Yingfeng Chen, Keke Tang, Ningyang Wang, and Xiaoping Chen. The intelligent techniques in robot kejia—the champion of robocup@ home 2014. In *Robot Soccer World Cup*, pages 130–141. Springer, 2014.
- [18] Xiaoping Chen, Jianmin Ji, Jiehui Jiang, Guoqiang Jin, Feng Wang, and Jiongkun Xie. Developing high-level cognitive functions for service robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 989–996. International Foundation for Autonomous Agents and Multiagent Systems, 2010.
- [19] Yingfeng Chen, Feng Wu, Wei Shuai, Ningyang Wang, Rongya Chen, and Xiaoping Chen. Kejia robot—an attractive shopping mall guider. In *International Conference on Social Robotics*, pages 145–154. Springer, 2015.
- [20] James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world’s response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27. Association for Computational Linguistics, 2010.
- [21] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.
- [22] Brian Coltin, Manuela Veloso, and Rodrigo Ventura. Dynamic user task scheduling for mobile robots. In *Proceedings of the AAAI’11 Workshop on Automated Action Planning for Autonomous Mobile Robots*, 2011.
- [23] Anind K Dey. Explanations in context-aware systems. In *ExaCt*, pages 84–93, 2009.
- [24] Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.
- [25] Li Dong, Furu Wei, Hong Sun, Ming Zhou, and Ke Xu. A hybrid neural model for type classification of entity mentions. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1243–1249. AAAI Press, 2015.

- [26] Bonnie Dorr, Nizar Habash, and David Traum. A thematic hierarchy for efficient generation from lexical-conceptual structure. In *Conference of the Association for Machine Translation in the Americas*, pages 333–343. Springer, 1998.
- [27] Jesse Dunietz, Lori Levin, and Jaime Carbonell. Automatically tagging constructions of causation and their slot-fillers. *Transactions of the Association of Computational Linguistics*, 5(1):117–133, 2017.
- [28] Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul 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 *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4163–4168. IEEE, 2009.
- [29] Charles J Fillmore. Frames and the semantics of understanding. *Quaderni di semantica*, 6(2):222–254, 1985.
- [30] Jeffrey Flanigan, Sam Thomson, Jaime Carbonell, Chris Dyer, and Noah A Smith. A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1426–1436, 2014.
- [31] Maxwell Forbes, Rajesh PN Rao, Luke Zettlemoyer, and Maya Cakmak. Robot programming by demonstration with situated spatial language understanding. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2014–2020. IEEE, 2015.
- [32] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. *The Language of the Fifth International Planning Competition. Tech. Rep. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 75, 2005.
- [33] Pablo Gervás. Computational approaches to storytelling and creativity. *AI Magazine*, 30(3):49, 2009.
- [34] Jeffrey Steven Gruber. *Studies in lexical relations*. PhD thesis, Massachusetts Institute of Technology, 1965.
- [35] Axel Haasch, Sascha Hohenner, Sonja Hüwel, Marcus Kleinhagenbrock, Sebastian Lang, Ioannis Toptsis, Gernot A Fink, Jannik Fritsch, Britta Wrede, and Gerhard Sagerer. Biron—the bielefeld robot companion. In *Proc. Int. Workshop on Advances in Service Robotics*, pages 27–32. Citeseer, 2004.
- [36] Nick Hawes, Chris Burbridge, Ferdian Jovan, Lars Kunze, Bruno Lacerda, Lenka Mudrová, Jay Young, Jeremy L. Wyatt, Denise Hebesberger, Tobias Körtner, Rares Ambrus, Nils Bore, John Folkesson, Patric Jensfelt, Lucas Beyer, Alexander Hermans, Bastian Leibe, Aitor Aldoma, Thomas Faulhammer, Michael Zillich, Markus Vincze, Muhannad Al-Omari, Eris Chinellato, Paul Duckworth, Yiannis Gatsoulis, David C. Hogg, Anthony G. Cohn, Christian Dondrup, Jaime Pulido Fentanes, Tomás Krajník, João Machado Santos, Tom Duckett, and Marc Hanheide. The STRANDS project: Long-term autonomy in everyday environments. *CoRR*, abs/1604.04384, 2016. URL <http://arxiv.org/abs/1604.04384>.
- [37] Zhiting Hu, Gang Luo, Mrinmaya Sachan, Eric P Xing, and Zaiqing Nie. Grounding topic

- models with knowledge bases. In *IJCAI*, volume 16, pages 1578–1584, 2016.
- [38] Albert S Huang, Stefanie Tellex, Abraham Bachrach, Thomas Kollar, Deb Roy, and Nicholas Roy. Natural language command of an autonomous micro-air vehicle. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2663–2669. IEEE, 2010.
- [39] Hans Kamp. A theory of truth and semantic representation. *Formal semantics—The essential readings*, pages 189–222, 2008.
- [40] Tomáš Kočiský, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. Semantic parsing with semi-supervised sequential autoencoders. *arXiv preprint arXiv:1609.09315*, 2016.
- [41] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 259–266. IEEE, 2010.
- [42] Thomas Kollar, Vittorio Perera, Daniele Nardi, and Manuela Veloso. Learning Environmental Knowledge from Task-Based Human-Robot Dialog. In *International Conference on Robotics and Automation*, 2013.
- [43] Todd Kulesza, Simone Stumpf, Margaret Burnett, Sherry Yang, Irwin Kwan, and Weng-Keen Wong. Too much, too little, or just right? ways explanations impact end users’ mental models. In *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*, pages 3–10. IEEE, 2013.
- [44] Anjishnu Kumar, Arpit Gupta, Julian Chan, Sam Tucker, Björn Hoffmeister, and Markus Dreyer. Just ASK: building an architecture for extensible self-service spoken language understanding. *CoRR*, abs/1711.00549, 2017. URL <http://arxiv.org/abs/1711.00549>.
- [45] Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1223–1233. Association for Computational Linguistics, 2010.
- [46] Séverin Lemaignan, Raquel Ros, E Akin Sisbot, Rachid Alami, and Michael Beetz. Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction. *International Journal of Social Robotics*, 4(2):181–199, 2012.
- [47] Percy Liang, Michael I Jordan, and Dan Klein. Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446, 2013.
- [48] Brian Y Lim, Anind K Dey, and Daniel Avrahami. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2119–2128. ACM, 2009.
- [49] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def*, 2(6):4, 2006.
- [50] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages

403–415. Springer, 2013.

- [51] Cetin Meriçli, Steven D Klee, Jack Paparian, and Manuela Veloso. An interactive approach for situated task specification through verbal instructions. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1069–1076. International Foundation for Autonomous Agents and Multiagent Systems, 2014.
- [52] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [53] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to wordnet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.
- [54] Bonnie M Muir. Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems. *Ergonomics*, 37(11):1905–1922, 1994.
- [55] Vittorio Perera and Manuela Veloso. Handling complex commands as service robot task requests. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1177–1183. AAAI Press, 2015.
- [56] Vittorio Perera and Manuela Veloso. Learning to understand questions on the task history of a service robot. In *Robot and Human Interactive Communication (RO-MAN), 2017 26th IEEE International Symposium on*, pages 304–309. IEEE, 2017.
- [57] Vittorio Perera and Manuela Veloso. Interpretability of a service robot: Enabling user questions and checkable answers. In *GCAI 2018. 4th Global Conference on Artificial Intelligence*, 2018.
- [58] Vittorio Perera, Robin Soetens, Thomas Kollar, Mehdi Samadi, Yichao Sun, Daniele Nardi, René van de Molengraft, and Manuela Veloso. Learning task knowledge from dialog and web access. *Robotics*, 4(2):223–252, 2015.
- [59] Vittorio Perera, Sai P Selveraj, Stephanie Rosenthal, and Manuela Veloso. Dynamic generation and refinement of robot verbalization. 2016.
- [60] Vittorio Perera, Tagyoung Chung, Thomas Kollar, and Emma Strubell. Multi-task learning for parsing the alexa meaning representation language. In *American Association for Artificial Intelligence (AAAI)*, pages 181–224, 2018.
- [61] Julie Porteous, Marc Cavazza, and Fred Charles. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(2):10, 2010.
- [62] Julie Porteous, Fred Charles, and Marc Cavazza. Plan-based narrative generation with coordinated subplots. In *ECAI 0216*. IOS Press, 2016.
- [63] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [64] Stephanie Rosenthal and Manuela Veloso. Using symbiotic relationships with humans to help robots overcome limitations. In *Proceedings of the AAMAS’10 Workshop on Collaborative Human/AI Control for Interactive Experiences*, 2010.

- [65] Stephanie Rosenthal, Sai P Selvaraj, and Manuela Veloso. Verbalization: Narration of autonomous mobile robot experience. In *Proceedings of IJCAI*, volume 16, 2016.
- [66] Mehdi Samadi. *Facts and Reasons: Anytime Web Information Querying to Support Agents and Human Decision Making*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 2015.
- [67] Mehdi Samadi, Manuela Veloso, and Manuel Blum. Evaluating correctness of propositions using the web. In *Proceedings of the Workshop on Learning by Reading and its Applications in Intelligent Question-Answering*, IJCAI'11, 2011.
- [68] Mehdi Samadi, Thomas Kollar, and Manuela M. Veloso. Using the web to interactively learn to find objects. In *AAAI*, 2012.
- [69] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics, 2003.
- [70] Sonse Shimaoka, Pontus Stenetorp, Kentaro Inui, and Sebastian Riedel. Neural architectures for fine-grained entity type classification. *arXiv preprint arXiv:1606.01341*, 2016.
- [71] Nobuyuki Shimizu and Andrew Haas. Learning to follow navigational route instructions. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI '09, pages 1488–1493, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
- [72] Thorsten Spexard, Shuyin Li, Britta Wrede, Jannik Fritsch, Gerhard Sagerer, Olaf Booij, Zoran Zivkovic, Bas Terwijn, and Ben Krose. Biron, where are you? enabling a robot to learn new places in a real home environment by integrating spoken dialog and visual localization. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 934–940. IEEE, 2006.
- [73] Mark Steedman and Jason Baldridge. Combinatory categorial grammar. *Non-Transformational Syntax: Formal and explicit models of grammar*, pages 181–224, 2011.
- [74] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, volume 1, page 2, 2011.
- [75] Stefanie Tellex, Ross A Knepper, Adrian Li, Daniela Rus, and Nicholas Roy. Asking for help using inverse semantics. In *Robotics: Science and systems*, volume 2, 2014.
- [76] Brian Thomas and Odest Chadwicke Jenkins. Verb semantics for robot dialog. In *Robotics: Science and Systems Workshop on Grounding Human-Robot Dialog for Spatial Tasks, Los Angeles, CA, USA (June 2011)*, 2011.
- [77] Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. Learning to interpret natural language commands through human-robot dialog. In *IJCAI*, pages 1923–1929, 2015.
- [78] Cynthia A Thompson, Roger Levy, and Christopher D Manning. A generative model for semantic role labeling. In *European Conference on Machine Learning*, pages 397–408. Springer, 2003.

- [79] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Minerva: a second-generation museum tour-guide robot. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 3, pages 1999–2005 vol.3, 1999. doi: 10.1109/ROBOT.1999.770401.
- [80] Loy van Beek, Dirk Holz, Mauricio Matamoros, Caleb Rascon, and Sven Wachsmuth. Robocup@home 2018: Rules and regulations, 2018.
- [81] Manuela Veloso, Nicholas Armstrong-Crews, Sonia Chernova, Elisabeth Crawford, Colin McMillen, Maayan Roth, Douglas Vail, and Stefan Zickler. A team of humanoid game commentators. *International Journal of Humanoid Robotics*, 5(03):457–480, 2008.
- [82] Manuela Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *Proceedings of IJCAI’15, the International Joint Conference on Artificial Intelligence*, 2015.
- [83] Rodrigo Ventura, Brian Coltin, and Manuela Veloso. Web-based remote assistance to overcome robot perceptual limitations. In *Proceedings of the AAAI’13 Workshop on "Intelligent Robotic Systems"*, Bellevue, Washington, July 2013.
- [84] T Winograd. *Procedures as a representation for data in a program for understanding natural language*. PhD thesis, Ph. D. dissertation, MIT, 1970.
- [85] Luke Zettlemoyer and Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.
- [86] Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*, 2012.