4-2016

Grounding robot motion in natural language and visual perception

Scott Alan Bronikowski

Purdue University

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Scott Alan Bronikowski

Entitled
GROUNDING ROBOT MOTION IN NATURAL LANGUAGE AND VISUAL PERCEPTION

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Jeffrey Mark Siskind Chair	Lisa A. Shay
J. Eric Dietz	
Robert L. Givan	
Barry L. Shoop	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Jeffrey Mark Siskind

Approved by: Venkataramanan Balakrishnan 4/15/2016
Head of the Departmental Graduate Program Date

GROUNDING ROBOT MOTION IN

NATURAL LANGUAGE AND VISUAL PERCEPTION

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Scott Alan Bronikowski

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2016

Purdue University

West Lafayette, Indiana

To Kara and Emma, without whose love, understanding, and unwavering support I would never have been able to complete this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Figure Page

ABSTRACT

Bronikowski, Scott Alan PhD, Purdue University, May 2016. Grounding Robot Motion in Natural Language and Visual Perception. Major Professor: Jeffrey Mark Siskind.

The current state of the art in military and first responder ground robots involves heavy physical and cognitive burdens on the human operator while taking little to no advantage of the potential autonomy of robotic technology. The robots currently in use are rugged remote-controlled vehicles. Their interaction modalities, usually utilizing a game controller connected to a computer, require a dedicated operator who has limited capacity for other tasks.

I present research which aims to ease these burdens by incorporating multiple modes of robotic sensing into a system which allows humans to interact with robots through a natural-language interface. I conduct this research on a custom-built six-wheeled mobile robot.

First I present a unified framework which supports grounding natural-language semantics in robotic driving. This framework supports learning the meanings of nouns and prepositions from sentential descriptions of paths driven by the robot, as well as using such meanings to both generate a sentential description of a path and perform automated driving of a path specified in natural language. One limitation of this framework is that it requires as input the locations of the (initially nameless) objects in the floor plan.

Next I present a method to automatically detect, localize, and label objects in the robot's environment using only the robot's video feed and corresponding odometry. This method produces a map of the robot's environment in which objects are differentiated by abstract class labels.

Finally, I present work that unifies the previous two approaches. This method detects, localizes, and labels objects, as the previous method does. However, this new method

integrates natural-language descriptions to learn actual object names, rather than abstract labels.

1. INTRODUCTION

As the world's battlefields have gotten increasingly more lethal over the last several decades of technological advancement, there has been a rising interest in battlefield robotics—protecting soldiers by replacing them with robots in the most dangerous battlefield situations, such as reconnaissance and IED interrogation [1]. However, the current state of the art in military ground robotics, which also finds wide adoption in the public safety and disaster relief fields, places a heavy physical and cognitive burden on the human operator while taking little to no advantage of the potential autonomy of which robotic technology is capable. The types of robots currently in use, such as the iRobot PackBot (Fig. 1.1), are not much more than extremely rugged remote-controlled vehicles with task-specific attachments, such as manipulator arms.

In order to operate such a robot in a combat environment, a soldier must typically use two hands on a video game controller (Fig. 1.2). While mapping the robot's controls to a game controller provides a familiar interface for soldiers who grew up playing video games, it is far from an optimal solution. Having two hands on a game controller means the soldier will not be able to employ his or her assigned weapon while operating the robot. Such is the physical burden. Additionally, the cognitive burden of robot operation requires the soldier to devote his or her attention to a computer screen displaying the output of the robot's cameras, as well as data on the robot's current state, *e.g.,* GPS position, battery charge, camera orientation, and manipulator arm orientation. This requirement for attention to the robot means the operator is not able to pay attention to his or her own immediate surroundings, which can be a deadly mistake in a combat situation. However, since the robot lacks any form of autonomy and cannot execute any tasks without direct control from a human operator, to divert the soldier's attention away from the robot is to render the robot useless.

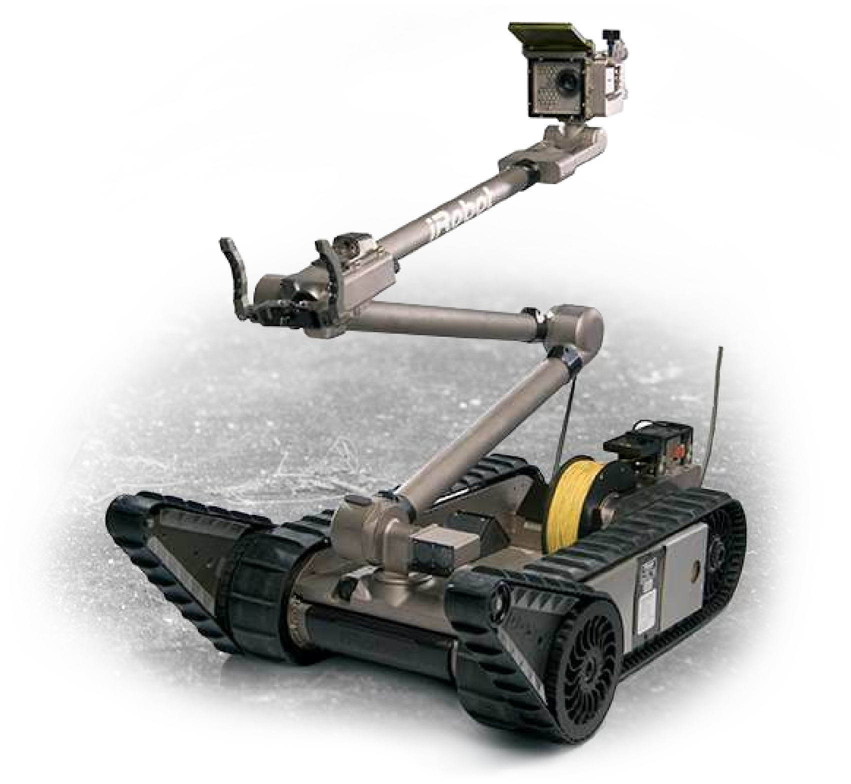


Fig. 1.1.: Image of the iRobot 501 PackBot® [2]

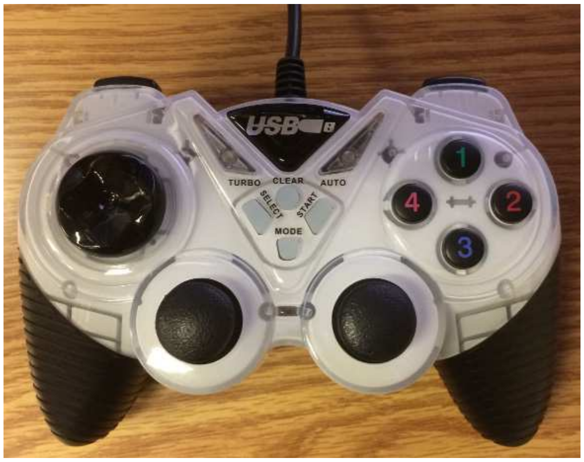


Fig. 1.2.: The type of controller usually used to operate a PackBot®

If a typical nine soldier infantry squad were to employ such a robot, the squad leader would need to dedicate two soldiers, or approximately one-fourth of his or her available combat power, to the robot—one to operate the robot, and one to maintain security for the operator. Furthermore, since the control system for the robot typically involves a laptop computer with a two-handed video game controller, it is ill-suited to mobility in a dismounted environment. Even recent innovations such as moving the control interface onto a touchscreen tablet [3] still require the soldier to use two hands and significant mental capacity to directly control the robot. In effect, the decision to employ a robot necessitates the shift from offensive to defensive operations, at least until the robot is either recovered or abandoned.

It is my hypothesis that the best way to increase the utility of such robots is to fundamentally change how humans interact with them. Instead of being required to explicitly *drive* the robot, the operator should instead be able to *direct* the robot to achieve navigational goals. The most useful modality through which humans can interact with such robots is via natural language.[1]

To investigate this, I have developed and built a research system called **VADER**, an acronym which stands for **V**isually and **AI** **D**irected **E**xperimental **R**obot.

The main hardware component of my system is the mobile robot, or rover, shown in Fig. 1.3 and Fig. 1.4. The **VADER** system also includes custom software that runs on both the rover and on a companion Linux computer. I present the technical details on the construction of this system in Chapter 3.

I have used this system to investigate incorporating multiple modes of robotic sensing to allow a human operator to interact with the robot through a natural language interface. I first constructed a unified framework which supports grounding natural language semantics in robotic driving. I present this work in Chapters 4 and 5. This framework supports learning the meanings of nouns and prepositions from sentential descriptions of paths driven by the

[1] While not an uninteresting or trivial problem, the task of speech recognition is nonetheless well-studied and not within the scope of my research. All components of my research assume that the natural language instructions to the robot will be in the form of text.

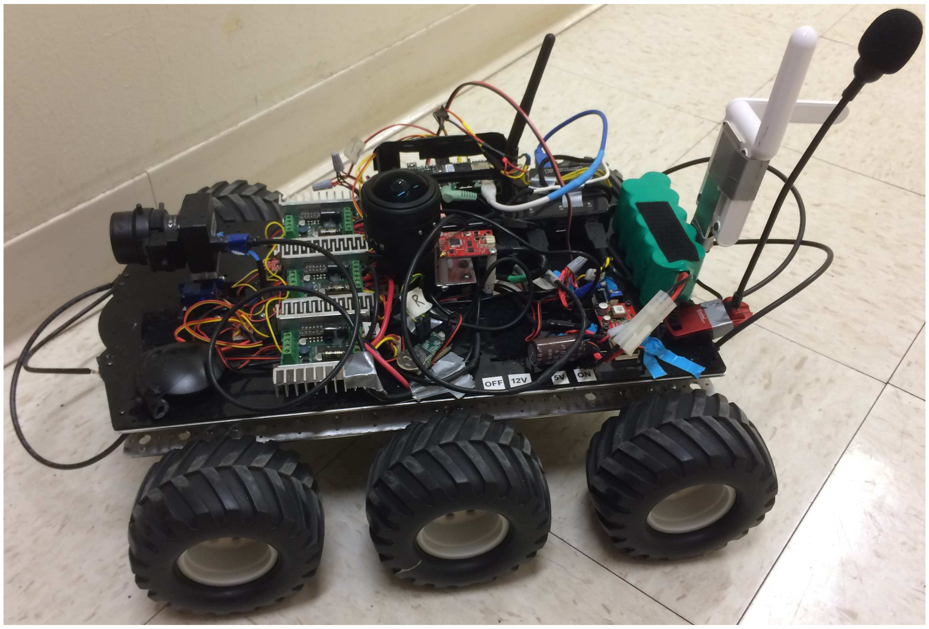


Fig. 1.3.: The **VADER** rover facing to the left

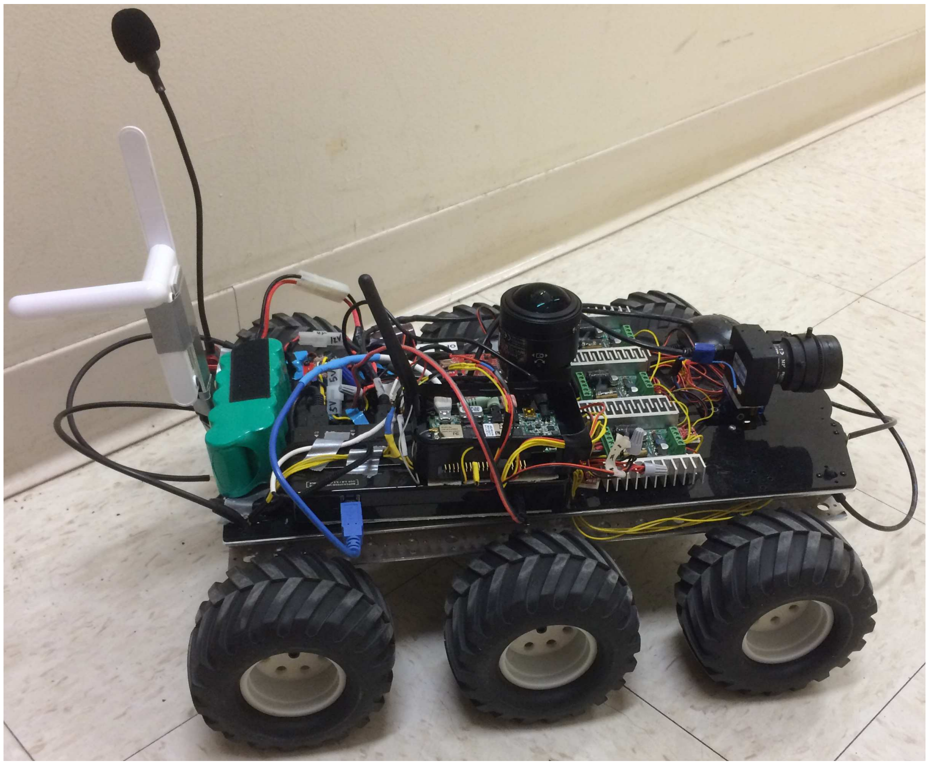


Fig. 1.4.: The **VADER** rover facing to the right

robot. It also supports using such meanings to generate sentential descriptions of a driven path and performing automatic driving of a path specified in natural language.

This initial framework is limited by the fact that it requires as input the locations of the (initially nameless) objects in the floor plan within which the robot drives. Therefore, I incorporated computer vision and machine learning techniques to automatically learn the locations and labels of the objects within a floor plan. I present this work in Chapters 6 and 7.

While my work is far from a complete solution to the problem of allowing humans to interact with robots via natural language, I believe it is a significant and novel step in the right direction.

1.1 Organization of this Report

In Chapter 2 I review some of the prior work related to my research. In Chapter 3 I describe **VADER** in depth, with detail of the hardware construction and software development.

In Chapter 4 I present my initial work on the unified framework to ground natural language semantics in robotic driving. This work was accomplished in collaboration with Daniel Barrett and Haonan Yu. With minor changes for formatting and length, it was submitted to arXiv on 25 August 2015. It is accessible at `http://arxiv.org/abs/1508.06161`.

Following this, my colleagues and I expanded upon the work in Chapter 4 by removing constraints on the natural language input and implementing a more rigorous evaluation scheme. This manuscript, entitled “Driving Under the Influence (of Language),” was submitted to IEEE Transactions on Neural Networks and Learning Systems (TNNLS) on 26 January 2016. It is currently in review. I provide excerpted and summarized sections of this manuscript, highlighting the differences with the initial work, in Chapter 5.

In Chapter 6, I present a method to automatically detect, localize, and label objects—also known as codetection—using the video feed and corresponding odometry from **VADER**.

This work was accomplished in collaboration with Daniel Barrett. With minor changes for formatting and length, it was submitted to the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) on 1 March 2016. It is currently in review.

In Chapter 7, I present my most recent work on unifying object codetection with natural language descriptions of driven paths. This work has not yet been submitted for publication.

1.2 Publications

Chapter 4 was previously published as:

> D.P. Barrett, S.A. Bronikowski, H. Yu, and J.M. Siskind, ”Robot Language Learning, Generation, and Comprehension,” arXiv, vol. abs/1508.06161, 25 August 2015.

Chapter 5 contains excerpts and summaries taken from a manuscript which is in review as:

> D.P. Barrett, S.A. Bronikowski, H. Yu, and J.M. Siskind, ”Driving Under the Influence (of Language),” submitted to IEEE Transactions on Neural Networks and Learning Systems (TNNLS), 2016.

Chapter 6 is in review as:

> S.A. Bronikowski, D.P. Barrett, and J.M. Siskind, “Object Codetection from Mobile Robot Video,” submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016.

2. OVERVIEW OF RELATED PRIOR WORK

To put it bluntly, there is little to no prior work that does what I am doing. While I would not be so bold as to say that my search of the literature has been exhaustive, I would say that it has been thorough. Throughout this search, I have been unable to find work that uses a physical robot to study both natural language and computer vision in an attempt to unify control of said robot in both of these domains.

There are papers that explore the use of natural language in a robotic context, such as [7–15] , but the majority of them do so in a discrete simulated environment. In such an environment, they are free to use the internal representation of the simulation to obtain discrete symbolic primitives. My work, conversely, uses a real mobile robot in the physical world. This simple fact requires that my robot be able to localize itself using only the sensors with which I equipped it. **VADER** thus does not have easy-to-use discrete symbolic primitives to represent its state, but rather primitives that are densely sampled from continuous-valued data, with all of the uncertainty and noise that is attendant therein.

There are also papers that use a real mobile robot with natural language commands [16–27], but the language used in these tends to be constrained (*e.g.,* a small set of specific utterances or a grammar with constrained expressiveness). While the grammar used in Chapter 4 does have a limited number of words, the grammar itself supports infinite recursion, so that sentences can be of arbitrary length and complexity. Additionally, I show in Chapter 5 that **VADER** can be used with language that is not constrained by a certain grammar, but rather can utilize free-form sentences collected from anonymous Amazon Mechanical Turk (AMT) workers.

Some of the work using a physical robot has involved learning in the context of language and navigation (*e.g.,* [16, 17]), but none of these do all three of the tasks (acquisition, generation, and comprehension) which **VADER** does in Chapters 4 and 5. Other work explores the topic of natural-language interaction with robots, such as [18–23], both within and outside the realm of robotic navigation; however, none of these involve learning. Still other work performs learning in the context of language and robotics, but not navigation (*e.g.,* [24–27]).

In the realm of object codetection, I have found no prior work that seeks to codetect objects from the video feed of a mobile robot. Existing methods that detect, localize, and label previously unseen objects operate on images [28–31] or video [32–35] collected from a stationary, human-centric point of view. Most of these methods require the object of interest to be prominent and close to the center of the field of view. Additionally, all of them localize an object only within the 2D image frame. In contrast, **VADER**'s video originates from a ground-based mobile robot, which has a different visual perspective that is both moving and much closer to the ground. **VADER** is also able to both detect small objects that are off-center and localize objects in both the 2D image and 3D world frames.

I will review several related prior works in the following sections, highlighting both their similarities and differences with my work.

2.1 Walk the Talk: Connecting Language, Knowledge, and Action in Route Instructions [7]

This paper introduces MARCO, a software agent that follows free-form natural language route instructions. The authors also introduce the concept of the *compound action specification*, which they use to model natural language instructions as a set of actions to take under certain conditions. MARCO is able to infer implicit actions from knowledge of linguistic conditional phrases, spatial action, and local configuration.

To test MARCO, the authors collect a corpus of 682 route instructions from six people who navigate through three virtual environments. These virtual environments are simple dungeon worlds of corridors that intersect at right angles, visually similar to the *Wolfenstein 3D* and *Doom* series of games from the early 1990s, with different floor and wall patterns used to distinguish between locations. The environments are naturally constrained, reducing the freedom of an agent in this environment to simply turning left or right, or going forward.

As a validation of their corpus, the authors have 36 testers rate the instructions' clarity on a 1-to-6 scale. The testers also attempt to follow the directions from the six corpus contributors. The testers achieve success—defined as reaching the intended destination—69% of the time. The MARCO system, in its most capable form, achieves success 61% of the time.

MARCO operates by interpreting human-written route instructions and following an inferred model of the described route. This inference of implicit actions requires knowledge of language and spatial actions. For example, to execute the action *walk to the further end of the hall*, an agent must infer the unstated action of looking down the hallway in both directions to determine which end is farther away. MARCO accomplishes this by applying a total of six separate modules. Three of these modules deal with the linguistic interpretation of route instruction text, while the other three interpret the instructions spatially within the environmental context.

The first three layers of MARCO, which deal with the interpretation of the text instructions, are the syntactic parser, the content framer, and the instruction modeler. The syntactic parser, as built by the authors, parses the raw route instruction text using the Python Natural Language Toolkit [36]. However, in testing the authors chose to take this layer out of play, instead using a set of hand-verified parse trees as input. The content framer layer translates the structure of the parsed text into a model, called a content frame, of the meaning as a nested attribute-value matrix. This model captures both the nested structure and the sense—drawn from WordNet [37]—of the text. When MARCO finds a word that is not already in its lexicon, it searches WordNet for a synonym or hypernym to gain the word's sense. Finally, the instruction modeler translates the content frame into an imperative model of what to do under which conditions. The authors call this model a *compound action specification*. The *compound action specification* models which simple actions to take. The instruction modeler layer infers the imperative model by applying the aforementioned linguistic knowledge of verbs and prepositions, along with spatial knowledge of the environment. If it encounters ambiguity, resolution is deferred until the agent is in the correct to resolve the ambiguity. Each clause in the instruction is interpreted as a *compound action specification* based on the verb; adverbs, objects, and prepositional phrases are interpreted as pre-, post-, or while-conditions. It also recognizes conditional clauses as possibly requiring an action to achieve. These imperative instructions take the form of four low-level simple actions: TURN, TRAVEL, VERIFY, and DECLARE-GOAL (terminate route following). The authors make an unsupported claim that these four actions are both necessary to follow all directions and sufficient to follow most. As an example of inference, the authors state that the phrase *go down the hall to the chair* gets interpreted with *along* and *until* as parameters of the TRAVEL action.

The final three layers of MARCO, which do spatial interpretation of the instructions within the environmental context, are the executor, the robot controller, and the view description matcher. The executor sequences the simple action. In its initial form, it simply executes each *compound action specification* before moving to the next. It also plans its sequences to gain information and achieve pre- and post-conditions of actions. The robot controller executes the simple actions on the robot hardware. This is simply an abstraction layer to allow the system to run on different platforms. In their testing, the authors ran a completely simulated robot in their virtual environments. The final layer is the view description matcher. It checks the view description against the current sensory observations by treating the view description as a set of constraints on the observation stream. Like other layers, it also defers ambiguity resolution until the environment can provide the correct context.

The authors claim that MARCO is robust to unexpected input. However, in their description of this robustness, they explain that the system simply ignores any linguistic segments that it cannot parse or understand. Their claim, for which they provide no evidence, is that the language around the ignored parts will provide enough information to make the ignored parts irrelevant.

As mentioned previously, the MARCO system achieves 61% success on the authors' corpus, compared to 69% for human testers. The authors also test their system without allowing implied TRAVEL actions, without implied TURN actions, and without both implied TRAVEL and implied TURN actions. At each removal, performance declined significantly, showing that the execution of the implied actions are key to correct direction following. However, the authors fail to give any quantitative assessment of their corpus with respect to the prevalence of implied actions within the directions in the corpus. This leads one to wonder how explicit their directions are; would more explicit instructions improve performance? Finally, the authors show that when considering the only the subset of the instructions that were rated most highly by the testers, there was no statistically significant difference between human and MARCO performance. This affirms the conclusion, which most people would regard as common sense, that when trying to follow route directions, the quality of the instructions matters.

The problem addressed by MARCO in this paper is one that is much simpler than the one I address with the unified framework to ground natural language semantics in robotic driving with **VADER** in Chapters 4 and 5. MARCO operates in a discrete simulated environment, rather than the real world as **VADER** does. This simulated environment allows MARCO to access the discrete symbolic primitives that represent the system's state. Conversely, **VADER** must determine its state from sensor data that is noisy, continuous-valued, densely sampled, and often ambiguous. The fact that both environment and instruction set are discrete in the MARCO system mean that there is a countably finite number of configurations and actions that can be undertaken. **VADER**, since it exists in the physical world, can take on an uncountably infinite number of configurations and actions while moving to any location in the continuous 2D Cartesian plane. MARCO only does a very high-level interpretation of the given natural language instructions, using *compound action specifications* to map between a series of words and a series of actions. Furthermore, the paper describes no learning within the MARCO system, which leads one to the conclusion that such *compound action specifications* are simply pre-programmed mappings between phrases and actions. Combined with the fact that the authors use hand-generated parse trees for their natural language input, MARCO is really nothing more than a fixed mapping between a set of words and a set of keyboard commands within a simulation. **VADER** is much more complex. As my work in Chapters 4 and 5 shows, **VADER** is capable of learning the meanings of nouns and prepositions, given paths within a known floor plan paired with sentences describing such paths, both when such sentences are constrained to a particular grammar and when the sentences are unconstrained responses from humans. **VADER** can then take such learned meanings and either generate sentential descriptions for previously unseen paths, or generate and automatically drive a path described in natural language. In short, **VADER**, even without its computer vision components, is a much more advanced system than MARCO.

2.2 Toward Understanding Natural Language Directions [8]

This paper develops a formulation of the problem of understanding natural language directions as inferring a sequence of viewpoints v_i given a set of natural language directions. To do this, the authors introduce the concept of *spatial description clauses*, or SDCs, which exploit the structure of the language in the directions. Their goal is to take natural language directions as input and infer the intended path through the environment, providing a set of waypoints (which they refer to as viewpoints v_i) as output. They accomplish this by extracting the linguistic structure from the directions, grounding the elements of this structure in the environment, and then performing inference to find the most probable path given the directions and observations of the environment.

The first part of this process is to extract the linguistic structure using SDCs. The SDCs comprise a hierarchy of structured clauses. Each SDC consists of four fields: a FIGURE (the subject of the clause), a VERB (the action to take), a LANDMARK (the object of the clause), and a SPATIAL RELATION (a geometric relationship between the figure and the landmark). These fields may be specified implicitly, such as the implicit "you" in an imperative sentence. These SDCs are hierarchical, so that every sentence has a single top-level SDC that may have child SDCs in one or more of its fields.

The authors collected a corpus of 150 natural language route instructions from volunteers who were unfamiliar with the test environment. Each of 15 subjects was given a tour of the test environment and then asked to write directions between 10 pairs of starting and ending locations. To cross-validate these directions, subjects were later asked to follow the directions given by a different subject; they achieved an 85% success rate at reaching the intended destination. To prove that the SDC formalism is capable of capturing the linguistic structure of the natural language directions, the authors conducted hand-annotation of their corpus into SDCs, finding that the limitations of the SDC model were minor and were outweighed by the model's ability to capture the important parts of the semantics of the route instructions.

Since hand-annotation of the instruction corpus is an infeasible limitation on the system, the authors instead devise an automatic SDC extraction system. Instead of being strictly hierarchical SDCs, as theoretical SDCs are, the automatic system approximates the hierarchy with a sequence of SDCs. The system labels each word each sentence with one of the four possible fields (FIGURE, VERB, SPATIAL RELATION, and LANDMARK) or none. Then a greedy algorithm groups continuous chunks together into SDCs. Although this automatic model lacks a hierarchical structure, the authors claim it still manages to sufficiently preserve the sequential nature of the natural language directions, since 60% of the automated SDCs correspond exactly to the hand-annotated SDCs.

The authors next explain their probabilistic model formulation. They formulate the problem of understanding natural language directions as inferring a sequence of viewpoints given a set of natural language directions. With S representing the sequence of SDCs, P representing the path (expressed as a series of viewpoints), and O representing the set of detected objects, the solution to the model is the path P^* that maximizes the joint probability of P and S, given O, or $P^* = \operatorname{argmax}_P p(P, S|O)$.

As the authors discuss their methods for grounding their SDCs in the environment, three pertinent differences with my **VADER** system arise. First, the authors model the verbs as one of three options: "turn left," "turn right," and "go straight." This choice, while it fits their model well, is somewhat perplexing. They combine verbs with spatial prepositions and then attempt to quantify the amount of turning in a particular instance of a verb through a penalty function. In actuality, there is only one verb that a mobile robot can do: it can “go.” Thus **VADER** uses a single fixed verb and instead focuses on using the meanings of the spatial prepositions to determine the shape of the path. My approach thereby leads to both better clarity and greater generality in the application of natural language to robot paths. Second, when grounding spatial prepositions, the authors make the conscious decision to focus only on dynamic spatial prepositions, or those which describe the features of a path. They ignore all prepositions which localize an object, known as static spatial relation prepositions. **VADER**, on the other hand, utilizes both spatial relation and path prepositions. Finally, the authors of this paper admit that their system has no way to disambiguate between multiple objects of the same class (*e.g., the box behind the table* versus *the box in front of the table*). This limitation is a direct result of their choice to ignore spatial relation prepositions. In contrast, **VADER** takes advantage of the information contained in spatial relation prepositions and can differentiate between same-class objects as long as there exists a way to describe the objects differently (see Section 4.4.2 and Section 5.4.1).

The SDCs represent one major component of the input to the authors’ system. The other components are a semantic map of the environment, and the starting location of the robot. The latter is a simple given location. The former is built from a topological map of the environment, which is simply a discretized form of the SLAM data collected from a physical robot in an unrelated experiment. Starting from a grid map, the system automatically segments spaces based on visibility and detected objects, then extracts a topology from the segmentation. This topology is then turned into a semantic map by manually seeding it with the locations of 21 types of known objects. The system then uses object-object context [38], trained with over one million labeled images downloaded from Flickr, to predict the locations of unknown landmarks.

The simplifications discussed above lead directly to stark and important differences between this work and mine. The use of a grid (*i.e.,* discrete) map of the environment, while a popular choice (*e.g.,* [7, 11] and others), does not have the complexity of the noisy, continuous-valued, real-world data on which **VADER** operates. Additionally, all of the experiments conducted in this paper are done in a simulated environment; the authors did not move a real robot with the system they describe in this paper. Even though the authors use data (odometry, LIDAR scans, and video) from a real robot, this is a *post hoc* use of a dataset from a SLAM experiment unrelated to this work. Thus they are able to completely discretize the environment and represent all state variables as discrete symbolic primitives. **VADER**, on the other hand, operates strictly in the continuous physical world and must deal with sensor data that is densely sampled and sometimes ambiguous. My work has seen **VADER** perform automatic driving of paths described in natural language, detailed in Section 4.5.3.The authors' system also requires hand-drawn examples of the meanings of path prepositions in order to learn the prepositions. In contrast, my system learns meanings of prepositions—which can be used to describe both a path and a spatial relationship—from odometry data of actual driven paths of the robot paired with sentences that describe that path. This work is described in Section 4.5.1.Finally, their approach of classifying object locations through a visual bag-of-words model mined from Flickr separates the computer vision problem from the natural language problem. While my work in Chapter 6 also addresses the vision problem independently of language, my work in Chapter 7 utilizes sentential descriptions of robot paths to both localize and label objects in the robot's environment.

2.3 Following Directions Using Statistical Machine Translation [9]

This work describes experiments in performed in simulation on a discrete topological grid map derived from SLAM data from a physical robot, similar to Kollar *et al.* [8], although their maps are segmented using the approach in Friedman *et al.* [39]. They focus on using statistical machine translation to map from imperative natural-language directions (*e.g., take the second left*) to a *path description language* which directs the motion of their simulated robot, taking advantage of the geometry of the map to constrain the number of possible actions at a given location. They learn no individual word meanings, as **VADER** does in Chapters 4 and 5. Rather, they learn the mappings between phrases, such as *take the second left*, and the sequence of simulated robot commands that follow that particular instruction. Their system is not capable of handling natural-language descriptions of objects or other landmarks that appear in the environment. Their evaluation is on a very small corpus of only fourteen sets of route instructions. They also use an oracle to evaluate whether the simulated robot reaches the correct destination by the intended route.

2.4 Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation [10]

This paper introduces a new model for understanding natural language commands, which they call Generalized Grounding Graphs (G^3). In a departure from previous approaches which used a fixed, flat structure to infer the likelihood of an action sequence given the command and the environment [8, 9, 40], G^3 uses a model that incorporates the command's hierarchical and compositional semantic structure. In G^3, this structure is used to dynamically instantiate a probabilistic graphical model for the natural language command. The authors' system then performs inference on this model after training the parameters of the model on a corpus of commands paired with videos in the domain of a simulation of a robotic forklift operating in a warehouse. The authors collect these command/video pairs through crowdsourcing via Amazon Mechanical Turk (AMT). The authors ask untrained users to write a natural language command that they would give to an expert human operator to describe the maneuver depicted in the video created from the simulation of the forklift performing tasks in a warehouse environment. The authors once again return to AMT for evaluation of the end-to-end performance of their system. After inferring motion plans from natural language commands and executing the plans in a robot simulator to produce a video depiction of the actions, the authors then ask AMT users to evaluate how well the command matches the simulation video on a 1-to-5 Likert scale.

The authors frame the problem as inferring the most likely robot state sequence that would result from a natural language command. Their hierarchical and compositional structure comes from the recognition of the innate linguistic structures in the natural language command, such as realizing that the verb "put" often has two arguments attached to it: the object being put and the destination of the object. The authors use Spatial Description Clauses (SDCs) (a semantic structure introduced by [8]) as the building blocks of their grounding graphs. Each SDC is a linguistic constituent (a word or phrase) from the command, automatically extracted from the natural language input by use of the Stanford Parser [41], that can be mapped, or *grounded*, to an aspect of the world. These groundings are derived from a semantic map of the environment, which the authors define as a metric map with the location, shape, and name of each object and place, along with a topology defining the connectivity. The details of each grounding are captured in the fields of its SDC. The authors note that the SDCs, groundings, and semantic maps they use must be manually annotated; every EVENT (action sequence), OBJECT (physical thing in the simulated environment), PLACE (location in the simulated environment), and PATH (route through the simulated environment), as well as the mapping of relationships between such, is built by hand.

The main differences between this work and mine are similar to the differences with [7, 11]. Namely, it deals only with a simulated robot environment, rather than a real robot in the physical world. As such, it does not deal with noisy, ambiguous, and sometimes conflicting sensor data, as **VADER** does. Rather, it is able to use unambiguous discrete symbolic primitives for its state. This system also reduces continuously-valued quantities from its world map, such as distance and angle between objects, into discrete uniform bins. This effectively simplifies their environment to a set of discrete locations that, while larger than the set of locations in [11], is still countably finite. Additionally, their use of hand-annotated data for all of their groundings reduces their problem to one much simpler than the one **VADER** solves. They do not learn the meanings of the nouns or prepositions in their environment, they simply learn how to navigate between given locations based on natural language input. Finally, the full extent of this work corresponds to the Acquisition task I do with **VADER** in Section 4.5.1.This system is incapable of generating sentential descriptions from given paths, as I do in Section 4.5.2, or automatically computing and following paths from given sentences, as I do in Section 4.5.3.

2.5 Learning to Interpret Natural Language Navigation Instructions from Observations [11]

This paper describes a general framework for learning to interpret navigation instructions given only sample observations of humans following similar instructions. The instructions are text-based, and the training samples have varying levels of occurrence for the various words to be learned. The authors' system infers a formal navigation plan for each instruction based on the observed actions. The system then learns a semantic parser that can map novel instructions into navigation plans. The system assumes no prior linguistic knowledge, and only receives supervision in the form of the observations of how humans respond to sample instructions.

More formally, the system uses e_i to represent a natural language instruction, w_i to represent a description of the state of the world at a discrete instant in time, and a_i to represent an observed action sequence. It thus uses sets of training data of the form $\{(e_1, w_1, a_1), (e_2, w_2, a_2), \ldots, (e_n, w_n, a_n)\}$. The goal of the system is to produce the correct a_j given a previously unseen (e_j, w_j) pair. Since there is a many-to-one correspondence between navigation instructions and routes (*i.e.,* many different sets of instructions can describe the same route), there is not always a direct correspondence between e_i and a_i. The authors overcome this problem by finding an unobserved plan, p_i, corresponding to a given e_i, that when executed in w_i will produce a_i.

This paper's main contribution is the formulation of the unobserved plan p_i, which is similar in concept to the Hidden Markov Models used in the Acquisition section of my language grounding work (Section 4.5.1). However, here this mapping exists only at the instruction level. In contrast, **VADER** learns at the individual word level.

The system described in this paper only exists as a software simulation. Furthermore, the environment is a simple dungeon world of corridors that intersect at right angles and at distances that are integer multiples of a discrete step size. These corridors differ in their floor patterns and wall decorations in order to make different locations identifiable. Curiously, this simplistic environment also appears in [7] in a nearly identical figure, despite the fact that these two papers share no overlap in authorship.

The instruction set available to the simulated robot is limited to exactly three instructions, represented as symbolic primitives: TURN left or right an integer multiple of 90°, TRAVEL forward N steps, and VERIFY surroundings with an expected wall or floor pattern. Thus their system is able to utilize the internal representation of the simulation to obtain discrete symbolic primitives, and then learn the mappings from the natural language elements of the instruction to such primitives. Additionally, since they only implement three instructions and have a fixed-grid environment, the number of configurations that their simulation can take is countably finite.

My system, **VADER**, on the other hand, exists in the continuous physical world. It can thus take an uncountably infinite number of configurations, and it can move to any location in the 2D Cartesian plane. Furthermore, **VADER** does not map a complete instruction to an action within a specific environment. Rather, it learns the labels of objects located within the environment as well as the parameters of continuous-valued functions used to represent the spatial relations expressed by prepositions. It thus has infinite combinatorial complexity—via noun phrase recursion—in its use of natural language. As long as the meanings of the nouns and prepositions in a sentence have been learned, **VADER** can parse and comprehend sentences of arbitrary length.

2.6 Learning to Parse Natural Language Commands to a Robot Control System [12]

In this paper, the authors seek to learn a semantic parsing model which takes natural-language instructions and produces robot-executable commands in what they call *Robot Command Language* (RCL). They learn a distribution over possible RCL sequences through the use of natural-language commands, which are first hand-segmented into discrete movement phrases and then paired with expert-created annotations of such in RCL. This is similar to both [7] and [11]; all learn the mappings from phrases to commands rather than representations of the meanings of individual words. Finally, the authors test their system by simulating the movement of a robot within a discrete grid-based virtual indoor environment and making a binary judgment on whether or not the simulated robot reaches the intended destination by the desired route. Overall, this represents a simpler problem than the one I solve with **VADER** in Chapters 4 and 5.

2.7 Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions [13]

This work seeks to learn a semantic parser as well as individual word meanings, represented as the grounding of objects or relationships within the test environment. The authors utilize the same dataset as [7] and [11]. Thus their system operates in simulation within the same discrete grid-based virtual environment used in those papers; however, they hand-filter the dataset to include only correct sentence-trace pairs. Similarly, this work also requires a training set in which natural-language instructions are manually paired with executable action sequences.

2.8 Learning Perceptually Grounded Word Meanings from Unaligned Parallel Data [14]

This paper is an extension on the work previously done in [10], with the same lead author on both. The aim of this work is to relax the previous requirement to have manually annotated groundings within their G^3 framework. They seek to build a system that can learn the meanings of words from a corpus of video simulations of a robotic forklift performing different actions paired with natural language descriptions of such actions. This is similar to the Acquisition task (Section 4.5.1) that **VADER** does. This system also follows paths derived from sentential descriptions, much like **VADER**'s Comprehension task (Section 4.5.3).

The authors claim success at their word-learning task, although they give scant detail on which words they learn or how they represent these meanings. They give only a single example of the word “to” with an accompanying heat map to show the meaning. It is thus an improvement over their previous work in that the system no longer requires manual annotation of the grounding of each word. However, their system still requires manual temporal segmentation and alignment between paths and the pieces of multi-part descriptions. **VADER**, on the other hand, can learn meanings for nouns and prepositions without any such manual intervention. Furthermore, since this work still occurs within a simulated environment, it still suffers from many of the same simplifications that plague [10]. Namely, the use of a discrete map, a discrete space of possible groundings, and a discrete set of robot actions reduces their set of possible configurations to a countably finite number. Also, the use of a simulated environment enables the use of unambiguous and discrete symbolic primitives to represent state. Conversely, **VADER** exists in the continuous physical world. It can move to any position in the 2D Cartesian plane, thereby taking an uncountably infinite number of configurations. **VADER** also relies on sensor data that is noisy, continuously valued, densely sampled, and sometimes ambiguous to determine its state. Simply stated, the language-related work with **VADER** shown in Chapters 4 and 5 solves a more complex problem within a more complex environment.

2.9 Where to Go: Interpreting Natural Directions Using Global Inference [15]

This work attempts to devise a method to determine the correct path through a previously-mapped area from a set of ambiguous, noisy, and possibly incorrect natural-language directions. This is similar to **VADER**'s Comprehension task addressed in Section 4.5.3.particularly in the use of global inference and Hidden Markov Models. However, Wei *et al.* solve a simpler problem. They use a discretized map of a real-world environment, which limits their navigation to a finite number of locations and constrains movement possibilities at each location. Also, they only perform experiments in a simulated environment, allowing them to use symbolic primitives which perfectly describe their simulated agent's state. **VADER**, conversely, must contend with the uncertainty and noise attendant with using sensor data in a real-world environment. Additionally, they assume that a set of directions will never loop back to revisit previously-occupied locations; **VADER** has no such limitation. Finally, their evaluation of direction-following involves only a simple binary decision on whether or not their software agent reached the correct destination. This is much less rigorous than the evaluation used in either Section 4.6 or Section 5.6.2.

2.10 Teaching a Robot Spatial Expressions [16]

This paper seeks to learn the meanings of spatial expressions (*i.e.,* prepositions) using a real robot. To do so, the authors collected a corpus of a set of paths generated from robot odometry data paired with human-generated commentary of such paths. This is very similar to **VADER**'s Acquisition task in Section 4.5. Additionally, their selection of prepositions to be learned is analogous to the prepositions in **VADER**'s grammar, both in number of prepositions and actual words to be learned.

The authors are able learn meanings for prepositions, but only for simple natural language phrases like *A is left of B*. To do this, they require hand-grounded nouns; their system possesses no capability to classify objects based on natural language, as **VADER** does.

Another puzzling aspect of this system is the authors' choice to use binary decision trees to represent their learned prepositions. The trees they present seem to show that their learning system keys on parameters unrelated to the actual word meanings, such as the x-coordinate of an object when trying to make an *in front of/behind* decision that should rightly be based on the y-coordinate. The authors recognize this deficiency and suggest that bias in their training corpus could be to blame. In contrast, **VADER**'s use of Von Mises distributions [42] to represent preposition meanings better captures the continuous and spatial nature of the words. This also has the added benefit of making such meanings easily intelligible to human readers (Fig. 4.2, bottom two rows).

While the system described in this paper does learn a type of preposition meaning from real robot data, **VADER** is far more advanced. In addition to learning nouns as well as prepositions, **VADER** can also generate sentences describing driven paths and automatically drive a path described by a sentence (see Chapters 4 and 5). The system described herein can do none of these.

2.11 Mobile Robot Programming Using Natural Language [17]

This work uses a collected corpus of high-level route instructions given by naïve users to a small robot operating in a miniaturized outdoor setting in order to learn symbolic representations of motion tasks that can generalize such instructions. The system assumes the use of pre-programmed primitives which encode the basic sensory-motor actions of the robot. The authors manually analyze their collected corpus and extract a set of 14 primitives, from the simple (TURN RIGHT) to the complex (TAKE THE NTH TURN AFTER X), which they ground to robot sensory-motor actions by hand. Their learning occurs at a higher level of abstraction than mine, focusing on finding the mappings from natural-language route instructions to sequences of these hand-crafted action primitives. Conversely, **VADER** learns meanings for individual nouns and prepositions, using such to both generate sentences which describe paths and compute and follow paths from sentential descriptions, as shown in Chapters 4 and 5.

2.12 A Voice-Commandable Robotic Forklift Working Alongside Humans in Minimally-Prepared Outdoor Environments [18]

This paper describes the development of a multi-ton robotic forklift intended to operate near people. Specifically, this robot is designed to handle palletized cargo within semi-structured outdoor military warehouse facilities, commonly known as Supply Support Activities. The authors highlight three principal novel characteristics of their work. First, they develop a multimodal tablet interface that allows the human supervisor to use both speech and stylus-based touchscreen gestures to assign tasks to the robot. Second, the robot operates in minimally-prepared, semi-structured environments, handling palletized cargo of variable dimensions and mass using only local sensing. The robot is equipped with GPS, but it is only used for coarse localization, not navigation. Third, the robot operates in close proximity to people, which is made possible by novel interaction mechanisms that facilitate safety and effectiveness in the often chaotic environment of a military warehouse.

The authors' robot is capable of executing a limited set of commands to approach, engage, transport, and place pallets in an outdoor warehouse area with little formal structure. It understands a small set of spoken commands directing movement—an example given was “come to receiving”—issued by the supervisor via the tablet computer interface. The speech recognition on capabilities, both on the tablet and on the robot itself, are provided by an off-the-shelf system, SUMMIT [43]. This system, coupled with beam-forming microphones covering a full 360° circle in the horizontal plane, makes up the *shouted warning detector* that pauses operation whenever a bystander shouts a command to stop. The robot also has an *annunciation subsystem*, consisting of LED signs, marquee lights, and speakers, that announces the robot's state and intended actions to bystanders through visual and audible cues. Additionally, the robot uses lidar to detect obstacles in its path and conservatively assumes all detected obstacles to be people. The default behavior is for the robot to slow down and attempt to navigate away from obstacles that breach an outer threshold, then stop when a moving obstacle, such as a pedestrian or vehicle, breaches an inner threshold. Furthermore, the integrity of the system is maintained through a layered arrangement of heartbeat signals between the sensors/actuators, on-board computers, and supervisor's tablet. When a specified number of heartbeats are missed, the robot defaults to a stopped state. These features combine to allow the robot to operate safely, meeting all OSHA requirements for warehouse safety.

The robot utilizes an extensive suite of sensors to enable its autonomy. The sensors, as well as the actuators necessary to provide computerized control of the forklift robot, are controlled by a set of four networked quad-core laptops mounted on the robot, with a fifth on-board laptop serving as a diagnostic display. A commodity wireless access point provides connectivity to the supervisor's tablet. The proprioceptive sensors, including wheel encoders, an IMU, and a GPS, provide self-sensing. Navigation is primarily based on dead-reckoning via the encoders, with GPS providing only coarse position estimates. The IMU estimates short-term 6-DOF vehicle motion, which is necessary because the packed earth and gravel surface of the outdoor warehouse environment requires a non-planar terrain representation, leading to full 6-DOF chassis dynamics. The exteroceptive sensors, including a total of 15 lidars and top-mounted cameras that published a 360° view to the supervisor's tablet, provide sensing of the external environment. For each of these sensors, the system estimates the rigid-body transformation relating the sensor's frame to the chassis frame in order to have the entire robot extrinsically calibrated to a common coordinate system.

This common coordinate system, provided to the supervisor through the camera view on the tablet controller, allows for supervisory gestures to be grounded in a world model common to both human and robot. Once the supervisor issues a verbal movement command, he or she can give more explicit instructions through the tablet by circling the image of a destination location or that of an object with which the robot is to interact. Since the cameras and lidars are in a common coordinate frame, this circle can be translated into a *volume of interest* corresponding to the interior of the cone emanating from the camera and having the circle as its planar cross section. The lidar is thus able to identify the object or area circled by the supervisor.

The path planning module on the robot is adapted from the framework developed at MIT for the DARPA Urban Challenge vehicle [44, 45]. It has a navigator subsystem that identifies a waypoint path with a closed-loop prediction model that incorporates pure pursuit steering control [46] and PI speed control. The motion planner subsystem then uses this prediction model to generate sets of feasible and safe trajectories towards waypoints. The controller subsystem then executes a trajectory that is selected in real-time to minimize an objective function. In testing, the authors observed a difference between predicted trajectory and actual path that averaged 12 cm, with a maximum of 35 cm, for a set of 97 paths with lengths from 6 m to 90 m.

While the robot described in this paper is clearly larger and more complex than **VADER**, it shares some striking similarities and differences. Like this robot, **VADER** also relies on wheel encoders and an IMU for self-localization via dead reckoning. However, instead of having over twenty on-board processors to deal with sensor data, **VADER** has only three. Yet **VADER** is still able to handle all motor control, localization, video streaming, and communication tasks with this order-of-magnitude less processing power. Furthermore, this system does not do much to advance the state of the art in controlling robots with either natural language or computer vision. The spoken commands that this robot can respond to are rendered as text via the SUMMIT speech recognition system and are thus no different than the text input to **VADER**. Additionally, while the commands that **VADER** understands are infinite in their combinatorial complexity, the commands that this robotic forklift understands are limited to a small set of specific utterances. This so-called understanding is simply pre-programmed responses to commands; this system does no language learning and requires manual guidance when a user wishes to have the robot exceed the limits of these responses. Finally, the use of the tablet interface to have the user circle *volumes of interest* shows that the system also does no learning in the computer vision realm. The object detection in this system is directly driven by human input. **VADER**, while undoubtedly a smaller and less complex system, is nonetheless more advanced in the realm of natural language understanding (Chapters 4 and 5). **VADER** can also detect and classify objects in its environment with minimal to no human input (Chapters 6 and 7).

2.13 Report on the Second NLG Challenge on Generating Instructions in Virtual Environments (GIVE-2) [19]

This work shows some similarity to the Generation task in Section 4.5.2;while **VADER** generates sentences that describe a robot path, their systems provide instructions to guide human users through virtual game worlds. Unlike **VADER**, however, the systems described therein operate in a virtual environment, which gives them a complete and unambiguous discrete symbolic representation of the world. Additionally, their systems do no learning and instead rely on hand-coded mappings between words and actions in order to generate instructions. **VADER**, on the other hand, learns word meanings from sentential descriptions of physically-driven paths.

2.14 Heterogeneous Multi-Robot Dialogues for Search Tasks [20]

This paper focuses on using spoken language to allow humans to interact with multiple robots in a “treasure-hunt” scenario. The authors go into great detail on how they built their front-end architecture for turning spoken language into logical form using components from [47–53]. However, they provide no detail on the actual speech input they use or how they parse it into a logical form, effectively treating this system as a black box. Therefore, it is impossible to compare their unknown grammar with the grammar I use with **VADER** in Chapter 4. However, I can compare their logical form to the one I use with **VADER** (Fig. 4.7). Their logical form has exactly three types of formulas to represent three distinct commands that a user can give a robot: a REPORTCOMMAND, a LOCATIONQUERY, and a MOVEVECTOR. These formulas have a fixed number of arguments for each, and have no combinatorial complexity. My logical form used with **VADER**, on the other hand, has a single formula type but supports infinite combinatorial complexity through the unlimited addition of conditions (see Section 4.4.1).

Furthermore, while they claim to implement and test this system on physical Carmen robots [54], they only provide results of a Carmen simulation. Additionally, the experiment they describe uses a fixed, pre-determined map of the environment. Their robotic participants have perfect knowledge of not only this map, but of their own location within such. **VADER** also uses a pre-determined map of its environment in Chapters 4 and 5, but it must contend with noisy, real-world sensor data to determine its location in real time. Furthermore, in Chapters 6 and 7, I show how **VADER** can operate without the need for a human-generated map of its surroundings.

2.15 Exploring Spoken Dialog Interaction in Human-Robot Teams [21]

This work describes TEAMTALK, a human-robot interface which allows verbal interaction between multiple people and multiple robots in a virtual treasure-hunt scenario similar to Harris *et al.* [20]. Its primary goal is to construct a policy which is able to handle dynamic and asynchronous conversation between a group of four or more humans and robots. Like [20], TEAMTALK uses a discrete grid-based simulation environment with a symbolic representation. TEAMTALK also does no learning.

2.16 The Structure and Generality of Spoken Route Instructions [22]

This paper collects and present the NAVIGATI corpus, a dataset of human-generated route instructions in an indoor environment. The authors' goal is to analyze the corpus and construct a grammar that would generalize to other navigational corpora. The authors focus solely on the linguistic content of the human-generated instructions and neither learn word meanings nor control robots, simulated or real, via language.

2.17 Using Semantic Fields to Model Dynamic Spatial Relations in a Robot Architecture for Natural Language Instruction of Service Robots [23]

This work investigates methods for enabling service robots to better understand spatial language from non-expert users. To do this, the authors use semantic fields to represent prepositions. Their system requires prepositions to be manually encoded as such. It also requires the hand-grounding of nouns to locations on a manually-created map. With such human-generated knowledge, the system is able to perform path planning in a variety of discrete simulation environments. This is similar to the Comprehension task in Section 4.5.3,although **VADER** operates in the continuous physical world using knowledge learned in the Acquisition task (Section 4.5.1).The system in this paper does no learning.

2.18 Multi-Modal Human-Machine Communication for Instructing Robot Grasping Tasks [24]

This work uses gesture recognition and speech commands to interact with a robot arm by human demonstration. This allows a user to train the robot arm to associate a natural-language command with a specific manipulation task. However, this system is unable to respond to a natural-language command for which it is not specifically programmed. Thus, this work merely maps from an utterance to a robotic motion primitive while doing no learning on the underlying meanings of words, as **VADER** does in Section 4.5.1.

2.19 Spoken Language Interaction With Model Uncertainty: an Adaptive Human-Robot Interaction System [25]

This paper presents a dialogue management system designed to learn to overcome the limitations of noisy speech recognition within the paradigm of a robotic wheelchair. All testing of this learning approach is done in simulation using a small number of discrete states. As the authors' focus is on dialogue management, they do no navigation. Rather, they assume that if their system is able to determine the correct goal state, it is also able to navigate to such via a manually-programmed route.

2.20 A Joint Model of Language and Perception for Grounded Attribute Learning [26]

Matuszek *et al.* here propose a joint framework to learn linguistic and perceptual (primarily visual) models for grounding language to physical objects in a static scene, based on the semantic parsing model described in [55]. Their system trains color and shape classifiers on objects segmented from images captured with an RGB-D camera. Then it learns the mapping between those classifiers and linguistic descriptors in the form of adjectives (*e.g., green*) and nouns (*e.g., triangle*). While both this work and the work in Chapters 4 and 5 ground language in robotic perception, the modalities of such perception are so different as to preclude direct comparison. **VADER** perceives its environment via mechanical sensors such as wheel encoders and an IMU (Section 4.3 and Section 5.3) as well as via visual sensors (Chapters 6 and 7), while their system employs only computer-vision techniques on camera data. Furthermore, while their system learns nouns and static adjectives (colors) that describe such, our system learns nouns as well as prepositions that describe noun properties which are both static (spatial relations between stationary objects) and dynamic (spatial relations between a stationary object and a moving robot).

2.21 Back to the Blocks World: Learning New Actions through Situated Human-Robot Dialogue [27]

She *et al.* explore teaching a physical manipulator arm to interact with objects through student-teacher dialogue. Their work connects high-level symbolic representations of language with low-level sensory-motor representations internal to the robot. Like [26], their system perceives the world solely through visual means.**VADER**, on the other hand, uses both visual and mechanical perception in its environment. Additionally, their system learns manipulation tasks through interactive dialogue with a human, while **VADER** learns individual word meanings from a fixed training corpus.

2.22 Simultaneous Object Detection and Ranking with Weak Supervision [28]

The codetection approach used by Blaschko *et al.* operates on weakly-annotated data to detect and localize objects by training SVM classifiers for each object class independently. This weak annotation can be either a simple binary indication of the presence or absence of an object, or an indication of a general object location without precise bounding box coordinates. Additionally, the fact that the training data is split into individual classes is another form of annotation of the dataset. **VADER**'s approach in Chapter 6 uses no human annotation and does not require independent training of object classifiers.

2.23 Learning the Easy Things First: Self-Paced Visual Category Discovery [29]

Lee and Grauman here present an iterative clustering approach to learn object models individually over many passes through an image corpus. Their system estimates the likelihood both that an image region contains an object, and that nearby regions in that image contain instances for which their system has previously-trained models. This requires them to seed their system with a small number of pre-trained object models to start, to which their system adds the models learned in each iteration. **VADER**, on the other hand, does not require pre-trained object models and can classify multiple object classes simultaneously.

2.24 Unsupervised Joint Object Discovery and Segmentation in Internet Images [30]

This work seeks to discover and segment common objects from diverse image collections. The authors' system takes as input collections of images retrieved from Internet search, where each collection comprises the results of a search for a specific object, such as a car. They are able to either automatically localize the common object in each image, or decide that an image does not contain such an object. Like the approach of [28], their approach requires the manual separation of these image collections by object class; this is clearly a form of supervision, even though their title leads with the word "unsupervised." Also, their system cannot learn multiple object models simultaneously. In contrast, **VADER** discovers and segments objects both with (Chapter 7) and without (Chapter 6) additional supervisory data. **VADER** also does so on multiple object classes simultaneously.

2.25 Co-Localization in Real-World Images [31]

Tang *et al.* perform codetection by solving a joint image-box formulation for each object class. Like Blaschko *et al.* [28] and Rubinstein *et al.* [30], their system operates on data that has been manually grouped by object class. Their dataset is constructed so that each image in a set either contains the object of interest or is considered noise because it does not contain such an object. **VADER**, in Chapter 6, uses an unsorted collection of images and is able to separate them into object-based groups automatically.

2.26 Learning Object Class Detectors from Weakly Annotated Video [32]

This work describes an approach for codetection in videos which uses motion segmentation to create spatio-temporal tubes that identify candidate objects. Like several previously-mentioned approaches (*e.g.,* [28,30,31]), this approach requires input data that is grouped by class and thus must learn each object individually. Additionally, this method also requires that each input video has a binary annotation indicating whether or not it contains the class of interest. While **VADER** uses spatio-temporal tubes for candidate objects in Chapter 7, my method does not require such grouping or annotation. **VADER** also learns multiple objects in a single pass.

2.27 Unsupervised Object Discovery and Segmentation in Videos [33]

This paper presents a system which discovers and segments unknown objects in videos through the use of optical flow to estimate motion segmentation. From this the system learns appearance models by clustering both superpixels and bounding boxes. It can thus classify multiple objects simultaneously, as **VADER** can, although the authors test it on fewer object classes.

2.28 Efficient Image and Video Co-Localization with Frank-Wolfe Algorithm [34]

Joulin *et al.* here build upon their previous work [31] to extend such work to video in addition to images. This new approach incorporates temporal consistency between the frames of a video, as **VADER** does in Chapter 7. However, like [31], this approach also requires input data that is manually grouped by object class. The methods in Chapters 6 and 7 require no such manual input.

2.29 Discovering Object Classes from Activities [35]

Unlike most previous codetection work, which can only detect objects that are large, prominent, and centered in the field of view, Srikantha and Gall present an approach which can detect and localize objects which are small and off-center. However, their approach relies upon human pose estimation and thus can only detect objects with which humans interact. The methods in Chapters 6 and 7 can also detect small and off-center objects, but is independent of human activity within a video.

3. DESCRIPTION OF THE EXPERIMENTAL PLATFORM, VADER

In order to conduct my research, I built a custom mobile robot, or rover. I built my rover from hardware components acquired from many different vendors. The total cost, including spares for all components except the cameras and lenses, was approximately $3,000. For the software to run it, I use a mixture of licensed, open-source, and custom-written code. The majority of the code running on the rover itself is in C/C++, with a small fraction of code in Arduino sketches. I also have code that runs on a companion laptop in order to communicate with the rover. This code is all either open-source or custom-written. It is about 60% in C/C++ and 40% in Scheme.

I have named my robot **VADER**, which stands for **V**isually and **AI** **D**irected **E**xperimental **R**obot. The finished product is pictured in Figs. 3.1 and 3.2.

In the following sections, I discuss the overall system, the hardware by subsystem, the software, and the testing performed prior to using **VADER** to collect the data used in Chapters 4 through 7.

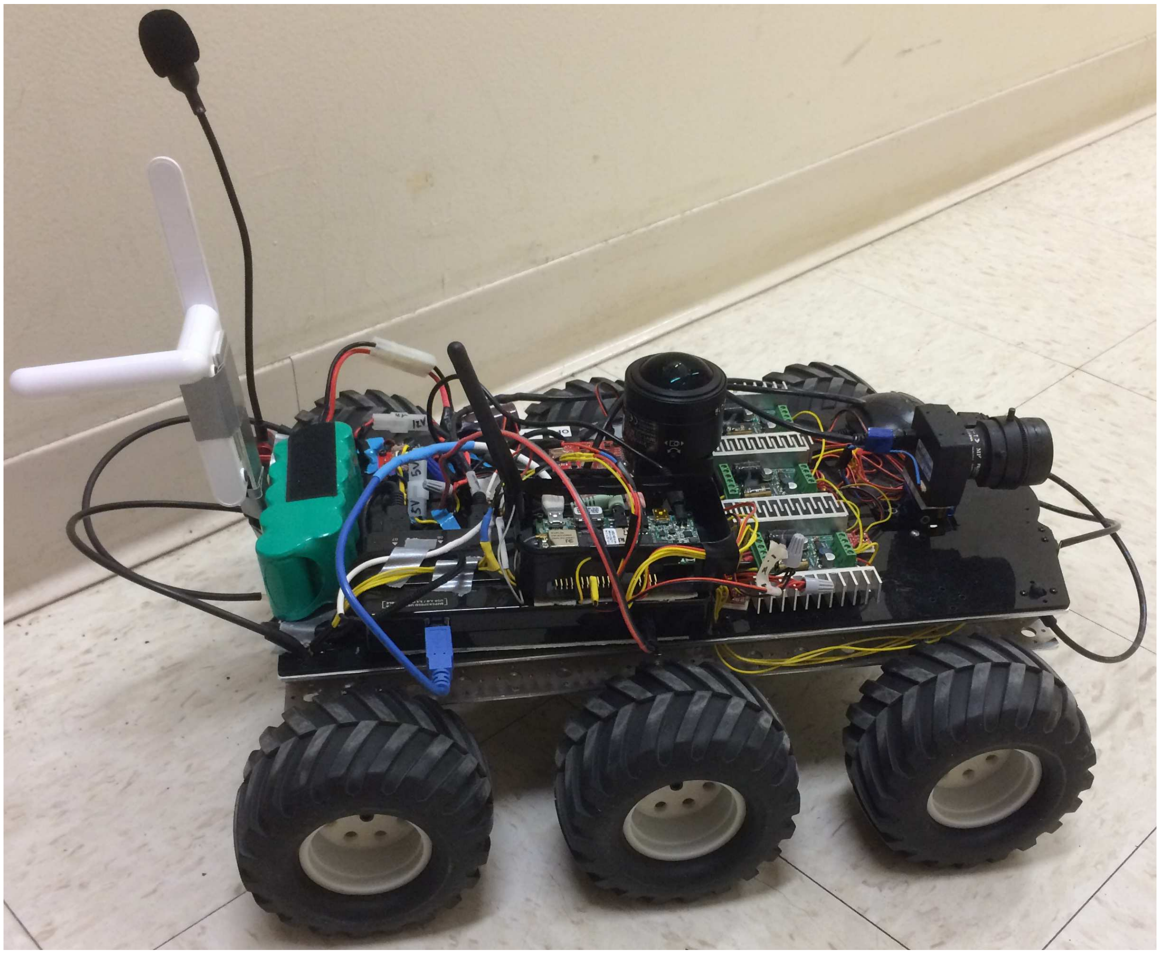
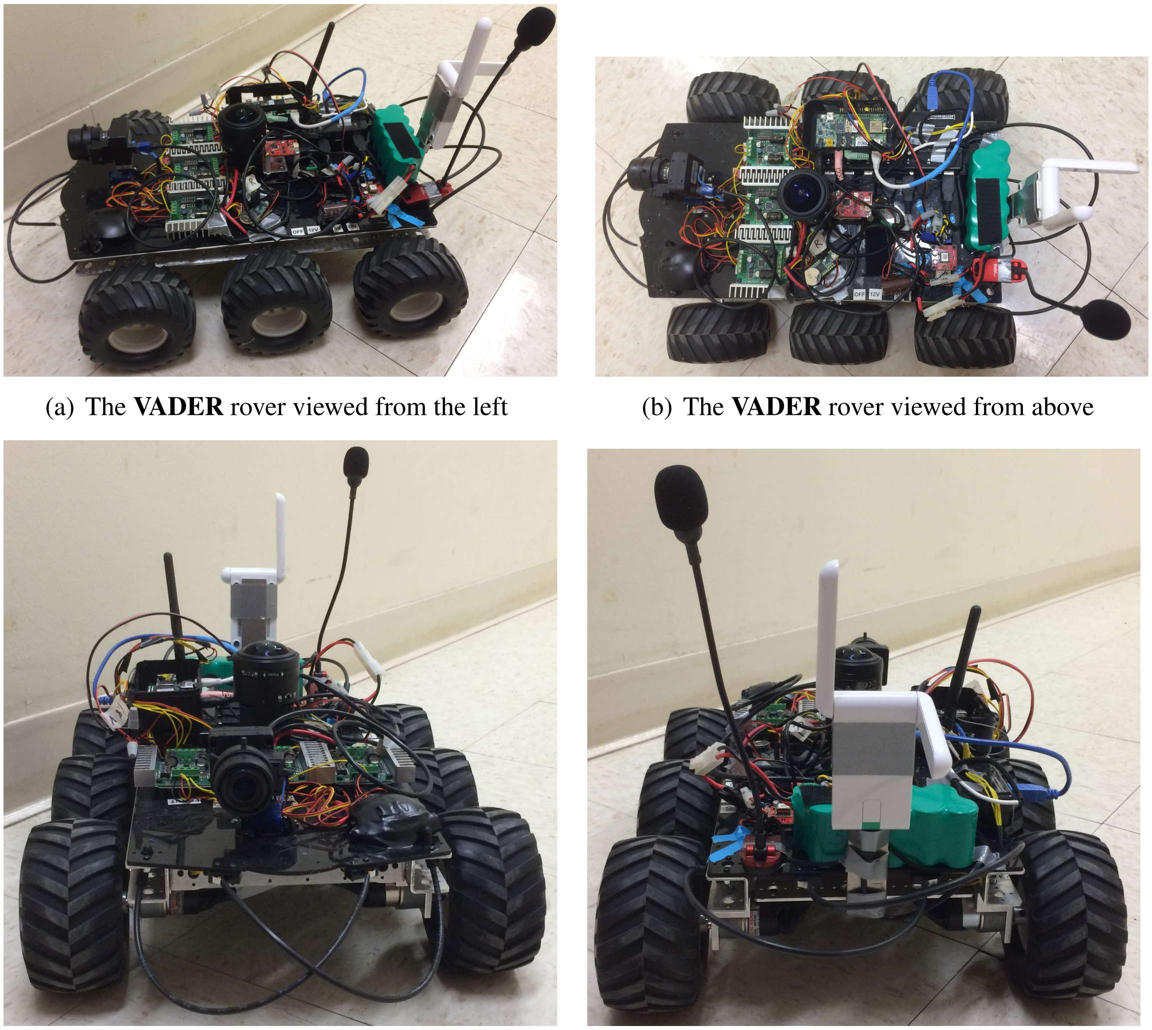


Fig. 3.1.: The **VADER** rover viewed from the right



(a) The **VADER** rover viewed from the left

(b) The **VADER** rover viewed from above

(c) The **VADER** rover viewed from the front

(d) The **VADER** rover viewed from the rear

Fig. 3.2.: The **VADER** rover viewed from multiple angles

3.1 System Overview

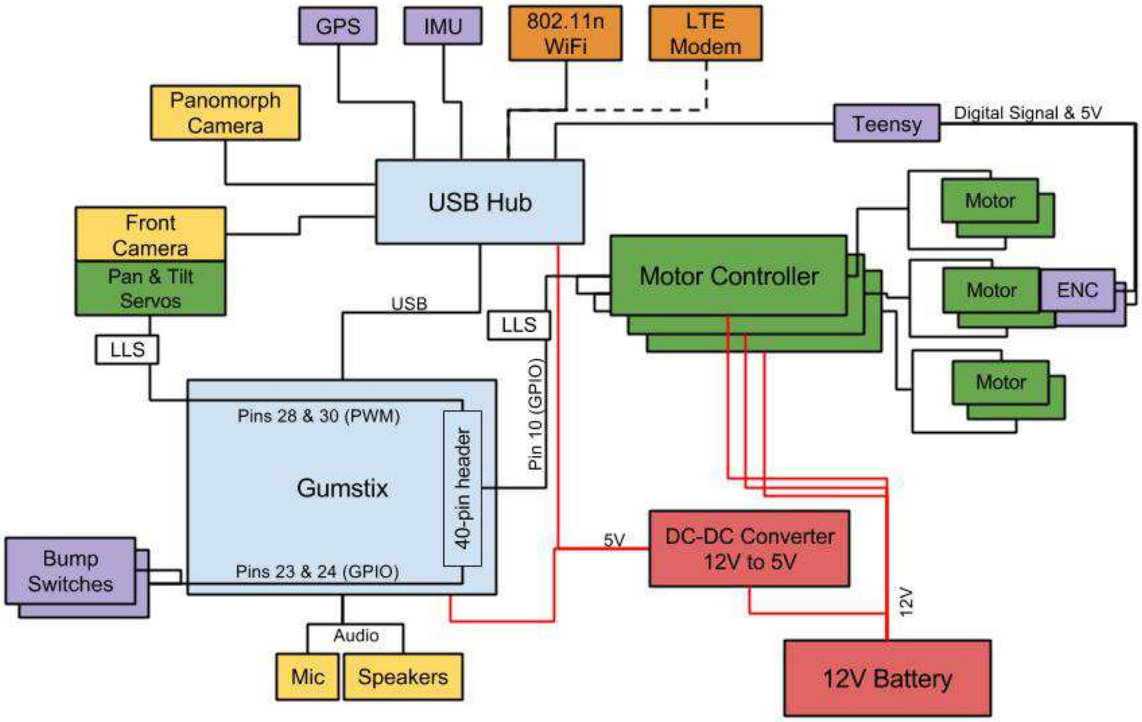


Fig. 3.3.: System diagram for the **VADER** rover. Each subsystem is depicted in a different color. Black lines connecting components represent data or signal lines, and red lines represent power lines. **NOTE**: The power lines going to and from the logic level shifters (LLS) are not shown in order to reduce clutter.

The system diagram for the **VADER** rover is shown in Fig. 3.3. There are six subsystems in **VADER**:

1. The Chassis, Body and Power Subsystem is depicted in red. Since the Chassis and Body do not have any electrical or electronic components, they are not depicted in the diagram.
2. The Main Processor Subsystem is depicted in blue.
3. The Mobility Subsystem is depicted in green.
4. The Communication Subsystem is depicted in orange.
5. The Audio and Video Subsystem is depicted in yellow.

6. The Sensor subsystem is depicted in purple.

The black lines in Fig 3.3 represent data or signal lines. The red lines represent power lines. The Mobility Subsystem also requires two logic level shifters (LLS) in order to communicate with the Main Processor. These are depicted in white. The power lines going to and from these are not shown in Fig. 3.3 in order to increase clarity. I discuss each of the listed subsystems in individual sections below.

3.2 Hardware by Subsystem

3.2.1 Chassis, Body and Power Subsystem

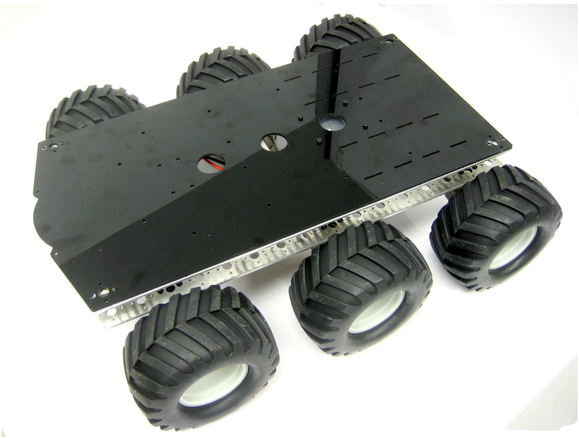
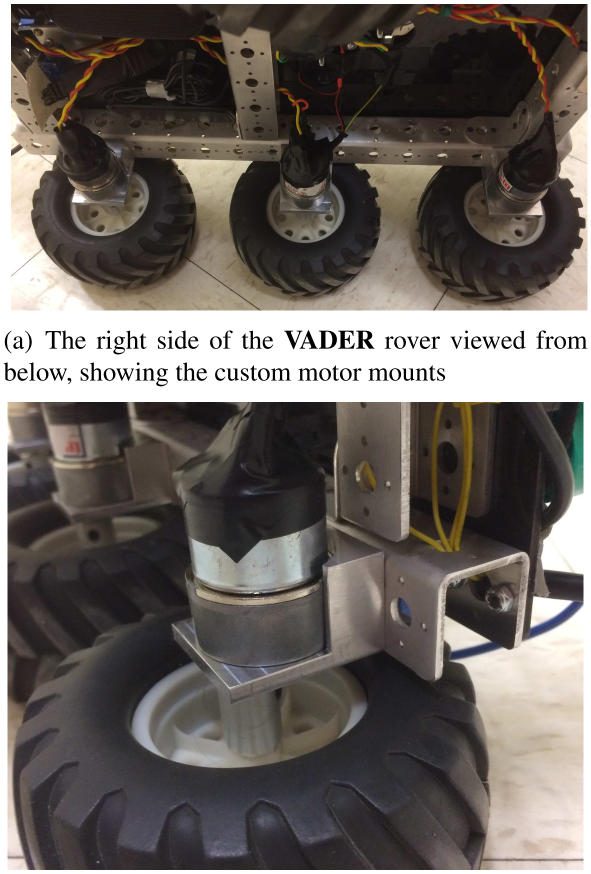


Fig. 3.4.: The A6WD2 rover kit from Lynxmotion [66]

To start building a vehicle, one must have a chassis and body on which to build. For the **VADER** rover, I use the A6WD2 Kit from Lynxmotion [66]. This is a bare-bones kit that only comes with a frame, motors, wheels, and tires. The kit is pictured in Fig. 3.4. The motors will be discussed further in Section 3.2.3. The tires and wheels measure 12 centimeters tall and have 64 millimeter wide angled treads. This kit is designed for differential steering; much like a tank, all wheels on a side will speed up, slow down, or reverse direction to accomplish a turn or pivot.

One interesting thing about the chassis, which I only discovered when the rover was complete and driving, was that the motor mounting brackets included with the kit were not strong enough to support my payload without flexing or shifting against their mounting screws. These original motor mounts are simple aluminum brackets bent at a 90°angle. Once the rover was driving, these brackets made it impossible to keep the wheels aligned on a common axis, which played havoc with doing things as simple as keeping the rover driving in a straight line. Therefore, I replaced them with custom-made mounts milled from a solid block of aluminum. These custom mounts are the work of the Purdue ECE Machine Shop, and once installed they solved all wheel alignment issues. The custom mounts are pictured in Fig. 3.5.



(a) The right side of the **VADER** rover viewed from below, showing the custom motor mounts

(b) Closeup view of the custom motor mount

Fig. 3.5.: Images of the custom motor mounts

The next component of this subsystem is the battery. I use a 12V Ni-MH 2800mAh battery pack, also sold by Lynxmotion [67]. The battery is mounted using Velcro strips at the rear of the robot, shown in Fig. 3.6. When collecting data, this battery allowed me to run the rover for about 2 hours before it needed to be changed. This was sufficient time to gather a useful amount of data, and with much practice I was able to accomplish the battery change drill (shut down the robot, change the battery, boot the robot back up, and ensure communications were back online) in less than a minute.

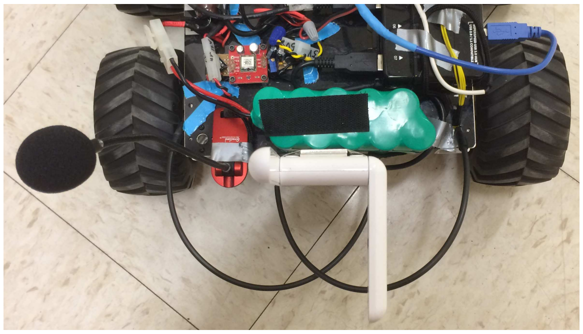


Fig. 3.6.: Closeup of the rear of the **VADER** rover, showing the battery (green)

The final component of this subsystem is the DC-DC converter. Since my robot requires both 12V and 5V power for different subsystems, I have to find a way to provide both. For simplicity I want to do so from a single battery. Therefore I use a DC-DC converter made by SparkFun [68]. My installation of this device is shown in Fig. 3.7. This converter allows the user to select the output voltage through a simple voltage divider. I use a potentiometer to set this resistance in order to achieve 5V exactly. I also found through experience that I need to put low pass filters on both the input and output terminals of the converter. Additionally, I learned the hard way that having exposed pins on a voltage converter is a bad idea—I managed to blow up my first DC-DC converter when I accidentally grounded the 5V output with a screwdriver. Therefore, as Fig. 3.7 shows, the new installation got the upgrade of all exposed pins being covered in hot glue.

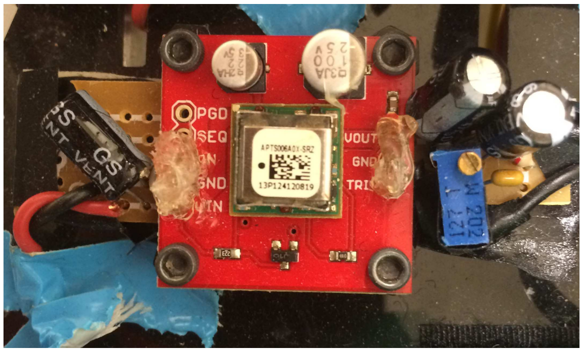


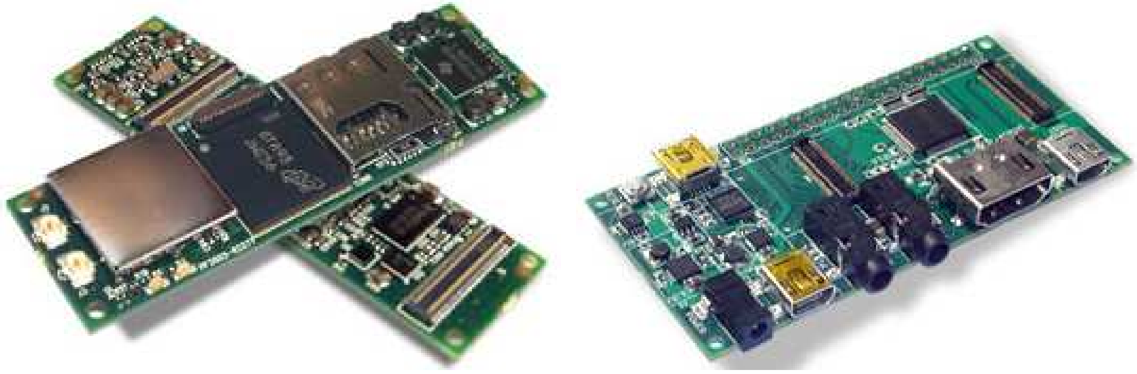
Fig. 3.7.: Closeup of the DC-DC converter, which is located just forward and to the left of the battery

3.2.2 Main Processor Subsystem

The heart, or, perhaps more accurately, the brain, of the **VADER** rover is the Gumstix® Overo® FireSTORM COM (Computer-On-Module) [69]. This is a single-board computer that takes its name from the fact that it is nearly the same size as a stick of gum (2.28 by 0.67 inches). It has a single-core ARM Cortex-A8 processor rated at up to 1GHz. It also has 512MB of RAM and a microSD expansion slot for on board storage. The Overo® runs Linaro Linux, a variant of Ubuntu optimized for ARM processors.

In order to interface with an Overo® COM, one must use an expansion board, also sold by Gumstix®. For my application, I chose the Summit [70] expansion board, primarily for its small size (3.15 by 1.54 inches) and the fact that it has a 40-pin header for GPIO and PWM signals, audio input and output, and multiple USB ports. Fig. 3.8 shows the product photos of the Overo® and Summit from the Gumstix® website. Fig. 3.9 shows **VADER** rover with these items mounted. They are on the right side of the rover, between the middle and rear wheels, and atop the USB hub.

Because of the number of USB devices I connect to the Gumstix®, and because many of them require 5V power over USB, I use a USB hub on the **VADER** rover. One would think that something so simple as a powered USB hub would be a simple commodity item. However, I found that not to be true. I started using a few powered USB hubs with the requisite number of ports and soon ran into a host of problems, including insufficient power



(a) Overo® FireSTORM COM

(b) Summit expansion board

Fig. 3.8.: Product photos from [69, 70]

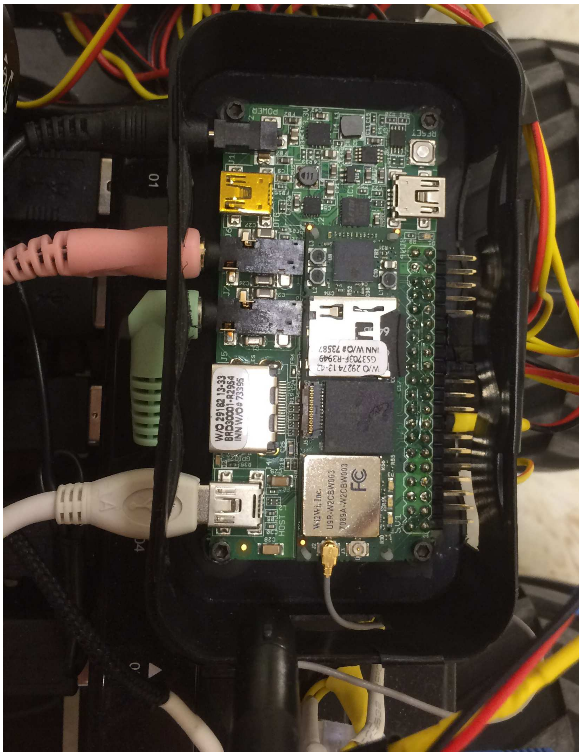


Fig. 3.9.: Image of the Gumstix® system mounted on the **VADER** rover

output and data connections that failed under load. After several rounds of trial and error, I found the USB 3.0 from Pluggable® [71] to be a workable solution. Although the USB 3.0 speed was not useful with the USB 2.0 port on the Gumstix® Summit board, the steady supply of power and solid data connections solved my USB problems. This hub is visible beneath the Gumstix® enclosure in Fig 3.10.

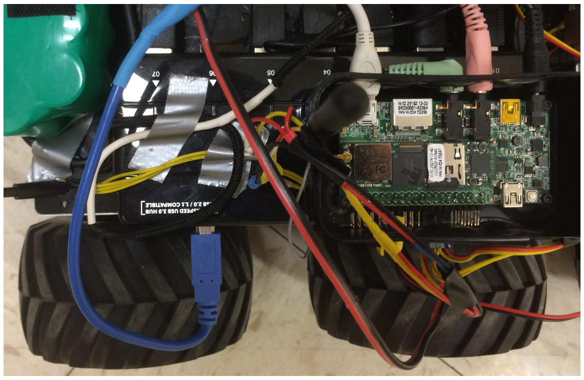


Fig. 3.10.: The Pluggable® USB hub mounted on the **VADER** rover

3.2.3 Mobility Subsystem

The first component of the Mobility Subsystem is the motor controller. I use the Sabertooth 2x12 Regenerative Dual Channel Motor Controller from Lynxmotion [72]. It is pictured in Fig. 3.11. The Sabertooth receives serial data to set motor speeds from the Gumstix® via a GPIO line on the Summit board. However, since the Summit uses 1.8V logic and the Sabertooth uses 5V logic, I also use a logic level shifter (denoted as "LLS" in Fig. 3.3) to do the conversion. The level shifter can also be seen in Fig. 3.11 in the bottom right of the image below the wires.

The Sabertooth is capable of controlling two motors independently. Therefore, in order to control six wheel motors, I use three Sabertooth controllers on **VADER**. The controllers are divided into front, middle, and rear wheels, with each controller handling one right side motor and one left side motor. The serial data that the Sabertooth uses to control the motor speeds is a single byte for each motor channel. Therefore there are 127 forward and 127 reverse speed bytes, with two stop bytes. The Sabertooth then translates this byte into an analog voltage and sends the voltage to the motor. Since 127 speeds in each direction is overkill, I use my motor control software running on the Gumstix® to simplify down to four speeds in each direction, roughly approximated as 25%, 50%, 75% and full throttle. In drive testing I discovered that due to the weight distribution on the rover, I actually have to slightly retard the speeds on the left side motors in order to prevent a rightward pull.

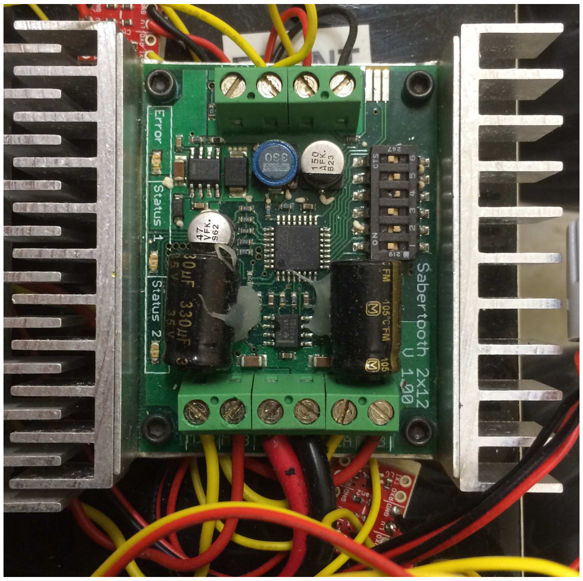


Fig. 3.11.: The Sabertooth motor controller and logic level shifter mounted on the **VADER** rover

The motors I use on the **VADER** rover are the ones included in the A6WD2 kit from Lynxmotion [66]. One is shown in Fig. 3.12. These are simple 12V motors with 30:1 gearing and an exposed motor shaft for mounting a wheel encoder.

In order to measure the distance traveled by my robot, I attach optical digital encoders to my motors. These are shown in the system diagram (Fig. 3.3) as “ENC.” The encoders I use are made by US Digital [73] and are pictured in Fig. 3.13. Since all wheels on each side receive the same motor command and thus should be rotating at the same speed, I only mount encoders to the middle wheel on each side. To ensure minimal slippage and maximal ground contact on these wheels, I add 1/8 inch shims between the robot frame and the motor mounts for these motors. The encoders output a two-channel quadrature signal, so that the channels are 90° out of phase. This allows me to find which direction the motor is turning by seeing which channel is leading and which is lagging. I can find speed and distance traveled by simply counting the pulses. These quadrature signals go to the Teensy

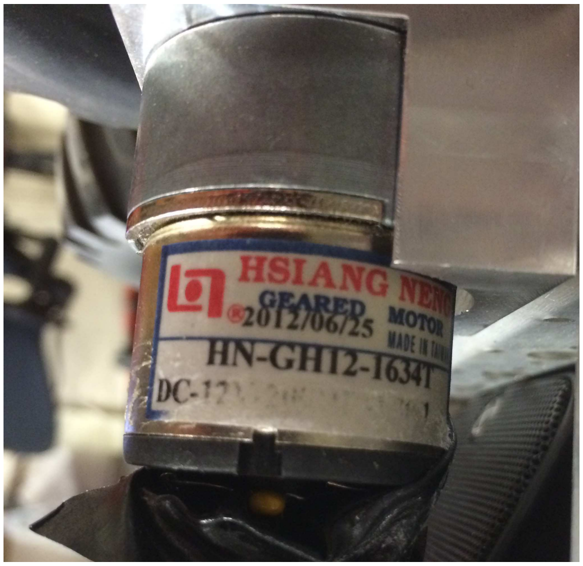


Fig. 3.12.: A wheel motor mounted on the **VADER** rover

board, which reads them and sends data to the Gumstix® at regular intervals (see Section 3.2.6).

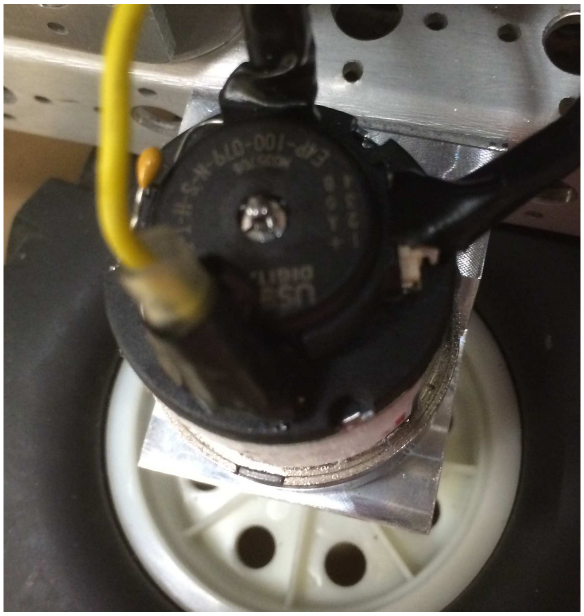
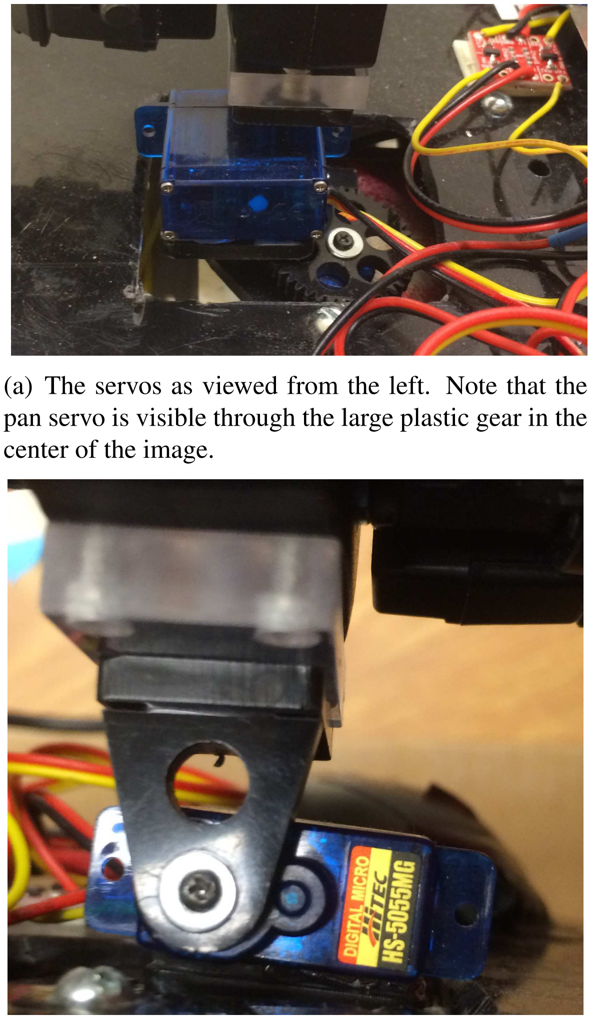


Fig. 3.13.: An optical digital encoder mounted on the **VADER** rover

The final element of the Mobility Subsystem is the pan and tilt servo mechanism that is attached to the front camera. This mechanism is made up of two Hitec HS-5055MG micro servos [74], one each in the horizontal and vertical planes. It is depicted in Fig. 3.14. These servos receive control signals from PWM pins on the Summit 40-pin header. The servos require a logic level shifter just as the motor controllers do, since the servos operate on 5V logic while the Summit uses 1.8V logic. The logic level shifter is visible in the upper right corner of Fig. 3.14(a).



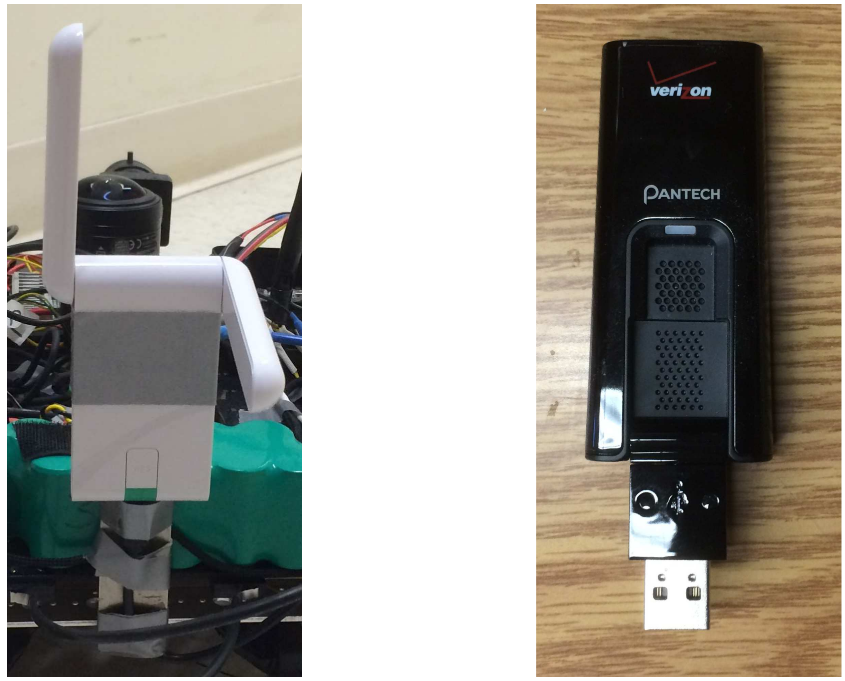
(a) The servos as viewed from the left. Note that the pan servo is visible through the large plastic gear in the center of the image.

(b) The tilt servo viewed from the right

Fig. 3.14.: Images of pan and tilt servo mechanism

3.2.4 Communication Subsystem

VADER rover's primary method of communication with the world is via WiFi. Even though the Gumstix® Overo® COM has a built-in 802.11b/g WiFi adapter, I found the need to use an external USB WiFi adapter instead. In testing the built-in WiFi, I found its connectivity to be intermittent. It also had bandwidth issues, seldom achieving a transfer rate higher than 802.11b (11 Mbps) speeds when it was connected. Therefore I have the TP-LINK TL-WN822N 802.11n USB WiFi adapter [75]. It is mounted on a vertical bracket at the rear of the rover, shown in Fig. 3.15(a). This adapter advertises speeds up to 300 Mbps. In use with Purdue's PAL3.0 WiFi I routinely achieve around 150 Mbps, which is sufficient for my purposes.



(a) TP-LINK TL-WN822N USB WiFi adapter mounted on the **VADER** rover

(b) Verizon UML295 LTE modem

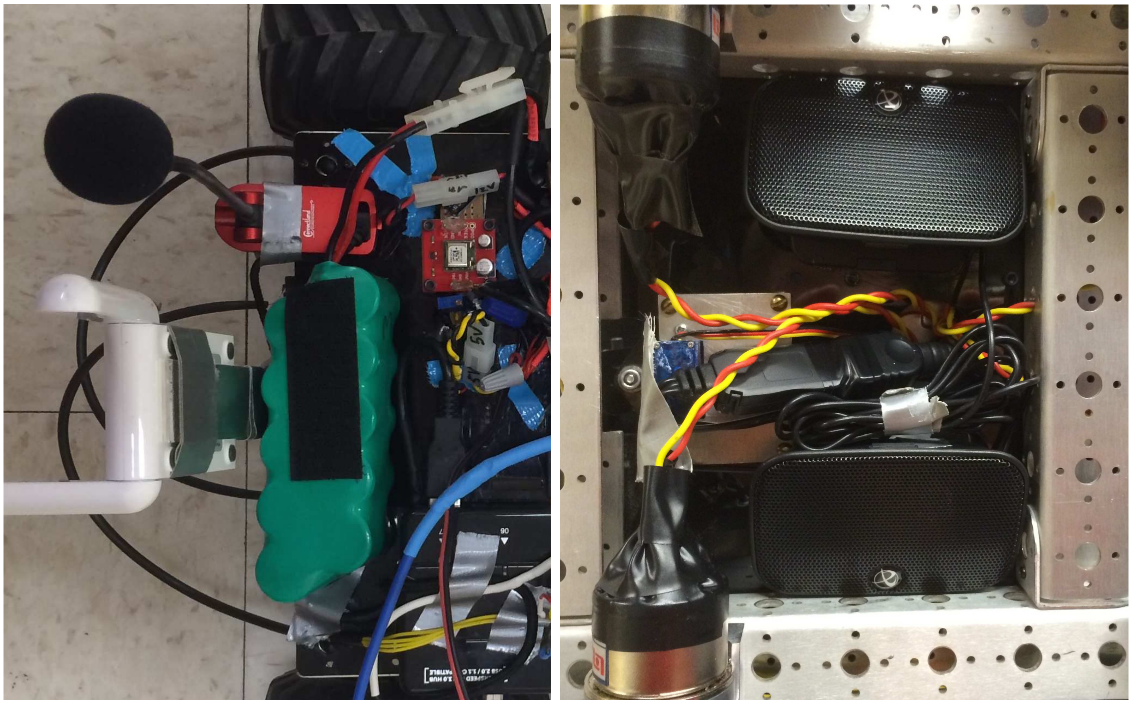
Fig. 3.15.: Communication Subsystem components

As a backup method of communication, and experiments that could be conducted outdoors away from WiFi, I have a Verizon UML295 4G LTE USB Modem [76], pictured in

Fig. 3.15(b). I have tested the modem and found its bandwidth to be sufficient, although I have not performed experiments or collected data while using it. This item is shown in Fig. 3.3 with a dashed line to the WiFi connection because it is intended to be used in lieu of the WiFi adapter if the situation warrants; I do not see a use case where I would use both the WiFi adapter and the LTE modem simultaneously.

3.2.5 Audio and Video Subsystem

The audio portion of this subsystem consists of a simple boom microphone, attached at the left rear of the rover, and a pair of 5V powered speakers, attached with epoxy to the underside of the rover between the front and middle wheels. They are connected to the Gumstix® through the line in and speaker out audio jacks on the Summit expansion board. They are pictured in Fig. 3.16. These systems represent a capability that I built into the system but did not use in any of my experiments.



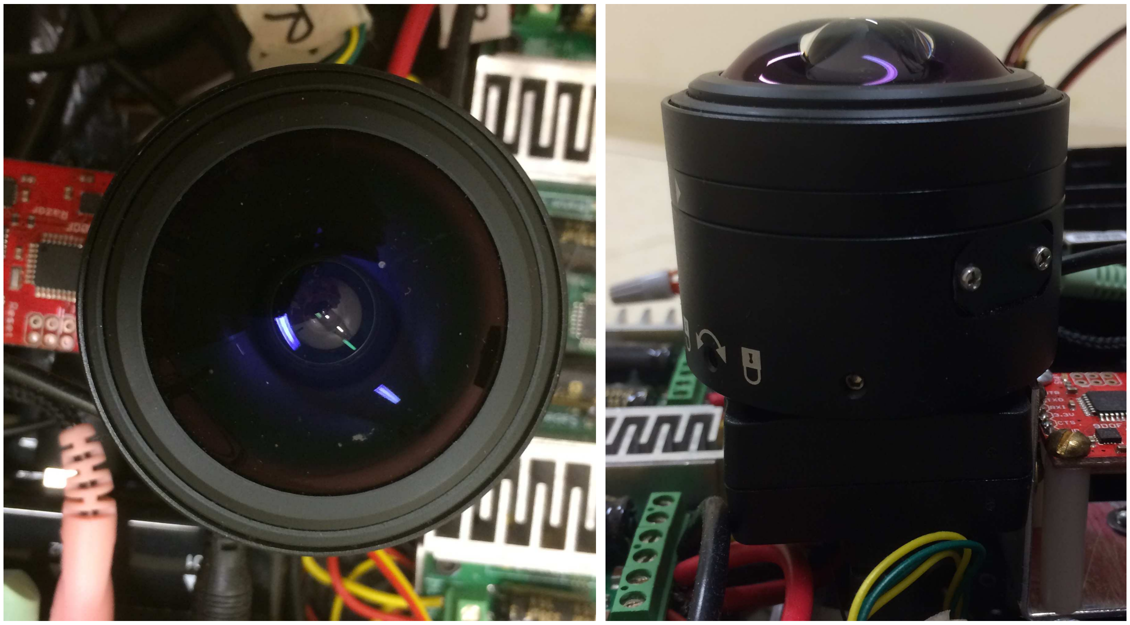
(a) Microphone

(b) Speakers

Fig. 3.16.: Audio components

There are two cameras which comprise the video portion of this subsystem. Both are Chameleon USB 2.0 cameras from Point Grey Research [77]. They differ in how and where they are mounted, as well as the lenses used.

The fixed camera on the rover is mounted near the center of the rover, pointing upwards. It is equipped with an ImmerVision IMV1-1/3 panomorphic lens [78]. This camera will thus be referred to as the panomorphic (or pano) camera, and it is pictured in Fig. 3.17. The lens on this camera enables a 360° field of view through unique optics that create an elliptical image. A sample image from this camera is pictured in Fig. 3.18.



(a) Overhead view

(b) Side view

Fig. 3.17.: Panomorphic camera

The movable camera is mounted on the pan and tilt servo mechanism on the front of the rover. It is oriented horizontally and points forward—with servos centered—and will thus be referred to as the front camera. The lens on this camera is a lightweight Fujinon lens with a manually adjustable zoom. A sample image from this camera is pictured in Fig. 3.20.

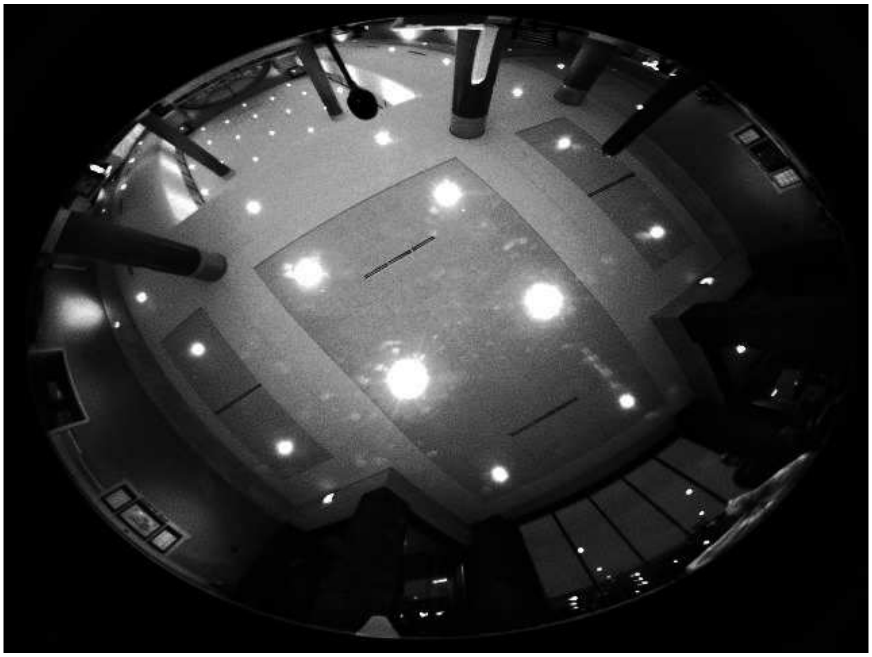
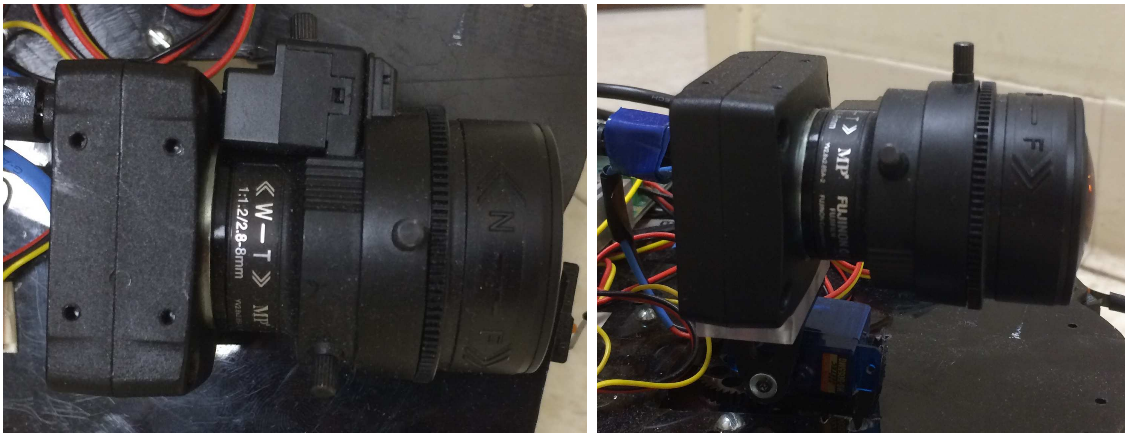


Fig. 3.18.: A sample image from the panomorphic camera



(a) Overhead view

(b) Side view

Fig. 3.19.: Front camera

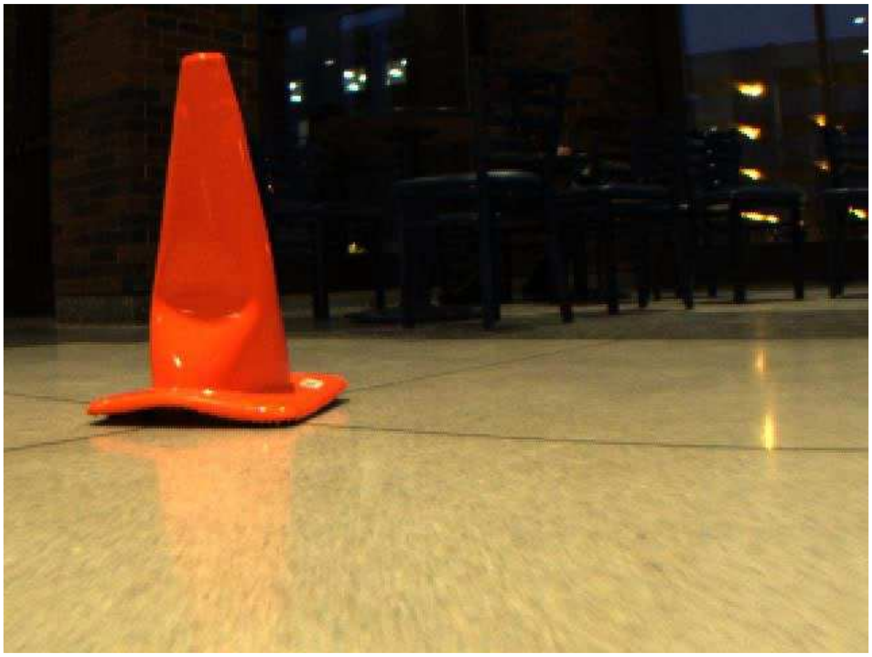


Fig. 3.20.: A sample image from the front camera

3.2.6 Sensor Subsystem

The simplest of the sensors on the **VADER** rover are the bump switches located at the front and rear of the robot. The purpose of these is to protect the robot from damage should it run into an object. A closeup of one of the front bump switches, viewed from underneath the robot, is shown in Fig. 3.21. These switches—two in front and two in back—are connected to "whiskers" made from semi-rigid vacuum tubing. When the rover makes contact with an object via a "whisker," the bump switch is closed. Since each pair of bump switches is wired through a pull-up resistor to a GPIO pin on the Summit expansion board, closing the switch sends a signal to the Gumstix® via GPIO. This signal is then handled in software; the rover automatically stops, briefly reverses direction to back off from the object, then stops again.

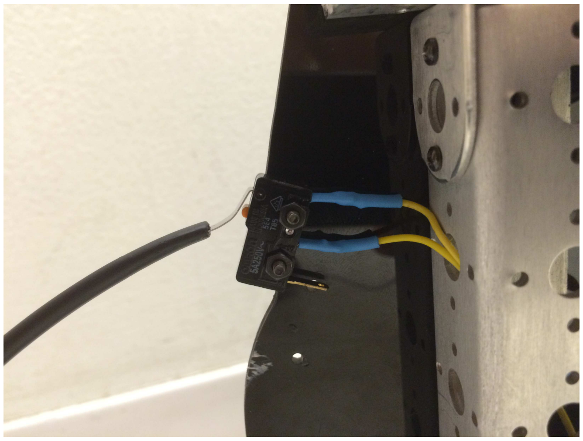
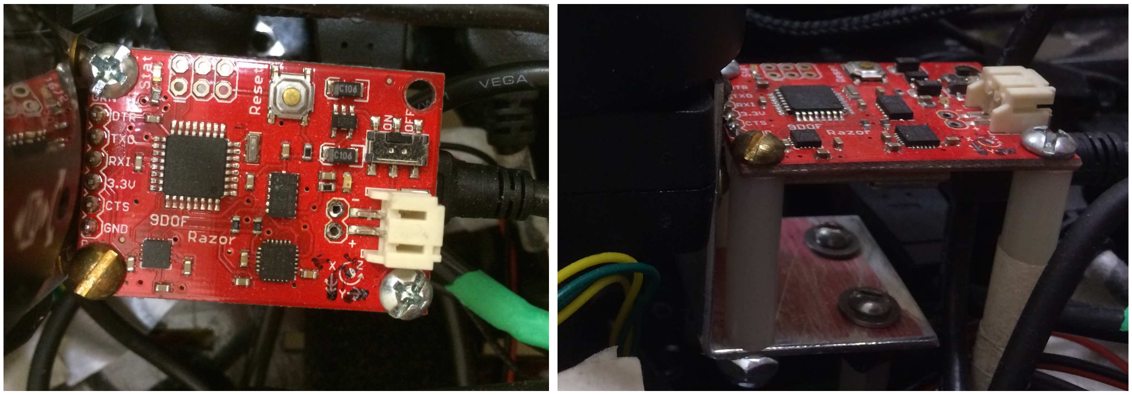


Fig. 3.21.: Bump switch

The GPS receiver on the rover, mounted near the front left corner, is pictured in Fig. 3.22. It is a GlobalSat BU-353 USB GPS Receiver [79]. It outputs several different types of standard NMEA sentences [80] as ASCII text via USB. There is software on the Gumstix® that listens for these messages and then parses and logs the relevant ones. I have tested the GPS and have obtained reasonable position and velocity readings. However, since all experiments to date have been conducted indoors, the GPS has not played a significant role in these experiments. This capability, like the audio capability, was built with future expansion in mind.

Fig. 3.22.: GPS receiver

The inertial measurement unit (IMU) on the rover is a Razor 9 Degrees of Freedom (DOF) IMU from SparkFun [81]. It is mounted near the center of the rover, just behind the camera. The IMU is shown in Fig. 3.23. It contains an accelerometer, a gyroscope, and a magnetometer, each with three axes. It also has at ATmega328 microcontroller programmed with the Arduino bootloader and test firmware. For my purposes, I adapted the open-source Attitude and Heading Reference System (AHRS) code [82] for my purposes. The original AHRS system was designed for aerial vehicles and is thus much more complex than I need. Furthermore, I found in testing that having the IMU within a few inches of six motors—a situation I could not avoid—rendered the magnetometer useless. What I did find useful was the gyroscope, specifically the z (vertical) axis. I was able to get readings from it that allowed me to reliably measure turn angles. Therefore I use my own customized firmware to read the gyro at a rate of 50 Hz and output the data to the Gumstix® via USB. With that functionality, I have half of the information I need to have the rover self-localize. The remaining half, distance traveled, comes from the wheel encoders via the Teensy board.



(a) Overhead view

(b) Side view

Fig. 3.23.: Inertial measurement unit

The final piece of data that I need to have the rover intrinsically monitor its own position is the distance traveled. The raw signals used to produce this data are generated by the wheel encoders, shown in Fig. 3.13. However, these signals are just a pair of quadrature pulses from each encoder and thus must be processed in order to create usable data. My initial efforts to read and process these pulses were focused on using the available GPIO lines on the Summit expansion board. However, I found that with the Gumstix® already running threads to capture and send images from the two cameras, send logging data, and listen for and execute commands sent from the companion laptop, there simply was not enough processing power available left on the single processor system. I could not even get my encoder test code to run at 10 Hz, much less than the 50 Hz—to match the data rate of the IMU—that was my goal. Therefore, I had to find a small microcontroller board to which I could offload this task.

I searched and found the Teensy 3.1 USB development board [83], shown in Fig. 3.24. It is an Arduino-like board with an ARM Cortex-M4 processor at 72 MHz and a total of 34 IO pins. The only pins I use are four of the interrupt-enabled digital IO pins, two for each encoder. The Teensy runs code written in the Arduino sketch language and is easily programmed through Teensyduino [84], an add-on to the Arduino IDE. The Teensy also has a fairly extensive list available libraries to help users get started on projects. I use the

Encoder Library [85] as the foundation on which I build my odometry code that runs on my Teensy. The library allows me to poll pairs of pins that are set up as encoders and receive the number of pulses—signed to indicate clockwise or counterclockwise rotation—since the last reset of the counter. Therefore my odometry code runs in a loop that monitors the millisecond clock on the Teensy. If 20 ms has elapsed since the last reading, it reads the encoders, resets them to zero, and sends that data over USB to the Gumstix®. This way I am able to keep the data from the encoders flowing to the Gumstix® at the same rate (50 Hz) as the data coming from the IMU.

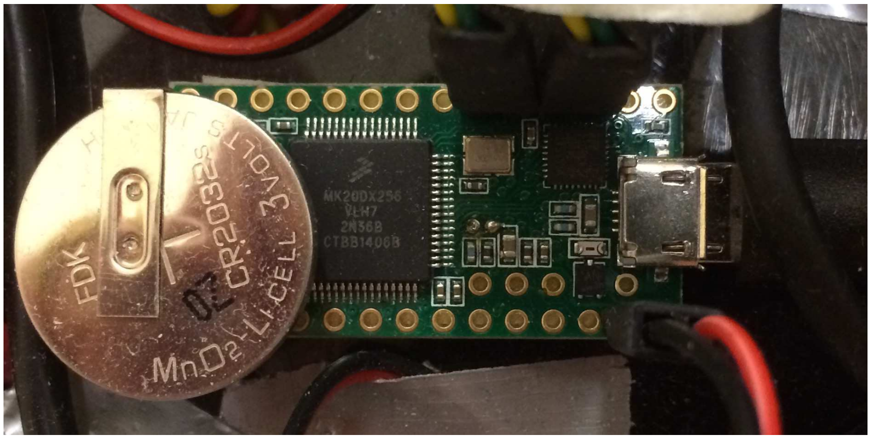


Fig. 3.24.: Teensy 3.1 USB development board. For a size comparison, the CR2032 memory battery affixed to the top of the Teensy board is approximately the same diameter as a nickel.

3.3 Software

While I briefly touched on some software issues on the rover in the previous section, in this section I discuss the software in more depth. I separate the discussion logically by discussing software on the rover and software on the companion laptop separately. I also briefly return to hardware in Section 3.3.2 in the description of the gamepad controller used for manual driving.

3.3.1 Software on the VADER rover

I maintain a publicly accessible GitHub repository of my code that runs on the rover at [86]. In fact, this repository contains the entire main partition of the Linux system that runs on **VADER** rover. It was my intention in creating this repository that it serve as a daily backup and a source for a full microSD card rebuild should I encounter a failure. I have tested this and was able to successfully create a running microSD card from the repository. However, since this is my working repository and not production-quality code, I presume that it will not be useful to anyone but me.

Startup Scripts

The first thing that **VADER** rover needs to do when it starts up is create a reliable communication channel back to the companion computer from which it will receive commands. Since the rover relies on WiFi that is behind a NAT for communications, I do not have a reliable way to establish an unchanging hostname for the rover. Therefore, I have a script at startup that essentially "phones home" to create a channel. I do this using a reverse SSH tunnel back to the companion laptop. During initial testing I was using a workstation with a fixed hostname in the `ecn.purdue.edu` domain, which made the tunnel easy. Unfortunately, when I moved the control system to a laptop, this laptop was also using WiFi behind the same NAT. Therefore, each time the laptop boots and gets a new local IP address, I have to change the text of the script—as well as some code within the low-level C functions on the rover—to match this IP. Although cumbersome, this setup has proven to be a workable solution. Additionally, there is another script on the rover that synchronizes its system clock with a time server, to ensure that I do not have any communication problems stemming from incorrect time.

Low-level Functions

There are two main libraries that run the low-level functions on the **VADER** rover, TOOLLIB-CAMERA and RUN-SENSORS. I describe each in turn.

TOOLLIB-CAMERA is the first set of libraries I wrote to run on the rover. In addition to running the cameras, this library also handles establishing sockets to send data back to the companion laptop. The communication tasks are based on open-source libraries. They establish sockets for image data from each of the two cameras, sockets to send timestamps (from the rover's internal clock) for each frame captured, sockets to send back command execution and sensor logs, as well as a socket to listen for commands from the companion laptop. Without these functions, the rover would have little to no functionality.

The communications functions within TOOLLIB-CAMERA handle different types of data differently. The rover sends the actual image data from the camera back to the companion laptop immediately after capture, since it is a large amount of data every time. However, the data from logging of executed commands, sensor data, and image timestamps is much smaller in size. Therefore, those communications functions create buffers for this data and only send when the buffer reaches capacity—as well as just before the program exits—in order to not waste the overhead of sending an entire packet for only a few bytes of payload data.

The final function that TOOLLIB-CAMERA handles, and from where it derives its name, is the actual operation of the Point Grey cameras. It is here that I interface with Point Grey's licensed FlyCapture2 API [87]. This provides a relatively flexible interface for using the Point Grey Chameleon cameras, although I did run into some obstacles that I had to work around. The Chameleon camera has a native resolution 1280x960. However, I want to limit my images to 640x480 to both conserve wireless bandwidth and limit the sheer number of pixels I will have to process when analyzing the videos. The FlyCapture API provides two methods to do this: sub-region selection or pixel averaging. Sub-region is more flexible and allows a user to choose any contiguous segment of the sensor as the region of interest. Pixel averaging brings the entire sensor's resolution down to 640x480 with a single command, but it has the disadvantage that the averaging process causes the loss of color data—pixel averaged images are only available as gray scale. Unfortunately, because of the panomorphic lens I use with the pano camera, a pixel averaged gray scale image is my only choice to get a 640x480 image that covers the entire sensor. This is why all pano camera images are in gray scale. The front camera images, on the other hand, are sub-region selected to only capture the central 640x480 region of the sensor. Within TOOLLIB-CAMERA I also use the OpenCV libraries [88] to perform compression on my image data prior to sending it across the network. I experimented with the Imlib2 libraries [89] as well, but found that OpenCV gave better performance.

RUN-SENSORS is a library that does just what its name implies: it runs the sensors on the rover. This library sets up serial communication with the Teensy, IMU, and GPS. It also receives messages from these sensors at the appropriate rates—50 Hz for the Teensy and IMU and approximately 1 Hz for each of the two types of GPS messages that I log. The functions then pass the messages to the main control programs (either EMPEROR or THE-FORCE below) for buffering and sending back to the companion laptop for logging. Finally, RUN-SENSORS also monitors the GPIO pins to which the bump switches are attached. When it detects and interrupt on one of those pins, it sends a message to the main control program to execute the stop–reverse–stop safety protocol described in Section 3.2.6.

Emperor

In keeping with my Star Wars theme, I named the program that establishes manual control of the rover EMPEROR, since Emperor Palpatine is always the one pulling Darth Vader's strings. This program is the main control program for manual control of the rover. It listens for and executes the commands sent from the user via the gamepad (Fig. 3.25) through the companion laptop. It is also responsible for calling all of the low-level functions in TOOLLIB-CAMERA and RUN-SENSORS that set up the communication channels and run the cameras, sensors, and data logging.

The Force

I named the program that does automatic driving based on self-localization is called THE-FORCE. This program is similar to EMPEROR in that it handles communications, logging and running the sensors and cameras. However, it differs from EMPEROR in two significant ways. First, the commands it receives from the companion laptop are not motor or servo commands to directly move the rover. Instead, it receives from the laptop a series of (x, y) points to which the rover must navigate autonomously. For the rover to do so requires the second major difference in software. THE-FORCE, rather than just sending back IMU and encoder data for logging, fuses this data to generate a position estimate every 20 milliseconds through the use of an Extended Kalman Filter [90, 91]. In testing, I found that the mean squared error difference between the estimated position and the ground truth position was on the order of 1 centimeter.

3.3.2 Software on the Companion Laptop

I maintain a publicly accessible GitHub repository of my code that runs on the rover's companion laptop at [92]. However, since this is my working repository and not production-quality code, I presume that it will not be useful to anyone but me.

The device I use to driver the rover manually is a Logitech F710 Bluetooth gamepad, shown in Fig. 3.25. To interface this device with the laptop, I use the open-source library found at [93]. This library works very well and gives me the ability to read device events from the gamepad from within C/C++.

The software that runs on the companion laptop is a mixture of low-level communication, file and device IO, and image processing functions written using open-source libraries [88, 89, 93] in C/C++, as well as higher-level wrapper functions written in Scheme in order to take advantage of my research group's extensive use of its own Scheme variant, QobiScheme [94]. The low-level C/C++ functions handle establishing communication sockets, sending commands (both gamepad movements and (x, y) points), receiving log data and writing such to files, receiving image data and constructing videos from still images, and sending images to a viewer application. I then wrap these low-level functions within Scheme functions in order to take advantage of the software infrastructure already established in my research group. This allows me to build my main application, VIEWER, atop the DEFINE-APPLICATION procedure defined within QobiScheme. This gives me a well-defined way to interface with the **VADER** rover. I can even choose whether to run EMPEROR or THE-FORCE from the VIEWER GUI. When running THE-FORCE, VIEWER can either send a series of waypoints directly to the rover, or it can run helper procedures that take sentences as input and generate a path (as a series of waypoints) for the rover to follow, as detailed in Section 4.5.3. An image of the VIEWER GUI running THE-FORCE during data collection for the work in Chapter 4 is shown in Fig. 3.26. Note that if EMPEROR had been running instead of THE-FORCE, the GUI would only have shown the two images from the cameras and not the floor plan and sentence below them.

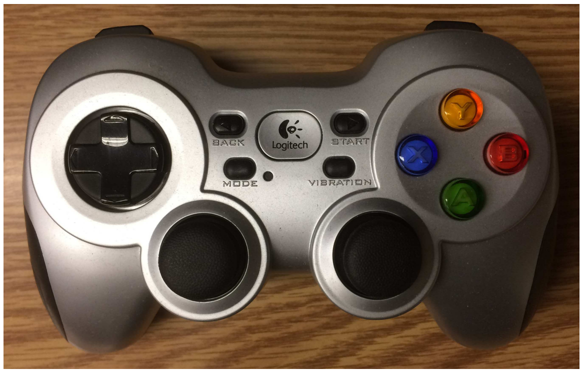
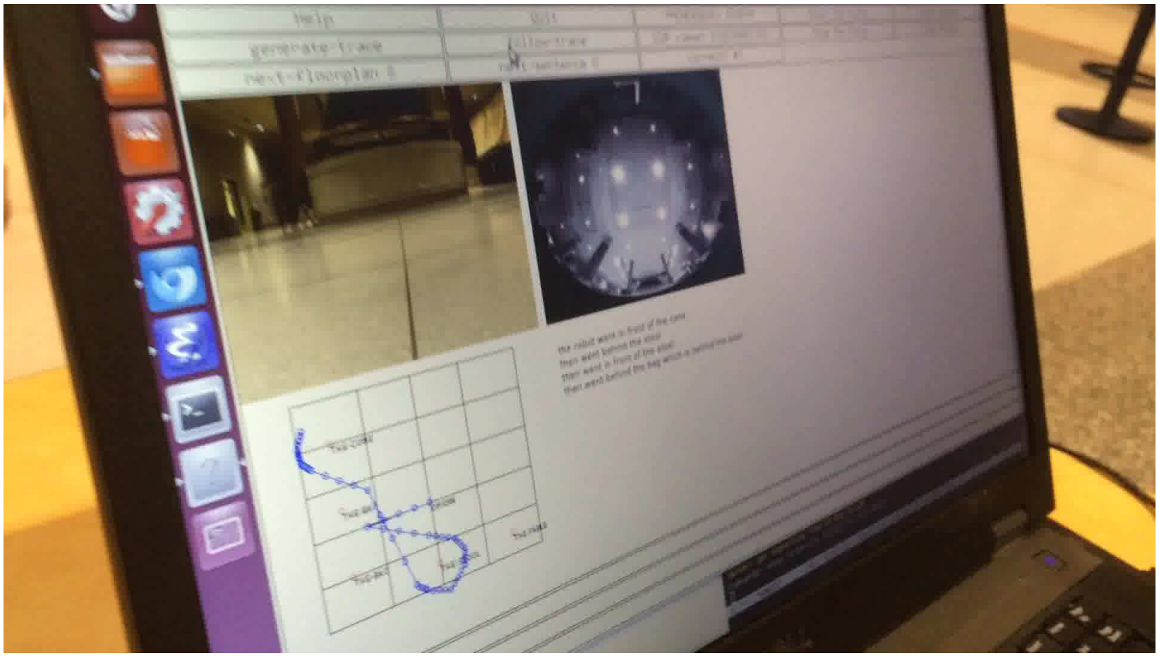


Fig. 3.25.: Logitech F710 Bluetooth gamepad



the robot went in front of the cone
then went behind the stool
then went in front of the stool
then went behind the bag which is behind the stool

Fig. 3.26.: A video screen capture showing the VIEWER interface while running THE-FORCE on the rover

4. ROBOT LANGUAGE LEARNING, GENERATION, AND COMPREHENSION

We present a unified framework which supports grounding natural-language semantics in robotic driving. This framework supports acquisition (learning grounded meanings of nouns and prepositions from human annotation of robotic driving paths), generation (using such acquired meanings to generate sentential description of new robotic driving paths), and comprehension (using such acquired meanings to support automated driving to accomplish navigational goals specified in natural language). We evaluate the performance of these three tasks by having independent human judges rate the semantic fidelity of the sentences associated with paths, achieving overall average correctness of 94.6% and overall average completeness of 85.6%.

4.1 Introduction

With recent advances in machine perception and robotic automation, it becomes increasingly relevant and important to allow machines to interact with humans in natural language in a *grounded fashion*, where the language refers to actual things and activities in the world. Here, we present our efforts to automatically drive—and learn to drive—a mobile robot under natural-language command. Our contribution is summarized in Fig. 4.1 through Fig 4.4. A human teleoperator is given a set of sentential instructions designating robot paths. The operator then drives a mobile robot under radio control according to these instructions through a variety of floorplans. The robot uses onboard odometry and inertial guidance sensors to determine its location in real time and saves traces of the driving

This work was accomplished in collaboration with Daniel Barrett and Haonan Yu. With only minor changes for formatting and length, it was submitted to arXiv on 25 August 2015. `http://arxiv.org/abs/1508.06161`

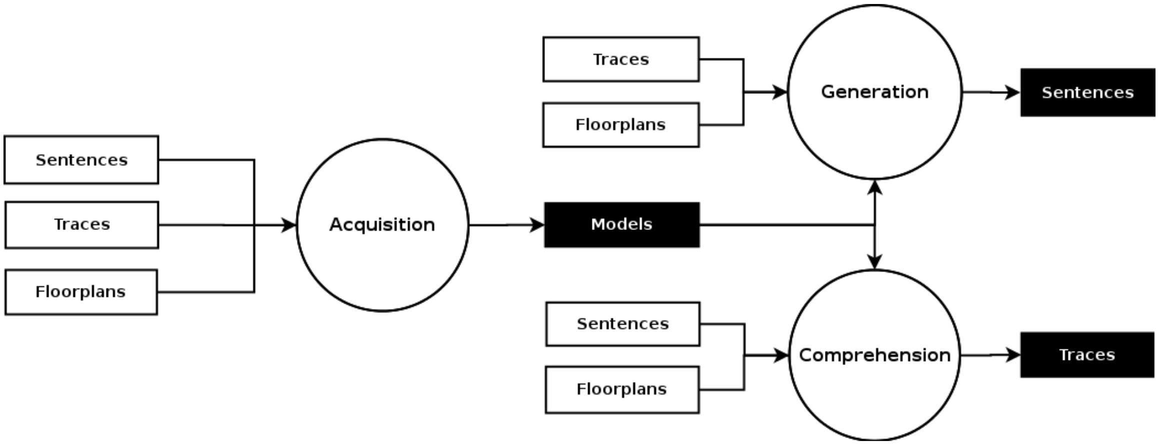


Fig. 4.1.: Data flow diagram overview of the system. Boxes with black text on white background represent data generated or collected by humans, while boxes with white text on black background represent data that is machine generated. The Acquisition subsystem (Fig. 4.2) takes sentential descriptions of paths, the traces of such paths as driven by a human teleoperator, and floorplan specifications as input. It produces learned models of the nouns and prepositions in such sentences as output. The Generation subsystem (Fig. 4.3) takes these learned models, along with traces of new manually driven paths and new floorplans, and produces sentences that describe the driven paths. The Comprehension subsystem (Fig. 4.4) also uses the learned noun and preposition models with new sentences and new floorplans to produce traces that meet the navigational goals specified in the sentences. Our custom mobile robot (Fig 4.5) can autonomously drive such traces.

path to log files. From a training corpus of paths paired with sentential descriptions and floorplan specifications, our system automatically learns the meanings of nouns that refer to objects in the floorplan and prepositions that describe both the spatial relations between floorplan objects and between such objects and the robot path. With such learned meanings, the robot can then generate sentential descriptions of new driving activity undertaken by the teleoperator. Moreover, instead of manually controlling the robot through teleoperation, one can issue the robot natural-language commands which can induce fully automatic driving to satisfy the path specified in the natural-language command.

We have conducted experiments with an actual radio-controlled robot that demonstrate all three of these modes of operation: acquisition, generation, and comprehension. We demonstrate successful completion of all three of these tasks on hundreds of driving ex-

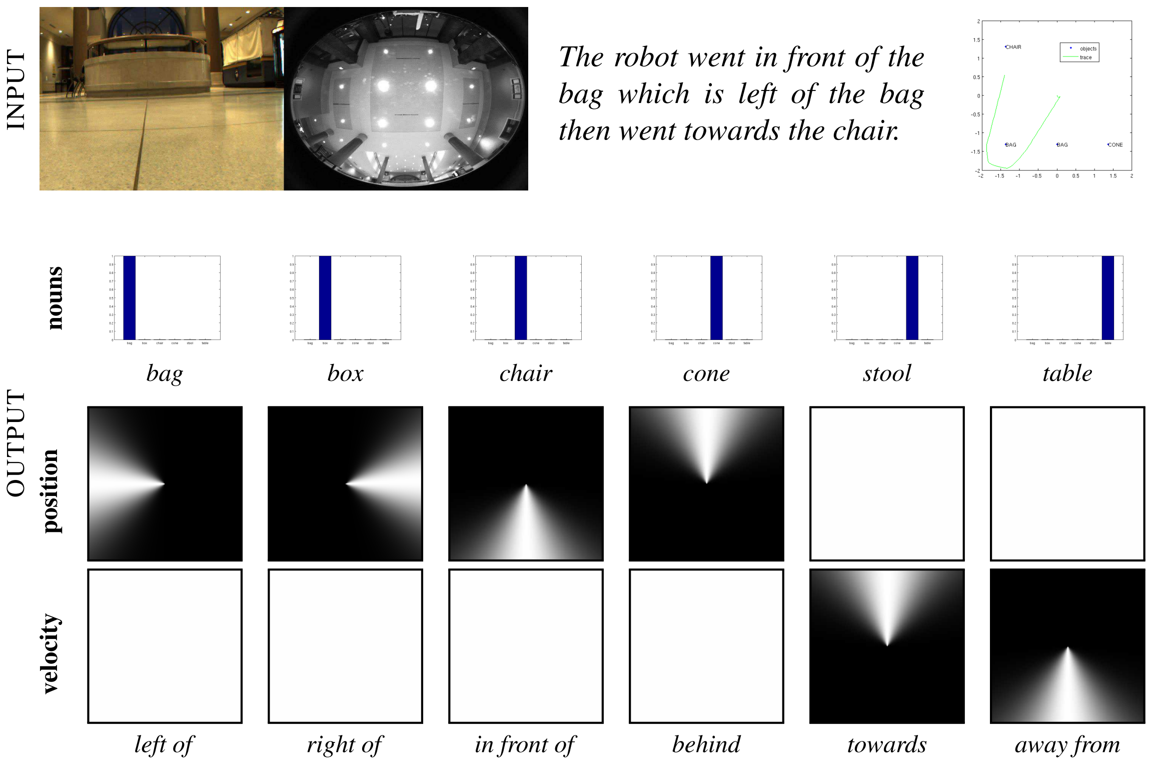
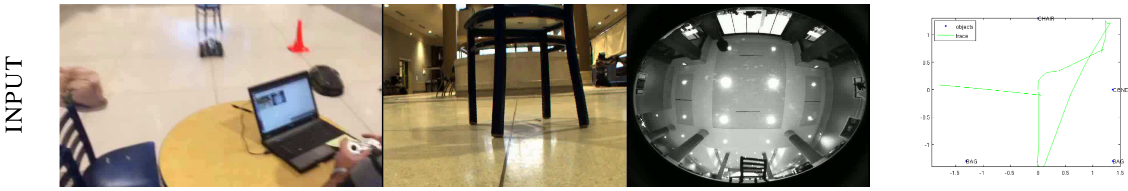


Fig. 4.2.: Overview of the Acquisition subsystem. A human drives the mobile robot through paths according to sentential instructions while odometry reconstructs the robot's paths. This allows the robot to learn the meanings of the nouns and prepositions. Hand-designed models are shown here for reference; actual learned models are shown in Fig. 4.11. Note that the distributions are uniform in velocity angle (bottom row) for *left of*, *right of*, *in front of*, and *behind* and in position angle (top row) for *towards* and *away from*.



OUTPUT

The robot went in front of the chair then went away from the chair and behind the cone then went right of the bag which is left of the cone then went left of the bag which is in front of the cone then went away from the cone and away from the chair.

Fig. 4.3.: Overview of the Generation subsystem. Noun and preposition models learned in the Acquisition subsystem support the generation of English descriptions of new paths driven by teleoperation.

INPUT

The robot went behind the bag which is in front of the bag then went in front of the bag which is left of the chair then went towards the cone then went away from the chair then went right of the chair then went right of the bag which is left of the cone.

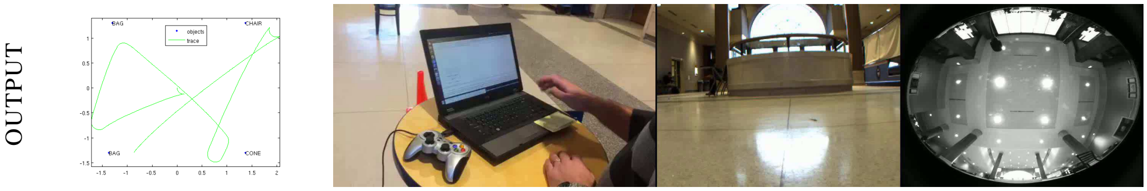


Fig. 4.4.: Overview of the Comprehension subsystem. Noun and preposition models learned in the Acquisition subsystem support autonomous driving of paths that meet navigational goals specified in English descriptions.

amples. We evaluate the fidelity of the sentential descriptions produced automatically in response to manual driving and the fidelity of the driving paths induced automatically to fulfill natural-language commands, by presenting the pairs of sentences together with the associated paths to human judges. Overall, the average "correctness" (the degree to which the description is true of the path) reported is 94.6% and the average "completeness" (the degree to which the description fully covers the path) reported is 85.6%.

4.2 Related Work

We know of no other work which presents a physical robot which learns word meanings from physical robot paths paired with sentences, uses these learned meanings to generate sentential descriptions of manually driven paths, and automatically plans and physically drives paths to satisfy input sentential descriptions.

While there is other work which claims to learn the meanings of words from robot paths or follow natural instructions, upon further inspection these systems operate only within discrete simulation, as they utilize the internal representation of the simulation to obtain discrete symbolic primitives [7, 8, 10, 11, 14, 19]. Their space of possible robot actions, positions and states are very small and are represented in terms of symbolic primitives like TURN LEFT, TURN RIGHT, and MOVE FORWARD N STEPS [11], or DRIVE TO LOCATION 1 and PICK UP PALLET 1 [14]. Thus, they take a sequence of primitives like {DRIVE TO LOCATION 1; PICK UP PALLET 1} and a sentence like *go to the pallet and pick it up* and learn that the word *pallet* maps to the primitive PALLET, that the phrase *pick up* maps to the primitive PICK UP, and that the phrase *go to X* means DRIVE TO LOCATION X.

In contrast, our robot and environment, being in the continuous physical world, can take an uncountably infinite number of configurations. We take a set of sentences matched with paths of the robot as input, where the paths are densely sampled points in the real 2D Cartesian plane. Not all points in the path correspond to words in the sentences, multiple (often undescribed) relationships can be true of any point, and the correspondence between described relationships and path points is unknown. This is a vastly more difficult problem.

Furthermore, previous work does not even solve the simplified problem without additional annotation. Kollar *et al.* [8] requires hand-drawn positive and negative paths depicting specific word meanings. Tellex *et al.* [10] requires manual annotation of the groundings of all words in the training sentences to specific objects and relationships in the training data. [14] does not require annotation of the grounding of each word, but does require manual temporal segmentation and alignment of paths and the pieces of multi-part sentences, whereas our method can learn without any such annotation.

Dobnik *et al.* [16] has an actual robot but only learns to classify simple phrases like *A is near B* from robot paths paired with such phrases that have hand-grounded nouns. They can neither generate sentences describing driven paths, nor automatically drive a path described by a sentence. Our system can do both of these, as well as learn meanings for both nouns and prepositions.

4.3 Our Mobile Robot

All experiments were performed on a custom mobile robot, shown in Fig. 4.5. This robot can be driven by a human teleoperator or drive itself automatically to accomplish specified navigational goals. During all operation, robot localization is performed onboard the robot in real-time via an Extended Kalman Filter [90, 91] with odometry from shaft encoders on the wheels and inertial-guidance from an IMU.

Due to sensor noise and mechanical factors such as wheel sliding, this localization is noisy, but generally within 20cm of the actual location. The video feed, localization, and all sensor and actuator data is logged in a time-stamped format. When conducting experiments on generation and acquisition, a human teleoperator drives the robot along a variety of paths in a variety of floorplans. The path recovered from localization supports generation and acquisition. When conducting experiments on comprehension, the path is first planned automatically, then the robot automatically follows its planned path by comparing the new odometry gathered in real time with the planned path and controlling the wheels accordingly.

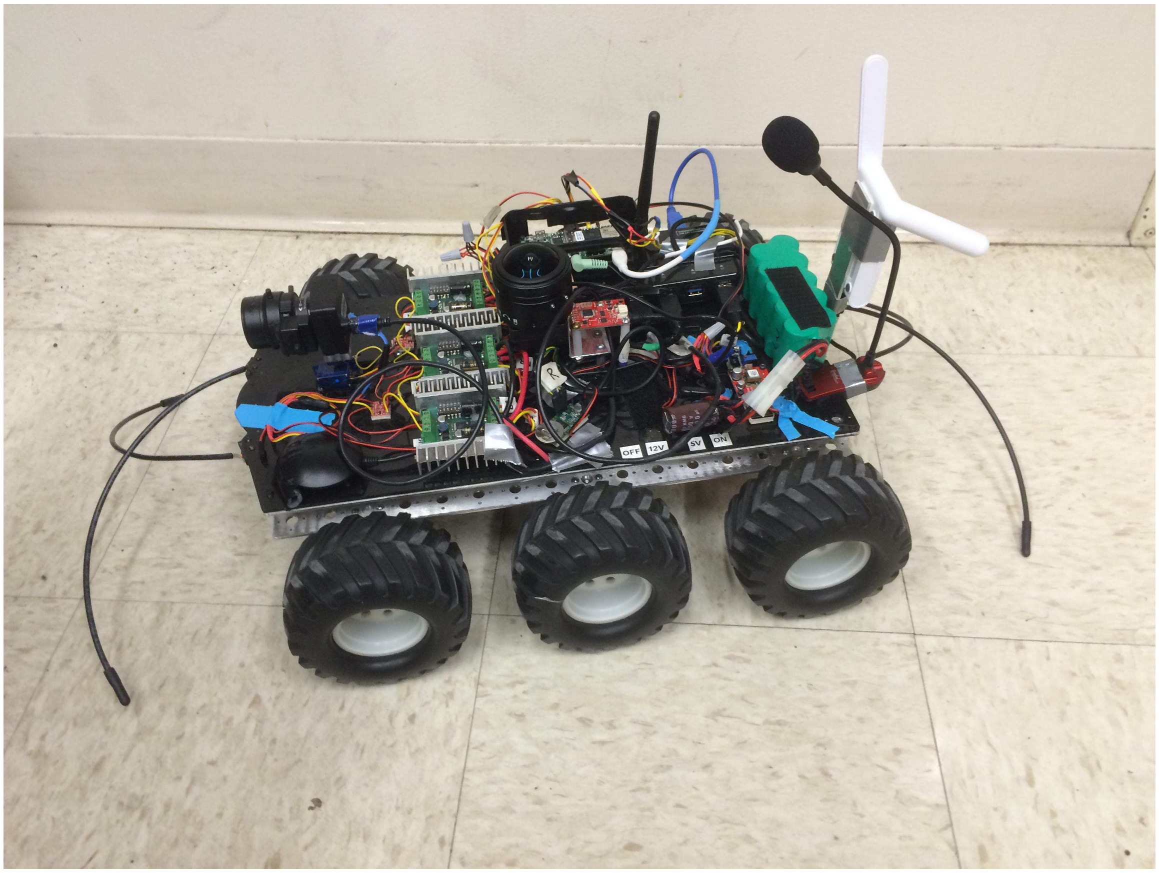


Fig. 4.5.: Our custom mobile robot.

The use of an actual robot with noisy real-world sensor data increases the difficulty of the tasks when compared to work which occurs in simulation. The noisy robot position is densely sampled in the continuous domain. For acquisition and generation, this adds an additional layer of uncertainty, as the correspondence between individual points in the robot path and the phrases of a sentence is unknown.

4.4 Technical Details

4.4.1 Grammar and Logical Form

We employ the grammar shown in Fig. 4.6, which, while small, supports an infinite set of possible utterances, unlike the grammars used in [20] and [18]. Nothing turns on this however. In principle, one could replace this grammar with any other mechanism for generating logical form. This paper concerns itself with semantics, not syntax, and only addresses issues relating to the grounding of logical form. This particular grammar is simply a convenient surface representation of our logical form.

Note that our surface syntax allows two uses of prepositions (and the associated prepositional phrases): as modifiers to nouns in noun phrases, indicated with a subscript ‘SR’ (*i.e.,* spatial relation), and as adjuncts to verbs in verb phrases, indicated with a subscript ‘path.’ Many prepositions can be used in both SR and path form. They share the same semantic representation and both uses are learned from the pooled data of both kinds of occurrences in the training corpus. Furthermore, note that the grammar supports infinite

S	→	*The robot* VP
VP	→	*went* PP_{path} [*then* VP]
PP_{path}	→	P_{path} NP [*and* PP_{path}]
NP	→	*the* N [PP_{SR}]
PP_{SR}	→	*which is* P_{SR} NP [*and* PP_{SR}]
P_{path}	→	*left of* | *right of* | *in front of* | *behind* | *towards* | *away from*
P_{SR}	→	*left of* | *right of* | *in front of* | *behind*
N	→	*bag* | *box* | *chair* | *cone* | *stool* | *table*

Fig. 4.6.: The grammar used by our implementation.

NP recursion: noun phrases can contain prepositional phrases that, in turn, contain noun phrases. Finally, note that the grammar supports conjunctions of prepositional phrases in both SR and path form.

We employ the logical form shown in Fig. 4.7. Informally, formulas in logical form denote paths through a floorplan. Both paths and floorplans are specified as collections of waypoints. A *waypoint* is a 2D Cartesian coordinate optionally labeled with the class of the object that resides at that coordinate, *e.g.,* $(3, 47, \textbf{bag})$ The waypoint is unlabeled, *e.g.,* $(3, 47)$, if no object resides at that coordinate. A *floorplan* is a set of labeled waypoints, while a *path* is a sequence of unlabeled waypoints (Fig. 4.8 right). A formula in logical form contains three parts: a *path quantifier*, a *floorplan quantifier*, and a *condition* that the path through the floorplan must satisfy. The condition is a conjunction of atomic formulas, predicates applied to variables bound by the path or floorplan quantifiers. The formula must be closed, *i.e.,* every variable in the condition must appear either in the path quantifier or the floorplan quantifier. The model of a formula is a set of bindings for each of the quantified path variables to unlabeled waypoints, and floorplan variables to labeled waypoints.

The one-argument atomic formulas constrain the class of waypoints to which the variables that appear as their arguments are bound. The two-argument atomic formulas constrain the spatial relations between pairs of waypoints to which the variables that appear as their arguments are bound. The logical form in Fig. 4.7 contains a particular set of six one-argument predicate and six two-argument predicates. Nothing turns on this however. This is simply the set of predicates that we use in the experiments reported. The framework clearly extends to any number of predicates of any arity, particularly since we learn the meanings of the predicates.

$$\begin{array}{lcl}
\langle \textit{formula} \rangle & \rightarrow & \langle \textit{path quantifier} \rangle \langle \textit{floorplan quantifier} \rangle \\
 & & \langle \textit{atomic formula} \rangle (\wedge \langle \textit{atomic formula} \rangle)^{*} \\
\langle \textit{path quantifier} \rangle & \rightarrow & [\langle \textit{var} \rangle (; \langle \textit{var} \rangle)^{*}] \\
\langle \textit{floorplan quantifier} \rangle & \rightarrow & \{\langle \textit{var} \rangle (, \langle \textit{var} \rangle)^{*}\} \\
\langle \textit{atomic formula} \rangle & \rightarrow & \langle \textit{atomic formula}_1 \rangle \\
 & | & \langle \textit{atomic formula}_2 \rangle \\
\langle \textit{atomic formula}_1 \rangle & \rightarrow & \textsc{bag}(\langle \textit{var} \rangle) \\
 & | & \textsc{box}(\langle \textit{var} \rangle) \\
 & | & \textsc{chair}(\langle \textit{var} \rangle) \\
 & | & \textsc{cone}(\langle \textit{var} \rangle) \\
 & | & \textsc{stool}(\langle \textit{var} \rangle) \\
 & | & \textsc{table}(\langle \textit{var} \rangle) \\
\langle \textit{atomic formula}_2 \rangle & \rightarrow & \textsc{leftOf}(\langle \textit{var} \rangle, \langle \textit{var} \rangle) \\
 & | & \textsc{rightOf}(\langle \textit{var} \rangle, \langle \textit{var} \rangle) \\
 & | & \textsc{inFrontOf}(\langle \textit{var} \rangle, \langle \textit{var} \rangle) \\
 & | & \textsc{behind}(\langle \textit{var} \rangle, \langle \textit{var} \rangle) \\
 & | & \textsc{towards}(\langle \textit{var} \rangle, \langle \textit{var} \rangle) \\
 & | & \textsc{awayFrom}(\langle \textit{var} \rangle, \langle \textit{var} \rangle)
\end{array}$$

Fig. 4.7.: The logical form used by our implementation.

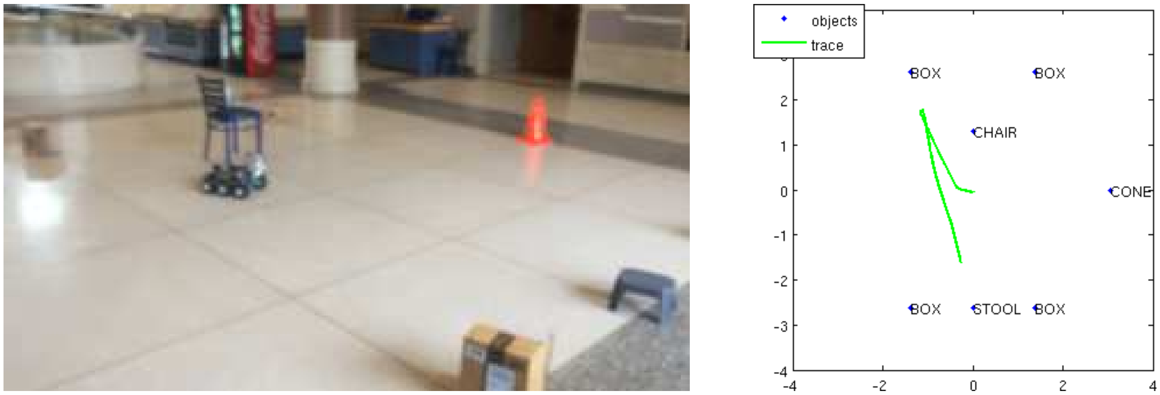


Fig. 4.8.: Sample floorplan with robot path. (left) Extrinsic image taken during operation. (right) Internal representation of floorplan consisting of labeled waypoints and localized path consisting of unlabeled waypoints.

Straightforward (semantic) parsing and surface generation techniques map bidirectionally between the surface language form as specified by the grammar in Fig. 4.6 and the logical form in Fig. 4.7. For example, a surface form like

> *The robot went towards the stool, then went behind the chair which is right of the stool, then went towards the cone, then went away from the chair which is left of the cone, then went in front of the table.*

(commas added for legibility) would correspond to the following logical form:

$$[\alpha,\beta,\gamma,\delta,\epsilon]\{t,u,v,w,x,y,z\}\left(\begin{array}{l}\textsc{towards}(\alpha,t)\wedge\textsc{stool}(t)\wedge\\ \textsc{behind}(\beta,u)\wedge\textsc{chair}(u)\wedge\textsc{rightOf}(u,v)\wedge\textsc{stool}(v)\wedge\\ \textsc{towards}(\gamma,w)\wedge\textsc{cone}(w)\wedge\\ \textsc{awayFrom}(\delta,x)\wedge\textsc{chair}(x)\wedge\textsc{leftOf}(x,y)\wedge\textsc{cone}(y)\wedge\\ \textsc{inFrontOf}(\epsilon,z)\wedge\textsc{table}(z)\end{array}\right) \tag{4.1}$$

Note that in the above, nouns all correspond to one-argument predicates while prepositions all correspond to two-argument predicates. But nothing turns on this. One could imagine lexical prepositional phrases, like *leftward*, that correspond to one-argument predicates. Moreover, path uses of prepositions specify waypoints in the path. These appear in logical form as predicates whose first argument is a variable in the path quantifier. Similarly, SR uses of prepositions specify waypoints in the floorplan. These appear in logical form as predicates whose first argument is a variable in the floorplan quantifier. Thus, in the above, the atomic formulas TOWARDS(α, t), BEHIND(β, u), TOWARDS(γ, w), AWAYFROM(δ, x), and INFRONTOF(ϵ, z) constitute path uses while the atomic formulas RIGHTOF(u, v) and LEFTOF(x, y) constitute SR uses. Note that each (path) prepositional phrase consists of a subset of the atomic formulas in the condition, as indicated above by the line breaks.

4.4.2 Representation of the Lexicon

The lexicon specifies the meanings of the one- and two-argument predicates in logical form. The meanings of one-argument predicates are discrete distributions over the set of class labels. Note that the one-argument predicates, like BAG, are distinct from the class labels, like **bag**. The mapping between such is learned. Moreover, a given floorplan might have multiple instances of objects of the same class. These would be disambiguated with complex noun phrases such as *the chair which is right of the stool* and *the chair which is left of the cone*. Such disambiguating prepositional phrase modifiers of noun phrases can be nested and conjoined arbitrarily. Similarly, waypoints can be disambiguated by conjunctions of prepositional phrase adjuncts.

Two-argument predicates specify relations between target objects and reference objects. In SR uses, the reference object is the object of the preposition while the target object is the head noun. For example, in *the chair to the left of the table*, *chair* is the target object and *table* is the reference object. In path uses, the target object is a waypoint in the robot path while the reference object is the object of the preposition. For example, in *went towards the table*, *table* is the reference object. The lexical entry for each two-argument predicate is specified as the location μ and concentration κ parameters for multiple independent von Mises distributions [42] for a variety of angles between target and reference objects.

The meanings of two-argument predicates are specified as a pair of von Mises distributions on angles. One, the *position angle*, is the orientation of a vector from the coordinates

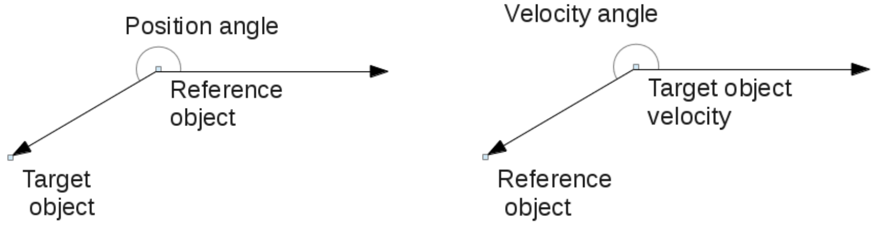


Fig. 4.9.: (left) How position angles are measured. (right) How velocity angles are measured.

of the reference object to the coordinates of the target object (Fig. 4.9 left).[1] The same distribution is used both for SR and path uses. The second, the *velocity angle*, is the angle between the velocity vector at a waypoint and a vector from the coordinates of the waypoint to the coordinates of the reference object (Fig. 4.9 right). This is only used for path uses, because it requires computation of the direction of robot motion which is determined from adjacent waypoints in the path. This angle is thus taken from the frame of reference of the robot.

Fig. 4.2 (bottom) illustrates how this framework is used to represent the meanings of prepositions. Here, we render the angular distributions as potential fields around the reference object at the center for the position angle, and the target object at the center for the velocity angle. The intensity of a point (target object for position angle) reflects its probability mass. Note that the distributions are uniform in velocity angle for *left of*, *right of*, *in front of*, and *behind* and in position angle for *towards* and *away from*.

4.5 Tasks

We formulate sentential semantics as a variety of relationships between a sentence $\mathbf{s}$ (a formula in logical form), a path $\mathbf{p}$ (a sequence of unlabeled waypoints), a floorplan $\mathbf{f}$, (a set of labeled waypoints), and a lexicon Λ (the collective μ and κ parameters for the angular

[1] Without loss of generality, angles are measured in the frame of reference of the robot prior to the beginning of action, which is taken to be the origin.

distributions for each of the two-argument predicates and the discrete distributions for each of the one-argument predicates). Thus the three tasks our method accomplishes are:

acquisition Learn a lexicon Λ from a collection of observed paths $\mathbf{p}_i$ taken by the robot in the corresponding floorplans $\mathbf{f}_i$ as described by human-generated sentences $\mathbf{s}_i$.

generation Generate a sentence $\mathbf{s}$ that describes an observed path $\mathbf{p}$ taken by the robot in a given floorplan $\mathbf{f}$ with a known lexicon Λ.

comprehension Generate a path $\mathbf{p}$ to be taken by the robot that satisfies a given sentence $\mathbf{s}$ issued as a command in a given floorplan $\mathbf{f}$ with a known lexicon Λ.

4.5.1 Acquisition

To perform acquisition, we formulate a large hidden Markov model (HMM), with a state k for every path prepositional phrase PP$_{\text{path},k}$ in each sentence in the training corpus. The observations for this HMM are the sequences of path waypoints in the training corpus. Each state's output model sums over all mappings m between object references in the PP$_{\text{path},k}$ and floorplan waypoints. Given such a mapping, the output model for a state k consists of the product of the probabilities P determined by each atomic formula i in the logical form derived from PP$_{\text{path},k}$, given the probability models for the predicates as specified by the current estimates of the parameters in Λ:

$$R_k(\mathbf{PP_{path,k}}, \mathbf{p}, \mathbf{f}, \Lambda, m) = \quad \prod_i P(w_{a_{i0}} \ldots w_{a_{iN_i}} | \Lambda_i, m) \tag{4.2}$$

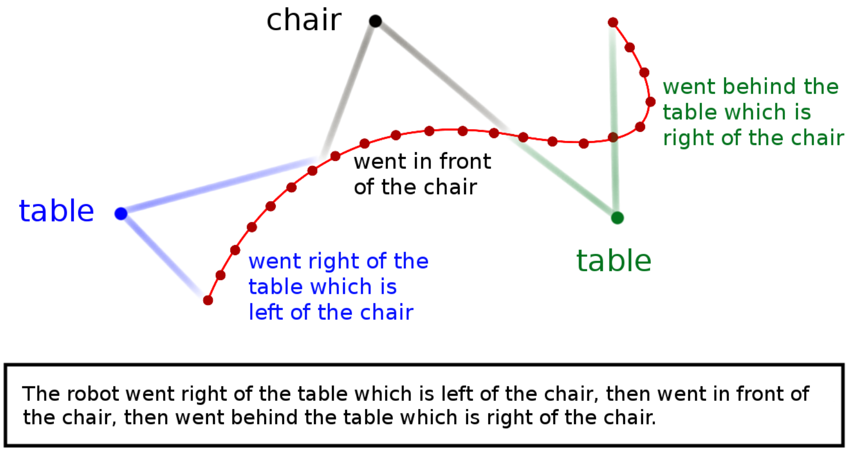
where w is the set of all path and floorplan waypoints, and where a_{ij} is the index in w of the jth argument of the ith atomic formula.

The transition matrix for the HMM is constructed from the sentences in the training corpus to allow each state only to self loop or to transition to the state for the next path prepositional phrase in the training sentence. The HMM is constrained to start in the state associated with the first path prepositional phrase in the sentence associated with each path. We add dummy states, with a small fixed output probability, between the states for each pair of adjacent path prepositional phrases, as well as at the beginning and end of each sentence, to allow for portions of the path that are not described in the associated sentence. We then train this HMM with Baum-Welch [95–97]. This trains the distributions for the words in the lexicon Λ as they are tied as components of the output models. Specifically, it infers the latent alignment between the noisy robot path waypoints and the phrases in the training data while simultaneously updating the meanings of the words to match the relationships between waypoints described in the corpus. In this way, the meanings of both the nouns and the prepositions are learned.

4.5.2 Generation

Language generation takes as input a path $\mathbf{p}$ obtained by odometry during human teleoperation of the robot. This path consists of a collection of 2D floor positions sampled at 50Hz. To generate a formula in logical form, and thus the corresponding sentence, one must select a subsequence of this dense sequence worthy of description.

During generation, we care about three properties: "correctness," that the sentence be logically true of the path, "completeness," that the sentence differentiate the intended path from all other possible paths, and "conciseness," that the sentence be the shortest that does so. We attempt to find a balance between these properties with the following heuristic algorithm, depicted in Fig. 4.10. First, we sample path waypoints in a way that the sampled points are evenly distributed along the path. To this end, we downsample the path by computing the integral distance traveled from the initial position for each point in the dense path and selecting a subsequence whose points are separated by 5cm of integral path length. We then produce a path prepositional phrase to describe each path waypoint by selecting that atomic formula with maximum posterior probability constructed out of a two-argument predicate with the path waypoint as its first argument and with a floorplan waypoint as its second argument. Identical such choices for consecutive sets of waypoints in the path are coalesced and short intervals of path prepositional phrases are discarded. We then generate a noun phrase for the object of each waypoint preposition that refers to that referenced floorplan waypoint. We take a one-argument predicate to be true of that


chair
went behind the table which is right of the chair
went in front of the chair
table
table
went right of the table which is left of the chair

The robot went right of the table which is left of the chair, then went in front of the chair, then went behind the table which is right of the chair.

Fig. 4.10.: Illustration of the generation algorithm. A disambiguating noun phrase is generated for each floorplan waypoint. Path waypoints are described by prepositional phrases, and then sets of identical phrases are merged into intervals, which are combined to form the sentence.

class with maximum posterior probability and false of all others. Similarly, for each pair of floorplan waypoints, we take that two-argument predicate with maximum posterior probability to be true of that tuple and all other predicates applied to that tuple to be false. Thus when the floorplan contains a single instance of a class, it can be referred to with a simple noun. But when there are multiple instances of a class, the shortest possible noun phrase, with one or more SR prepositional phrases, is generated to disambiguate.

More formally, let $q(o)$ be the class name of the object at the floorplan waypoint o. For each pair of floorplan waypoints (o, o'), there exists at least one two-argument spatial-relation predicate ϕ that is true of this tuple. Let $u(o)$ be the noun phrase we want to generate to disambiguate the floorplan waypoint o from others o'. Then o can be referred to with $u(o)$ unambiguously if **(a)** $u(o) = (q(o), \{\})$ is unique; or **(b)**, there exists a collection of two-argument predicates $\{\phi(o, o')\}$ such that formula $u(o) = (q(o), \{(\phi, u(o'))\})$ is unique. To produce a concise sentence, we want the size of the collection of two-argument predicates in step **(b)** above to be as small as possible. However, finding the smallest collection of modifiers is NP-hard [98, 99]. To avoid exhaustive search, we use a greedy heuristic that biases towards adding the least frequent pairs $(\phi, u(o'))$ into the collection until $u(o)$ is unique. This results in a tractable polynomial algorithm. After we get $u(o)$, we turn it into a noun phrase by simple realization, for example:

$$(\text{TABLE}, \{(\text{LEFT-OF}, \text{CHAIR}), (\text{BEHIND}, \text{TABLE})\})$$

$$\downarrow$$

the table which is left of the chair and behind the table

4.5.3 Comprehension

To perform comprehension, we use gradient ascent to optimize the scoring function with respect to an unknown path $\mathbf{p}$

$$\mathbf{p}^* = \underset{\mathbf{p}}{\operatorname{argmax}} \mathcal{R}(\mathbf{s}, \mathbf{p}, \mathbf{f}, \Lambda) \tag{4.3}$$

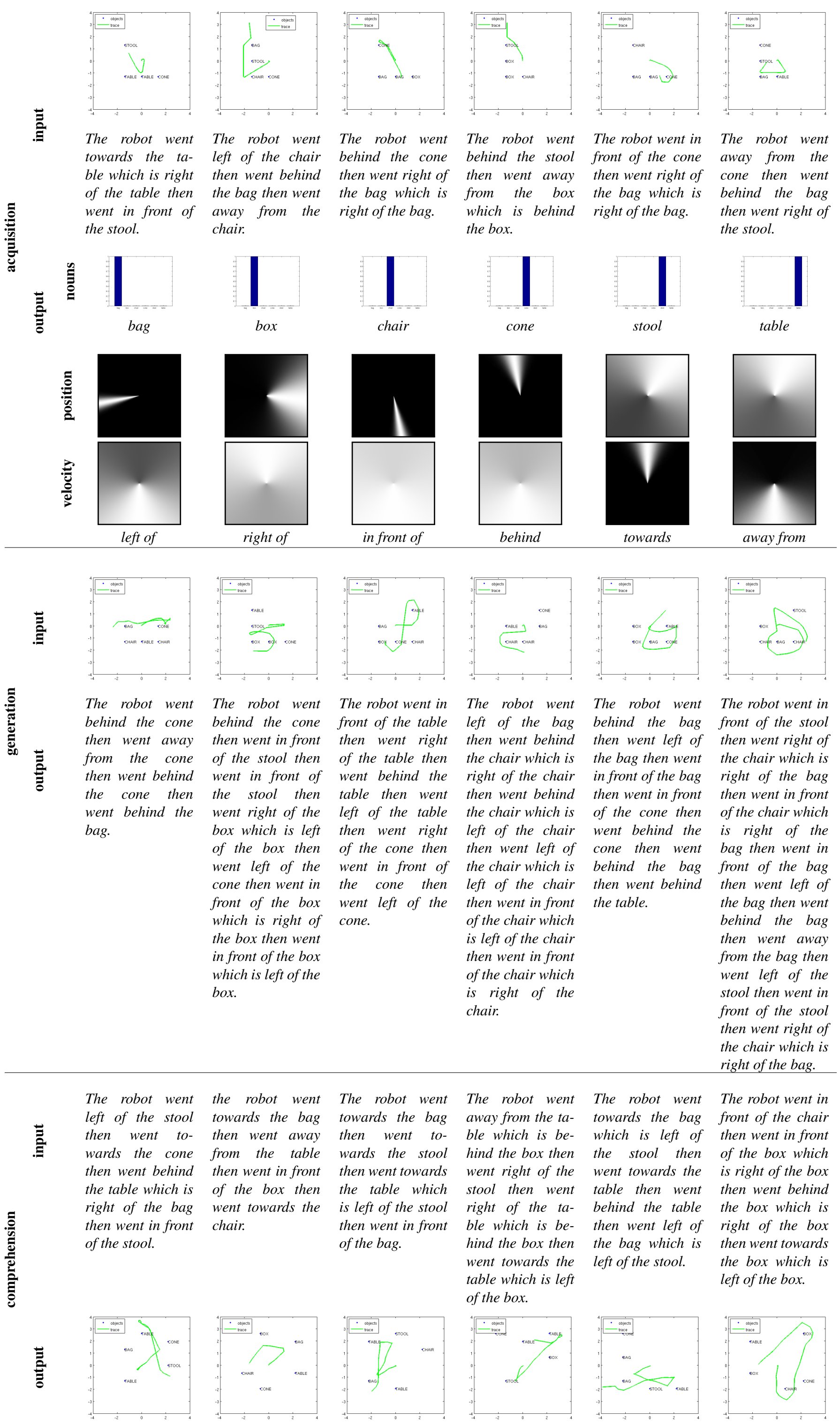
where $\mathcal{R}(\mathbf{s}, \mathbf{p}, \mathbf{f}, \Lambda)$ is the product of all R_k from (4.2). We are computing a MAP estimate of the joint probability of satisfying the conjunction of atomic formulas assuming that they are independent.

The above scoring function alone is insufficient. It represents the strict meaning of the sentence, but does not take into account constraints of the world, such as the need to avoid collision with the objects in the floorplan. It can also be difficult to optimize because the cost associated with the relative orientation between two waypoints becomes increasingly sensitive to small changes in position as they become closer together. To remedy the problems of the path waypoints getting too close to objects and to each other, a barrier penalty term is added between each pair of a path waypoint and floorplan waypoint as well as between pairs of temporally adjacent path waypoints to prevent them from becoming too close. This term is 1 until the distance between the two waypoints becomes less than a threshold, at which point it decreases rapidly. Finally, our formulation of the semantics of prepositions is based on angles but not distance. Thus there is is a large subspace of the floor that leads to equal probability of satisfying each atomic formula, *i.e.,* the cones in Fig. 4.2. This allows a path to satisfy a prepositional phrase like *to the left of the chair* by being far away from the chair. To remedy this, we add a small attraction between each path waypoint and the floorplan waypoints selected as its reference objects to prefer short distances. A postprocessing step performs obstacle avoidance by adding additional path waypoints as needed.

4.6 Experiments

We conducted an experiment as outlined in Fig. 4.1 through Fig. 4.4 . We generated 250 random sentences from the grammar in Fig. 4.6, 25 in each of 10 different floorplans that were randomly generated to place either 4 or 5 objects, with 2 objects always being of the same class, to introduce ambiguity requiring disambiguation via SR prepositional phrases, at one of 12 possible grid positions. Path data was logged while a human teleoperator manually drove the robot to comply with these sentential instructions in these floorplans (Fig. 4.11 top). Models were learned for each of the nouns and prepositions. These were used to automatically generate descriptions for 10 different new paths manually driven by a human teleoperator in each of 10 new random floorplans (Fig. 4.11 middle). These were also used to automatically drive the robot to follow 10 different new random sentences in each of 10 different new random floorplans where the same objects could be placed at one of 56 possible grid positions (Fig. 4.11 bottom). The random sentences used for training had either 2 or 3 path waypoints while those used for generation and comprehension had either 5 or 6 path waypoints.

Odometry and inertial guidance were used to determine paths driven. Pairs of sentences and paths obtained during both generation and comprehension were given to a pool of 6 independent judges to obtain 3 judgments on each. Judges were asked to label each path prepositional phrase in each sentence paired with the entire path as being either ‘correct’ or ‘incorrect’, *i.e.,* whether it was true of the intended portion of the path as determined by that judge. For generation, judges were also asked to assess how much of the path was described by the sentence, giving a completeness judgment ranging from 0 (worst) to 5 (best). These were converted to percentages. For comprehension, judges were also asked



The robot went towards the table which is right of the table then went in front of the stool.

The robot went left of the chair then went behind the bag then went away from the chair.

The robot went behind the cone then went right of the bag which is right of the bag.

The robot went behind the stool then went away from the box which is behind the box.

The robot went in front of the cone then went right of the bag which is right of the bag.

The robot went away from the cone then went behind the bag then went right of the stool.

The robot went behind the cone then went away from the cone then went behind the cone then went behind the bag.

The robot went behind the cone then went in front of the stool then went in front of the stool then went right of the box which is left of the box then went left of the cone then went in front of the box which is right of the box then went in front of the box which is left of the box.

The robot went in front of the table then went right of the table then went behind the table then went left of the table then went right of the cone then went in front of the cone then went left of the cone.

The robot went left of the bag then went behind the chair which is right of the chair then went behind the chair which is left of the chair then went left of the chair which is left of the chair then went in front of the chair which is left of the chair then went in front of the chair which is right of the chair.

The robot went behind the bag then went left of the bag then went in front of the bag then went in front of the cone then went behind the cone then went behind the bag then went behind the table.

The robot went in front of the stool then went right of the chair which is right of the bag then went in front of the chair which is right of the bag then went in front of the bag then went left of the bag then went behind the bag then went away from the bag then went left of the stool then went in front of the stool then went right of the chair which is right of the bag.

The robot went left of the stool then went towards the cone then went behind the table which is right of the bag then went in front of the stool.

the robot went towards the bag then went away from the table then went in front of the box then went towards the chair.

The robot went towards the bag then went towards the stool then went towards the table which is left of the stool then went in front of the bag.

The robot went away from the table which is behind the box then went right of the stool then went right of the table which is behind the box then went towards the table which is left of the box.

The robot went towards the bag which is left of the stool then went towards the table then went behind the table then went left of the bag which is left of the stool.

The robot went in front of the chair then went in front of the box which is right of the box then went behind the box which is right of the box then went towards the box which is left of the box.

Fig. 4.11.: Example experimental runs, 6 out of 250 for acquisition and 100 for each of generation and comprehension. Videos available at `http://drivingundertheinfluenceoflanguage.blogspot.com`.

Table 4.1.: Correctness and completeness results of human evaluation of sentences automatically generated from manually driven paths and automatically driven paths produced by comprehension of provided sentences.

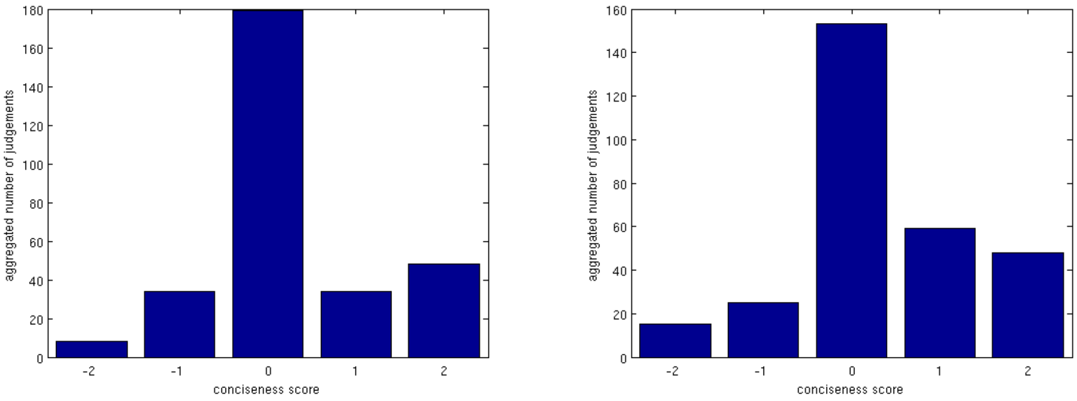
	correctness		completeness	
	mean	std dev	mean	std dev
generation (hand-constructed models)	94.6%	4.54%	85.5%	2.26%
generation (learned models)	92.0%	6.11%	84.2%	6.35%
comprehension (planned path)	96.2%	0.38%	88.5%	11.5%
comprehension (measured path)	95.5%	1.42%	84.7%	9.9%

to assess what fraction of the path constitutes motion that is described by the sentence (quantized as 0 to 5). These were again converted to percentages to measure completeness. For generation, judgments were obtained twice, pairing each input path with sentences generated using the hand-constructed models from Fig. 4.2 as well the learned models from Fig. 4.11. For comprehension, judgments were also obtained twice, pairing each input sentence with both the planned path as well as the actually driven path as determined by odometry and inertial guidance. Table 4.1 summarizes the judgments aggregated across the 3 judges and 100 samples. The standard deviations are across the mean value of the 3 judges for each sample. Overall, the average "correctness" reported is 94.6% and the average "completeness" reported is 85.6%.

For generation, we also measured "conciseness" by having the 3 human judges score each generated sentence as -2 (much too short), -1 (too short), 0 (about right), 1 (too long), or 2 (much too long). Fig. 4.12 summarizes these judgments as histograms. Overall, judges assessed that the generated sentence length was 'about right' a little over half of the time, with generation erring more towards being too long than too short.

4.7 Conclusion

We demonstrate a novel approach for grounding the semantics of natural language in the domain of robot navigation. Sentences describe paths taken by the robot relative to other objects in the environment. The meanings of nouns and prepositions are trained from



(hand-constructed models)

(learned models)

Fig. 4.12.: Conciseness results of human evaluation of sentences automatically generated from manually driven paths and automatically driven paths produced by comprehension of provided sentences.

a corpus of paths driven by a human teleoperator annotated with sentential descriptions. These can then support both automatic generation of sentential descriptions of new paths driven as well as automatic driving of paths to satisfy navigational goals specified in provided sentences. This is a step towards the ultimate goal of grounded natural language that allows machines to interact with humans when the language refers to actual things and activities in the real world.

Acknowledgments

This research was sponsored, in part, by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0060. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

5. EXCERPTS AND SUMMARIES FROM “DRIVING UNDER THE INFLUENCE (OF LANGUAGE)”

5.1 Introduction

After completing the work in Chapter 4, my coauthors Daniel Barrett, Haonan Yu, and I expanded that work into a journal-length manuscript which we entitled “Driving Under the Influence (of Language).” The overall contribution is similar to the contribution summarized in Fig. 4.1 through Fig. 4.4. For the convenience of the reader, Fig. 4.1 is reproduced here as Fig. 5.1.

The most significant difference between the two works is the removal of the fixed grammar and logical form (Fig. 4.6 and Fig. 4.7). Instead of using the randomly-generated sentences as the linguistic description of the driven paths, we instead just use them as a seed for the manual driving. We then place the traces recovered from odometry and IMU data on Amazon Mechanical Turk (AMT) to gather free-form sentential descriptions from anonymous workers. These sentences form the language input to our Acquisition method. We also use AMT workers, in a separate task, to evaluate the accuracy of both the sentential descriptions of driven paths produced by our Generation method and the automatically induced driving paths produced by our Comprehension method in response to natural-language commands. We evaluate on the same metrics as in Chapter 4. For “sentence correctness” (how true the sentence is of the path), our method achieves an average score of 74.2%; for “sentence completeness” (how fully the sentence covers the path), 74.5%, and for “path completeness” (how fully the path covers the sentence), 76.0%. The overall average is 74.9%.

We submitted the full manuscript to IEEE Transactions on Neural Networks and Learning Systems on 26 January 2016. It is currently under review.

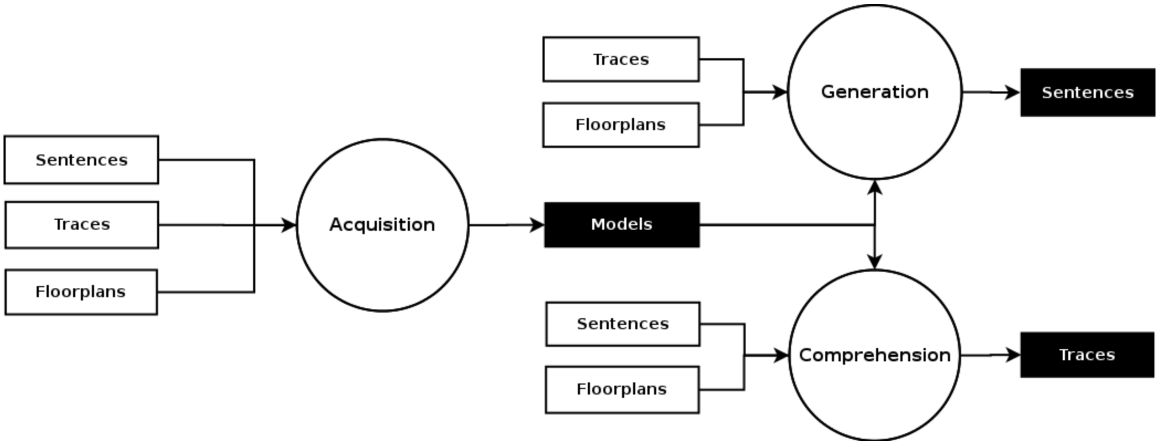


Fig. 5.1.: Data flow diagram overview of the system. Boxes with black text on white background represent data generated or collected by humans, while boxes with white text on black background represent data that is machine generated. The Acquisition subsystem (Fig. 4.2) takes sentential descriptions of paths, the traces of such paths as driven by a human teleoperator, and floorplan specifications as input. It produces learned models of the nouns and prepositions in such sentences as output. The Generation subsystem (Fig. 4.3) takes these learned models, along with traces of new manually driven paths and new floorplans, and produces sentences that describe the driven paths. The Comprehension subsystem (Fig. 4.4) also uses the learned noun and preposition models with new sentences and new floorplans to produce traces that meet the navigational goals specified in the sentences. Our custom mobile robot (Fig 5.2) can autonomously drive such traces.

In this chapter I excerpt some sections of the manuscript verbatim, while I summarize others. These are denoted with **(excerpt)** or **(summary)**, respectively.

5.2 Related Work (excerpt)

We know of no other work which presents a physical robot which learns word meanings from driven paths paired with sentences, uses these learned meanings to generate sentential descriptions of driven paths, and automatically plans and physically drives paths satisfying sentential descriptions.

Table 5.1 compares the properties of the work reported in this chapter with the work reported in twenty recent related papers which are further discussed below.

Table 5.1.: A comparison of the properties of the work reported in this chapter with that reported in twenty recent related papers. Unless otherwise noted, the green and red boxes mean yes and no respectively.

	This Chapter	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[16]	[17]	[18]	[19]	[20]	[21]	[22]	[23]	[24]	[25]	[26]	[27]
Physical Robot?	🟩	🟥	🟥[4]	🟥[4]	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥[23]	🟩	🟥	🟥	🟩
Learn word meanings?	🟩	🟥[1]	🟩	🟥	🟩	🟥[1]	🟥[8]	🟩	🟩[13]	🟩[15]	🟥[19]	🟥	🟥	🟥	🟥	🟥	🟥	🟥[22]	🟥	🟩	🟥[22]
Generate descriptions of paths?	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
Generate then drive or follow paths from descriptions?	🟩	🟩	🟩	🟩	🟥	🟩	🟩	🟩	🟩	🟥	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟥	🟥	🟥	🟩[26]
Noisy Data?	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥[16]	🟥[21]	🟩	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩
Domain? Continuous 🟩 Discrete 🟥	🟩	🟥	🟥[4]	🟥[4]	🟥	🟥	🟥	🟥	🟥	🟩	🟥[21]	🟩	🟥	🟥	🟥	🟩	🟥	🟩	🟥	🟥[25]	🟩
Environment? Real 🟩 or Virtual 🟥	🟩	🟥	🟥[4]	🟥[4]	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟥	🟥	🟥	🟩	🟥	🟩	🟥	🟩	🟩
Language Corpus Size (# of words) Unspecified ▨, N/A ⬛	12,616 total 247 unique	587 [2]	▨	▨	▨	▨	▨[9]	▨[11]	266 [2]	▨	▨	▨[18]	⬛	⬛	⬛	⬛	⬛	▨	⬛	▨	▨
Learned Lexicon Size (# of words) **N**ouns, **P**repositions, or **A**djectives Unspecified ▨, N/A ⬛	12 6 **N**, 6 **P** 🟩	⬛	11 11 **P** 🟩	▨	▨	⬛	⬛	▨	▨	16 7 **P**, 9 **A** 🟩	▨	⬛	⬛	⬛	⬛	⬛	⬛	▨	⬛	▨	▨
Learns without manual annotation? does not learn ⬛	🟩	🟥[3]	🟥[5]	🟩	🟥[6]	🟥[7]	🟥[10]	🟥[12]	🟥[14]	🟥[17]	🟥[20]	⬛	⬛	⬛	⬛	⬛	🟥[24]	🟩	🟩	⬛	🟥[27]

Notes:

[1] Does not learn individual word meanings, instead maps from NL phrases to *compound action specifications* which apply contextual cues of the phrase to the parameters of one of a small number of predetermined actions.

[2] Uniqueness of such is not stated.

[3] Requires hand-verified parse trees as system input.

[4] Authors collect dataset from a SLAM-equipped mobile robot, then create from this a discrete grid-map (a graph with vertices and edges) which is used to conduct experiments in simulation.

[5] Requires hand-drawn examples of paths that depict spatial relations as well as hand-grounded locations of objects.

[6] Requires the manual grounding of all words to specific objects (nouns) or relationships (prepositions) in the training corpus.

[7] Utilizes training data from [7] that has been manually parsed and aligned so that each sentence corresponds to a single predetermined action.

[8] Like [7] and [11], does not learn individual word meanings, but learns the mappings from NL phrases to one of a small number of movement, location, or logic terms in their Robot Control Language.

[9] Language corpus consists of 189 unique sentences.

[10] Requires natural-language commands to be hand-segmented into individual movement phrases and then manually annotated in RCL for training.

[11] Language corpus consists of 1863 sentences for training and 816 sentences for testing, uniqueness of such is not stated.

[12] Uses same dataset as [7] and [11], and thus needs the same manual pairing of NL sentences to action sequences.

[13] Learns mappings from natural-language noun phrases to manually-annotated object groundings.

[14] Dataset has hand-annotated sentence parses and object groundings for each natural-language command.

[15] Learns only preposition/adverb meanings, not nouns.

[16] Uses robot's internal state and environment representation to suppress noise.

[17] Requires hand-grounding of nouns.

[18] Only language used is a small set of utterances directing movement, which map to specific motor responses.

[19] Uses hand-grounded primitives that map from word or phrase to robot action; learns mappings from higher-level instructions to sequences of action primitives.

[20] Authors do manual analysis of collected NL instruction corpus to determine list of required robot action primitives.

[21] System uses a Robot Manager module that abstracts the noisy, continuous data into a discrete, symbolic form.

[22] Learns mappings of phrases/instructions to robot arm actions.

[23] All experiments are done in simulation, but some environment maps are created from SLAM data from a physical robot.

[24] Uses prepositions which are represented as hand-coded semantic fields and nouns which are manually grounded to map data.

[25] System's perceptual framework takes continuous RGB-D data and segments it into discrete objects.

[26] Generates and follows paths and sets of actions for the robot arm.

[27] All learning is done through interaction with a human teacher.

While there is other work which learns the meanings of words in the context of description of navigation paths, these systems operate only within discrete simulation, as they utilize the internal representation of the simulation to obtain discrete symbolic primitives [7–14]. They have a small space of possible robot actions, positions, and states which are represented in terms of symbolic primitives like TURN LEFT, TURN RIGHT, and MOVE FORWARD N STEPS (*e.g.,* [11, 12]), or DRIVE TO LOCATION 1 and PICK UP PALLET 1 (*e.g.,* [14]). Thus, they take a sequence of primitives like $\{$DRIVE TO LOCATION 1; PICK UP PALLET 1$\}$ and a sentence like *go to the pallet and pick it up* and learn that the word *pallet* maps to the primitive PALLET, that the phrase *pick up* maps to the primitive PICK UP, and that the phrase *go to X* means DRIVE TO LOCATION X.

We solve a more difficult problem. Our robot and environment are in the continuous physical world and can take an uncountably infinite number of configurations. Our input is a set of sentences matched with robot paths, which are sequences of points in the real 2D Cartesian plane, densely sampled in time. Not all points in the path correspond to words in the sentences; multiple (often undescribed) relationships can be true of any point, and the correspondence between described relationships and path points is unknown. Furthermore, our system does not require the additional manually annotated data upon which much of the previous work depends.

MacMahon *et al.* [7] introduce a software agent that follows natural-language route instructions in a discrete grid-based virtual indoor environment. This system does not learn individual word meanings. Instead, it maps from natural-language phrases to what they call a *compound action specification*, which applies the linguistic contextual cues of the instruction to the parameters of one of the four predetermined actions that their system can take (TURN, TRAVEL, VERIFY, and DECLARE-GOAL). This system also requires manually annotated data, in the form of hand-verified parse trees of the natural-language route instructions, as input.

Kollar *et al.* [8] present a system for natural-language direction following that operates in simulation on a discrete virtual environment derived from SLAM data collected by a physical robot. This system requires two different types of manual annotation. To locate objects (nouns), it takes a manually seeded map of the names and locations of such objects. To learn the meanings of prepositions, this system requires a corpus of hand-drawn examples of paths, both positive and negative, that depict such prepositions.

Matuszek *et al.* [9] also do experiments in simulation on a discrete topological grid map derived from SLAM data from a physical robot, similar to Kollar *et al.* [8], although their maps are segmented using the approach in Friedman *et al.* [39]. They focus on using statistical machine translation to map from imperative natural-language directions (*e.g., take the second left*) to a *path description language* which directs the motion of their simulated robot, taking advantage of the geometry of the map to constrain the combinatorics of possible actions at a given location. They learn no individual word meanings, as we do; instead, they learn the mappings between phrases, such as the previous example, and the sequence of simulated robot commands that follow such an instruction. Their system is not capable of handling natural-language descriptions of objects or other landmarks that appear in the environment. Finally, their evaluation is on a small corpus of only fourteen sets of route instructions and uses an oracle to evaluate whether the simulated robot reaches the correct destination by the intended route.

Tellex *et al.* [10] describe a system for understanding natural-language commands prescribing navigation and limited manipulation in discrete simulated environments. Like our system, this system utilizes natural-language commands collected from AMT. However, this system is only capable of learning word meanings, represented as the words' groundings within the environment, not using the learned word meanings for subsequent tasks. To do so, it requires manual annotation of the groundings of all words in the training sentences to specific objects (nouns) and relationships (prepositions) in the training data. It is incapable of generating either sentential descriptions from given paths, or paths from given descriptions.

Chen & Mooney [11] present a system that learns to interpret natural-language navigation instructions by mapping them to executable plans. This system uses the discrete grid-based virtual environments, data, and simulation execution module developed and used by MacMahon *et al.* [7]. Like that system [7], this system does not directly learn word meanings, but rather it maps phrases to *compound action specifications* which in turn map to the parameters of one of three predetermined actions (TURN, TRAVEL, and VERIFY). Additionally, Chen & Mooney manually parse and align the data from MacMahon *et al.* [7] to form training data in which each sentence in a human-generated instruction is paired with a corresponding predetermined action.

Matuszek *et al.* [12] seek to learn a semantic parsing model which takes natural-language instructions and produces robot-executable commands in what they call *Robot Command Language* (RCL). They learn a distribution over possible RCL sequences through the use of natural-language commands, which are first hand-segmented into discrete movement phrases and then paired with expert-created annotations of such in RCL. This is similar to both MacMahon *et al.* [7] and Chen & Mooney [11]; all learn the mappings from phrases to commands rather than representations of the meanings of individual words. Finally, the authors test their system by simulating the movement of a robot within a discrete grid-based virtual indoor environment and making a binary judgment on whether or not the simulated robot reaches the intended destination by the desired route.

Artzi & Zettlemoyer [13] learn a semantic parser as well as individual word meanings, represented as the grounding of objects or relationships within their test environment. They utilize the same dataset as MacMahon *et al.* [7] and Chen & Mooney [11] and thus operate in simulation within the same discrete grid-based virtual environment used in the work described in those papers; however, they hand-filter the dataset to include only correct sentence-trace pairs. Similarly, this work also requires a training set in which natural-language instructions are manually paired with executable action sequences.

Tellex *et al.* [14] present a system that learns grounded word meanings from a training corpus consisting of natural-language commands paired with video or log data depicting such actions, similar to our acquisition task (Section 4.5.1). This system also follows paths derived from sentential descriptions, much like our comprehension task (Section 4.5.3) However, this system operates only in simulation in a grid-based virtual environment with a discrete space of possible robot actions. Furthermore, this system also requires manual annotation of sentence parses and object groundings for each natural-language command in the training corpus. The system then learns the mappings from natural-language noun phrases to these object groundings.

There has been work on learning in the context of language and mobile robot navigation using a physical robot (*e.g.,* [16,17]), but none of these do all three of the tasks (acquisition, generation, and comprehension) which we do.

Dobnik *et al.* [16] have an actual robot but only learn to classify prepositional phrases like *A is near B* from robot paths paired with such phrases that have hand-grounded nouns. They neither generate sentences, nor automatically drive paths. Our system can do both of these, as well as learn meanings for both nouns and prepositions.

Lauria *et al.* [17] use a collected corpus of high-level route instructions given by naïve users to a small robot operating in a miniaturized outdoor setting in order to learn symbolic representations of motion tasks that can generalize such instructions. The system described therein assumes the use of pre-programmed primitives which encode the basic sensory-motor actions of the robot. The authors manually analyze their collected corpus and extract a set of 14 primitives, from the simple (TURN RIGHT) to the complex (TAKE THE NTH TURN AFTER X), which they ground to robot sensory-motor actions by hand. Their learning occurs at a higher level of abstraction than ours, focusing on finding the mappings from natural-language route instructions to sequences of these hand-crafted action primitives.

There is also recent work on the topic of natural-language interaction with robots (*e.g.,* [18–23]), both within and outside the realm of robotic navigation. However, such work does not involve any learning.

Teller *et al.* [18] present a robotic forklift which accepts speech and pen-based input, operates in semi-structured outdoor environments, and has a robust sensing architecture which allows it to operate in close proximity to humans. This system uses an off-the-shelf speech recognizer [43] to turn spoken utterances into text. Such utterances are limited to a small set of phrases that direct movement. The system's responses to the text derived from these utterances is based on predefined knowledge of locations in the environment and hand-programmed mappings between the words of an utterance and robotic actions required to fulfill such. This system does no learning and cannot respond to commands for which it does not possess manually-defined primitives.

Koller *et al.* [19] shows similarity to our generation task (Section 4.5.2) in that while we generate sentences that describe a robot path, their systems provide instructions to guide human users through virtual game worlds. Unlike our system, however, the systems described therein operate in a virtual environment, which gives them a complete and unambiguous discrete symbolic representation of the world. Additionally, their systems do no learning and instead rely on hand-coded mappings between words and actions in order to generate instructions.

Harris *et al.* [20] present a dialog system in which a single human interacts verbally with multiple robots in a simulation of a treasure-hunt scenario. Human participants are given a map of a maze and must direct robots from unknown starting locations to a destination using only imperative commands like MOVE NORTH 5 METERS and responses to queries like REPORT. Like many other simulation systems, this system uses a discrete grid-based virtual environment with a complete and unambiguous internal symbolic representation. This system does no learning and is instead focused on disambiguation of instructions from a single human sender to multiple robot receivers.

Marge *et al.* [21] describe TEAMTALK, a human-robot interface which allows verbal interaction between multiple people and multiple robots in a virtual treasure-hunt scenario similar to Harris *et al.* [20]. Its primary goal is to construct a policy which is able to handle dynamic and asynchronous conversation between a group of four or more humans and robots. Like Harris *et al.* [20], TEAMTALK uses a discrete grid-based simulation environment with a symbolic representation. TEAMTALK also does no learning.

Pappu & Rudnicky [22] collect and present the NAVIGATI corpus, a dataset of human-generated route instructions in an indoor environment. Their goal is to analyze the corpus and construct a grammar that would generalize to other navigational corpora. The authors focus solely on the linguistic content of the instructions and neither learn word meanings nor control robots, simulated or real, via language.

Fasola & Mataric [23] investigate methods for enabling service robots to better understand spatial language from non-expert users. To do this, they use semantic fields to represent prepositions. Their system requires prepositions to be manually encoded as such. It also requires the hand-grounding of nouns to map data. With such human-generated knowledge, the system is able to perform path planning in a variety of discrete simulation environments. This is similar to our comprehension task (Section 4.5.3), although our system operates in the continuous physical world using knowledge learned in the acquisition task (Section 4.5.1). Their system does no learning.

There is also work which performs learning in the context of language and robotics, but not navigation (*e.g.,* [24–27]).

McGuire *et al.* [24] use gesture recognition and speech commands to interact with a robot arm by human demonstration. This allows a user to train the robot arm to associate a natural-language command with a specific manipulation task. However, this system is unable to respond to a natural-language command for which it is not specifically programmed.

Doshi & Roy [25] present a dialogue management system designed to learn to overcome the limitations of noisy speech recognition within the paradigm of a robotic wheelchair. All testing of this learning approach is done in simulation using a small number of discrete states. As their focus is on dialogue management, they do no navigation; they assume that if their system is able to determine the correct goal state, it is also able to navigate to such via a manually-programmed route.

Matuszek *et al.* [26] propose a joint framework to learn linguistic and perceptual, primarily visual, models for grounding language to physical objects in a static scene. Their system trains color and shape classifiers on objects segmented from images captured with an RGB-D camera and then learns the mapping between such classifiers and linguistic descriptors in the form of adjectives (*i.e., green*) and nouns (*i.e., triangle*). While both this work and ours ground language in robotic perception, the modalities of such perception are so different as to preclude direct comparison. Our system perceives its environment via mechanical sensors such as wheel encoders and an IMU (Section 5.3), while their system employs computer-vision techniques on camera data. Our current work does not utilize any vision data, although we intend to do so in the future. Furthermore, while their system learns nouns and static adjectives (colors) that describe such, our system learns nouns as well as prepositions that describe noun properties which are both static (spatial relations between stationary objects) and dynamic (spatial relations between a stationary object and our mobile robot).

She *et al.* [27] teach a physical manipulator arm to interact with objects through student-teacher dialogue. Their work connects high-level symbolic representations of language with low-level sensorimotor representations internal to the robot. Like Matuszek *et al.* [26], their system perceives the world primarily through visual means, which is starkly different from the mechanical perception that our system uses. Additionally, their system learns manipulation tasks through interactive dialogue with a human, while our system learns individual word meanings from a fixed training corpus.

5.3 Our Mobile Robot (excerpt)

All experiments were performed on a custom mobile robot (Fig. 5.2). This robot can be driven by a human teleoperator or drive itself automatically to accomplish specified navigational goals. During all operation, robot localization is performed onboard the robot in real time via an Extended Kalman Filter [90, 91] with odometry from shaft encoders on the wheels and inertial guidance from an IMU.

Due to sensor noise and mechanical factors such as wheel slippage, this localization is noisy, but on average is within 20cm of the actual location, which is approximately one-half of the robot's length. The video feed, localization, and all sensor and actuator data are logged in a time-stamped format. When conducting experiments on acquisition and generation, a human teleoperator drives the robot along a variety of paths in a variety of floorplans. The paths recovered from localization support acquisition and generation. When conducting experiments on comprehension, a path is first planned automatically, then the robot follows this planned path by comparing the new odometry gathered in real time with the planned path and controlling the wheels accordingly.

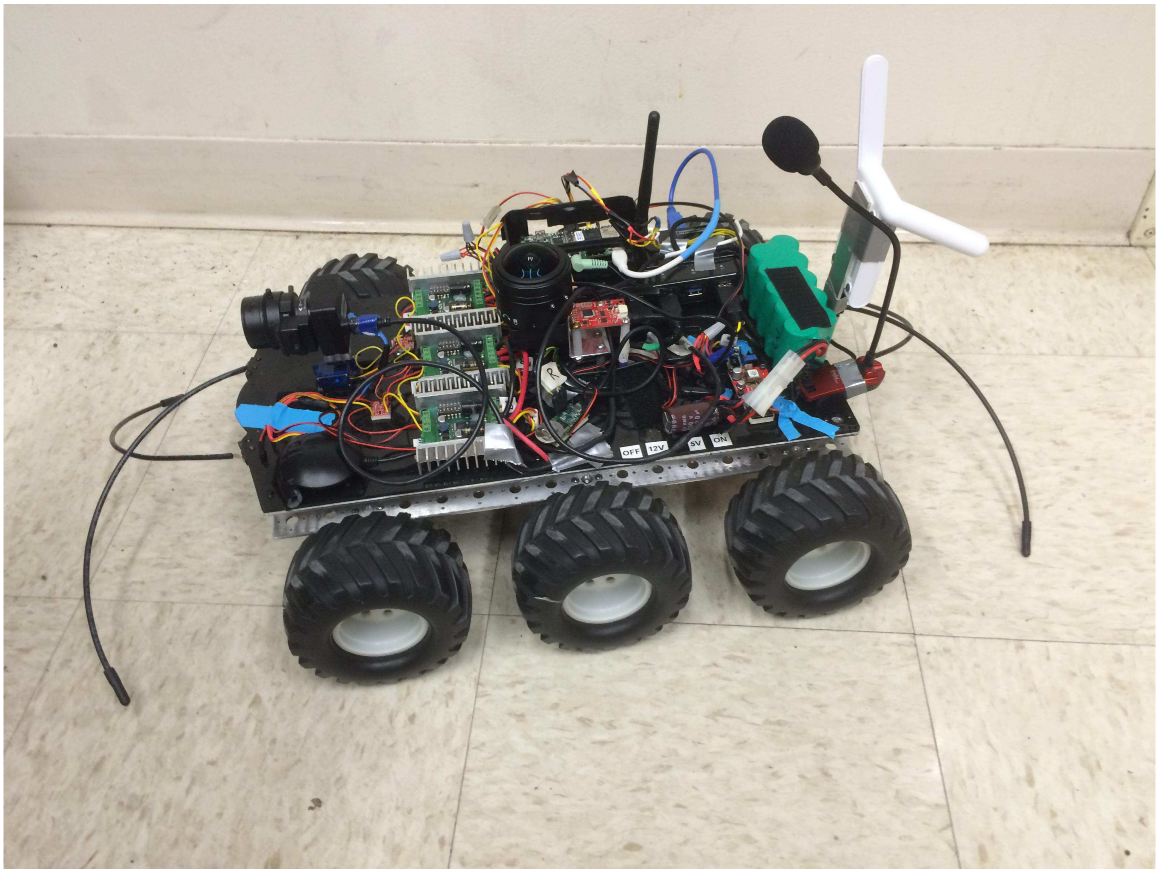


Fig. 5.2.: Our custom mobile robot.

The use of an actual robot with noisy real-world sensor data increases the difficulty of these tasks when compared to work which occurs in simulation. The noisy robot position is densely sampled in the continuous domain. For acquisition and generation, this adds an additional layer of uncertainty, as the correspondence between individual points in the robot path and the phrases of a sentence is unknown. For comprehension, the robot must find a path by choosing waypoints in the continuous domain that both maximally satisfy the meaning of the instructions and avoid collision with objects. It does not rely on a small set of discrete locations with associated symbolic commands like *goto(table)*, as many (simulated and robotic) systems do.

5.4 Extracting Meaning from a Sentence (summary)

Since in this work we no longer use the fixed grammar and logical form described in Section 4.4.1 and depicted in Fig. 4.6 and Fig. 4.7, we must devise a method to determine the meaning of free-form natural-language sentences which describe robot paths. We do so by representing a sentence as a sequence of graphical models, with each graphical model representing a clause which describes a temporal segment of the path. These graphical models are joint distributions over a single *path variable* (a pair of 2D vectors which represent the robot's position and velocity at a given instant) and a set of *floorplan variables* (representations of the class and position of each object in a floorplan via labeled 2D coordinates).

5.4.1 Constructing graphical models from a sentence (summary)

Our graphical models are constructed from the nouns and the prepositions in each sentential description of a robot path. Each clause in a sentence becomes a path variable. The nouns in each clause become floorplan variables, each with a discrete distribution over possible object labels. The prepositions are represented as joint distributions between target and referent objects; these may be adverbial or adjectival uses of the preposition.

Constructing the graphical models is a three-step process: parsing the sentence into sequential segments, identifying nouns and prepositions, and finding the arguments to each preposition. To parse a sentence, we use all verbs which do not follow a determiner (*e.g., that, which*), as well as adverbial transition words (*e.g., then*), as segment boundaries. We developed this method because we found that off-the-shelf parsers, such as the Stanford parser [100], give erroneous parse trees on our corpus. However, we do use the Stanford parser's part-of-speech tagging in a manner similar to that used in [23], along with a list of prepositions from Wikipedia [101], to identify nouns and prepositions within our sentences. Finally, we use a small set of rules to identify the arguments to prepositions.

5.4.2 Representation of the lexicon (summary)

This section is substantially the same as Section 4.4.2.

5.5 Tasks (summary)

This section is substantially similar to Section 4.5. However, instead of representing a sentence $\mathbf{s}$ as a formula in logical form, here we represent it as a sequence of graphical models. We represent a path $\mathbf{p}$, a floorplan $\mathbf{f}$, and a lexicon Λ in the same manner as in Section 4.5. We also accomplish the same **acquisition**, **generation**, and **comprehension** tasks as previous.

5.5.1 Acquisition (summary)

This section is similar to Section 4.5.1, except for the fact that we now generate the Hidden Markov Models from the graphical models instead of from formulas in logical form.

5.5.2 Generation (summary)

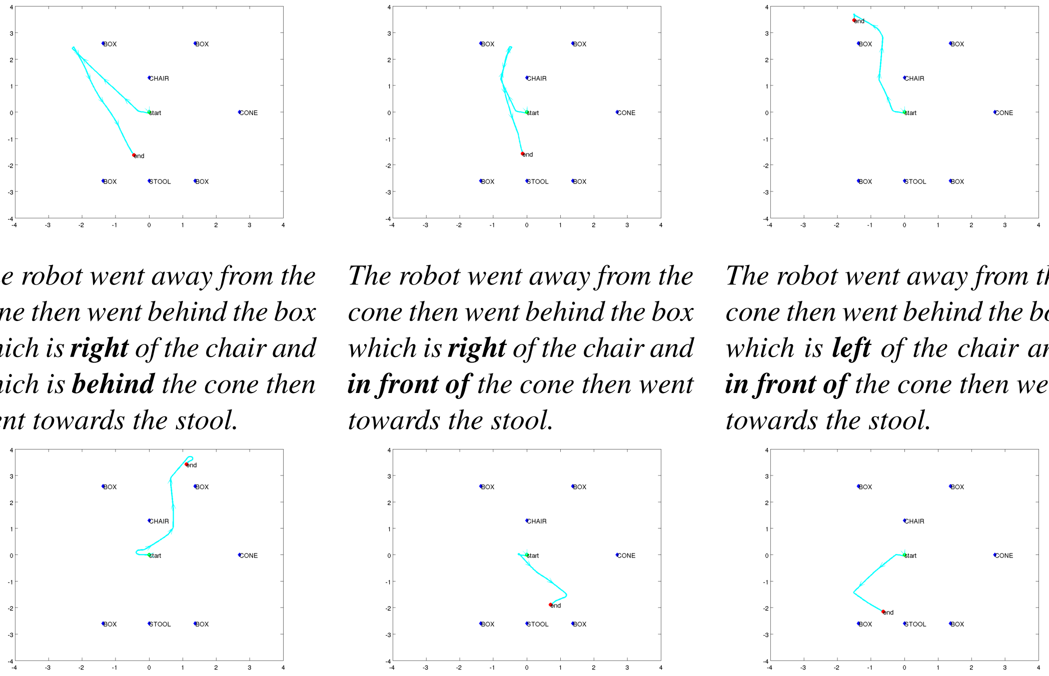
This section is substantially the same as Section 4.5.2.

5.5.3 Comprehension (summary)

This section is similar to Section 4.5.3. One noteworthy addition in the new manuscript is Fig. 5.3, which depicts what we came to call the “minimal pairs” experiment. This shows the effect that changing just one or two prepositions in an input sentence has on our Comprehension system. The top row shows the difference that using the prepositions **left**, **right**, or **behind** makes when describing a position relative to a single object. The bottom row shows how varying the preposition pairs of **right / behind**, **right / in front of**, and **left / in front of** changes the object to which a sentential clause refers. The fact that the paths reflect a reasonable interpretation of the input sentences gives evidence that our Comprehension method works as intended.

5.6 Experiments (summary)

We repeated the experiments outlined in in Fig. 4.2 through Fig. 4.4, but with a different data set. Instead of using randomly-generated sentences as input to our Acquisition system, we posted path traces on AMT and obtained human-generated sentences from anonymous workers. We also collected new human-generated sentences to use as input to the Comprehension system. Fig. 5.4 shows examples of input to and output from each of our Acquisition, Generation, and Comprehension systems. This figure is similar in appearance to Fig. 4.11, but the learned lexicon (*i.e.,* the Acquisition output) is obviously different because we used different inputs. Additionally, the sentences shown as input for Acquisition and Comprehension are examples of the human-generated sentences collected via AMT. We also obtained an independent set of AMT worker judgments on how well the sentences and paths matched, both for human- and machine-generated sentences and paths. The results of these judgments are shown in Tables 5.2 through 5.5.



The robot went away from the cone then went ***left*** *of the box which is left of the chair and behind the cone then went towards the stool.*

The robot went away from the cone then went ***right*** *of the box which is left of the chair and behind the cone then went towards the stool.*

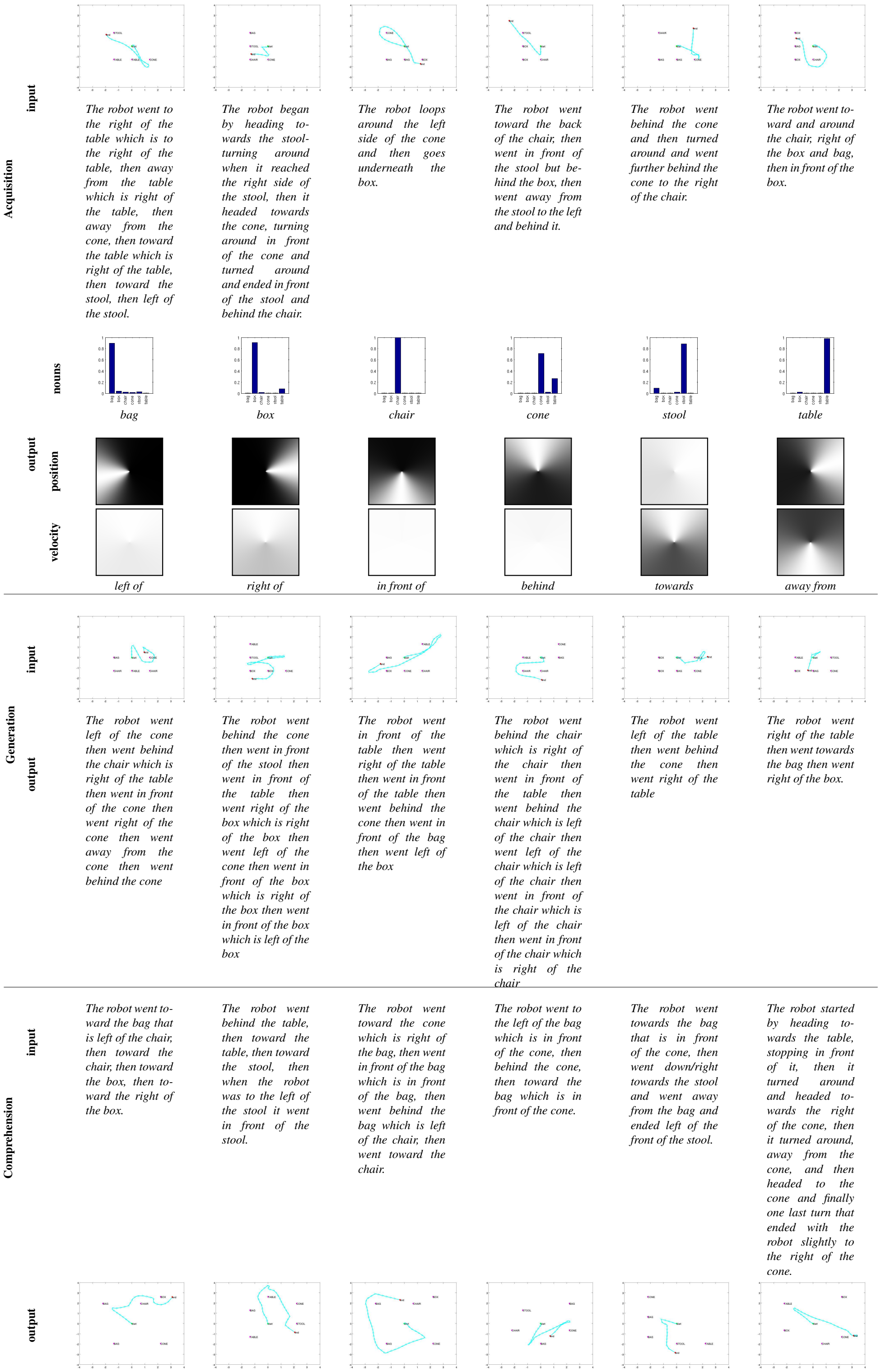
The robot went away from the cone then went ***behind*** *the box which is left of the chair and behind the cone then went towards the stool.*

The robot went away from the cone then went behind the box which is ***right*** *of the chair and which is* ***behind*** *the cone then went towards the stool.*

The robot went away from the cone then went behind the box which is ***right*** *of the chair and* ***in front of*** *the cone then went towards the stool.*

The robot went away from the cone then went behind the box which is ***left*** *of the chair and* ***in front of*** *the cone then went towards the stool.*

Fig. 5.3.: A depiction of the "minimal pairs" experiment. The top row shows how differing path prepositions can change the position of the robot relative to a single object. The bottom row shows how differing spatial relation prepositions can change the object to which a sentential clause refers, thus allowing our method to distinguish between multiple objects of the same type.



The robot went to the right of the table which is to the right of the table, then away from the table which is right of the table, then away from the cone, then toward the table which is right of the table, then toward the stool, then left of the stool.

The robot began by heading towards the stool-turning around when it reached the right side of the stool, then it headed towards the cone, turning around in front of the cone and turned around and ended in front of the stool and behind the chair.

The robot loops around the left side of the cone and then goes underneath the box.

The robot went toward the back of the chair, then went in front of the stool but behind the box, then went away from the stool to the left and behind it.

The robot went behind the cone and then turned around and went further behind the cone to the right of the chair.

The robot went toward and around the chair, right of the box and bag, then in front of the box.

The robot went left of the cone then went behind the chair which is right of the table then went in front of the cone then went right of the cone then went away from the cone then went behind the cone

The robot went behind the cone then went in front of the stool then went in front of the table then went right of the box which is right of the box then went left of the cone then went in front of the box which is right of the box then went in front of the box which is left of the box

The robot went in front of the table then went right of the table then went in front of the table then went behind the cone then went in front of the bag then went left of the box

The robot went behind the chair which is right of the chair then went in front of the table then went behind the chair which is left of the chair then went left of the chair which is left of the chair then went in front of the chair which is left of the chair then went in front of the chair which is right of the chair

The robot went left of the table then went behind the cone then went right of the table

The robot went right of the table then went towards the bag then went right of the box.

The robot went toward the bag that is left of the chair, then toward the chair, then toward the box, then toward the right of the box.

The robot went behind the table, then toward the table, then toward the stool, then when the robot was to the left of the stool it went in front of the stool.

The robot went toward the cone which is right of the bag, then went in front of the bag which is in front of the bag, then went behind the bag which is left of the chair, then went toward the chair.

The robot went to the left of the bag which is in front of the cone, then behind the cone, then toward the bag which is in front of the cone.

The robot went towards the bag that is in front of the cone, then went down/right towards the stool and went away from the bag and ended left of the front of the stool.

The robot started by heading towards the table, stopping in front of it, then it turned around and headed towards the right of the cone, then it turned around, away from the cone, and then headed to the cone and finally one last turn that ended with the robot slightly to the right of the cone.

Fig. 5.4.: Example experimental runs, 6 for each system (Acquisition, Generation, and Comprehension). The output of each system is depicted below the input.

5.6.1 Dataset collection (summary)

We used the human-driven paths from our original experiment in Section 4.6 as the starting point for our new dataset. For the *Acquisition* corpus, we took the 250 manually-driven robot paths and collected 3 sentences for each from AMT workers, for a total of 750 unique path-sentence pairs. The *Generation* corpus remained the same, with 100 manually-driven paths. To create the *Comprehension* corpus, we automatically drove 100 new robot paths (using the ideal word models in Fig. 4.2 and randomly-generated sentences as in Section 4.6), then collected 3 sentences for each from AMT workers, for a total of 300 path-sentence pairs.

The AMT workers were given an image depicting a robot path within a floorplan and were asked to provide a sentence describing that path in relation to the objects in the floorplan. They were told neither the origin of the paths nor the purpose of their sentences, and were given no restrictions on the syntax of the sentence. As such, the sentences are as natural as possible. The sentences contain all the problems one would expect from human-generated data, such as misspellings, ambiguity, grammatical errors, and logical errors (*e.g.,* describing an object's position relative to itself). To quantify the quality of these human-generated sentences relative to our machine-generated data, we conducted an independent round of AMT judgments on both human- and machine-generated data. These judgments are shown in Tables 5.2 through 5.5.

5.6.2 Experimental evaluation (summary)

We conducted experiments similar to those described in Section 4.6. We first learned noun and preposition meanings from the *Acquisition* corpus. However, without the defined grammar of Section 4.4.1, we first had to determine which words to learn. For prepositions, we selected those words which both appear in the Wikipedia prepositions list [101] and appear in our corpus of 1050 human-generated sentences (750 from *Acquisition* and 300 from *Comprehension*) more than 100 times, excluding the prepositions *in*, *on*, and *to*. The relevant prepositions were: *left of*, *right of*, *in front of*, *behind*, *towards*, and *away*. For nouns, we selected words tagged by the Stanford parser [100] as nouns more than 100 times, excluding the word *robot*. The relevant nouns were: *bag*, *box*, *chair*, *cone*, *stool*, *table*.

While using the 750 sentences in the *Acquisition* corpus to learn the noun and preposition meanings, we found a number of extremely lengthy sentences which slow down both the Stanford parser and our learning system. Therefore, we excluded sentences with more than 16 object references, which gave us the set of 600 path-sentence pairs used to learn our lexicon, shown in the top portion of Fig. 5.4. With the word meanings thus learned, we then automatically produced sentences describing the paths in the *Generation* corpus and used the sentences in the *Comprehension* corpus to automatically drive the robot. Examples of such are shown in the middle and bottom portions, respectively, of Fig. 5.4.

We next used AMT to obtain independent judgments as to the degree to which the path and sentence in each pair match. We obtained judgments for human-generated sentences in response to human-driven paths (denoted as *Acquisition (human)* in Tables 5.2 through 5.5), for human-generated sentences in response to machine-driven paths (denoted as *Comprehension (human)*), for machine-driven paths in response to human-generated sentences (denoted as *Comprehension (machine)*) and for machine-generated sentences in response to human-driven paths (denoted as *Generation (machine)*. The first two categories allow us to establish a baseline for the quality of anonymous human responses, against which we can compaire the latter two categories, which give us metrics on our automatic system.

We asked our AMT judges four multiple-choice questions:

1. **Sentence Correctness:** Approximately how much of the sentence is true of the path?
2. **Sentence Completeness:** Approximately how much of the path is described by the sentence?
3. **Path Completeness:** Approximately how much of the sentence is depicted by the path?
4. **Sentence Conciseness:** Rate the length of the sentence.

For questions one through three, the judges chose from one of five percentage-based categories *0–20%*, *20–40%*, *40–60%*, *60–80%*, and *80–100%*. For the fourth question, the choices were *much too short*, *somewhat too short*, *about right*, *somewhat too long*, and *much too long*. The first question evaluates how true a sentence is. The second question evaluates how fully a sentence describes a path. The third question evaluates how completely a driven path fulfills the sequence of actions in a sentence. The final question evaluates the verbosity of our Generation system relative to human-generated sentences.

We obtained judgments from three different AMT workers for each question on each path-sentence pair. All three judgments agreed 26.5% of the time, two judgments agreed 54.1% of the time, while for the remaining 19.4% all three judgments differed.

Tables 5.2 through 5.5 show the distribution of judgments for each question. The rows in the table represent the *Acquisition (human)*, *Comprehension (human)*, *Comprehension (machine)*, and *Generation (machine)* corpora, as described above. For sentence length, human-generated sentences were judged in the middle three categories (*somewhat too short*, *about right*, and *somewhat too long*) in 92.4% of cases, while our machine-generated sentences were judged the same 81.3% of the time. The average judgment for the first three questions on the *Acquisition (human)* and *Comprehension (human)* corpora were 82.4% and 85.3%, respectively. This gives the overall average performance for the human annotators as 83.8%. The average judgment for the first three questions on the *Comprehension (machine)* and *Generation (machine)* corpora were 71.1% and 78.6%, respectively, for an overall machine average of 74.9%. Thus our system can produce sentences and paths that are 89.2% as good as human performance.

These results show that our system is capable of learning the proper meanings of words from noisy, imperfect, and inaccurate human-generated sentences. Our system also achieves a reasonably high level of performance compared to human-generated data.

Table 5.2.: Sentence Correctness responses from AMT judges

	fraction of responses by category				
corpus	*0-20%*	*20-40%*	*40-60%*	*60-80%*	*80-100%*
Acquisition (human)	0.04	0.04	0.11	0.20	0.61
Comprehension (human)	0.01	0.04	0.11	0.21	0.63
Comprehension (machine)	0.12	0.08	0.16	0.20	0.44
Generation (machine)	0.04	0.06	0.17	0.18	0.55

Table 5.3.: Sentence Completeness responses from AMT judges

	fraction of responses by category				
corpus	*0-20%*	*20-40%*	*40-60%*	*60-80%*	*80-100%*
Acquisition (human)	0.03	0.06	0.12	0.25	0.54
Comprehension (human)	0.01	0.04	0.11	0.27	0.57
Comprehension (machine)	0.11	0.07	0.12	0.23	0.47
Generation (machine)	0.04	0.05	0.18	0.26	0.47

Table 5.4.: Path Completeness responses from AMT judges

	fraction of responses by category				
corpus	*0-20%*	*20-40%*	*40-60%*	*60-80%*	*80-100%*
Acquisition (human)	0.03	0.05	0.09	0.19	0.64
Comprehension (human)	0.01	0.04	0.07	0.19	0.69
Comprehension (machine)	0.11	0.08	0.14	0.18	0.49
Generation (machine)	0.03	0.04	0.17	0.21	0.55

Table 5.5.: Sentence Conciseness responses from ATM judges

	fraction of responses by category				
corpus	*much too short*	*somewhat too short*	*about right*	*somewhat too long*	*much too long*
Acquisition (human)	0.06	0.23	0.58	0.11	0.02
Comprehension (human)	0.03	0.17	0.59	0.17	0.04
Comprehension (machine)	0.04	0.15	0.53	0.19	0.09
Generation (machine)	0.07	0.16	0.39	0.27	0.11

5.7 Conclusion

In this chapter we expand on the work in Chapter 4 by removing the fixed grammar and logical form shown in Fig. 4.6 and Fig. 4.7. We instead collect human-generated sentences from AMT workers and use such with Acquisition method to learn noun and preposition meanings in the presence of noise. We then use these meanings to automatically generate sentences from paths, and to automatically drive paths from sentences. An independent round of AMT judgments show that our system achieves 89.2% of human performance. We believe this represents significant progress in using grounded natural language to interact with robots.

6. OBJECT CODETECTION FROM MOBILE ROBOT VIDEO

We present a method for detecting, localizing, and labeling objects encountered by a ground-based mobile robot in its environment. This method does not require any object detectors or models, which allows it to codetect previously unseen objects. Our experimental results show that our method can detect 74.2% of objects presented, localize such with a mean accuracy of 17.7cm, label such with an accuracy of at least 82.6%, and associate images to locations with a mean accuracy of 93.2%.

6.1 Introduction

A well-established computer-vision community studies *codetection*, which is the detection, localization, and labeling of previously unseen objects, in both images (*e.g.,* [28–31]) and video (*e.g.,* [32–35]). We present a new method for codetecting novel objects encountered by a mobile robot in its environment. One crucial aspect of our method is that it does *not* require any pretrained object detectors or models, allowing it to detect, localize, and label objects that have never been seen. Our work differs from prior codetection methods in three important ways.

a) We process a video feed from a ground-based mobile robot. The properties of this video are very different from that of static images or video from a stationary camera. We capitalize on these properties, particularly that we get multiple views of the same objects as the robot moves.

b) We avail ourselves of the odometry and IMU information from the robot, integrating such into the codetection process.

This work was accomplished in collaboration with Daniel Barrett. With minor changes for formatting and length, it was submitted to the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) on 1 March 2016. It is currently under review.

c) The above allow localization of the objects in a 3D world coordinate system, not just the 2D image frame.

We formulate the problem as a graphical model to determine which proposals from a general-purpose object proposal-generation mechanism [102] are most likely to be objects, utilizing similarity measures between proposals [103, 104]. Our method also locates such objects in the world coordinate system and finds consistent object class labels across several different floor plans without human intervention.

We present experiments conducted on a custom-built robot that demonstrate the feasibility of our approach. Our method can detect 74.2% of objects presented, localize such with a mean accuracy of 17.7cm, label such with an accuracy of at least 82.6%, and correctly associate images to locations with a mean accuracy of 93.2%.

6.2 Related Work

To the best of our knowledge, there is no prior work on codetection of objects from the video feed of a mobile robot. Most existing codetection methods (*e.g.,* [28–34]) operate on images or video taken from a stationary point of view, require the object of interest to be prominent and close to the center of the field of view, and localize objects only within the 2D image frame. Conversely, our method operates on video from a mobile robot, which has a moving visual perspective. Our method is also able to both detect small objects that are off-center and localize objects in both the 2D image and 3D world frames.

The codetection approach of Blaschko *et al.* [28] uses weakly-annotated data to detect and localize objects by training SVM classifiers for each object class independently. The training data is split into individual classes, and each image has either a binary indicator of the presence or absence of an object, or a general location of an object. Our approach uses no human annotation and does not require independent training of object classifiers.

Lee and Grauman [29] present an iterative clustering approach to learn object models individually over many passes through an image corpus. This requires that their system start with a small number of pretrained object models, to which it adds models learned in each iteration. Our method, on the other hand, does not require pretrained object models and can classify multiple object classes in a single iteration.

Rubinstein *et al.* [30] discover and segment out common objects from diverse image collections. Their system can automatically localize the common object in images retrieved from Internet search for a specific object, such as a car, or decide that an image does not contain such an object. Like the approach of Blaschko *et al.* [28], their approach requires the manual separation of these image collections by object class; they cannot classify multiple object classes simultaneously.

Tang *et al.* [31] perform codetection by solving a joint image-box formulation for each object class. Like Blaschko *et al.* [28] and Rubinstein *et al.* [30], their system operates on data that has been manually grouped by object class. Our method uses an unsorted collection of images and is able to separate them into object-based groups automatically.

Prest *et al.* [32] use motion segmentation to create spatio-temporal tubes that identify candidate objects for codetection in videos. Like several previously-mentioned approaches (*e.g.,* [28,30,31]), this approach must learn each object individually; it also requires input videos that are grouped by class and annotated to indicate whether or not it contains an object of the class. Our approach does not require such grouping or annotation and learns multiple objects at once.

Schulter *et al.* [33] present a system which uses optical flow to estimate a motion segmentation to discover unknown objects in videos. From this they learn appearance models by clustering both superpixels and bounding boxes. They can thus classify multiple objects simultaneously, although they require pretrained class-specific Hough Forests [105], along with the exact number of classes present in the dataset, to do so.

Joulin *et al.* [34] build upon their previous work [31] by incorporating temporal consistency between the frames of a video. However, like their previous work [31], this approach also requires input data that is manually grouped by object class. Our approach requires no such manual input.

In contrast to most previous codetection work, Srikantha and Gall [35] present an approach which can detect and localize objects which are small and off-center. However, their approach relies upon human pose estimation and thus can only detect objects with which humans interact. Our approach can also detect small and off-center objects, but is independent of human activity within a video.

6.3 Overall Concept

The work presented here is a intended to be a component of a larger system to automatically drive—and learn to drive—a mobile robot under natural-language command [4, 5], presented in this document as Chapters 4 and 5. This method learns the meanings of the nouns and prepositions in sentential descriptions of paths driven by the robot as measured with odometry. However, it requires, as input, a manually-crafted floor plan consisting of a set of 2D object coordinates in the world, each labeled with an object class. Different floor plans can contain objects of the same class at different coordinates. It learns a mapping from nouns, such as *box*, to object class labels, such as OBJECT3. The objective of the work described here is to eliminate the need for manual input by producing the object locations and abstract labels such as OBJECT3 automatically from sensor input.

6.4 Experimental Platform

In order to conduct our experiments, we built a custom mobile robot, or rover, shown in Fig. 6.1. The use of a physical robot with noisy real-world sensor data increases the difficulty of our task. We use the front-facing camera on the rover to collect observations of objects. We use odometry and IMU data as input into an Extended Kalman Filter [91] which enables the rover to localize itself in real time. Such localization is not perfectly accurate due to sensor noise, wheel slippage, and other mechanical factors. These factors create localization error as high as 20cm after long driving paths, but our method is robust enough to handle this.

To collect data, a human teleoperator drives the rover within a floor plan (Fig. 6.2) which is populated by objects. During such operation, the video feed, localization data, and

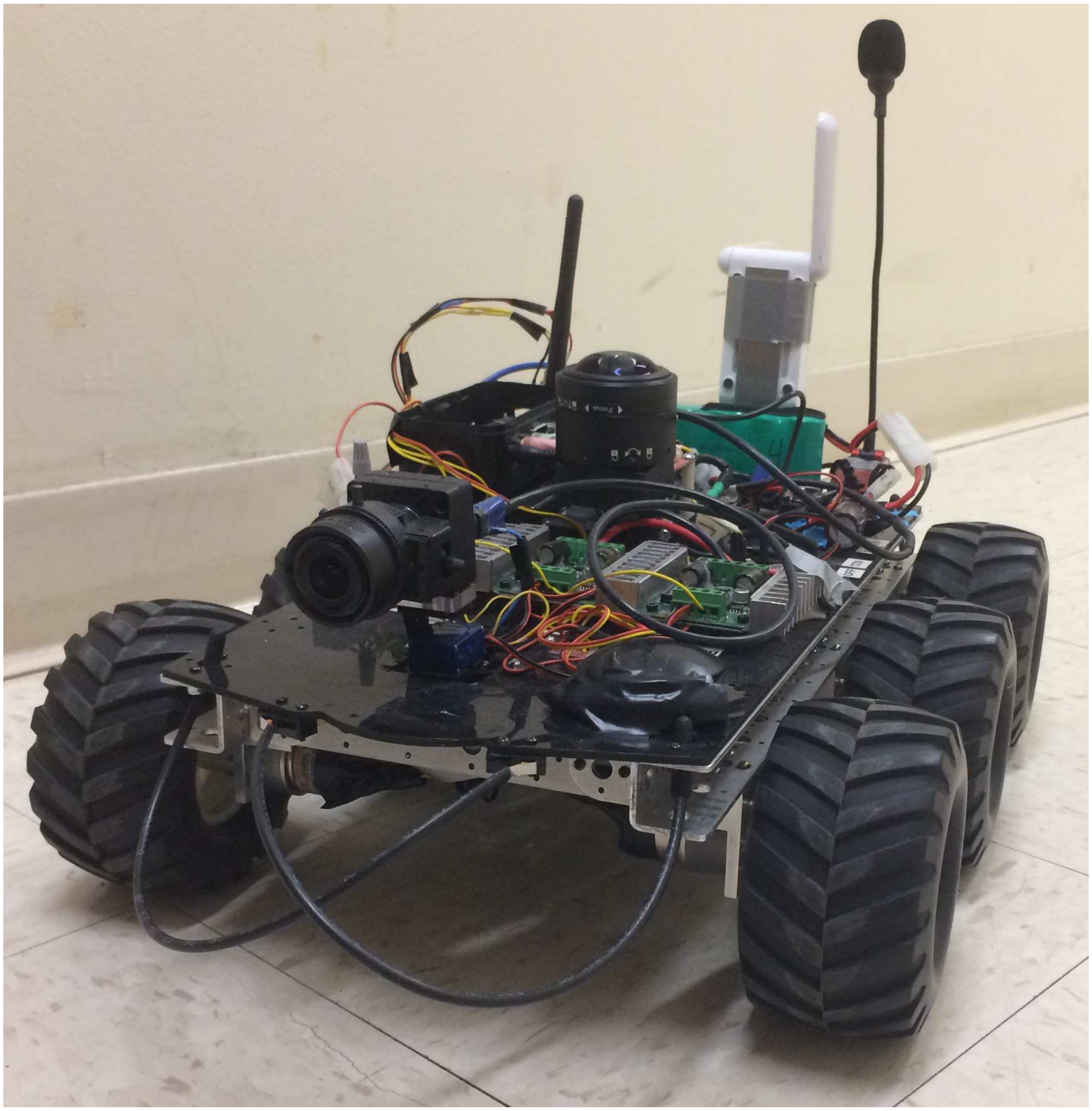


Fig. 6.1.: The custom mobile robot used in our experiments.

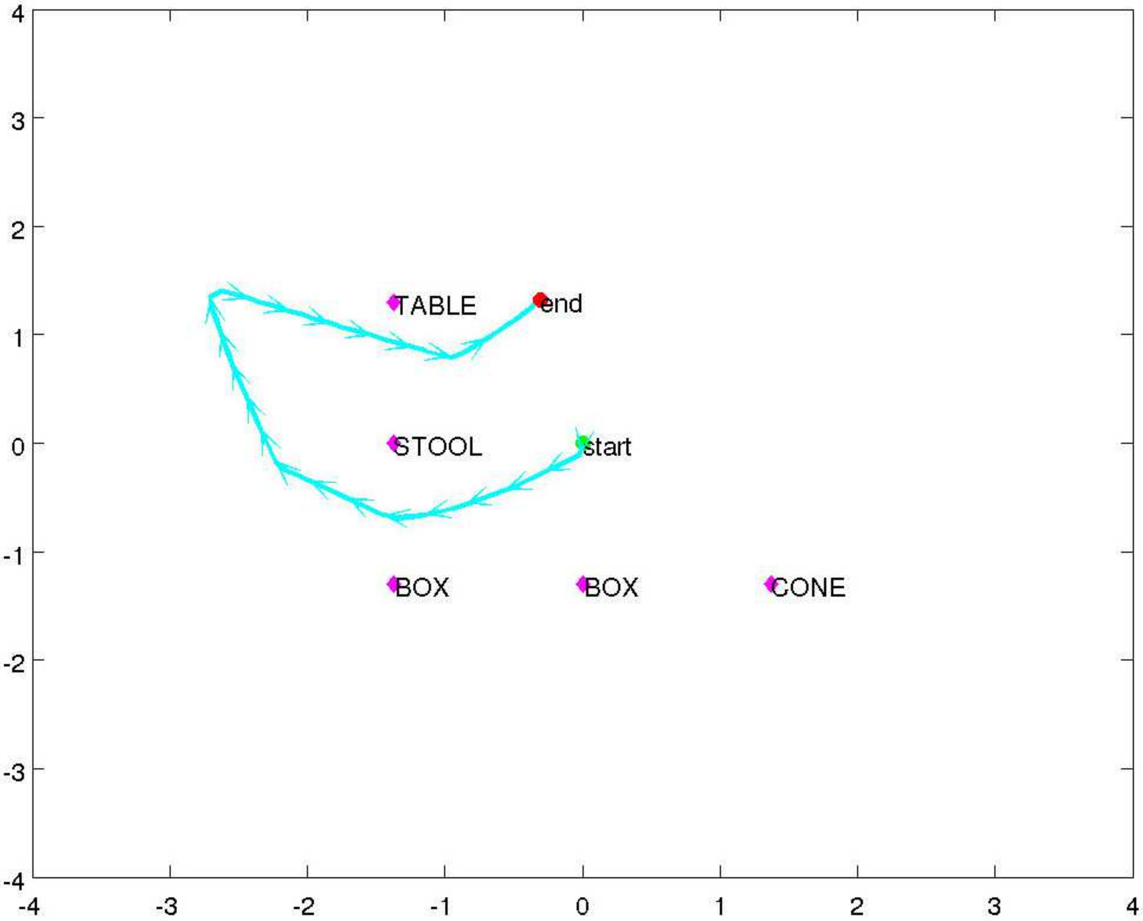


Fig. 6.2.: An example floor plan with a trace of robot motion.

all other sensor and actuator data are logged for later use in finding real-world locations for objects observed in video.

6.5 Technical Details

Our approach to codetection consists of three steps, described in greater detail in the following subsections:

detection and localization Detect candidate objects within video frames and find 3D world locations for such.

clustering Perform clustering on the detection locations to find object locations within the floor plan.

labeling Learn class labels that are consistent within and across different floor plans.

6.5.1 Detecting and localizing objects

Detection and localization of objects is performed by generating a large number of candidate object proposal boxes in each video frame, using projective geometry to locate such in the world, and solving for the most consistent sets of proposal boxes by performing inference on a graphical model. The proposals are generated by applying an off-the-shelf object proposal mechanism [102] to each video frame. This uses general-purpose visual cues such as edges, within-region similarity, and closed contours to place bounding boxes around candidate objects. No class-specific object detectors are used. These proposals therefore support detection of previously unseen objects.

Because the video feed from our robot is time stamped and synchronized with localization data from odometry and the IMU, each video frame is associated with the camera location in the world coordinate frame. This information is used to determine, via projective geometry [106], the world location of each box under the assumption that it rests on the ground. The world location (w_x, w_y, w_z) and world width w_w of an object proposal are thus determined for each box.

However, the proposal-generation mechanism is highly inaccurate; it often produces both false positives and false negatives. To compensate for this, we bias the proposal-generation mechanism to overgenerate, producing ten proposals per frame in attempt to reduce false negatives at the expense of false positives, which are filtered out by performing inference on a graphical model.

For each video, we construct a graphical model with a vertex for each frame that ranges over a set of labels that denote the proposals generated for that frame. Each possible assignment of a vertex to a proposal box has a corresponding unary score which represents the likelihood that the image contained within that box depicts an object. There is also a binary score for each pair of vertex assignments which represents how consistent that pair of assignments is. These binary scores take into account both the image similarity between the two boxes and other geometric information available through knoweledge of the robot's trajectory. Solving this graphical model produces an assignment from vertices to labels which selects a single proposal as depicting the most prominent object in that frame. Because there is not always an object visible to the robot, we augment the potential label set of each vertex to include a dummy proposal that indicates that no object is prominent in the field of view.

Our graphical model optimizes the score

$$\max_{v_1 \in L_1} \cdots \max_{v_T \in L_T} \prod_{i<j} f_{v_i} g_{v_i,v_j} \tag{6.1}$$

where i and j denote frames from a video feed of T frames, v_i denotes the vertex constructed for frame i, and L_i denotes the set of proposals generated for frame i. Also, f_l denotes the unary factor for proposal l, and $g_{k,l}$ denotes the binary factor for a pair of proposals k and l, where k and l are particular proposals. This graphical model is fully-connected, as shown in Fig. 6.3, because we seek to maximize similarity between independent object proposals. The hypothetical optimum for this is a single cluster of proposals in one location. However, in the absence of such, our graphical model will choose as small a number of cliques as possible, each with as many frames as possible, with dummy boxes in those frames that do not fit sufficiently well into any clique. We determine the highest scoring set of vertex assignments through belief propagation [107–109].

The unary factors f are obtained through the proposal-generation mechanism, which associates a proposal score b with each proposal. We find, however, that b is not a reliable indicator. Since we overgenerate proposals, we get many proposals that do not depict valid objects. Thus we use additional problem-specific assumptions and information to filter the proposal boxes.

Because the proposal mechanism treats the borders of the image frame as edges, it tends to give high score to boxes whose borders coincide with the image boundaries; we filter

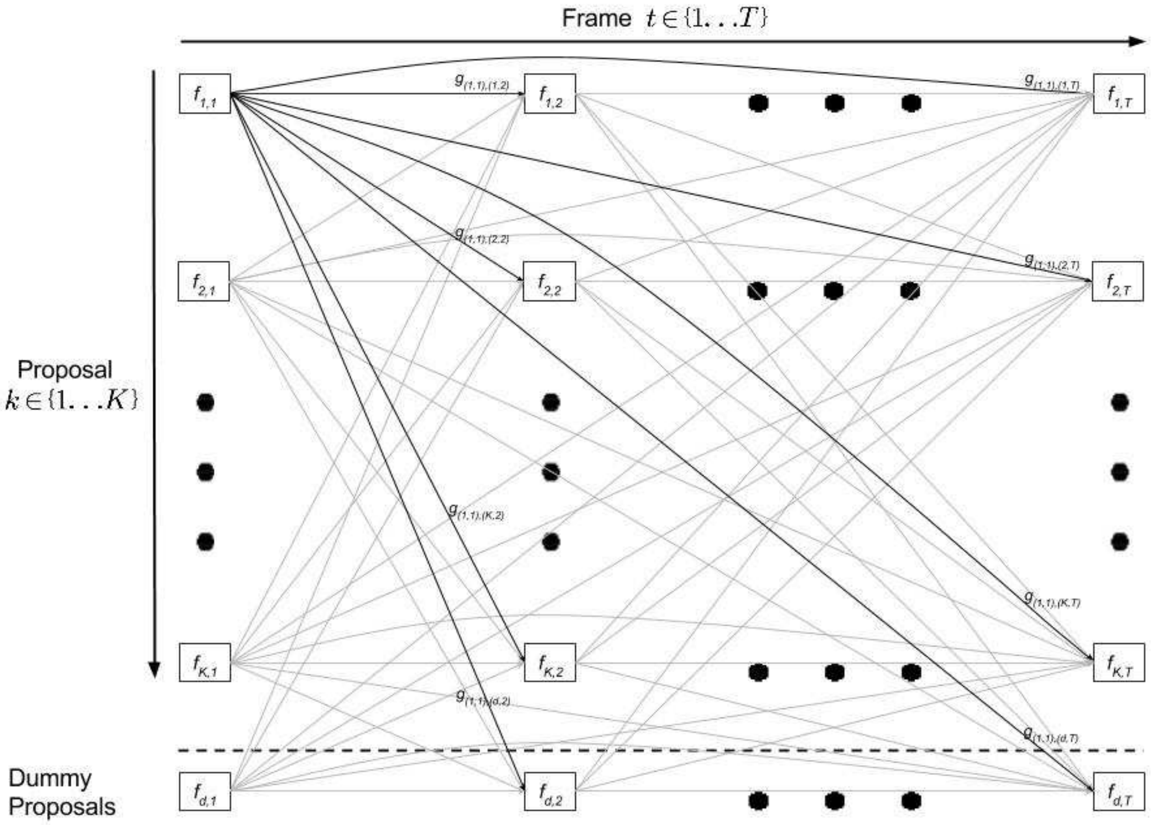


Fig. 6.3.: A visualization of the graphical-model framework for proposal selection. Here $f_{1,1}$ represents the unary factor of proposal 1 in frame 1 and $g_{(1,1),(1,2)}$ represents the binary factor between proposal 1 in frame 1 and proposal 1 in frame 2. For clarity, only the binary factors for proposal 1 in frame 1 are labeled and shown in black. All other binary factors are shown in gray.

such boxes. Boxes whose bottom is above the horizon line of the image cannot be located on the ground, violating an assumption of our problem. Thus we filter such boxes. Due to the size of our robot and our experimental area, we assume that all objects we wish to detect will be less than 2 meters wide, so we filter proposal boxes that are wider than 2m. We finally filter proposals that reside outside of our floor plan boundaries.

The similarity measure between pairs of proposals k and l in different frames used as the binary factors g in the graphical model are a sum of three terms $s_{k,l}$, $d_{k,l}$, and $w_{k,l}$ that denote different aspects of similarity.

$$g_{k,l} = s_{k,l} + d_{k,l} + w_{k,l} \tag{6.2}$$

The first similarity measure, $s_{k,l}$, encodes visual similarity. It is the normalized χ^2 distance between PHOW dense SIFT descriptors [103], as implemented in [104], for the image inside each proposal in each frame. The second similarity measure, $d_{k,l}$, encodes the Euclidean distance between the world coordinates of two proposed objects, reflecting the constraint that an object should have the same position in the world, even when viewed from different viewpoints. The final similarity measure, $w_{k,l}$, encodes the difference in the world width of two proposals, reflecting the constraint that an object should be of similar size when detected from different viewpoints We normalize $d_{k,l}$ and $w_{k,l}$ to [0 1], in order to match them to the scale of the χ^2 distance, by passing them through a zero-mean Gaussian membership function.

Some visualizations of these results are shown in Fig. 6.4. All images are taken from navigational paths driven on the same floor plan. These results show that both false positives (Fig. 6.4 *bottom right*) as well as false negatives (Fig. 6.4 *top right, middle right*) can arise. However, our overall goal is to determine the collection of objects present in each floor plan, their world positions, and a unique labeling. For this, it is not necessary to have correct detections of prominent objects in every frame of the video feed. Subsequent processing is resilient to such false positives and negatives.

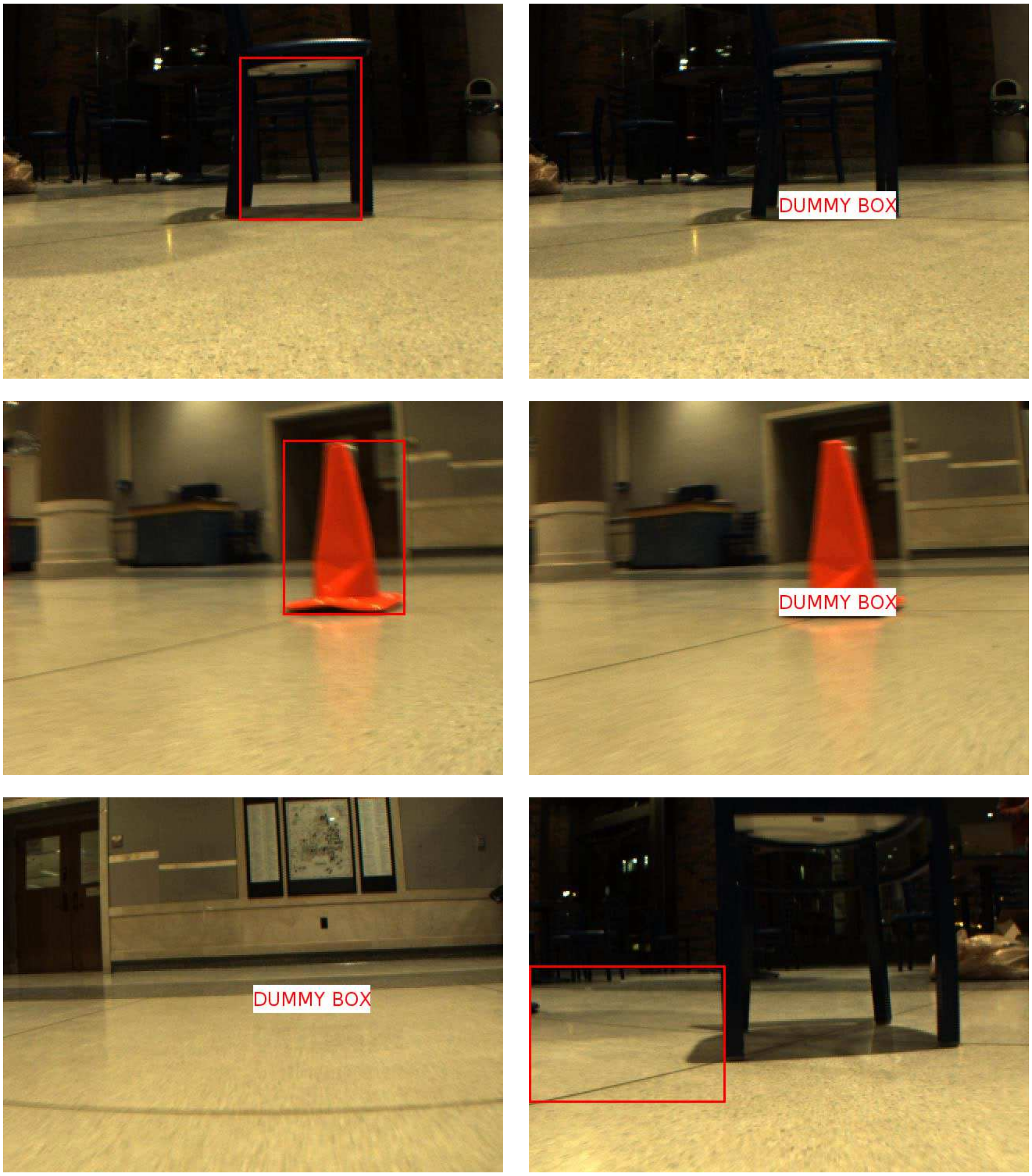


Fig. 6.4.: Visualizations of the solution to the graphical model used to select prominent objects from proposals. The left column shows correct results, while the right column shows failure modes. (*top left, middle left*) Correct detections of a chair and cone. (*bottom left*) Correct selection of the dummy proposal when no object is visible. (*top right, middle right*) False negatives: missed detections of the chair and cone. (*bottom right*) A false positive: spurious detection of what appears to be a shadow.

6.5.2 Clustering detected objects

After using the graphical model to find the most prominent objects in each video frame and localizing such in the world, the next step is to cluster these sets of detections in order to find the object locations in each floor plan. Fig. 6.5 through Fig. 6.10 (*top* of each) show plots of the selected proposals from all ten navigational paths driven in each of the six floor plans. The solid lines show the boundaries of the experimental area and the axes of the world coordinate system. The clustering of selected proposals around actual objects is apparent.

To determine these cluster centers, we assume that the proposals were drawn from a probability distribution with mass centered around the actual world objects. This density is estimated and the peaks in this distribution are taken to be object locations. Examples of such estimated densities can be seen in the center plots of Fig. 6.5 through Fig. 6.10. A Gaussian kernel density estimator, $S_{x,y}$, is used, with $\sigma = 0.25$m and samples weighted by their proposal score f_n and a visibility measure v_n.

$$S_{x,y} = \sum_{n=1}^{N} \frac{f_n}{\sigma v_n} \exp\left(-\frac{\|(x,y)-(x_n,y_n)\|^2}{2\sigma^2}\right) \tag{6.3}$$

$S_{x,y}$ is computed for each point (x, y) in each floor plan, where n ranges over all nondummy selected proposals, (x_n, y_n) denotes the world location of proposal n, f_n denotes the unary factor of proposal n, and v_n denotes a visibility measure of proposal n.

The visibility measure v_n is taken as the number of times the world location (x_n, y_n) was in the camera's field of view. This encodes the idea that when an object truly exists at world location (x, y), it should be detected a high fraction of the time that (x, y) is within the robot's field of view, and eliminates bias in the estimation caused by viewing some regions of the world more often than others.

Fig. 6.5 through Fig. 6.10 (*center* and *bottom* plots of each) show surface and contour plots of (6.3) for the floor plans in Fig. 6.5 through Fig. 6.10 (*top* plot of each). The ground truth object locations are labeled with magenta diamonds in the contour plot. The peaks, found by weighted centroid with a threshold set at two standard deviations above the mean of each $S_{x,y}$, are marked with blue squares. We report the distance error for all floor plans used in our experiment in Section 6.6.

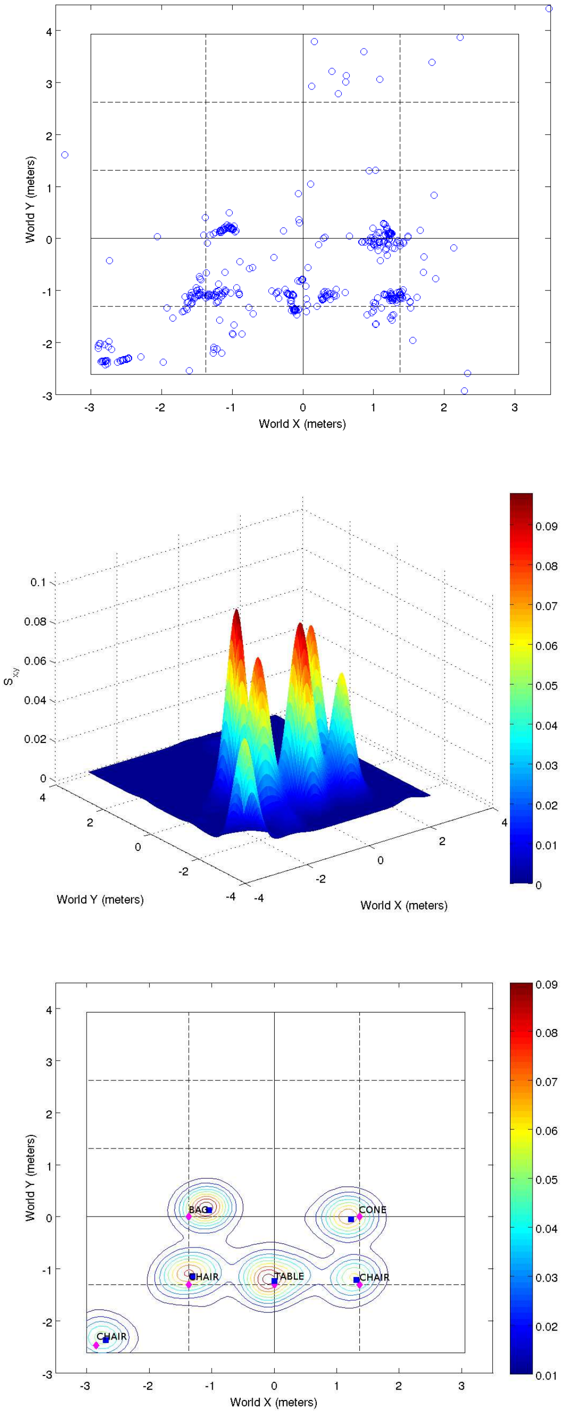


Fig. 6.5.: (*top*) Scatter plot of selected proposal locations for floor plan 1. Surface plot (*center*) and contour plot (*bottom*) of (6.3) for the selected proposals. On the contour plot, blue squares represent detected object locations, while magenta diamonds represent ground truth object locations.

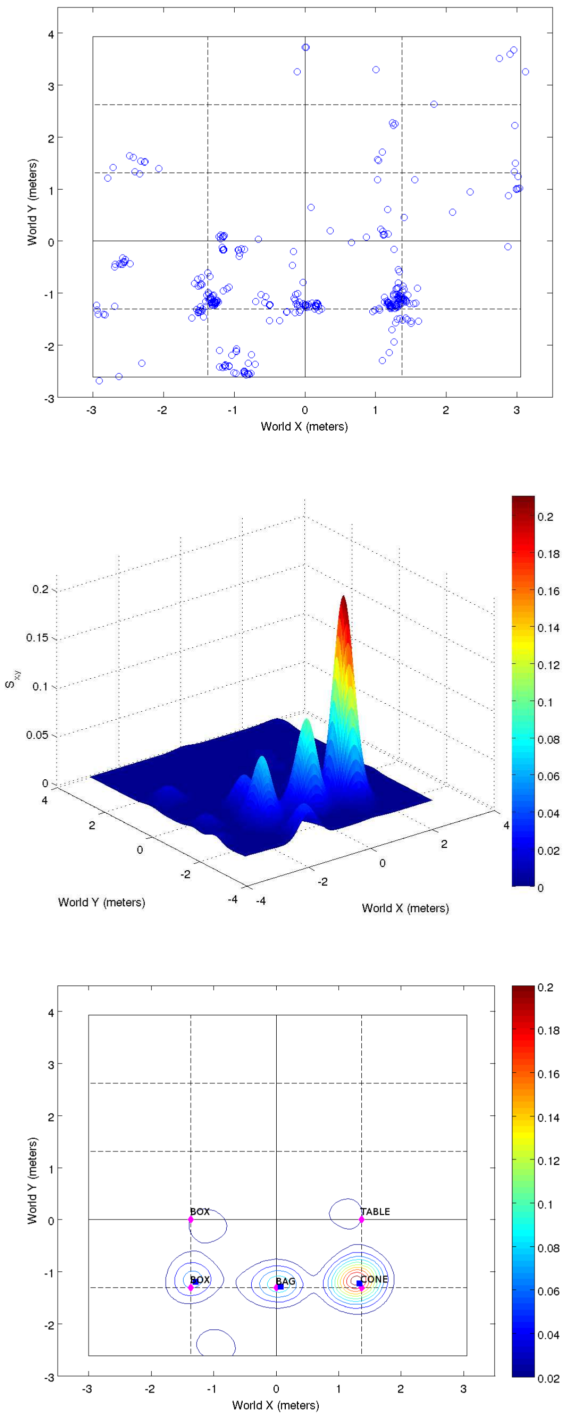


Fig. 6.6.: (*top*) Scatter plot of selected proposal locations for floor plan 2. Surface plot (*center*) and contour plot (*bottom*) of (6.3) for the selected proposals. On the contour plot, blue squares represent detected object locations, while magenta diamonds represent ground truth object locations.

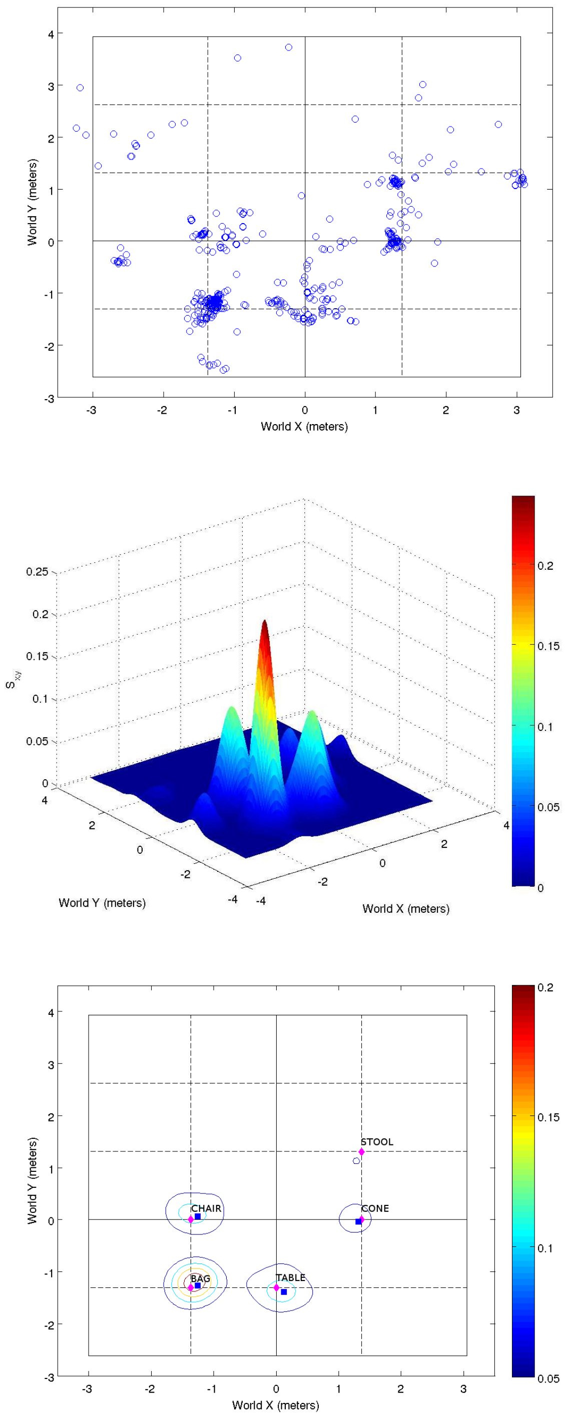


Fig. 6.7.: (*top*) Scatter plot of selected proposal locations for floor plan 3. Surface plot (*center*) and contour plot (*bottom*) of (6.3) for the selected proposals. On the contour plot, blue squares represent detected object locations, while magenta diamonds represent ground truth object locations.

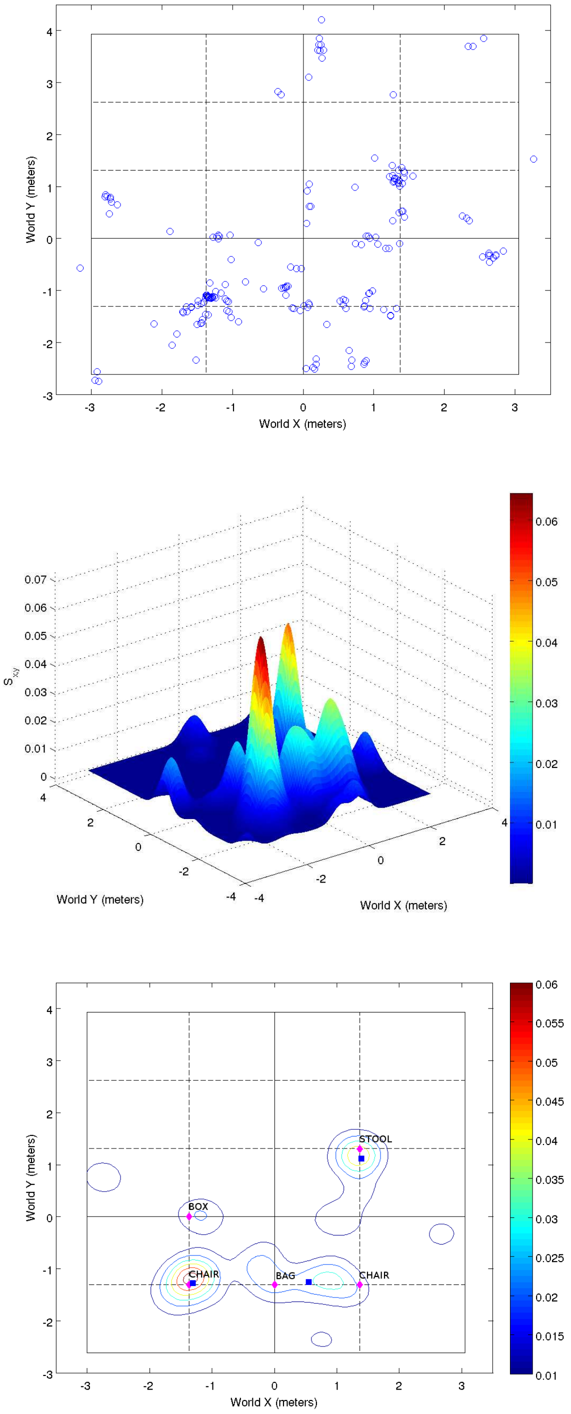


Fig. 6.8.: (*top*) Scatter plot of selected proposal locations for floor plan 4. Surface plot (*center*) and contour plot (*bottom*) of (6.3) for the selected proposals. On the contour plot, blue squares represent detected object locations, while magenta diamonds represent ground truth object locations.

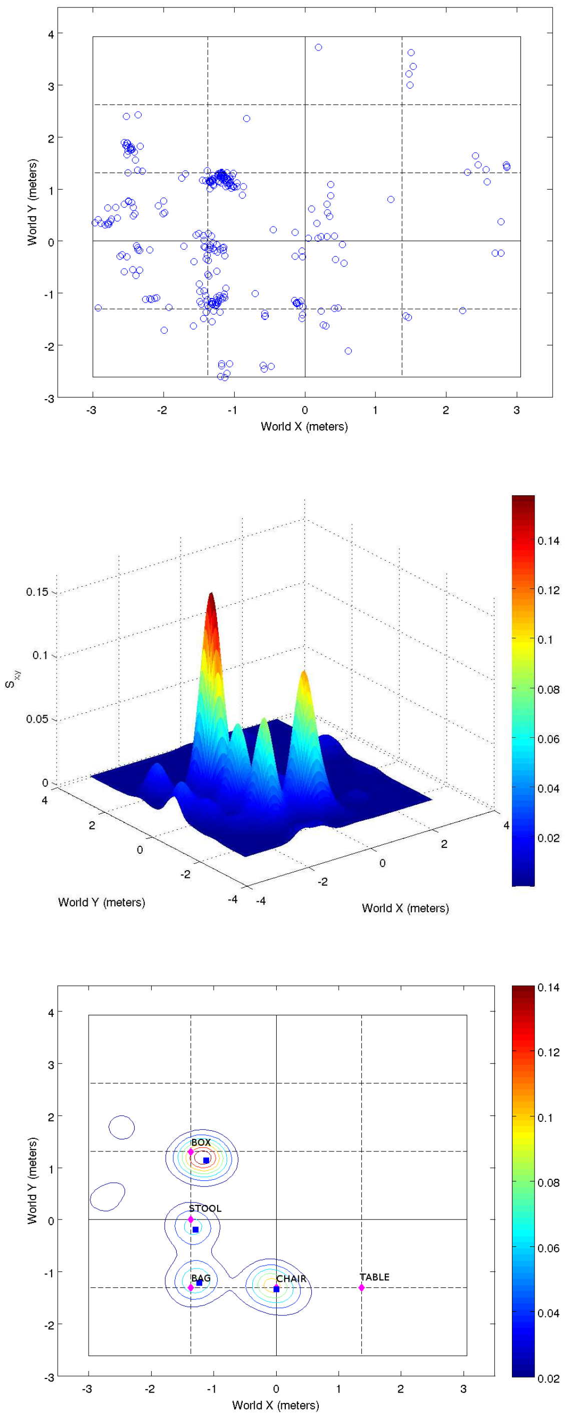


Fig. 6.9.: (*top*) Scatter plot of selected proposal locations for floor plan 5. Surface plot (*center*) and contour plot (*bottom*) of (6.3) for the selected proposals. On the contour plot, blue squares represent detected object locations, while magenta diamonds represent ground truth object locations.

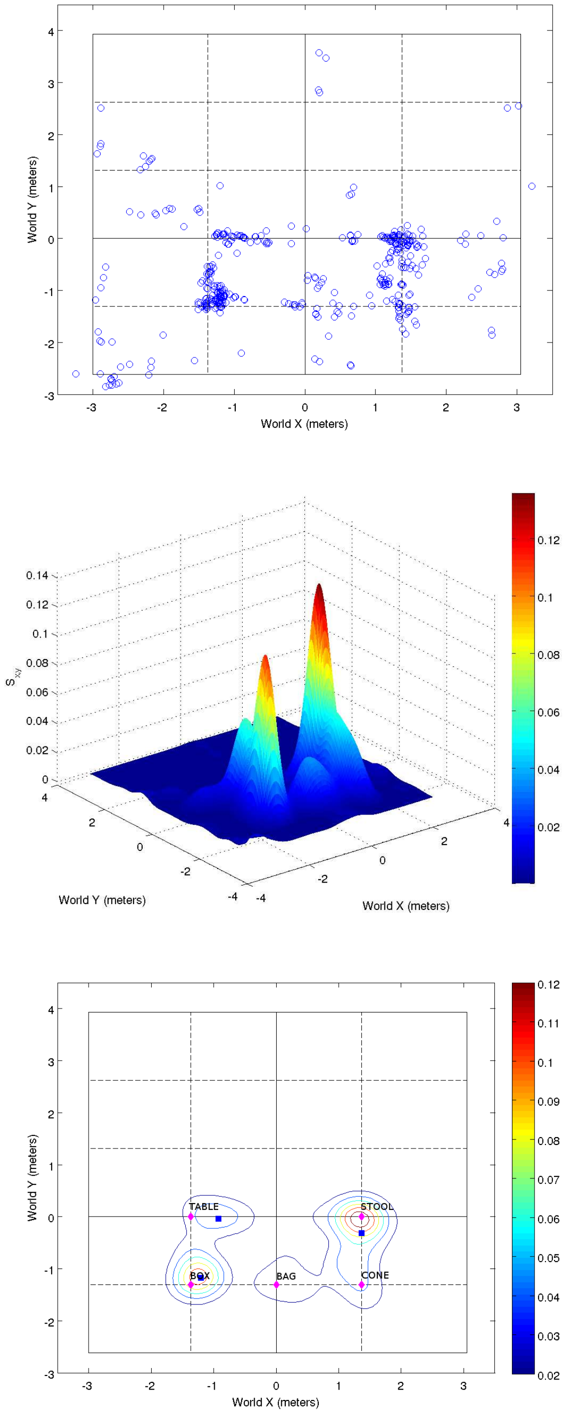


Fig. 6.10.: (*top*) Scatter plot of selected proposal locations for floor plan 6. Surface plot (*center*) and contour plot (*bottom*) of (6.3) for the selected proposals. On the contour plot, blue squares represent detected object locations, while magenta diamonds represent ground truth object locations.

6.5.3 Labeling object classes

The object locations must next be labeled in a consistent fashion. These labels are indended to be used as input to the method of [4, 5] (Chapters 4 and 5), which learns an assignment from abstract object labels to nouns. That method can handle a many-to-one mapping of labels to nouns. As such, while it is important that the labels produced are consistent, and do not assign the same abstract labels to world objects of differing class, it is not critical that a single abstract label be used to represent each class of objects. For example, it is perfectly acceptable for two abstract labels to be used to represent different kinds of bags, but would be problematic for a single label to be used to represent both bags and tables.

To assign class labels to each detected object location, we first assign each selected proposal box and its corresponding image region to the closest object location (peak) determined in the previous step, rejecting outliers based on a distance threshold of 50cm. Then we create a similarity matrix Q between pairs p_1, p_2 of object peaks detected in all floor plans. If the method has detected P object peaks, each with a set C_P of associated image regions, let $U_{a,b}$ denote the visual similarity between pairs a, b of image regions where a is associated with peak p_1 and b is associated with peak p_2. Visual similarity is measured by the same methods as s in (6.2). However, we take the mean of only the top 50% of similarity scores in an effort to suppress noise from incorrectly-associated images. We compute Q as

$$Q_{p_1,p_2} = \frac{\sum\limits_{a \in C_{p_1}} \max\limits_{b \in C_{p_2}} U_{a,b} + \sum\limits_{b \in C_{p_2}} \max\limits_{a \in C_{p_1}} U_{a,b}}{|C_{p_1}| + |C_{p_2}|} \tag{6.4}$$

We then formulate a second graphical model with a vertex for each of the P object peak locations to compute a common labeling across all floor plans. The vertex variables can range over the set of abstract class labels. Since abstract class labels are interchangeable, there are no unary factors in this graphical model. The binary factors represent visual similarity between the sets of images assigned to each object location.

Let $\ell(p)$ represent the abstract class label selected for object $p \in \mathbb{P} = \{1, \ldots, P\}$. We then seek the set of labels that maximizes the sum of all t scores:

$$\max_{\ell} \sum_{\substack{p_1,p_2 \in \mathbb{P} \\ p_1 \neq p_2}} t(p_1, p_2) \tag{6.5}$$

We assign a factor $t(p_1, p_2)$ for each pair of $p_1, p_2 \in \mathbb{P}$, $p_1 \neq p_2$, computed as follows:

$$t(p_1, p_2) = \begin{cases} -log(Q_{p_1,p_2}) & \text{if } \ell(p_1) = \ell(p_2) \\ -log(1 - Q_{p_1,p_2}) & \text{if } \ell(p_1) \neq \ell(p_2) \end{cases} \tag{6.6}$$

Belief propagation fails to solve this graphical model, so we use branch and bound [110] instead.

6.6 Experimental Results

We conducted an experiment on six different floor plans to test our method, utilizing a total of six object classes: **BAG**, **BOX**, **CHAIR**, **CONE**, **STOOL**, and **TABLE**. Fig. 6.5 through Fig. 6.10 (*bottom* of each) depict each of the six floor plans with the ground truth objects marked with magenta diamonds and labeled with an object class. We drove ten paths in each floor plan, allowing us to collect video and odometry data for 60 paths.

Table 6.1 shows both the number of objects detected and the location error of such detections. Our dataset contains a total of 31 object instances, of which 23 are detected. This yields a detection rate of 74.2%, with no false positives.

The right side of Table 6.1 gives statistics on localization error, defined as the distance between ground truth and detected object location. The overall mean localization error is 17.7cm with a standard deviation of 12.9cm. Note that while ground truth locations are taken as object center projected to the ground plane, all objects have a nonzero physical footprint. The localization error is within this footprint, which ranges from 21cm by 26cm for the **BOX** to 46cm by 46cm for the **TABLE**.

Table 6.1.: Object detection and localization results.

	number of objects		location error (cm)			
floor plan	present	detected	min	max	mean	std dev
1	6	6	7.3	34.7	17.3	9.6
2	5	3	7.1	13.5	9.8	3.3
3	5	4	6.0	15.1	11.5	3.9
4	5	3	7.3	55.4	27.3	25.1
5	5	4	3.2	29.5	17.4	11.0
6	5	3	21.1	45.2	32.4	12.2
overall	**31**	**23**	**3.2**	**55.4**	**17.7**	**12.9**

Table 6.2.: Labeling accuracy by object class.

	object instances		number labeled (6 abstract labels)						number labeled (9 abstract labels)								
class	ground truth	detected	1	2	3	4	5	6	1	2	3	4	5	6	7	8	9
BAG	6	5	4		**1**				4	1							
BOX	5	3		3							2	1					
CHAIR	7	6	**2**		3	**1**				**1**			3	2			
CONE	4	3				3									3		
STOOL	4	3					3									3	
TABLE	5	3						3									3

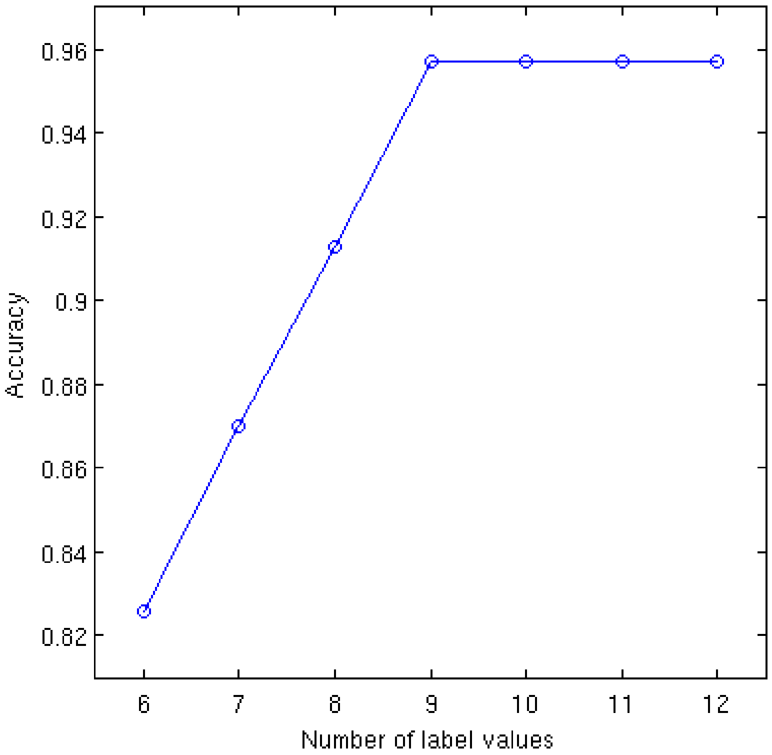


Fig. 6.11.: Labeling accuracy versus number of label values. We define an error in labeling as a single abstract class label value assigned to more than one ground truth object class.

We evaluate object class labeling (Section 6.5.3), using varying numbers of abstract class labels. We test with the minimum number of abstract class labels needed to distinguish all ground-truth classes, six for our dataset, through twice the minimum number of labels required, twelve for our dataset. This increasing number of labels allows for variability appearance of the objects and also models the case when the number of object classes is not known *a priori*. Such a labeling is still a valid input to the method of [4,5], (Chapters 4 and 5) which functions properly with a small number of labels per class.

Fig. 6.11 shows the accuracy of the labeling with varying numbers of label values, and Table 6.2 shows the labeling accuracy by object class for two of these trials. Labels are considered in error when more than one real object is assigned to the same abstract label, *i.e.,* a label column in Table 6.2 (*middle*) or (*right*) has entries in more than one row. We highlight these in bold. Accuracy is computed by dividing the total number of correctly-classified object instances by the total number of instances. The left section of Table 6.2 compares detected object instances to ground truth object instances. The middle section shows the number of detected instances assigned to each abstract class label with the minimum number of labels, shown as the numbers 1 through 6. The right section of Table 6.2 shows the same data for the test with nine abstract class labels, when the method achieves its best performance. We observe that in Table 6.2 (*middle*), there are four occurrences of abstract class labels being assigned to object instances of more than one class, which yields a labeling accuracy of 82.6%. In Table 6.2 (*right*), there is only one such occurrence, which yields a labeling accuracy of 95.7%.

We evaluate the images that our method produces in order to judge how accurately it associates object images to their detected locations. We define an incorrectly associated image as one that does not have any recognizable part of the appropriate object within the image. The image association is 93.2% accurate overall, varying from 85.4% on the **CHAIR** objects to 98.0% on the **CONE** objects. The accuracy for each object class is shown at the top of Fig. 6.12 and Fig. 6.13.

Fig. 6.12 and Fig. 6.13 show the images contained within three example boxes images for each of the six object classes, as grouped by the object class labeling test with nine class labels. Examples of images associated to the correct ground-truth object class are shown above the double line, while images of incorrectly associated images are shown below the double line. These figures clearly illustrates the effect of appearance variability on object class labeling. Those objects that demonstrate significant variability in appearance (Fig. 6.12) require multiple abstract classes, while those that have a consistent appearance (Fig. 6.13) can be modeled with one.

6.7 Conclusions and Future Work

We demonstrate a method for object codetection on a video feed from a mobile robot camera. It detects, localizes, and labels previously unseen objects. This approach differs

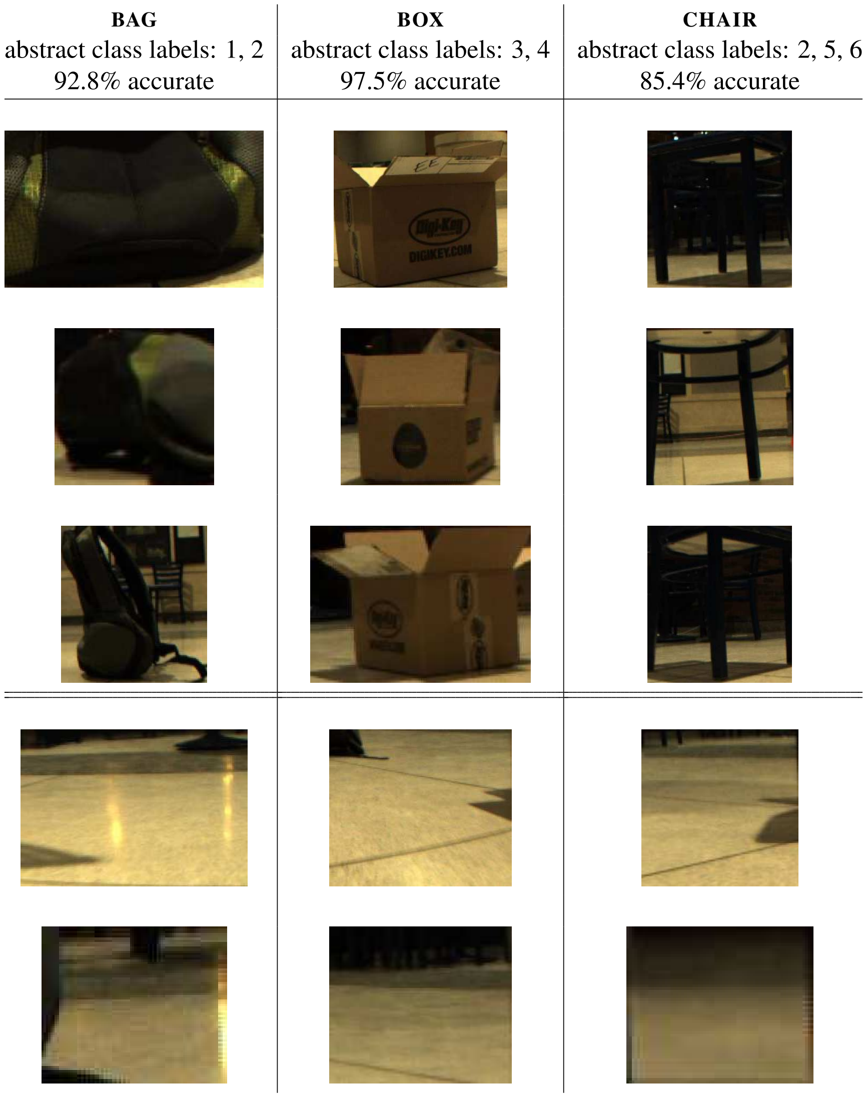

BAG
abstract class labels: 1, 2
92.8% accurate
BOX
abstract class labels: 3, 4
97.5% accurate
CHAIR
abstract class labels: 2, 5, 6
85.4% accurate
Digi-Key
DIGIKEY.COM

Fig. 6.12.: Examples of object images as grouped by our labeling method using nine abstract class labels (Table 6.2 *right*) for the **BAG**, **BOX**, and **CHAIR** classes. Image association accuracy by class is shown at the top of each column. Correct image-to-object-class assignments are shown above the double line, while incorrect assignments are shown below. The ground-truth object class names have been included at the top of the columns for clarity. Note that the images shown represent only the contents of proposal boxes (shown in red in Fig. 6.4). These object classes exhibit high appearance variability and thus each require multiple abstract class labels.

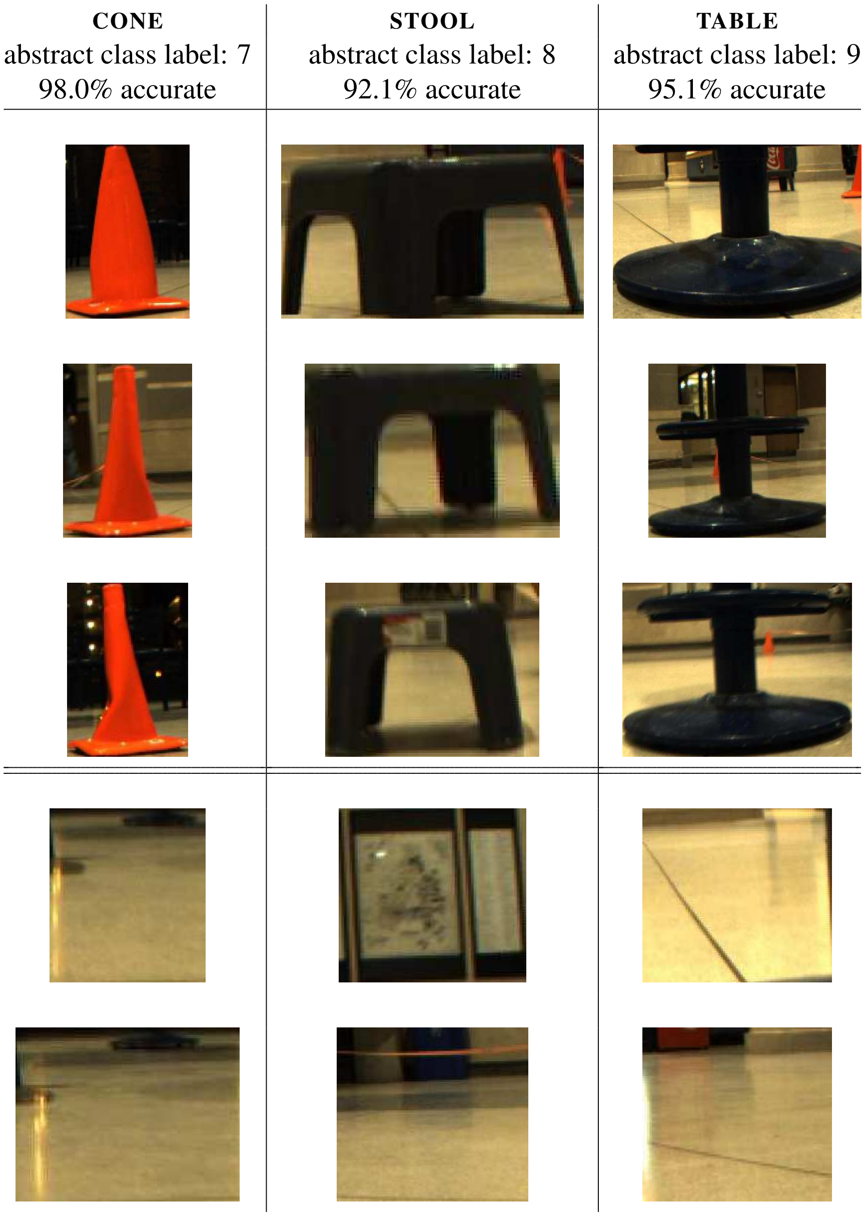


Fig. 6.13.: Examples of object images as grouped by our labeling method using nine abstract class labels (Table 6.2 *right*) for the **CONE**, **STOOL**, and **TABLE** classes. Image association accuracy by class is shown at the top of each column. Correct image-to-object-class assignments are shown above the double line, while incorrect assignments are shown below. The ground-truth object class names have been included at the top of the columns for clarity. Note that the images shown represent only the contents of proposal boxes (shown in red in Fig. 6.4). These object classes exhibit low appearance variability and thus need only a single abstract class label each.

from prior approaches to codetection in static images and video from a stationary camera in that it effectively combines egocentric robot video with odometry and IMU data to localize novel objects in the 3D world coordinate frame.

We formulate the problem as a graphical model to determine which proposals from a general-purpose object proposal-generation mechanism are most likely to be objects, incorporating several similarity measures between proposals. Our method can automatically determine which proposals are most likely to be objects, find 3D world locations for such objects, and consistently label them with abstract class labels unique to a single physical object class. The method has been shown to accurately solve this problem in our experiments. Future work includes integration of this method with that in [4, 5] (Chapters 4 and 5) to enable combined object localization and language learning.

6.8 Acknowledgments

This research was sponsored, in part, by the Army Research Laboratory, accomplished under Cooperative Agreement Number W911NF-10-2-0060, and by the National Science Foundation under Grant No. 1522954-IIS. The views, opinions, findings, conclusions, and recommendations contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Army Research Laboratory, the National Science Foundation, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

7. TOWARDS SENTENCE-BASED CODETECTION OF OBJECTS FROM MOBILE ROBOT VIDEO

7.1 Introduction

This chapter represents the final increment of work that I have been able to accomplish during my research program. While I had originally planned to combine the work in Chapters 4 and 5 with that in Chapter 6, further reflection led me away from that plan. I realized that doing so would not be very novel. I would simply be repeating previous experiments with a new data set. Therefore, I decided to change directions a bit.

The work here, while admittedly not as fully developed as it could be, integrates the natural language concepts explored in Chapters 4 and 5 with computer vision methods similar to those in Chapter 6. Rather than a simple combination of those systems, I instead develop a new method that first uses computer vision techniques to identify potential objects within **VADER**'s environment and then incorporates language to identify the objects described relative to both the robot's path and each other in sentences.

I conduct a proof-of-concept experiment which shows that the method has promise, but could still use some refinement.

7.2 Concept

The concept of my work here is inspired by the conclusions in [111], namely that sentential descriptions of video can be highly informative when labeling objects. However, that work uses video from a stationary camera and tracks objects that move within the frame. As such, it can only localize objects within the 2D image coordinate system. My robot data has not only video, but location data. This allows me to localize objects in the 3D world coordinate system as well as in the 2D image system. This location data, along with sentential descriptions of the driven paths relative to the objects of interest, allows me to combine vision and language to find and label objects.

7.3 Technical Details

My method is based on the use of a corpus of manually-driven robot paths and their corresponding videos, paired with one or more sentences describing each path in terms of the objects in the environment. It operates in four steps. First, I use an off-the-shelf object proposal generator [112] to produce boxes around possible objects in a selection of frames from each video. Next I use a tracking algorithm [113] to create from each proposal a spatio-temporal tube of boxes which covers a series of temporally adjacent frames. Then I use projective geometry [106], in conjunction with the odometry and IMU data that describe the manually-driven paths, to generate a real-world location for each box in each tube. From this I compute statistics on each tube which I then use to filter out unreasonable tubes; the tubes that remain form my set of candidate objects. Finally, I incorporate the sentential data by formulating and solving a graphical model in which the vertices are the nouns (objects) in the sentences and the labels are the tubes representing the candidate objects. The output of this graphical model is a single tube for each object which represents the most likely location for that object. Each step is described in more detail below.

7.3.1 Proposal Generation

The first step in the object codetection is to start with object proposals. To do this, I use Edge Boxes [112], a state-of-the-art algorithm which uses a count of the edges contained within a candidate box—itself derived from a sliding window approach—to determine a score which indicates the likelihood that the box contains an object. This method is computationally efficient, which allows me to over-generate proposals. The Edge Boxes algorithm's fast run time is the reason why I selected this proposal generation mechanism over the MCG algorithm [102] I used in Chapter 6. Early experiments using MCG proved it infeasible for the large number of proposals that this method requires.

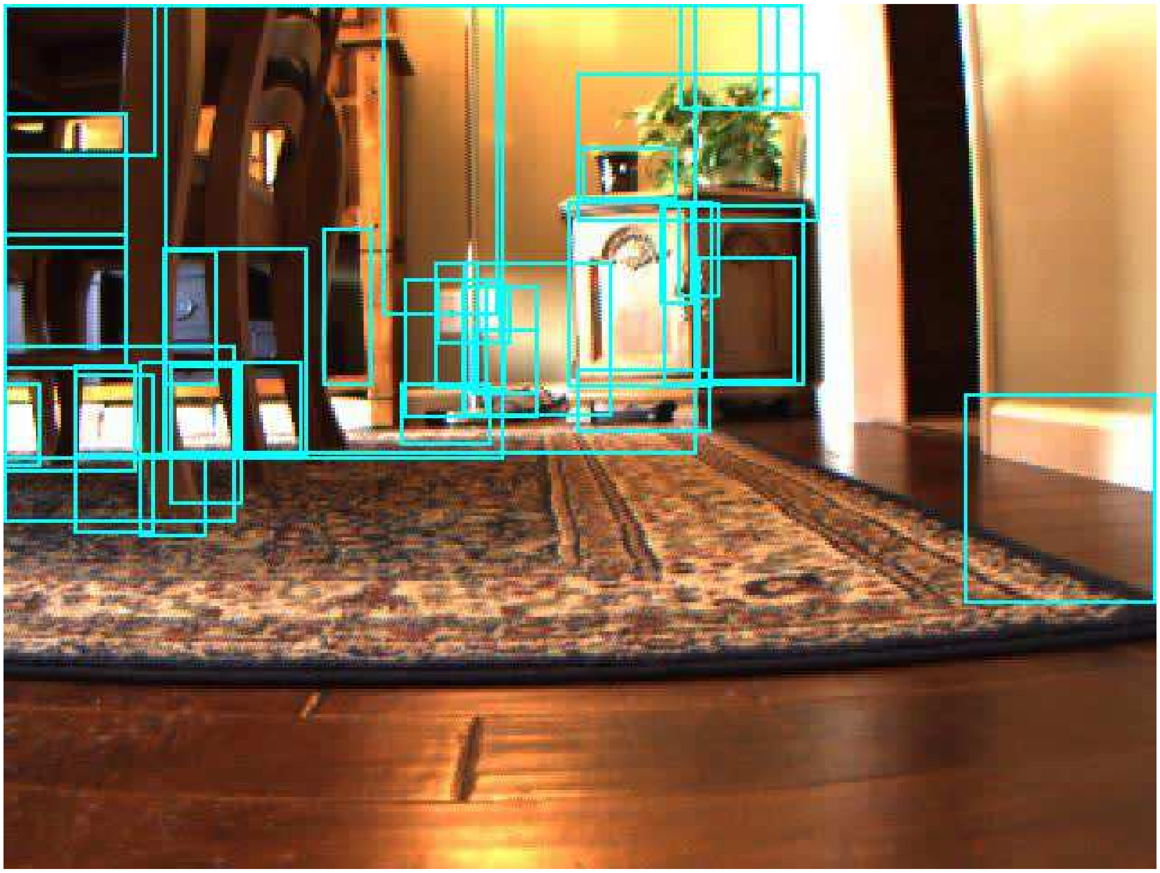


Fig. 7.1.: A frame with proposals for $K = 50$.

For each video, I first down sample every L^{th} frame. Given a video of T total frames, this yields $M = {}^{T}\!/_{L}$ frames. Then in each of those M frames I generate $10 \times K$ proposals, where K is a positive integer representing the final number of proposals I wish to have in each frame. I over-generate by a factor of 10 in order to run Non-Maximal Suppression (NMS) on the proposals. I run NMS using the intersection over union (IoU) metric, with the IoU threshold[1] set to 0.5. I then take the top K proposals from this sorted list. These proposal boxes all have a pairwise IoU score less than 0.5, which increases the probability that every object in view is contained in a proposal box. Fig. 7.1 shows an example frame from my data set rendered with its proposals for $K = 50$.

[1] I determined this IoU threshold empirically.

7.3.2 Tube Generation

The next step in my method is to turn the proposal boxes from the previous step into spatio-temporal tubes. These tubes should each ideally track a given object through all the frames in which the object appears. To do this, I use the Median Flow algorithm [113], as implemented in OpenCV [88].

My tube generation method takes as input a proposal box in a frame of a given video of a manually-driven path. I give this video, starting frame, and box to Median Flow, which then tracks that box through temporally adjacent frames. It is important to note that for starting frames which are not the first frame of the video—which is the majority of them—the algorithm runs both forwards and backwards in time within the video. This ensures that if a particular starting frame happens to occur in the middle of a tube, the frames which precede the starting frame are also considered for inclusion in the tube. In practice, each tube only exists in a small fraction of the frames of any video; this makes sense because the video feed from a moving robot will have objects entering and leaving the field of view throughout the run.

The output of the above is a tube for each of the K proposal boxes in each of the M starting frames. Since these tubes were generated independently from different starting frames, it is likely that there is significant overlap between pairs of tubes. This would unnecessarily complicate subsequent steps of my method, so I perform NMS on the tubes. However, since the tubes are not just spatial, but also temporal, I make some modifications to the standard IoU metric.

My method for tube NMS takes advantage of the fact that all pairwise combinations of tubes to be compared will have the same number of frames (including frames which do not contain a box) because the tubes have been generated from the same video. To apply NMS to $K \times M$ spatio-temporal tubes in each video, I keep running totals of both the intersection and union of the boxes in each pair of matching frames. There are three possible cases for the frame-by-frame comparison of a pair of tubes. If neither frame has a valid box, then nothing is added to either the intersection or union totals. If only one frame has a valid box, the intersection total obviously remains unchanged, and the area of the lone box is added to the union total. Only if both frames contain a valid box do I compute the intersection of those boxes and add it to the intersection total. Similarly, in this case I compute the union of the two boxes in the usual fashion and add it to the union total. Once I have compared each frame pair by this method, I simply divide the intersection total by the union total for a tube IoU score. As previously, I set the IoU threshold[2] at 0.5. After this round of NMS, each video has $N \leq K \times M$ tubes.

7.3.3 Candidate Object Tube Selection

At this point I have a set of tubes for each video in which the IoU for each pair of tubes is less than 0.5. However, so far I have only analyzed the image data; I still know nothing about the locations of these potential objects. Therefore, at this point I incorporate the odometry and IMU data collected from each run. All data from the robot is logged with a time stamp, so I am able to associate each frame of each video with the world location of the camera at that instant. This enables me to use projective geometry [106], in a manner similar to the method I describe in Section 6.5.1, to find a world location (*i.e.,* an (x, y) coordinate) and world width and height for each box in each tube. This method also makes the assumption, as does the method in Section 6.5.1, that the objects of interest rest on the ground plane. For a robot whose front-facing camera is approximately 20 centimeters off the ground, this is a reasonable assumption.

Using the computed world location of each box in each tube, I next compile some tube-level information which I use to filter out unreasonable tubes. I first compute the mean (x, y) location for the boxes in each tube. Then I measure the distance from the mean for each box in each tube. This serves as a measure of the variance of the location of a particular tube. I use this as a metric for how consistent a mean tube location is. Tubes with all boxes close to the mean represent give a reliable estimate of the location of the object depicted in that tube. Tubes which have boxes that lie far from the mean give an uncertain estimate

[2] As with the proposal IoU threshold, I also set the tube IoU threshold empirically.

for the tube's location; such tubes are not useful in my method. Therefore, I filter such tubes out. I also filter tubes based on number of frames in the tube, since a tube with fewer frames could have an artificially low location variance. For this filter I set a hard limit that tubes must contain more than 5 frames. Finally, I also filter tubes whose world location does not lie on the ground plane, since this violates one of the assumptions of my method.

The tubes which remain after this filtering protocol represent the tubes which depict candidate objects. These tubes are members of the set of possible labels for the vertices of the graphical model used in the next step of my method.

All steps of the method described up to this point have been applied to each run, video, or tube individually. At this point, I aggregate all tubes from all runs in a given room or floor plan into a single set. I do so because the set of objects with in a room or floor plan is constant. In the following step I seek to find the single tube which best fits the sentence semantics for each object, therefore I must consider all object candidates, regardless of the individual run from which each came. Indeed, once I have determined world location data for each tube, one can consider all tubes to exist within the same world domain.

7.3.4 Graphical Model Formulation and Solution

In order to select a single tube which represents each object, I formulate a graphical model which operates over all runs driven on a particular floor plan. This graphical model utilizes the sentential information contained in human-generated descriptions of the driven paths. Fig. 7.2 shows an example of a driven path with human-generated sentences which describe it in the caption. The nouns in these sentences (with the exception of *robot*) are the vertices of the graphical model. In order to test the concepts of my method without having to determine if there exist multiple instances of an object class, all objects are identified uniquely. Object classes with multiple instances in a given floor plan (*i.e., laundry-basket* in Fig. 7.2) have a unique integer appended to the end of the object name. These noun vertices are automatically extracted from the sentential descriptions of the paths.

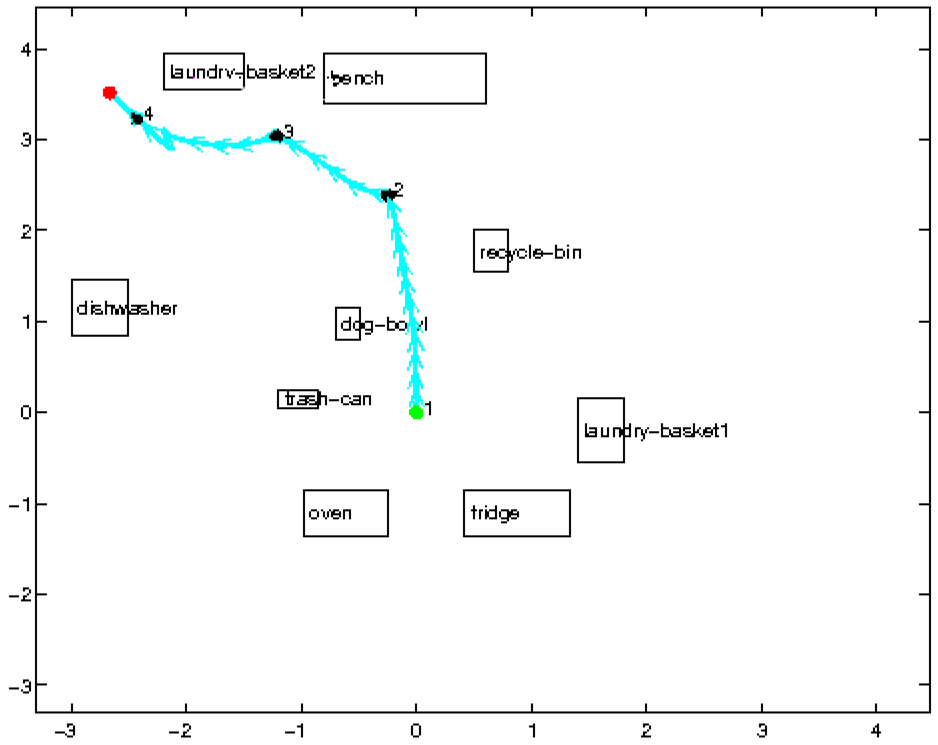


Fig. 7.2.: An example of a robot path in a typical household kitchen floor plan. The labeled rectangles in the floor plan represent the ground-truth locations of the objects. The human-generated sentences describing it are:

- *The robot went towards the bench, then went towards the laundry-basket2 which is left-of the bench, then went in-front-of the laundry-basket2 which is behind the dishwasher, then went left-of the laundry-basket2 which is left-of the bench.*
- *The robot went right-of the trash-can and left-of the laundry-basket1 which is behind the fridge, then went right-of the dog-bowl, then went left-of the recycle-bin, then went behind the dog-bowl, then went in-front-of the laundry-basket2 which is left-of the bench, then went left-of the laundry-basket2 which is behind the dishwasher.*
- *The robot went behind the fridge and behind the oven, then went in-front-of the bench, then went in-front-of the laundry-basket2 which is behind the dishwasher, then went behind the dishwasher and left-of the laundry-basket2 which is left-of the bench.*

The set of labels which can be assigned to these vertices is the set of all tubes for a given floor plan, as determined in the previous step. To assign a unary score to each tube, I use the information contained in the prepositions which describe the robot's path relative to the objects. I use von Mises distributions [42] to represent both the position and velocity information which a preposition encodes, in a manner similar to that in Chapter 4. The particular distributions I use for each preposition are the hand-designed models depicted in Fig. 4.2, but this is simply a matter of convenience; I could easily change to the learned models depicted in Fig. 4.11.

Each instance of a noun in a sentence is associated with a preposition and a segment of the robot path for which that prepositional phrase is true. In Fig. 7.2, the numbered waypoints correspond to the four clauses in the first descriptive sentence in the caption. To score each tube against each noun instance, I use the function for the preposition attached to the noun instance. I evaluate the preposition function for the noun and each point in the path segment, which gives me a score in $[0, 1]$. I then take the mean value for the entire path segment. For a noun class with N instances in the corpus, I denote each mean noun instance preposition score as p_n for $n \in \{1, \ldots, N\}$. To find the overall score for a tube relative to a noun, I multiply each mean noun instance preposition score for the tube; one can think of this as a logical AND operation. I also use a normalized tube location variance v, as a discount factor; tubes with lower variance have a score closer to 1. Note that v is a property of the tube and is independent of the particular noun instance against which the tube is evaluated. Let there exist a data corpus with C noun classes and D candidate object tubes. For noun class $c \in \mathbb{C} = \{1, \ldots, C\}$ and candidate object tube $d \in \mathbb{D} = \{1, \ldots, D\}$, I compute the unary score $f_{c,d}$ as

$$f_{c,d} = v_d \prod_{n=1}^{N} p_{c,d,n} \tag{7.1}$$

This score thus represents how well a tube satisfies all of the sentential clauses that describe a noun class relative to the robot path.

I draw binary scores for my graphical model from sentential clauses which describe nouns relative to each other, *e.g., the chair which is left of the hutch.* Such clauses are automatically extracted from the parsed sentences. I again use the preposition functions to generate a binary score for each pairwise combinations of tubes between vertices which have a prepositional relationship. This computation is simpler than the one used for the unary scores; since tubes have a fixed location, the preposition functions only need to be evaluated once for each pair of tubes. For noun classes $c_1, c_2 \in \mathbb{C}, c_1 \neq c_2$ and candidate object tubes $d_1, d_2 \in \mathbb{D}$, the binary score is

$$g_{(c_1,d_1),(c_2,d_2)} = v_{d_1} v_{d_2} p_{(c_1,c_2),d_1,d_2} \tag{7.2}$$

where $p_{(c_1,c_2)}$ is the preposition function which represents the sentential relationship between noun classes c_1 and c_2. Note that the prepositional relationships between noun classes are directional; $p_{(c_1,c_2)}$ is never the same as (and usually the opposite of) $p_{(c_2,c_1)}$. Because a tube can only represent one object class, the binary score between a tube and itself in two different object classes, *i.e.,* $g_{(c_1,d_1),(c_2,d_1)}$, is always zero.

Sentential descriptions of driven paths will rarely contain descriptions of all nouns relative to all other nouns. Therefore, the binary scores in this graphical model are sparse. Some noun classes may have many binary relationships while others may have none.

Fig. 7.3 shows a visualization of the graphical model used in this method. The columns in the figure represent the noun classes and the rows denote the candidate object tubes. Every noun class vertex draws its label from the same pool of candidate tubes, so the tubes in each column are the same. However, since each candidate tube is evaluated relative to the particular instances of each noun class, the unary scores $f_{c,d}$ are different for varying values of c while d is held constant. I illustrate the binary scores in different colors for clarity; blue shows a relationship between noun classes 1 and C, red between 2 and 3, and green between 3 and 1. If, for example, noun class 1 was *chair*, 2 was *table*, 3 was *lamp*, and C was *hutch*, then the depicted binary scores could represent such relationships as *the*

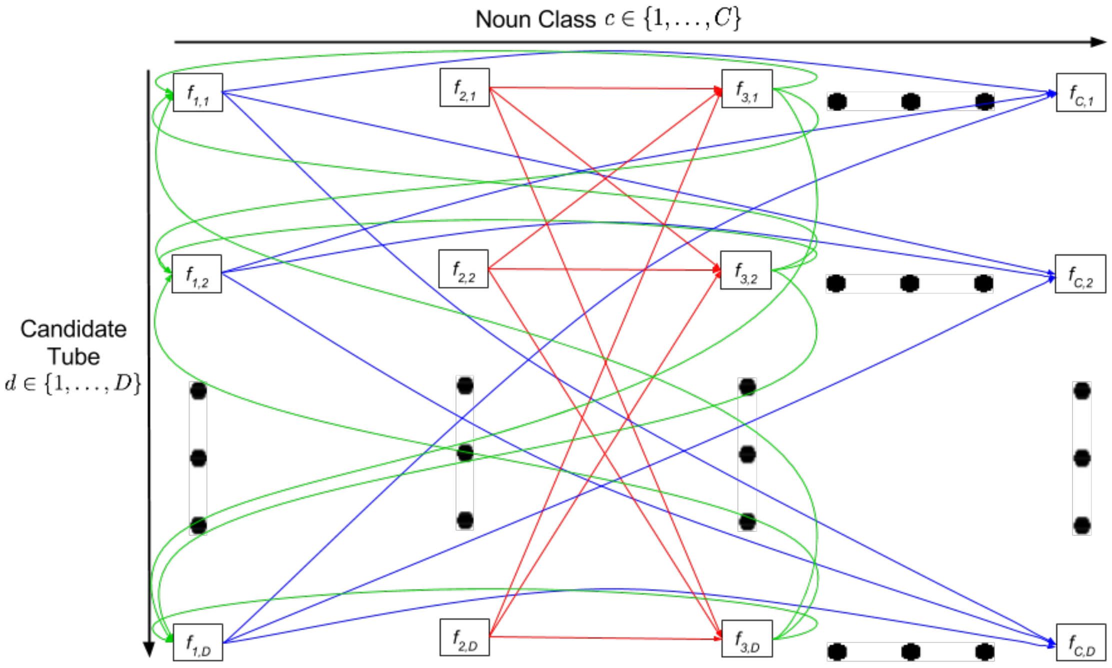


Fig. 7.3.: A visualization of the graphical model used in this method. Note that the binary scores in this model are both sparse and directional. Let noun classes $1, 2, 3$, and C represent *chair, table, lamp,* and *hutch,* respectively. The blue binary scores encode a relationship such as *the chair which is left of the hutch*; the red, *the table which is in front of the lamp*; and the green, *the lamp which is behind the chair*.

chair which is left of the hutch, *the table which is in front of the lamp*, and *the lamp which is behind the chair*.

With my graphical model built as described above, I apply belief propagation [107,109] to find the highest scoring set of assignments of candidate object tubes (labels) to noun classes (vertices).

7.4 Initial Experiments and Results

To test my method, I conducted a small proof-of-concept experiment. I would have preferred to conduct a more rigorous experiment, but my constrained time line did not allow for such. For my proof-of-concept experiment, I chose to use an environment that resembled a real-world situation a robot might encounter, since one complaint against the work in Chapter 6 has been that the floor plans used in that experiment were too synthetic. The environment I selected was a residential house, namely, my own. I used three rooms of my house, the dining room, the kitchen, and the living room. The objects in those rooms are predominantly ones that occur naturally in such rooms, although I did add extra objects from elsewhere in my house to increase the overall number of objects present. I placed between 8 and 13 objects in each room floor plan. I used a total of 25 different object classes in the rooms, with some of the classes appearing in multiple rooms. Figures 7.4 through 7.6 (*top*) show the three floor plans used in my experiment. The labeled rectangles represent the objects in their measured ground truth locations.

To collect data, I manually drove 25 paths in each floor plan, for a total of 75 paths. I then plotted the recovered path, as in Fig. 7.2, to verify that the odometry and IMU data from each run depicted the actual path (compared against the front-facing and panoramic video) with reasonable accuracy; all did. I next used the plots and video to write three sentences describing the driven paths relative to the objects in the room. The sentences for one example run are shown in the caption of Fig. 7.2. In a more thorough experiment I would have outsourced this sentence-writing task to independent parties, such as the workers on Amazon Mechanical Turk (AMT). However, since this is a proof-of-concept experiment and the ability to process free-form natural language is not a goal of my method, it was easier to do this myself.

With three sentences for each run, the next task was to align each phrase to a path segment and then parse these path-aligned phrases into a format suitable for automated processing. Since these tasks are ancillary to the focus of my method, I did them by hand for each run. Had I had more time available, I would have used AMT workers or an automated system based on Baum-Welch [95–97] to do this.

Once this data pre-processing was complete, I ran my method, as described in the previous section, on each of the three floor plans independently. I set the parameters for $K = 50$ proposals every $L = 10$ frames. After the first three steps of my method, each floor plan had a pool of candidate object tubes that numbered between approximately 6,000 and 8,000.

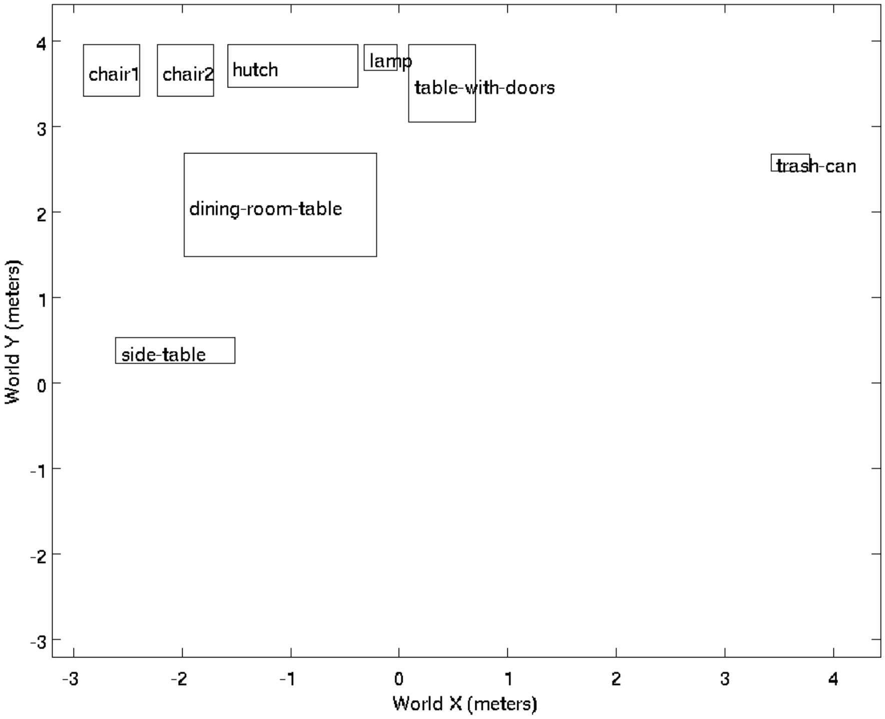
Table 7.1.: Statistics for selected tube locations. A selected tube is considered to be located correctly if its location is within 10 cm of measured ground truth.

	number of objects			location error (in meters) for incorrectly located objects			
floor plan	total	correctly located	incorrectly located	min	max	mean	std dev
dining room	8	6	2	1.90	9.31	5.61	5.24
kitchen	9	6	3	0.32	3.29	1.53	1.56
living room	13	11	2	0.51	0.80	0.66	0.21
total	30	23	7	0.32	9.31	2.45	3.19

The overall run time for my method varied from 4.5 to 6.5 hours on a six-core Intel Xeon workstation. The number of noun classes (vertices) in a floor plan was the controlling factor for run time, as expected.

Figures 7.4 through 7.6 (*bottom*) show the output of my method, with the selected tube location for each noun depicted with a labeled magenta diamond. Table 7.1 shows statistics on the location accuracy of my method. Figures 7.7 through 7.9 show example images from the tubes assigned to each noun class. The selected tubes are denoted by blue boxes within the images.

For the judgment of correct or incorrect location of the selected tubes relative to ground truth in Table 7.1, I consider any location within 10 cm of the ground truth boundaries of an object to be correct. This accounts for possible inaccuracies in my measurement of ground truth. Thus in the dining room floor plan (Fig. 7.4), the locations for *hutch* and *side-table* are correct because they are within 10 cm of ground truth, while *lamp* and *trash-can* are incorrect. As Table 7.1 shows, my method locates tubes correctly for 23 out of 30 instances, or 76.67% of the time.



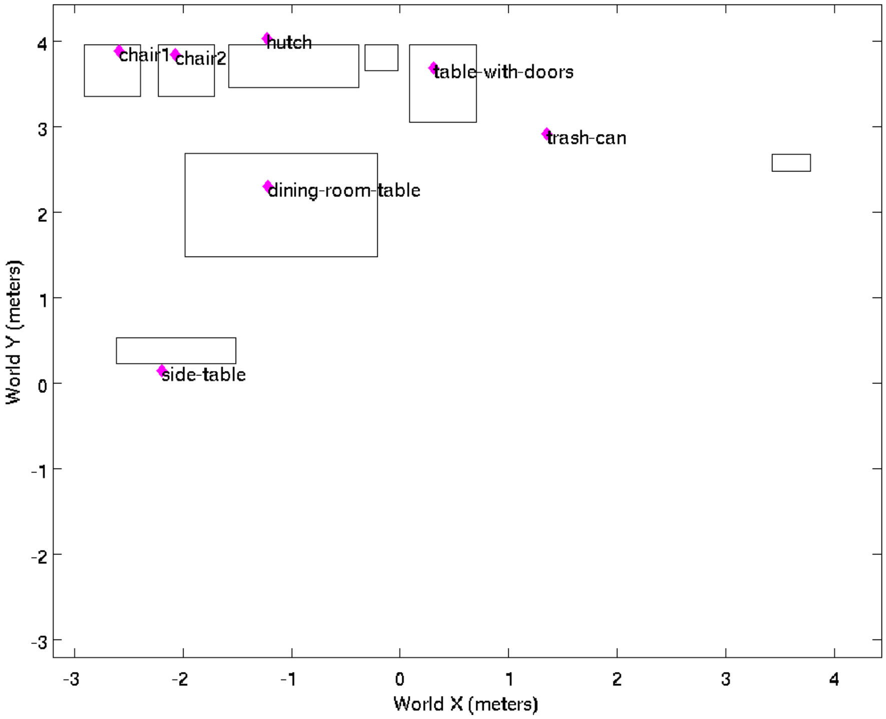


Fig. 7.4.: (*top*) The dining room floor plan shown with labeled ground truth object locations. (*bottom*) The same floor plan with detected object locations shown as magenta diamonds. Unlabeled ground truth rectangles are shown for reference. Note that the detected *lamp* is not shown in the bottom plot because it is located at $(0.71, 13.24)$.

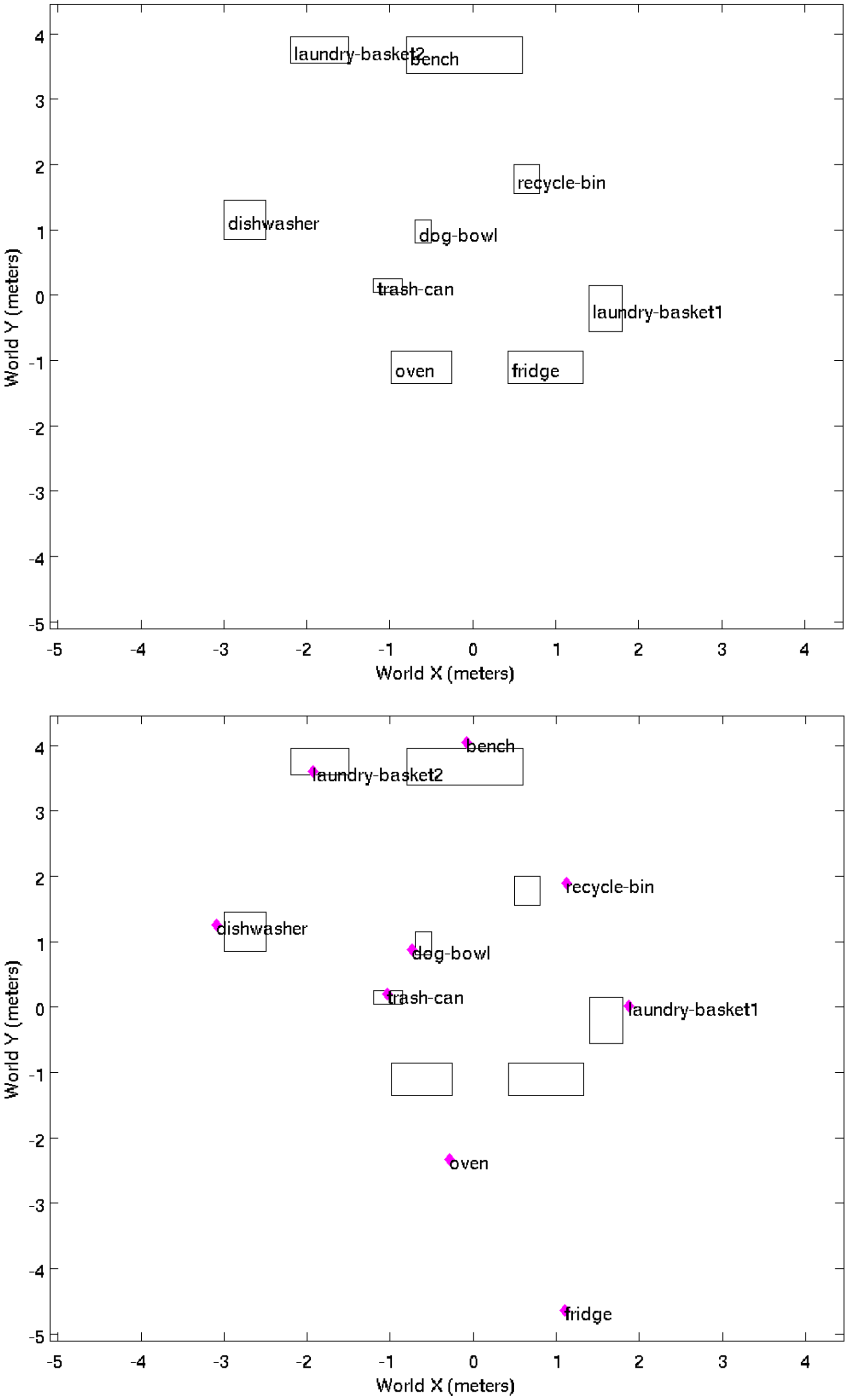


Fig. 7.5.: (*top*) The kitchen floor plan shown with labeled ground truth object locations. (*bottom*) The same floor plan with detected object locations shown as magenta diamonds. Unlabeled ground truth rectangles are shown for reference.

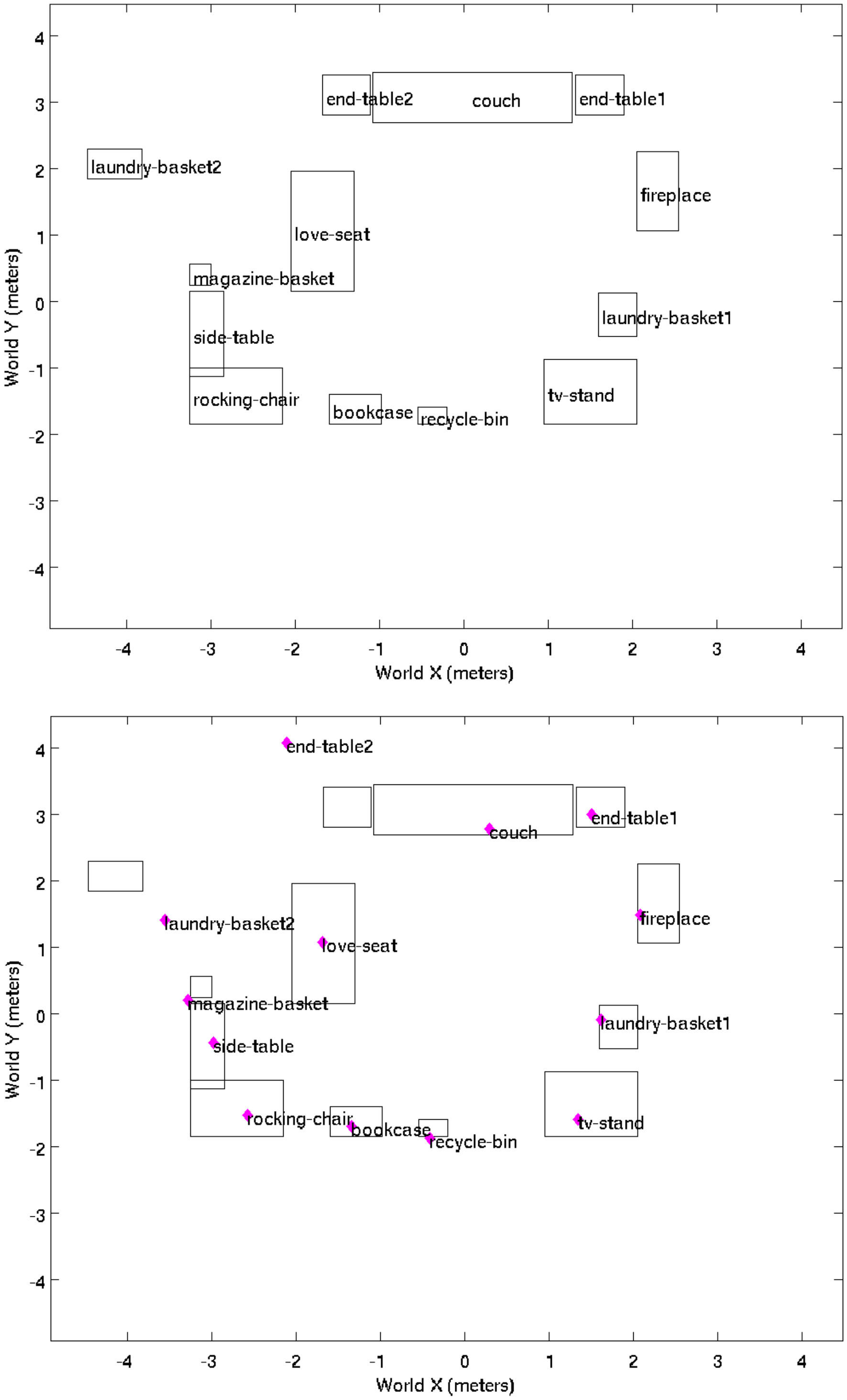
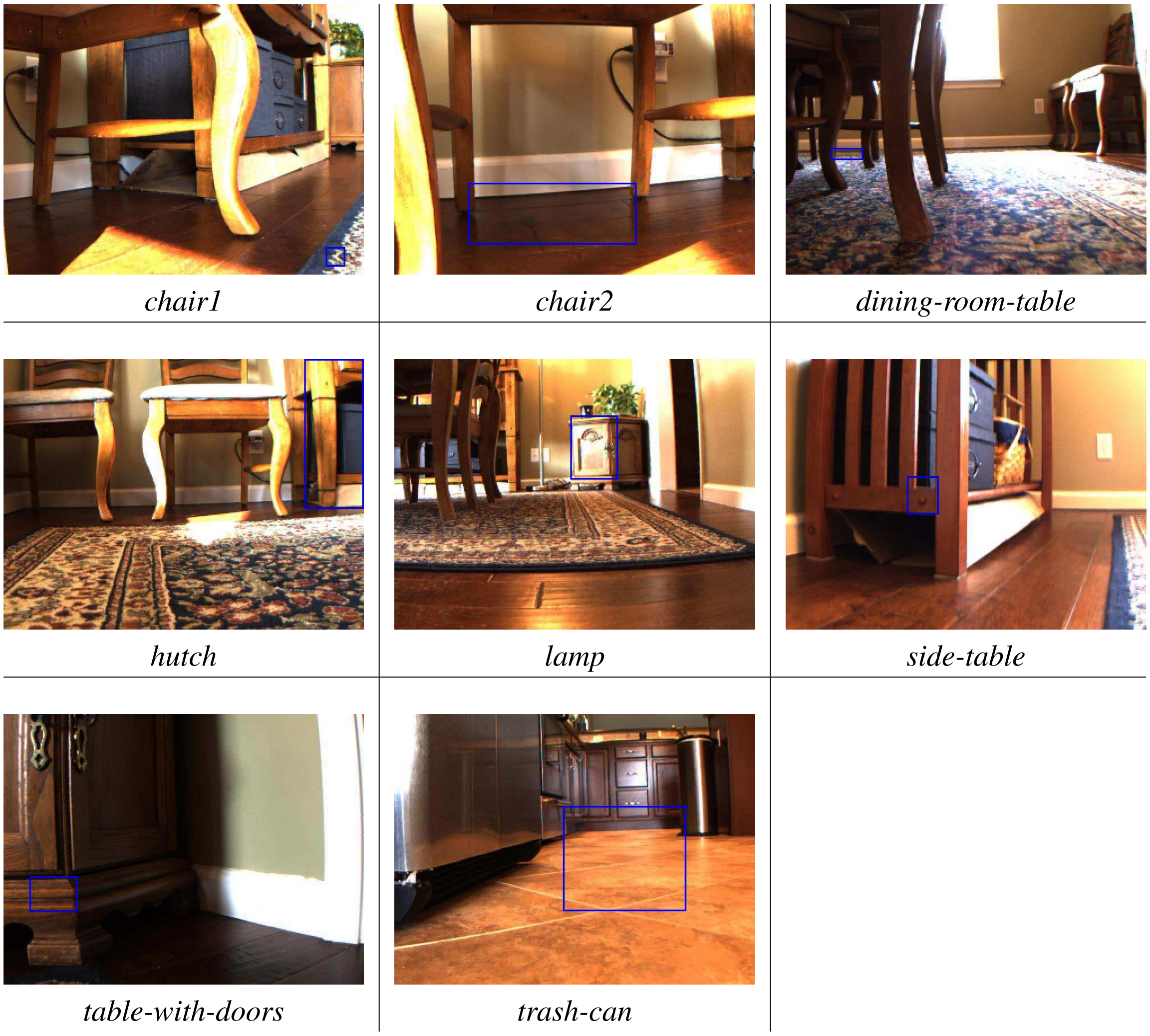


Fig. 7.6.: (*top*) The living room floor plan shown with labeled ground truth object locations. (*bottom*) The same floor plan with detected object locations shown as magenta diamonds. Unlabeled ground truth rectangles are shown for reference.



chair1

chair2

dining-room-table

hutch

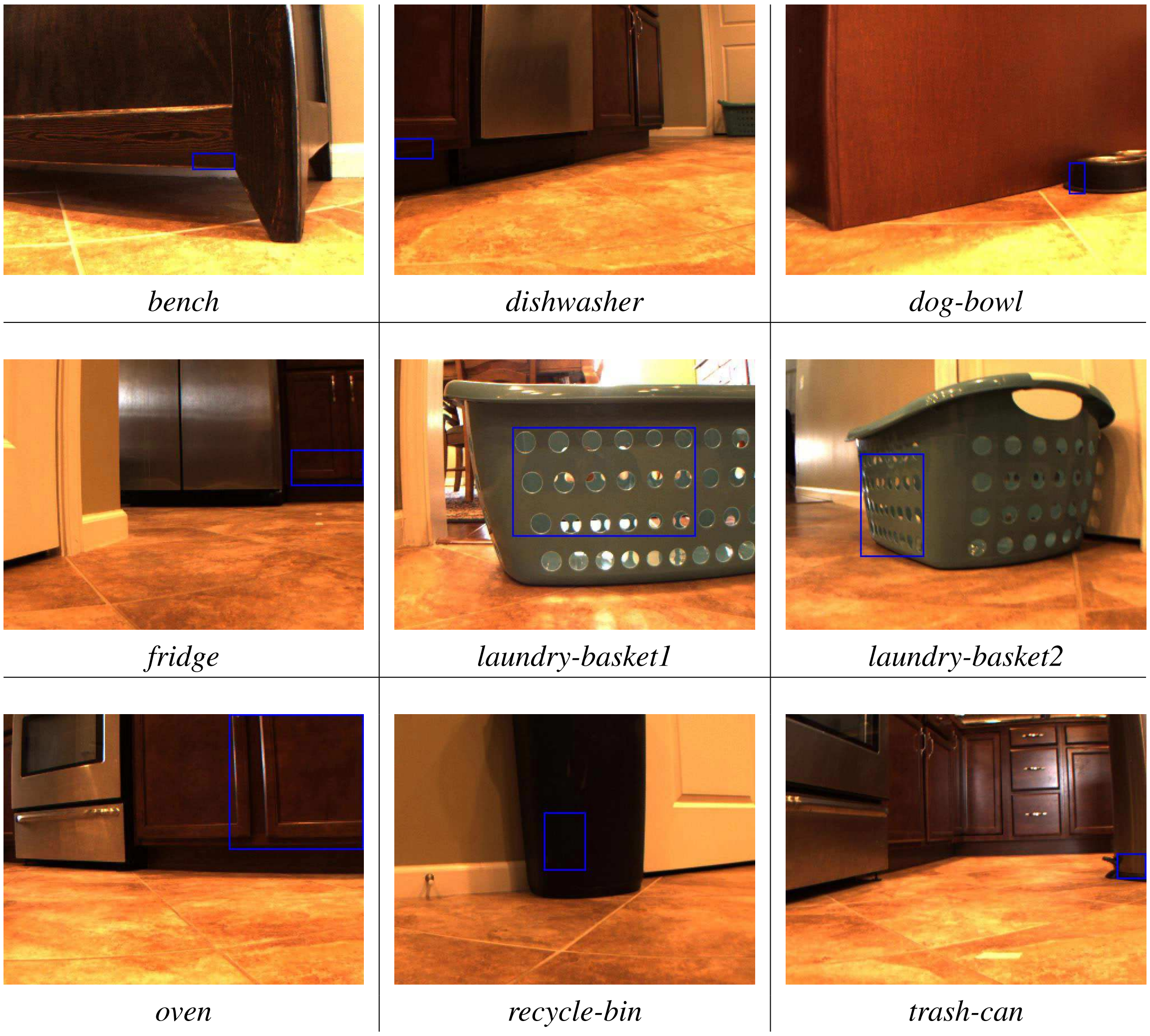
lamp

side-table

table-with-doors

trash-can

Fig. 7.7.: Example images from the tubes assigned to each noun class in the dining room floor plan. Tubes are represented by a blue box in each image.



bench

dishwasher

dog-bowl

fridge

laundry-basket1

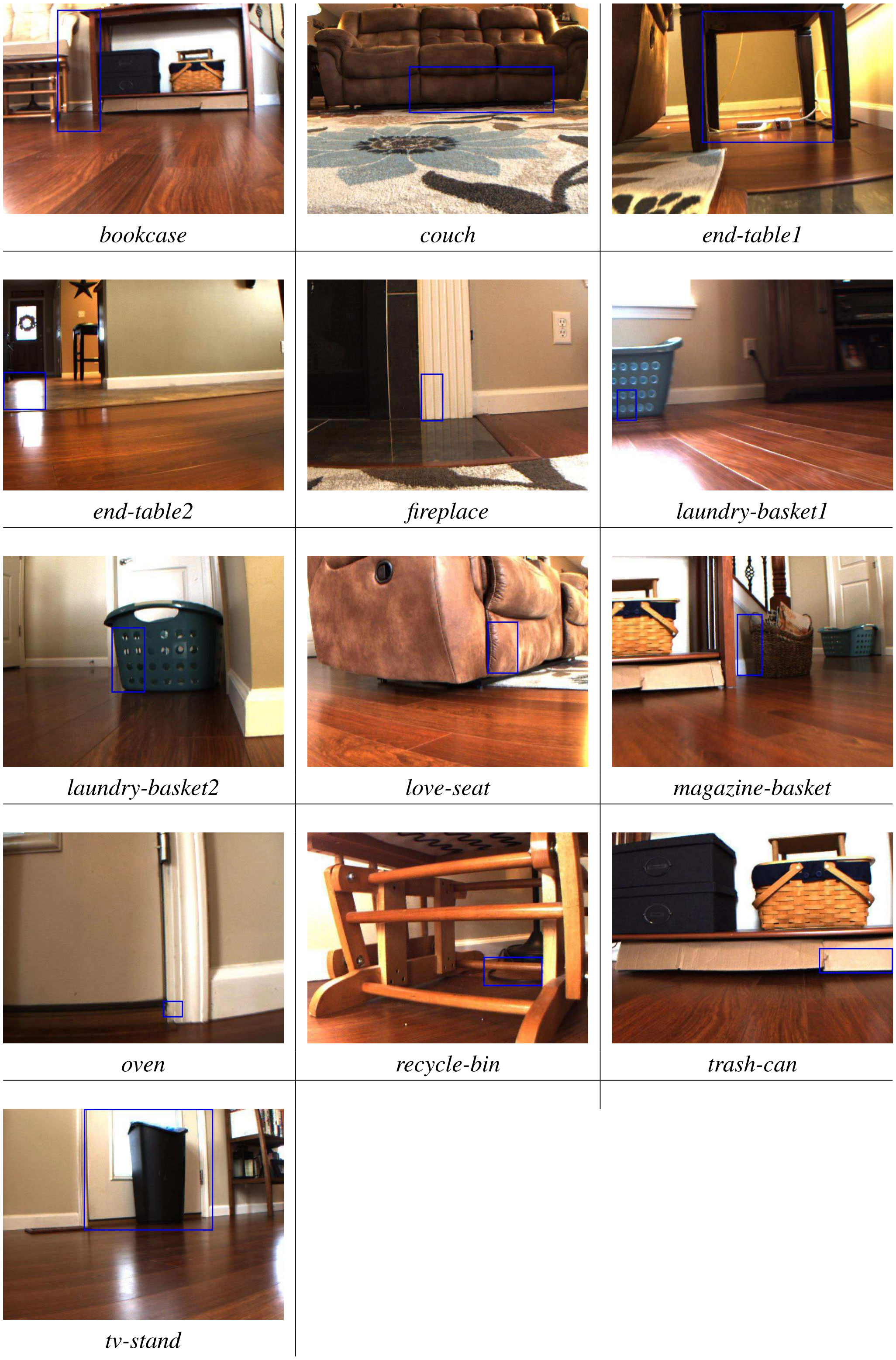
laundry-basket2

oven

recycle-bin

trash-can

Fig. 7.8.: Example images from the tubes assigned to each noun class in the kitchen floor plan. Tubes are represented by a blue box in each image.



bookcase

couch

end-table1

end-table2

fireplace

laundry-basket1

laundry-basket2

love-seat

magazine-basket

oven

recycle-bin

trash-can

tv-stand

Fig. 7.9.: Example images from the tubes assigned to each noun class in the living room floor plan. Tubes are represented by a blue box in each image.

While the results in Table 7.1 seem somewhat promising, the example images from the selected tubes in Figures 7.7 through 7.9 show evidence of two problems with my method that I have not yet been able to solve.

The first problem is the drift that accumulates in my dead reckoning localization system, which is based on data from odometry and IMU sensors. This problem is inherent to my localization system; as **VADER** covers more distance, its estimate of where it is becomes progressively less accurate. Tube locations which come from frames occurring later in runs are thus not as reliable as those which are based on early frames. One can most clearly see evidence of this in Fig. 7.9. Here the *bookcase* image shows a box that is actually located between the *rocking-chair* and *side-table*, even though the location for the *bookcase* in Fig. 7.6 appears within ground truth for that object. Similarly, the *tv-stand* is located within ground truth while its image actually shows the *recycle-bin*. The opposite outcome of this problem appears with *laundry-basket2*. Its image in Fig. 7.9 plainly shows a box located on the object, while its location in Fig. 7.6 shows over 50 cm of error.

I conceptualized two soft preferences that might help correct these problems, but due to time constraints I was unable to implement and test them. Both of these preferences could be implemented as additional multipliers to the unary score of a tube in (7.1). The first preference would encode the length of the tube, since a shorter tube can have an artificially low variance. Thus a tube whose length is close to the filter minimum (see Section 7.3.3) would receive a score close to 0, while a longer tube would receive a score closer to 1. The second preference would come from when a tube occurs within the course of a run. A tube which occurs early in a run would receive a better score than one which occurs near the end of a run. If I had more time to work on this project, these preferences would be the first modifications to my method that I would try.

The second problem in my method is the proposal boxes themselves. Without exception, all example images in Figures 7.7 through 7.9 show boxes that do not actually enclose the intended objects. I believe this is related to the use of Edge Boxes [112] as the initial proposal generation mechanism. If the initial proposals do not adequately enclose the potential objects, the subsequent steps have no way to correct this error. I would attempt to remedy this problem by trying MCG [102] instead of Edge Boxes for the initial object proposals. However, MCG is known to be at least an order of magnitude slower than Edge Boxes [114]. The initial proposal and tube generation step using Edge Boxes took approximately 576 core-days (three days running on four 48-core servers) to complete. Running this step again with MCG instead of Edge Boxes would require significantly more compute resources, time, or both. Given more time, I could also find and try other off-the-shelf proposal generation systems.

7.5 Conclusion

In this chapter I present the final portion of research that I have accomplished in my program. I develop a novel four-step method to locate and label real-world objects through the integration of computer vision, robotic odometry, and natural language. I show the results of my proof-of-concept experiment, which shows promising initial results. I identify some of the shortcomings of my method and propose remedies to try in the future.

8. CONCLUSION

The theme of my research program has been investigating methods for humans to interact with robots in a more natural fashion. My inspiration for this came from observing the current state of the art of battlefield robotics, where the physical and cognitive burden of operating a robot renders the soldier-operator incapable of self-defense. The interaction modalities currently in use require a robot operator to *drive* the robot by watching a screen and manually operating a controller. My long-term vision is to develop a robotic control system which allows a person to instead *direct* the robot through natural-language commands, not unlike the way in which characters in the *Star Wars* universe interact with C3PO and R2D2.

While I admit that my work is nowhere near a complete solution to this problem, I believe that it is a novel and significant addition to the body of research on this topic. In Chapters 4 and 5 I present a framework for grounding natural language semantics in robotic driving, including a method to both represent and learn word meanings. In Chapter 6 I apply codetection to robotic video paired with odometry by presenting a method to automatically detect, localize, and label objects in **VADER**'s real-world environment. Finally, in Chapter 7 I unify the two previous methods by integrating sentential information into the object codetection process.

It is my sincere hope that the work presented here can contribute in some small way to the advancement of the state of the art in human-robot interaction.

REFERENCES

REFERENCES

[1] R. Doaré, D. Danet, J.-P. Hanon, and G. de Boisboissel, “Robots on the battlefield. contemporary perspectives and implications for the future,” DTIC Document, Tech. Rep., 2014.

[2] iRobot Corporation. (2014) iRobot 510 PackBot the Multi-mission Robot. [Online]. Available: http://www.irobot.com/For-Defense-and-Security/Robots/510-PackBot.aspx

[3] ——. (2014) iRobot uPoint Multi-Robot Control System. [Online]. Available: http://www.irobot.com/For-Defense-and-Security/Robots/uPoint.aspx

[4] D. P. Barrett, S. A. Bronikowski, H. Yu, and J. M. Siskind, “Robot language learning, generation, and comprehension,” *arXiv*, vol. abs/1508.06161, 2015.

[5] ——, “Driving under the influence (of language),” 2016, submitted to IEEE Transactions on Neural Networks and Learning Systems.

[6] S. A. Bronikowski, D. P. Barrett, and J. M. Siskind, “Object codetection from mobile robot video,” 2016, submitted to IEEE/RSJ International Conference on Intelligent Robots and Systems.

[7] M. MacMahon, B. Stankiewicz, and B. Kuipers, “Walk the talk: connecting language, knowledge, and action in route instructions,” in *AAAI Conference on Artificial Intelligence*, 2006, pp. 1475–1482.

[8] T. Kollar, S. Tellex, D. Roy, and N. Roy, “Toward understanding natural language directions,” in *International Conference on Human-Robot Interaction*, 2010, pp. 259–266.

[9] C. Matuszek, D. Fox, and K. Koscher, “Following directions using statistical machine translation,” in *International Conference on Human-Robot Interaction*, 2010, pp. 251–258.

[10] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, “Understanding natural language commands for robotic navigation and mobile manipulation,” in *AAAI Conference on Artificial Intelligence*, 2011, pp. 1507–1514.

[11] D. L. Chen and R. J. Mooney, “Learning to interpret natural language navigation instructions from observations,” in *AAAI Conference on Artificial Intelligence*, 2011, pp. 859–865.

[12] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, “Learning to parse natural language commands to a robot control system,” in *International Symposium on Experimental Robotics*, 2012, pp. 403–415.

[13] Y. Artzi and L. Zettlemoyer, "Weakly supervised learning of semantic parsers for mapping instructions to actions," *Transactions of the Association for Computational Linguistics*, vol. 1, no. 1, pp. 49–62, 2013.

[14] S. Tellex, P. Thaker, J. Joseph, and N. Roy, "Learning perceptually grounded word meanings from unaligned parallel data," *Machine Learning*, vol. 92, no. 2, pp. 151–167, 2014.

[15] Y. Wei, E. Brunskill, T. Kollar, and N. Roy, "Where to go: Interpreting natural directions using global inference," in *IEEE International Conference on Robotics and Automation*, 2009, pp. 3761–3767.

[16] S. Dobnik, S. Pulman, P. Newman, and A. Harrison, "Teaching a robot spatial expressions," *Proceedings of the Second ACL-SIGSEM Workshop on The Linguistic Dimensions of Prepositions and their Use in Computational Linguistics Formalisms and Applications*, 2005.

[17] S. Lauria, G. Bugmann, T. Kyriacou, and E. Klein, "Mobile robot programming using natural language," *Robotics and Autonomous Systems*, vol. 38, no. 3, pp. 171–181, 2002.

[18] S. Teller, M. R. Walter, M. Antone, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J. P. How, A. S. Huang *et al.*, "A voice-commandable robotic forklift working alongside humans in minimally-prepared outdoor environments," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 526–533.

[19] A. Koller, K. Striegnitz, A. Gargett, D. Byron, J. Cassell, R. Dale, J. Moore, and J. Oberlander, "Report on the second NLG challenge on generating instructions in virtual environments (GIVE-2)," in *Proceedings of the 6th International Natural Language Generation Conference*, 2010, pp. 243–250.

[20] T. K. Harris, S. Banerjee, and A. I. Rudnicky, "Heterogeneous multi-robot dialogues for search tasks," in *Proceedings of the AAAI Spring Symposium Intelligence*, 2005.

[21] M. R. Marge, A. K. Pappu, B. Frisch, T. K. Harris, and A. I. Rudnicky, "Exploring spoken dialog interaction in human-robot teams," in *Proceedings of Robots, Games, and Research: Success Stories in USARSim*, 2009.

[22] A. Pappu and A. Rudnicky, "The structure and generality of spoken route instructions," in *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue, ACL*, 2012, pp. 99–107.

[23] J. Fasola and M. J. Mataric, "Using semantic fields to model dynamic spatial relations in a robot architecture for natural language instruction of service robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 143–150.

[24] P. McGuire, J. Fritsch, J. J. Steil, F. Rothling, G. A. Fink, S. Wachsmuth, G. Sagerer, and H. Ritter, "Multi-modal human-machine communication for instructing robot grasping tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, 2002, pp. 1082–1088.

[25] F. Doshi and N. Roy, "Spoken language interaction with model uncertainty: an adaptive human-robot interaction system," *Connection Science*, vol. 20, no. 4, pp. 299–318, 2008.

[26] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox, “A joint model of language and perception for grounded attribute learning,” in *International Conference on Machine Learning*, 2012, pp. 1671–1678.

[27] L. She, S. Yang, Y. Cheng, Y. Jia, J. Y. Chai, and N. Xi, “Back to the blocks world: Learning new actions through situated human-robot dialogue,” in *The Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2014, pp. 89–97.

[28] M. Blaschko, A. Vedaldi, and A. Zisserman, “Simultaneous object detection and ranking with weak supervision,” in *Neural Information Processing Systems*, 2010, pp. 235–243.

[29] Y. J. Lee and K. Grauman, “Learning the easy things first: self-paced visual category discovery,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2011, pp. 1721–1728.

[30] M. Rubinstein, A. Joulin, J. Kopf, and C. Liu, “Unsupervised joint object discovery and segmentation in internet images,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1939–1946.

[31] K. Tang, A. Joulin, L.-J. Li, and L. Fei-Fei, “Co-localization in real-world images,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1464–1471.

[32] A. Prest, C. Leistner, J. Civera, C. Schmid, and V. Ferrari, “Learning object class detectors from weakly annotated video,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3282–3289.

[33] S. Schulter, C. Leistner, P. M. Roth, and H. Bischof, “Unsupervised object discovery and segmentation in videos,” in *Proceedings of the British Machine Vision Conference*, 2013, pp. 53.1–53.12.

[34] A. Joulin, K. Tang, and L. Fei-Fei, “Efficient image and video co-localization with Frank-Wolfe algorithm,” in *European Conference on Computer Vision*, 2014, pp. 253–268.

[35] A. Srikantha and J. Gall, “Discovering object classes from activities,” in *European Conference on Computer Vision*, 2014, pp. 415–430.

[36] S. Bird, “NLTK: the natural language toolkit,” in *International Conference on Computational Linguistics/Annual Meeting of the Association for Computational Linguistics*, 2006, pp. 69–72.

[37] C. Fellbaum, *WordNet*. Wiley Online Library, 1998.

[38] T. Kollar and N. Roy, “Utilizing object-object and object-scene context when planning to find things,” in *IEEE International Conference on Robotics and Automation*, 2009, pp. 2168–2173.

[39] S. Friedman, H. Pasula, and D. Fox, “Voronoi random fields: Extracting topological structure of indoor environments via place labeling.” in *International Joint Conference on Artificial Intelligence*, vol. 7, 2007, pp. 2109–2114.

[40] A. Vogel and D. Jurafsky, “Learning to follow navigational directions,” in *Annual Meeting of the Association for Computational Linguistics*, 2010, pp. 806–814.

[41] M.-C. De Marneffe, B. MacCartney, C. D. Manning *et al.*, "Generating typed dependency parses from phrase structure parses," in *Proceedings of LREC*, vol. 6, 2006, pp. 449–454.

[42] M. Abramowitz and I. A. Stegun, *Handbook of mathematical functions*. Dover New York, 1972.

[43] I. L. Hetherington, "Pocketsummit: small-footprint continuous speech recognition." in *INTERSPEECH*, vol. 11, 2007, pp. 13–51.

[44] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman *et al.*, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[45] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[46] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," DTIC Document, Tech. Rep., 1992.

[47] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "GALAXY-II: a reference architecture for conversational system development." in *International Conference on Spoken Language Processing*, 1998, pp. 931–934.

[48] X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, K.-F. Lee, and R. Rosenfeld, "The SPHINX-II speech recognition system: an overview," *Computer Speech & Language*, vol. 7, no. 2, pp. 137–148, 1993.

[49] W. Ward, "Extracting information in spontaneous speech," in *International Conference on Spoken Language Processing*, 1994.

[50] D. Bohus and A. Rudnicky, "Integrating multiple knowledge sources for utterance-level confidence annotation in the cmu communicator spoken dialog system," DTIC Document, Tech. Rep., 2002.

[51] D. Bohus and A. I. Rudnicky, "Ravenclaw: Dialog management using hierarchical task decomposition and an expectation agenda," in *Eighth European Conference on Speech Communication and Technology*, 2003.

[52] A. H. Oh and A. I. Rudnicky, "Stochastic language generation for spoken dialogue systems," in *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational systems-Volume 3*, 2000, pp. 27–32.

[53] A. W. Black, P. Taylor, R. Caley, and R. Clark, "The festival speech synthesis system," *University of Edinburgh*, vol. 1, 2002.

[54] M. Montemerlo, N. Roy, and S. Thrun. (2002) Carnegie mellon robot navigation toolkit. [Online]. Available: http://www.cs.cmu.edu/carmen

[55] T. Kwiatkowski, L. Zettlemoyer, S. Goldwater, and M. Steedman, "Lexical generalization in CCG grammar induction for semantic parsing," in *Empirical Methods on Natural Language Processing*, 2011, pp. 1512–1523.

[56] S. R. K. Branavan, L. S. Zettlemoyer, and R. Barzilay, “Reading between the lines: Learning to map high-level instructions to commands,” in *Annual Meeting of the Association for Computational Linguistics*, 2010, pp. 1268–1277.

[57] B. Carpenter, *Type-logical semantics*. MIT Press, 1997.

[58] J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth, “Driving semantic parsing from the world’s response,” in *Conference on Computational Natural Language Learning*, 2010, pp. 18–27.

[59] Y. He and S. Young, “Semantic processing using the hidden vector state model,” *Computer Speech & Language*, vol. 19, no. 1, pp. 85–106, 2005.

[60] J. Hockenmaier and M. Steedman, “Generative models for statistical parsing with combinatory categorial grammar,” in *Annual Meeting of the Association for Computational Linguistics*, 2002, pp. 335–342.

[61] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[62] K. A. Papineni, S. Roukos, and T. R. Ward, “Feature-based language understanding,” in *European Conference on Speech Communication and Technology*, 1997.

[63] N. Siddharth, A. Barbu, and J. M. Siskind, “Seeing what you’re told: Sentence-guided activity recognition in video,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 732–739.

[64] M. Steedman, *Surface structure and interpretation*. MIT Press, 1996.

[65] ——, *The syntactic process*. MIT Press, 2000.

[66] Lynxmotion. (2013) A6WD2 No Electronics. [Online]. Available: http://www.lynxmotion.com/c-163-a6wd2-no-electronics.aspx

[67] ——. (2013) 12.0 Volt Ni-MH 2800mAh Battery Pack. [Online]. Available: http://www.lynxmotion.com/p-727-120-volt-ni-mh-2800mah-battery-pack.aspx

[68] SparkFun Electronics. (2009) SparkFun DC/DC Converter Breakout - BOB-09370 - SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/9370

[69] Gumstix, Inc. (2015) Overo®FireSTORM COM — Gumstix®. [Online]. Available: https://store.gumstix.com/index.php/products/267/

[70] ——. (2015) Summit — Gumstix®. [Online]. Available: https://store.gumstix.com/index.php/products/215/

[71] Amazon.com. (2014) Pluggable®7-Port USB 3.0 SuperSpeed Hub. [Online]. Available: http://www.amazon.com/gp/product/B008ZGKWQI/

[72] Lynxmotion. (2013) Sabertooth 2X12 Regenerative Dual Channel Motor Controller. [Online]. Available: http://www.lynxmotion.com/p-562-sabertooth-2x12-regenerative-dual-channel-motor-controller.aspx

[73] U. Digital. (2013) US Digital — Products — E4 Miniature Optical Kit Encoder. [Online]. Available: http://www.usdigital.com/products/e4

[74] Servocity.com. (2013) HS-5055MG Servo. [Online]. Available: https://www.servocity.com/html/hs-5055mg_servo.html

[75] Amazon.com. (2014) TP-LINK TL-WN822N Wireless N300 High Gain USB Adapter. [Online]. Available: http://www.amazon.com/gp/product/B00416Q5KI

[76] Verizon Wireless. (2014) 4G LTE USB Modem UML295. [Online]. Available: http://www.verizonwireless.com/internet-devices/4g-lte-usb-modem-uml295/

[77] Point Grey Research. (2013) Chameleon USB2.0 cameras. [Online]. Available: http://www.ptgrey.com/chameleon-usb2-cameras

[78] ImmerVision. (2014) IMV1-1/3 and IMV1-1/3NI 360° PANOMORPH LENS. [Online]. Available: https://www.immervisionenables.com/wp-content/uploads/2014/03/ImmerVision_Enables-OnePager-IMV1_2014.pdf

[79] Amazon.com. (2014) GlobalSat BU-353 USB GPS Receiver. [Online]. Available: http://www.amazon.com/gp/product/B000PKX2KA

[80] National Marine Electronics Association. (2014) NMEA. [Online]. Available: http://www.nmea.org/content/nmea_standards/nmea_0183_v_410.asp

[81] SparkFun Electronics. (2010) 9 Degrees of Freedom - Razor IMU - SEN-10736 - SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/10736

[82] P. Bartz. (2013) ptrbrtz/razor-9dof-ahrs. [Online]. Available: https://github.com/ptrbrtz/razor-9dof-ahrs

[83] PJRC. (2014) Teensy USB Development Board. [Online]. Available: https://www.pjrc.com/teensy/

[84] ——. (2014) Teensyduino – Add-on for Arduino IDE to use Teensy USB development board. [Online]. Available: https://www.pjrc.com/teensy/teensyduino.html

[85] ——. (2014) Encoder Library, for Measuring Quadarature Encoded Position or Rotation Signals. [Online]. Available: https://www.pjrc.com/teensy/td_libs_Encoder.html

[86] S. A. Bronikowski. (2013) scottbronikowski/rootfs. [Online]. Available: https://github.com/scottbronikowski/rootfs

[87] Point Grey Research. (2013) FlyCapture SDK. [Online]. Available: http://www.ptgrey.com/flycapture-sdk

[88] OpenCV.org. (2013) Documentation – OpenCV. [Online]. Available: http://opencv.org/documentation.html

[89] Enlightenment Foundation Libraries. (2013) Imlib2 Library Documentation. [Online]. Available: https://docs.enlightenment.org/api/imlib2/html/

[90] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[91] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*. Academic Press, 1970.

[92] S. A. Bronikowski. (2013) scottbronikowski/vader-rover. [Online]. Available: https://github.com/scottbronikowski/vader-rover

[93] S. Middleditch. (2014) elanthis/gamepad. [Online]. Available: https://github.com/elanthis/gamepad

[94] J. M. Siskind. (2002) Jeffrey Mark Siskind's Software. [Online]. Available: https://engineering.purdue.edu/~qobi/software.html

[95] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *The Annals of Mathematical Statistics*, vol. 37, pp. 1554–63, 1966.

[96] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–71, 1970.

[97] L. E. Baum, "An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process," *Inequalities*, vol. 3, pp. 1–8, 1972.

[98] R. Dale and E. Reiter, "Computational interpretations of the gricean maxims in the generation of referring expressions," *Cognitive Science*, vol. 19, no. 2, pp. 233–263, 1995.

[99] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Springer US, 1972, pp. 85–103.

[100] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks," in *Empirical Methods on Natural Language Processing*, 2014, pp. 740–750.

[101] Wikipedia, "List of English prepositions," 2015, [Online; accessed 4-June-2015]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=List_of_English_prepositions&oldid=662161282

[102] P. Arbelaez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik, "Multiscale combinatorial grouping," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 328–335.

[103] A. Bosch, A. Zisserman, and X. Munoz, "Image classification using random forests and ferns," in *IEEE International Conference on Computer Vision*, 2007, pp. 1–8.

[104] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library of computer vision algorithms," in *Proceedings of the International Conference on Multimedia*, 2010, pp. 1469–1472.

[105] J. Gall and V. Lempitsky, "Class-specific Hough Forests for object detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1022–1029.

[106] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[107] J. Pearl, "Reverend Bayes on inference engines: a distributed hierarchical approach," in *AAAI Conference on Artificial Intelligence*, 1982, pp. 133–136.

[108] B. Andres, J. H. Kappes, U. Köthe, C. Schnörr, and F. A. Hamprecht, "An empirical comparison of inference algorithms for graphical models with higher order factors using opengm," in *Pattern Recognition*, 2010, pp. 353–362.

[109] B. Andres, T. Beier, and J. H. Kappes, "OpenGM: A C++ library for discrete graphical models," *arXiv*, vol. abs/1206.0111, 2012.

[110] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.

[111] H. Yu and J. M. Siskind, "Sentence directed video object codetection," 2016, submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI).

[112] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *European Conference on Computer Vision*, 2014.

[113] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-backward error: Automatic detection of tracking failures," in *International Conference on Pattern Recognition*, 2010, pp. 2756–2759.

[114] J. Hosang, R. Benenson, and B. Schiele, "How good are detection proposals, really?" *arXiv*, vol. abs/1406.6962, 2014.

VITA

VITA

Scott Alan Bronikowski was born in Milwaukee, Wisconsin, on November 11, 1977. After completing high school at Thomas More High School in Milwaukee in 1995, Scott entered the United States Military Academy (USMA) at West Point, New York. Scott graduated from USMA in 1999 with a Bachelor of Science in Electrical and Computer Engineering and was commissioned as a Second Lieutenant in the Armor Branch of the United States Army. He then served in several different positions, including Tank Platoon Leader, Tank Company Executive Officer, Battalion Logistics Officer, Assistant Brigade Logistics Officer, Assistant Brigade Operations Officer, and Company Commander, at Fort Riley, Kansas, from 1999 through 2007. These assignments included one deployment to Kuwait in support of Operation Desert Spring in 2000 and two deployments to Iraq in support of Operation Iraqi Freedom in 2003 and 2005-2006.

In 2007 Scott changed his branch in the Army from Armor to Telecommunications Systems Engineering. Scott also entered the Graduate School of Kansas State University in Manhattan, Kansas, in 2007. He graduated from Kansas State with a Master of Science in Electrical and Computer Engineering in 2009. Scott was then assigned to the Department of Electrical Engineering and Computer Science at USMA, where he served as an Instructor and later earned promotion to Assistant Professor. He taught upper-level undergraduate electrical engineering courses and advised senior design projects from 2009 through 2012. In 2012 Scott again deployed to Kuwait to support Operation Enduring Freedom as Chief of Network Engineering for US Army Central.

In August 2013, Scott entered the Graduate School of Purdue University in order to pursue a PhD in Electrical and Computer Engineering prior to returning to the Department of Electrical Engineering and Computer Science at USMA.