DCG-UPUP-Away: Automatic Symbol Acquisition through Grounding to Unknowns

by

Mycal Tucker

S.B. Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Author ....
Department of Electrical Engineering and Computer Science
August 16, 2016

Certified by ....
Nicholas Roy
Professor
Thesis Supervisor

Accepted by ....
Dr. Christopher Terman
Chairman, Masters of Engineering Thesis Committee

DCG-UPUP-Away: Automatic Symbol Acquisition through Grounding to Unknowns

by

Mycal Tucker

Submitted to the Department of Electrical Engineering and Computer Science
on August 16, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Research in automatic Natural Language grounding, in which robots understand how phrases relate to real-world objects or actions, offers a tantalizing reality in which untrained humans can operate highly sophisticated robots. Current techniques for training robots to understand natural language, however, assume that there is a fixed set of phrases or objects that the robot will encounter during deployment. Instead, the real world is full of confusing jargon and unique objects that are nearly impossible to anticipate and therefore train for. This paper presents a model called the Distributed Correspondence Graph, Unknown Phrase, Unknown Percept, Away (DCG-UPUP-Away) that augments a previously successful model by explicitly modeling unknown phrases and objects as unknown, as well as reasoning about objects that are not currently perceived. Furthermore, experimental results in simulation, as well as a trial run on a physical turtlebot, validate the effectiveness of DCG-UPUP-Away in grounding phrases and learning new phrases.

Thesis Supervisor: Nicholas Roy
Title: Professor

Acknowledgments

This is where I will write my acknowledgments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Perhaps the defining vision in robotics since its inception as a field has been autonomy — how to make a robotic agent act intelligently and robustly without any guidance from humans. Although this view of autonomy seems to preclude interaction with humans, in fact it requires it. A truly autonomous robot operating in a human-filled environment must be able to interact with humans in a natural way; one would not argue that people who talk to other people are not autonomous, but someone who must be told how to respond to each individual phrase loses that title. Conversely, ignoring natural language from humans, although technically autonomous behavior, meets autonomy requirements in the same way a rock is autonomous. Therefore, natural language interaction is a necessary requirement for a robot to fulfill the grand promise of autonomous robotics.

Driven by such a vision, considerable work has been done on developing approaches to the "grounding problem". The grounding problem is the association of natural language to real world actions or objects - i.e. the realization that "chair" refers to the structure of wood and a cushion in a room full of clutter. Probabilistic graphical models in particular have yielded impressive results such as training forklifts to drive to pallets when told "pick up the pallet". Unfortunately, the models used in such techniques often make two assumptions:

1) the solution to the grounding problem will be an association between objects

and phrases the robot has been trained to recognize

2) the solution to the grounding problem will refer to objects in the robot's known environment (either in a map or in the robot's field of view)

If a robot is to operate in the real world, these assumptions will be violated. On the one hand, it is impossible to teach a robot all correct groundings before deploying it in the real world. Even if a robot were to learn every word in the dictionary, it would also have to recognize every possible object in the world in order to generate the right groundings. Furthermore, context-dependent jargon often attaches new meaning to words while specific environments often contain otherwise rarely-found objects. On the other hand, robots rarely have complete information about their environment ahead of time and must therefore be able to reason about un-perceived objects.

In this work, techniques for reasoning in environments in which both assumptions are relaxed are presented in a system called DCG-UPUP-Away. A previously successful model called DCG is adapted to reason over broader domains and solved using standard techniques. This model is implemented, trained, and tested on a turtlebot navigating a junkyard-like environment.

1.1 Problem Statement

The fundamental problem that DCG-UPUP-Away must solve is the grounding problem. The correct solution to a grounding problem is the interpretation of natural language to real-world objects and actions. For example, in a room with a box and a helmet the correct grounding of "drive to the helmet" is the execution of a path to the helmet. Generally, the natural language phrase is not required to be a command (e.g. "this is a helmet"), but in this work only commands will be considered.

In a probabilistic framework, the grounding problem can be rephrased as a probability maximization problem:

$$\vec{\gamma}^* = \arg\max_{\vec{\gamma}\epsilon\Gamma} P(\vec{\gamma}|\vec{\lambda}, \Upsilon) \tag{1.1}$$

where $\vec{\gamma}$ is an element of Γ, the set of all possible groundings, $\vec{\lambda}$ is a natural language phrase, and Υ is a model of the world. In other words, the grounding problem must find the vector of groundings $\vec{\gamma}$ that maximizes the probability of being the right groundings given a particular phrase $\vec{\lambda}$ and a model of the world. In this work, I propose adaptations to an existing probabilistic graphical model, DCG, that efficiently solves the grounding problem while expanding the domain of Γ in two ways: 1) Γ includes objects the robot has not been trained to recognize and 2) Γ includes objects not perceived by the robot.

1.2 Contributions

The main contribution of this work is the design of DCG-UPUP-Away, a probabilistic graphical model built upon DCG that increases the domain over which to solve the grounding problem. First, explicitly recognizing when phrases or objects are unknown allows the model to reason and learn in new environments. Second, "hallucinating" potential groundings allows the model to ground to out-of-perception objects.

The secondary contribution of this work is a validation of DCG-UPUP-Away's performance. On the one hand, a large-scale study using natural language commands in a simulated environment shows that a robot can correctly ground phrases around 80% of the time while in an environment in which it initially knows fewer than half of the objects present. On the other hand, a sample trial run on a turtlebot in a laboratory office demonstrates that, with some modifications to object recognition software, DCG-UPUP-Away can be used in the real world.

1.3 Outline

Chapter 2 is background. Chapter 3 is grounding to unknowns. Chapter 4 is grounding out of field of view. Chapter 5 is finer-grain disambiguation. Chapter 6 is evaluation. Chapter 7 is conclusion.

Chapter 2

Background

The work presented in this thesis builds upon research on the natural language grounding problem. As discussed in the introduction, the grounding problem is the problem of finding the most likely physical meaning for a natural language phrase. For example, when uttered in a room with a cone and a helmet, the phrase “the cone” probably refers to the physical cone itself. More generally, phrases other than nouns may also be grounded to their physical meanings (e.g. “skipping” to a hopping sort of run), but such examples are harder to describe.

A useful perspective in viewing the grounding problem is through the lens of symbols. On the natural language side, phrases (e.g. “cube”) are associated with symbols (e.g. a canonical cube type). On the real-world side, percepts (e.g. an RGB image of a cube) are associated with symbols (e.g. a canonical cube type). Such relations are shown in Figure 2-1. Using such a representation seems to simplify the grounding problem significantly: finding the correct grounding for a given phrase merely involves translating from language to symbols and from those symbols to percepts.

Unfortunately, although such a method may work in theory in a world with one-to-one mappings with symbols, natural language and the real world are too complex for such a naive approach to work. For example, consider the fact that several words may refer to the same symbolic type (e.g. “cat” and “feline” both refer to a canonical

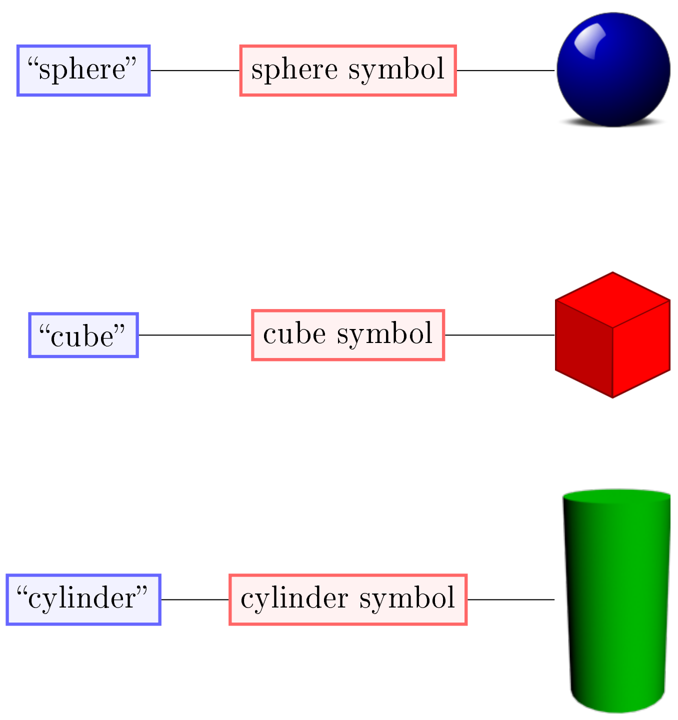


Figure 2-1: Mappings among phrases on the left, symbols in the middle, and percepts on the right.

cat), a single word may refer to multiple symbolic types (e.g. "dust" refers to a powder of detritus, the act of removing a powder of detritus, or the act of adding a powder like sugar), several percepts may contain the same symbolic type (e.g. two different photos of a cone), and a single percept may refer to multiple symbolic types (e.g. a photo containing a cone and a helmet). Thus, Figure 2-1 would be better represented with Figure 2-2. Even within a simplified block world, language and perception ambiguity is unavoidable.

Given the complexity of phrase-symbol-percept mappings, a simple and deterministic solution to the grounding problem appears impossible. A probabilistic view, however, simplifies the grounding problem considerably while reflecting the uncertainty inherent in language. For the remainder of this section, two probabilistic models used to solve the grounding problem will be introduced before discussing related - but less directly relevant - work in natural language robotics.

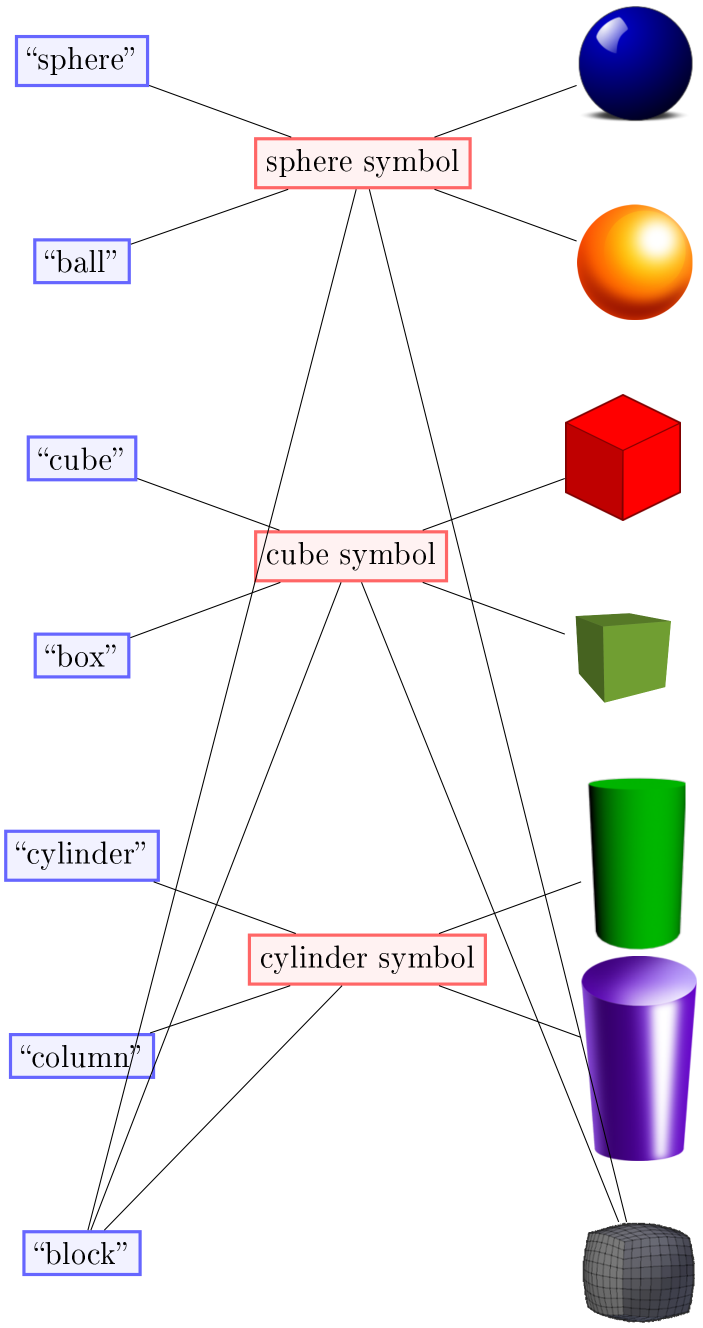


Figure 2-2: A more realistic but complex mapping among phrases on the left, symbols in the middle, and percepts on the right. Note how one-to-one mappings are not guaranteed.

2.1 Grounding in the Generalized Grounding Graph

Five years ago, one of the first models, called the Generalized Grounding Graph (G^3), was used for solving the grounding problem with probabilistic graphical models [11]. The main insight in G^3 that makes the model so powerful is that the hierarchy of Spatial Description Clauses (SDCs) generate from the natural language command can be used to build up a probabilistic graphical model.

As in any probabilistic view of the grounding problem, the overall equation to be solved by G^3 is Equation 1.1, rewritten below for convenience. The optimal grounding $\vec{\gamma}^*$ is the set of groundings that maximizes the likelihood of that set of groundings, given language and a model of the world.

$$\vec{\gamma}^* = \underset{\vec{\gamma}\epsilon\Gamma}{\arg\max}\, P(\vec{\gamma}|\vec{\lambda}, \Upsilon) \tag{2.1}$$

By introducing a binary correspondence variable ϕ that represents whether or not a phrase is paired with the correct grounding, Equation 2.1 may be rewritten as follows:

$$\vec{\gamma}^* = \underset{\vec{\gamma}\epsilon\Gamma}{\arg\max}\, P(\phi = True|\vec{\lambda}, \vec{\gamma}, \Upsilon) \tag{2.2}$$

One way of paraphrasing Equation 2.2 is that the optimal grounding $\vec{\gamma}^*$ is the grounding such that the probability that the language and the grounding correspond, in the world model, is maximized. If the most likely setting of a given ϕ_i is true, then language λ_i and grounding γ_i likely correspond; otherwise they likely do not. Fixing ϕ_i to be true for all i, and then searching over the most likely settings of γ_i therefore achieves the goal of finding the most likely correct groundings.

Unfortunately, without addition information or assumptions, such a search could be prohibitively computationally expensive. Instead, Tellex et al. use the structure of the SDCs of the natural language command to factor Equation 2.2.

$$P(\phi = True|\vec{\lambda}, \Gamma, \Upsilon) = P(\phi = True|SDCs(\vec{\lambda}), \Gamma, \Upsilon) \tag{2.3}$$

$$= \frac{1}{Z} \prod_i \Psi_i(\phi_i, SDC_i, \Gamma, \Upsilon) \tag{2.4}$$

The final form of Equation 2.4 reveals that the probability of a correct grounding equals the product of a feature function Ψ evaluated for every correspondence variable, SDC clause, grounding, and the world model (all normalized by the partition function Z). The feature function Ψ, while shown in full detail in Equation 2.5, may be thought of simply as a learned log-linear function that associates binary features of language (e.g. if a word is present) with binary features about groundings (e.g. if a box is present).

$$\Psi_i(\phi_i, SDC_i, \Gamma, \Upsilon) = exp\Big(\sum_k \mu_k s_k(\phi_i, SDC_i, \Gamma, \Upsilon)\Big) \tag{2.5}$$

An example parse tree and G^3 graphical model are shown in Figures 2-3 and 2-4.

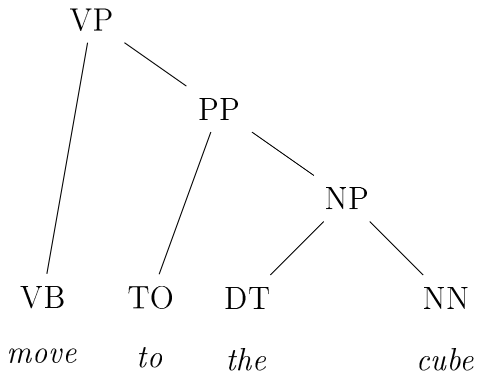


Figure 2-3: Parse tree for the sentence “move to the cube”

While the details of the algorithm for computing Ψ and building the graphical model are important, for the purposes of this literature review, the most important takeaway is that the G^3 algorithm fixes all the ϕ_i values to true and searches through Γ to find the most likely γ_i in each node.

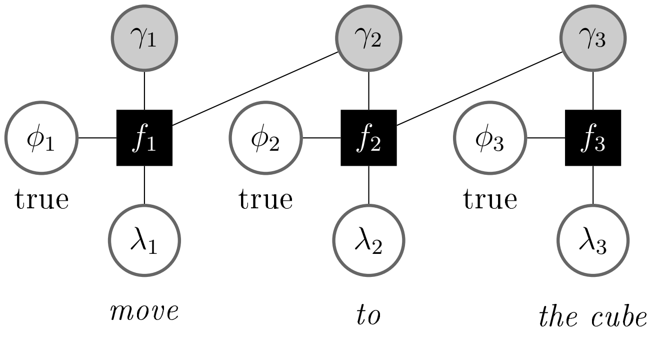


Figure 2-4: G^3 factor graph using the parse tree in Figure 2-3. The gray γ nodes are unknown, while the white ϕ and λ nodes are set to true and phrases, respectively.

2.2 Grounding in the Distributed Correspondence Graph

While G^3 formally describes the grounding problem in a probabilistic graphical model, inference on the graph remains relatively slow. The Distributed Correspondence Graph (DCG) addresses this problem in two ways - by building a larger, but ultimately more efficient, graphical model and inference procedure and by using ternary correspondence variables [4].

Although the ternary correspondence variables are important because they allow for planning over constraints that may be active, inactive, or inverted, ultimately they will not be used in DCG-UPUP-Away. The changed graph structure, however, is extremely important for this work. The key idea behind DCG is that, rather than fixing ϕ to true and then searching over Γ to find the most likely groundings, grounding nodes γ_{ij} may be introduced into the graph, each set to a possible grounding, and then search is over the most likely settings of ϕ_{ij}. Thus, Equation 2.2 from G^3 may be transformed to Equation 2.6.

$$\vec{\phi}^* = \underset{\vec{\phi} \epsilon \Phi}{\arg\max}\, P(\vec{\phi}|\vec{\lambda}, \vec{\gamma}, \Upsilon) \tag{2.6}$$

The equation can be paraphrased as saying that the most likely correspondence variables $\vec{\phi}$ can be found by finding the correspondence variables that maximize the likelihood of phrases and groundings corresponding, given a model of the world. By using the same factorization trick from G^3 assuming independence of child groundings, Equation 2.6 may be rewritten as Equation 2.7.

$$\vec{\phi} = \arg\max_{\phi_{ij} \epsilon \Phi} \prod_{i=1}^{n} P(\phi_{ij} | \gamma_{ij}, \lambda_{ij}, \Gamma_{c_{ij}}, \Upsilon) \tag{2.7}$$

Formal definitions of all the variables and domains are shown in Tables TODO, respectively.

symbol	meaning	example
Λ	set of possible phrases	{"cube", "sphere"}
Γ	set of possible groundings	{symbolic cube, symbolic sphere}
Φ	set of possible correspondence variables	{Active, Inactive, Inverted}
Υ	set of percepts from world	{RGB image of cube, sonar readings of wall }

Table 2.1: You should know these domains

And here are what what values the variables can take on:

symbol	domain	meaning	example
λ	Λ	a particular phrase	"cube"
γ	Γ	a particular grounding	symbolic cube
ϕ	Φ	a particular correspondence variable	Active

Table 2.2: And here are variables

Finally, it is possible to assemble all the variables in a probabilistic graphical model, with structure inferred from a parse tree. For example, consider the phrase "move to the cube." The parse tree for this command is identical to the one used for G^3, but this time the generated graph is significantly different. Note how, for this relatively simple parse tree, each non-leaf node has exactly two children. This structure of the parse tree, repeated in Figure 2-5, appears inside the DCG graph for the same command, shown in Figure 2-6.

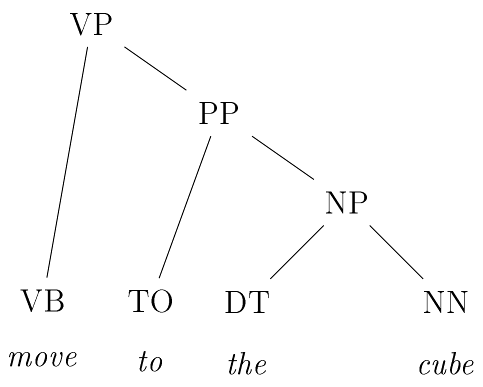


Figure 2-5: Parse tree for the sentence “move to the cube”

Although the graphical model may appear complex, it is actually quite simple. Each parsed phrase appears in a λ_i node at the bottom of the graph. Each λ_i node is connected to a factor linking the phrase to a correspondence variable ϕ_{ij} and a grounding variable γ_{ij}. Each column represents possible groundings for the phrase; in this simplified example, the domain of possible actions is restricted to a single `go_to` action, the domain of preposition groundings is restricted to `within` or `near`, and the domain of noun groundings is restricted to `cube` or `sphere`. Finally, the edges between columns, going from each factor in one column to each ϕ_{ij} and γ_{ij} in the next column, represent the reliance on child groundings. In this case, because the parse tree is binary, each column (except the rightmost one) has exactly one child grounding, so there are only edges between adjacent columns.

To demonstrate how a phrase is grounded with DCG, Figure 2-6 has been repeated and solved in Figure 2-7, this time with ϕs color-coded to green or red, representing whether the solved ϕ is set to true or false, respectively. After solving for all the ϕs, finding the grounded actions or objects is straightforward.

Finally, just as in G^3, DCG uses a log-linear model composed of binary features to the most likely associations between groundings, phrases, and correspondence variables. The factors fall into three groups, one for groundings, one for phrases, and one for correspondence variables. The grounding factors are a function of an object

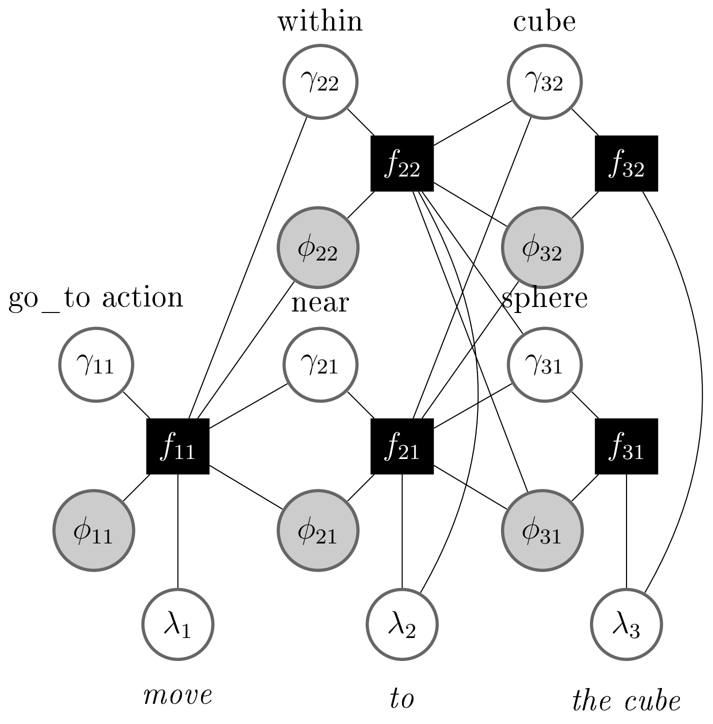


Figure 2-6: DCG diagram generically

or action being grounded to (e.g. whether or not the object is a cube). The phrase factors are similarly only a function of language (e.g. if the word "cube" appears in the phrase). The correspondence variables return whether or not the correspondence variable is true or false. It is precisely by modifying the DCG graph structure and introducing new grounding and phrase features that DCG-UPUP-Away will develop new behaviors.

2.3 Alternative Approaches

While DCG-UPUP-Away builds most directly on G^3 and DCG, other work in natural language robotics has guided the work in this thesis. Research into Human-Robot natural language dialog and question-asking indicates a broad area for future improvements to DCG-UPUP-Away. Separate work indicates how language can be used to intelligently hypothesize about extra-perception objects. Finally, significant progress has been made in answering the question of how to train systems to learn complex

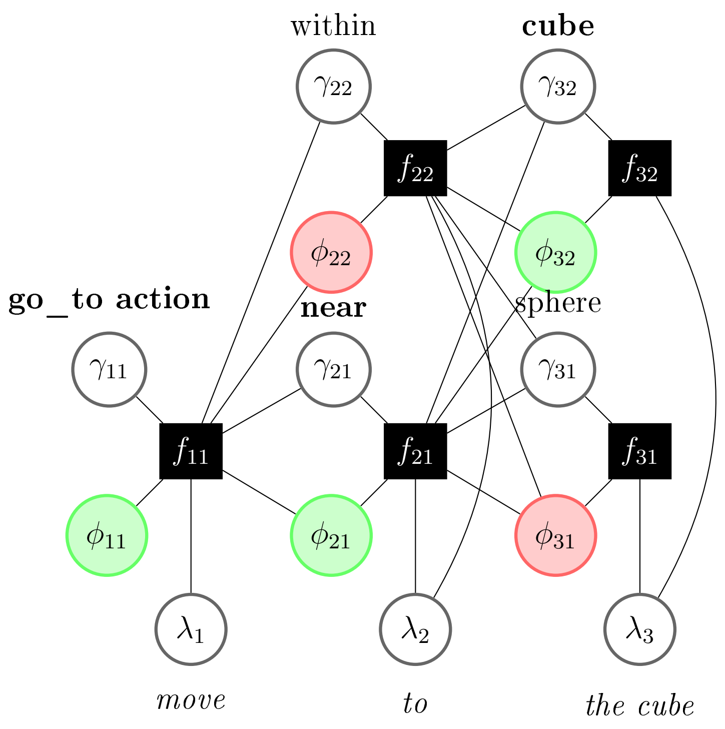


Figure 2-7: DCG diagram as a tikz picture with highlighted phis

semantic relations with as little information as possible.

2.3.1 Human-Robot Dialog

Instead of blindly executing commands, as is done in traditional language grounding research, a natural extension is having robots that talk back. For example, should a human issue an ambiguous command (e.g. "pick up the bottle" when facing two bottles), a robot might ask for more information rather than silently executing a low-confidence grounding.

Such scenarios are considered in [9]. Ros et al. broadly approach resolving language ambiguity using two techniques. First, a robot attempts to model the human's perspective on the scene to determine which objects may be visible to the human. This technique relies on insights from child development studies that show how children employ such reasoning on their own and has been successfully used in other robotics literature. [7][8][14][13] Importantly, this first strategy requires no additional dialog. The second strategy, however, relies on the robot asking a human for more information. For example, the robot may ask for spatial relations or object features in distinguishing between objects. Choosing exactly which question to ask, of course, requires reasoning about what information best discriminates among potential groundings.

Deits et al. explore precisely that issue of how to choose which questions to ask in [12]. As in much research in robotic question-asking, a balance must be struck between too many questions and not enough questions while simultaneously determining what sorts of question to ask [3][10]. [12] use the entropy of the probability distribution over groundings to estimate the grounding uncertainty. Higher entropy, therefore, leads to more questions. As expected, asking questions leads to a greater grounding accuracy rate for all sorts of questions asked.

Such research provides a clear path for future improvements upon DCG-UPUP-Away, which offers no dialog. In fact, [12] interfaces nicely with G^3 and thus may presumably be easily adapted to operate with DCG or DCG-UPUP-Away.

2.3.2 Object Hypotheses for Unknown Environments

Another field of study closely tied to the field of language grounding explores the topic of inferring information about unknown environments using language. For example, if a robot is place in a room and told to "pick up the ball outside the door," a robot should be able to easily deduce that a ball is located near the door. A more formal framing of that intuition indicates that grounding systems should reason over a belief distributions over objects rather than a deterministic known map.

Duvallet et al. use such a framework to propose a latent map that is partially observed by language [2]. Thus, in the example of a ball outside the door, the phrase "pick up the ball outside the door" generates a region of high probability near the door and low probability further away. Sampling from this distribution, as well as updating the distribution as more observations are made, yields a useful map to plan in. Similarly, other work generates distributions over maps or exactly placing objects in unknown environments if their locations are uniquely described [15][16].

In the absence of localizing language, however, these sophisticated techniques provide relatively little information about where an object may be located. Therefore, because this research is restricted to simple phrases that lack spatial relations, DCG-UPUP-Away uses a far simpler technique for hypothesizing objects in unknown locations. Future development, however, could incorporate some of the methods presented for more intelligent reasoning about unperceived objects.

2.3.3 Learning through Groundings

Joint model for learning via grounding: [6].
Learning nav instructions: [1].

2.3.4 Non-probabilistic Groundings

Earlier work by stefanie with sdcs [5].

Chapter 3

Grounding to Unknowns

`έοκια γοῦν τούτου γε σμικρῶ τινι αύτῶ τούτω σοφώτερος ε`ἵναι, `ότι `ά μὴ ο`ἵδα ούδὲ ο`ἵομαι είδέναι
"Indeed I seem to be wiser than him in this little matter in that I don't think I know what I don't know"
–Plato's Apology

Despite the impressive work done by Howard et al. in DCG, the natural language model they developed only reasons about objects and phrases that the system has already been trained on. For example, if DCG is taught how to ground the phrase "chair" and to recognize chairs in the real world, it can subsequently operate in an environment with chairs. DCG fails, however, to reason about new phrases or percepts in the absence of relevant training. In the same scenario as previously described (DCG has been thoroughly trained with chairs, but only chairs), if a robot is placed in a room full of chairs and a single sofa and is told "drive to the sofa," nothing happens. Given the context, however, any human would be able to deduce that "sofa" refers to the only unknown object in the scene - the sofa.

In this chapter, the DCG model is upgraded to DCG-UPUP (short for Distributed Correspondence Graph - Unknown Phrase Unknown Percept). The main technical insight is that explicitly recognizing when phrases or percepts are unknown allows for intelligent learned behavior in new scenes. The remainder of this chapter is divided among recognizing when phrases are unknown, recognizing when percepts are unknown, training the model to associate unknown phrases to unknown percepts, and permanently learning new symbols. By the end of chapter, DCG-UPUP will be able to handle exactly the chair-sofa DCG failure case described previously.

3.1 Unknown Phrases

A necessary step in learning to associate unknown phrases with unknown percepts is first recognizing when a phrase is unknown. For the sake of simplicity, development was limited to recognizing unknown nouns; expansions to verbs will be discussed in the future work section.

Before immediately recognizing unknown nouns, one must define what it means for a phrase to be unknown. In the context of the grounding problem, a natural definition arises based on whether or not the system has been trained on how to ground that noun. Thus, if the system has been trained to ground a phrase, that specific phrase is known – otherwise the phrase is unknown.

Keeping track of which nouns are known is similarly straightforward. The training procedure for the LLM (the log-linear model used for DCG) is augmented to also generate a set of all the nouns seen in training that were grounded to a known object type. (For now, the requirement that the phrase grounds to a known type may seem odd given that no unknown types have been introduced yet, but that will change in the next section.) The built up set of known phrases, henceforth referred to as *known_words*, therefore is a snapshot at any given moment of exactly which nouns are known. Note that *known_words* grows over the course of the LLM training, so a phrase that is unknown a quarter of the way through training may be known by the end.

For the purposes of adapting the DCG model, the final step in recognizing a noun as unknown is adding a feature that fires when an unknown phrase is heard. Using the definition of unknown phrases and the construction of the *known_words* set, one may easily see that the *feature_unknown_phrase* feature should fire precisely when a noun is not found within *known_words*. From an implementation perspective, this new feature - *feature_unknown_phrase* - is quite similar to many of the other features already used to train the LLM; from a theoretical standpoint, however, it is the only feature that is self-reflective, in a sense, and requires a model of its own intelligence.

3.2 Unknown Percepts

In parallel to detecting when a phrase is unknown, DCG-UPUP must recognize when a percept is unknown. In order to match up with unknown nouns, percepts are restricted to objects, in which case recognizing unknown percepts becomes the simpler task of detecting unknown objects (as opposed to unknown actions, for example).

Once again, before attempting to recognize unknown percepts, one must first define what it means for a percept to be "unknown." Just as phrases are defined as unknown if they never appear in training examples, one may similarly say that objects of a type that never appeared in training data are unknown. For example, if an object recognizer has been trained to recognize cubes, spheres, and cylinders, any instance of a cube, sphere, or cylinder would be known but a cone would be unknown. Of course, such a definition merely translates the problem of deciding whether an object is unknown into the problem of deciding how well it matches a known category. There are myriad ways to quantify how well objects match known categories: support vector machines (SVMs) report distances from boundaries while other multi-class detectors report probability distributions over known classes.

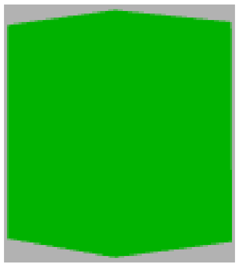
In this particular implementation, an image classifier based on the Histogram of Orientated Gradients (HOG) is used as part of a three-step method. First, an object proposal toolbox from [17] generates tight bounding boxes around objects in an RGB image of the scene. Second, each bounding box is fed into a standard SVM classifier offered by MATLAB that uses HOG features as well as the aspect ratio of the object. Third, given the reported distances from the boundaries in the SVM, if the classifier appears uncertain, the object is labeled as "unknown" – otherwise the most likely classification is used.

The confidence of the classifier is approximated by examining the difference between the signed margin of the most likely classification and the second most likely classification. Since the SVM is a discriminative model among non-overlapping classes, the first margin is always positive and the second is always negative. Thus, objects that fall squarely within a category generate large differences while objects near classification borders yield small differences. Comparing this difference to a threshold allows one to say whether or not an object is unknown. TODO cite how this technique is used in other situations. As the number of classes grows, however, classification margins necessarily shrink, so some method of normalizing the threshold by the number of categories is required. The general form of the final classification equation used in this experiment is shown in Equation 3.1 below:

$$classification = \begin{cases} \text{``unknown''}, & \text{if } m\_diff < \frac{m\_threshold}{(num\_classes-1)^{norm\_exp}} \\ known\_class, & \text{otherwise} \end{cases} \tag{3.1}$$

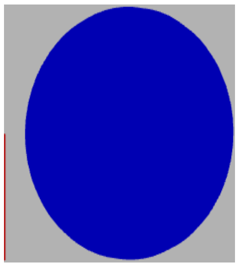
where m_diff is the difference between the margins of the best and second-best classifications and $known\_class$ is the class that maximizes the margin. Finally, by playing with intercept and normalization parameters until sample cases pass, one may generate the actual equation used for determining the unknown cutoff.

In this experiment, the best values for m_diff and $norm\_exp$ were found to be 0.6 and 1.2, respectively. There is absolutely no theoretical motivation for these numbers; because object recognition is not the emphasis of this research, though, all that is required is something that works most of the time. The values for m_diff and $norm\_exp$ were generated by systematically testing the classifier with several images of both known and unknown objects. Figure 3-1 shows sample test images of a cube, sphere, and fire hydrant. Below each image are the classification margins generated by a classifier trained only on cubes, spheres, and cylinders.


(a)

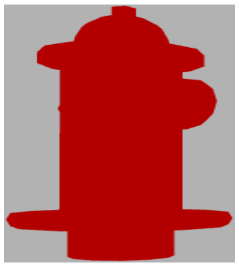
cube	**0.2281**
cylinder	-0.2281
sphere	-0.5657

A test image of a cube, with margins.


(b)

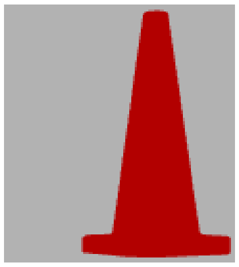
cube	-0.2959
cylinder	-0.6862
sphere	**0.2959**

A test image of a sphere, with margins.


(c)

cube	-0.0584
cylinder	-0.3859
sphere	**0.0584**

A test image of a fire hydrant (unknown), with margins.


(d)

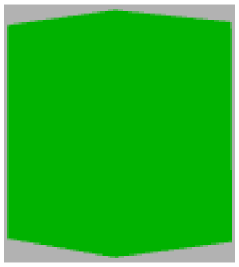
cube	-0.0443
cylinder	-0.1274
sphere	**0.0443**

A test image of a construction (unknown), with margins.

Figure 3-1: Sample generated test images. The bold classifications are the most likely known classification type. Note how the unknown fire hydrant and cone have very small margins, resulting in a final classification of "unknown."

A new image classifier, trained on cubes, spheres, cylinders, **and hydrants** results in different behavior: this time the fire hydrant is classified as a known type, but the cone remains unknown. Classification results and margins for that classifier are shown in Figure 3-2.

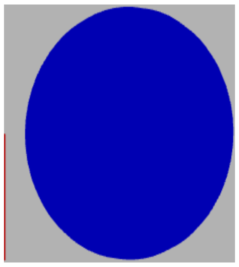
The final modification made to the object recognizer was adding color recognition. Because the HOG classifier is trained using color-invariant features, only the object type (e.g. "cube" or "unknown") is reported. Given the simplicity of the simulated environment, however, it is easy to average the RGB values across the cropped image of each object and report the greatest color value. Thus, the final object classifier returns the object type as well as color.



(a)

cube	**0.1160**
cylinder	-0.1160
hydrant	-0.4624
sphere	-0.2915

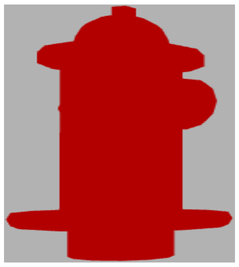
A test image of a cube, with margins.



(b)

cube	-0.1856
cylinder	-0.3956
hydrant	-0.3853
sphere	**0.1856**

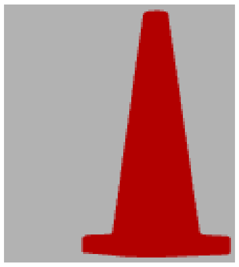
A test image of a sphere, with margins.



(c)

cube	-0.2775
cylinder	-0.4418
hydrant	**0.2497**
sphere	0.2497

A test image of a fire hydrant, with margins.



(d)

cube	-0.0435
cylinder	-0.0730
hydrant	-0.0852
sphere	**0.0435**

A test image of a construction (unknown), with margins.

Figure 3-2: Sample generated test images. The bold classifications are the most likely known classification type. The classifier has been trained on cubes, spheres, cylinders, **and hydrants**, so only the cone is unknown.

3.2.1 Alternate Approaches

It is important to emphasize that, while a specific object recognizer and unknown-detection mechanism have been proposed here, *any* object recognizer that can recognize objects and report when they are unknown is acceptable. In fact, initial development of DCG-UPUP used a SURF-based object recognizer that returned a probability distribution over known classes. In that instance, rather than using the difference of the top-two confidences to detect unknowns, the normalized entropy of the distribution was used. When the SURF object recognizer was given an object that did not clearly fall into a known category, the probability distribution tended to flatten across all categories. Thus, the entropy would increase, likely indicating an unknown object. Ultimately, this approach using SURF features was abandoned because it did not seem to distinguish well between objects when using few training examples.

It is likely that more reliable and faster object detectors exist that could increase the robustness of this work. However, the purpose of this research is not instantly deployable robots in the real world, merely to demonstrate a proof of concept. Future researchers, however, should incorporate more reliable object detectors before deploying in non-laboratory environments. In fact, during while evaluating the full DCG-UPUP-Away system, the majority of the failure cases arose simplify from improper object classification, so most results had to be generated using manual object labeling.

3.3 Training Unknown Phrase - Unknown Percept Associations

It is not enough, however, to merely recognize phrases and percepts as unknown; the DCG-UPUP model must be trained to ground unknown phrases to unknown objects. Although this may appear to be a daunting task, all the necessary framework is already in place. For one thing, *feature_unknown_phrase* already detects when phrases are unknown. Secondly, the original version of DCG already has features that detect what object type is being grounded to. All that is necessary, therefore, is to train the association between *feature_unknown_phrase* firing and a feature for object type "unknown_type".

In total, seven training examples were added to train the unknown_phrase - unknown_percept association. Each example uses a new word that has never been grounded to a known type (e.g. "hammer," "mallet," "bottle," and "crate"). Each file has exactly the same format as other training files, except that instead of grounding to a known data type such as cube_type, the noun is grounded to unknown_type. Crucially, by varying which unknown noun is used but continuing to ground to the unknown_type, the LLM learns that the exact phrase does not matter - only whether or not the noun appears in the known_words file.

These changes alone are enough to solve the sofa and chairs problem presented at the beginning of this section. In brief, the situation involves a robot that has only been trained with chairs but is facing a sofa and several chairs being told "go to the sofa." DCG-UPUP behaves exactly as desired:

1. The object detector segments out the sofa and chairs from its input image
2. The object classifier returns that each chair is a chair and the sofa is an unknown object
3. The *feature_unknown_phrase* returns that the noun "sofa" is unknown
4. The trained DCG-UPUP model recognizes that unknown phrases are often associated with unknown objects and returns that "sofa" is referring to the sofa

Despite its seeming simplicity, the importance of this achievement should be understated. DCG-UPUP has learned, without any hardcoding, how to use process of elimination, a procedure that exam-studying highschoolers often forget.

3.3.1 Redone Equations

Although a high level view of DCG-UPUP has been presented, the real difference between DCG and DCG-UPUP is revealed by looking at the domains of the variables in the new DCG-UPUP graphical model. First, it appears that the underlying equation that both DCG and DCG-UPUP are trying to solve remains the same.

DCG	$\vec{\phi} = \underset{\phi_{ij} \epsilon \Phi}{\arg\max} \prod_{i=1}^{n} P(\phi_{ij} \vert \gamma_{ij}, \lambda_{ij}, \Gamma_{c_{ij}}, \Upsilon)$
DCG-UPUP	$\vec{\phi} = \underset{\phi_{ij} \epsilon \Phi}{\arg\max} \prod_{i=1}^{n} P(\phi_{ij} \vert \gamma_{ij}, \lambda_{ij}, \Gamma_{c_{ij}}, \Upsilon)$

Table 3.1: Equations for DCG and DCG-UPUP

The crucial difference is the domains for λ, γ, and ϕ. In the next two tables, the domains for language and groundings are expanded to represent unknown phrases and objects, respectively.

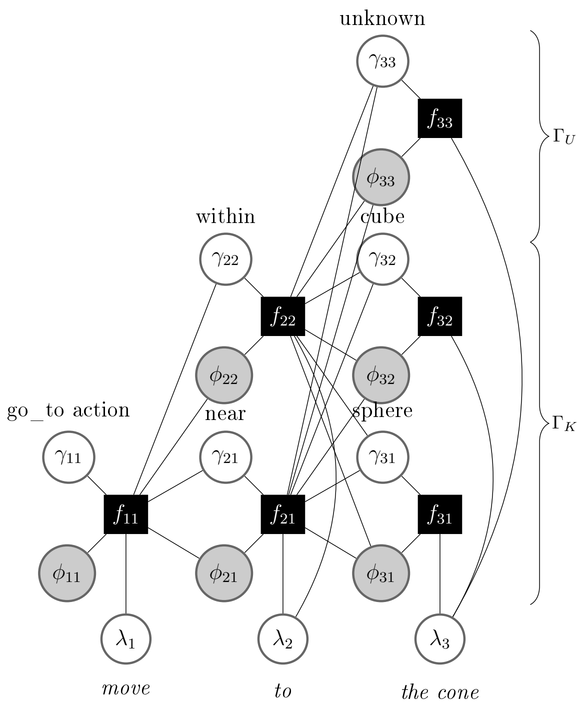
symbol	meaning	example
Λ_K	set of known phrases	{“cube”, “sphere”}
Λ_U	set of unknown phrases	{“cone”, “thingy”}
Γ_K	set of known groundings	{symbolic cube, symbolic sphere}
Γ_U	set of size one for unknown grounding	{symbolic unknown}
Φ	set of possible correspondence variables	{True, False}
Υ	set of percepts from world	{RGB image of cube, sonar readings of wall }

Table 3.2: You should know these domains

symbol	domain	meaning	example1	example2
λ	$\Lambda_K \cup \Lambda_U$	a particular phrase (known or unknown)	“cube”	“cone”
γ	$\Gamma_K \cup \Gamma_U$	a particular grounding	symbolic cube	symbolic unknown
ϕ	Φ	a particular correspondence variable	True	False

Table 3.3: And here are variables

Finally, one may assemble the probabilistic graphical model for DCG-UPUP, shown in Figure 3.3.1, using the expanded domains. The unknown object grounding appears in the graph as an extra element on top of the rightmost column, containing the noun.



By varying the natural language command, one may see how DCG-UPUP is solved for both known and unknown nouns. In this scenario, DCG-UPUP has been trained with cubes, spheres, and cylinders. The robot faces a cube, sphere, and cone, which is recognized as unknown.

3.4 Unsupervised Learning and Concept Acquisition

In the previous section, DCG-UPUP deduced that unknown phrases are associated with unknown objects, so when it hears "sofa" and sees a sofa for the first time, it claims that "sofa" must be referring to the sofa. Without any learning mechanism in place, however, the next time the robot sees a sofa or hears the phrase "sofa," it will once again treat the percept and phrase as unknown. This is undesirable. Instead, ideally the robot would remember what sofas look like and that the word "sofa" refers to sofas. In other words, the robot should learn about a symbolic sofa.

Perhaps the simplest way to teach DCG-UPUP about the new symbol is to gen-

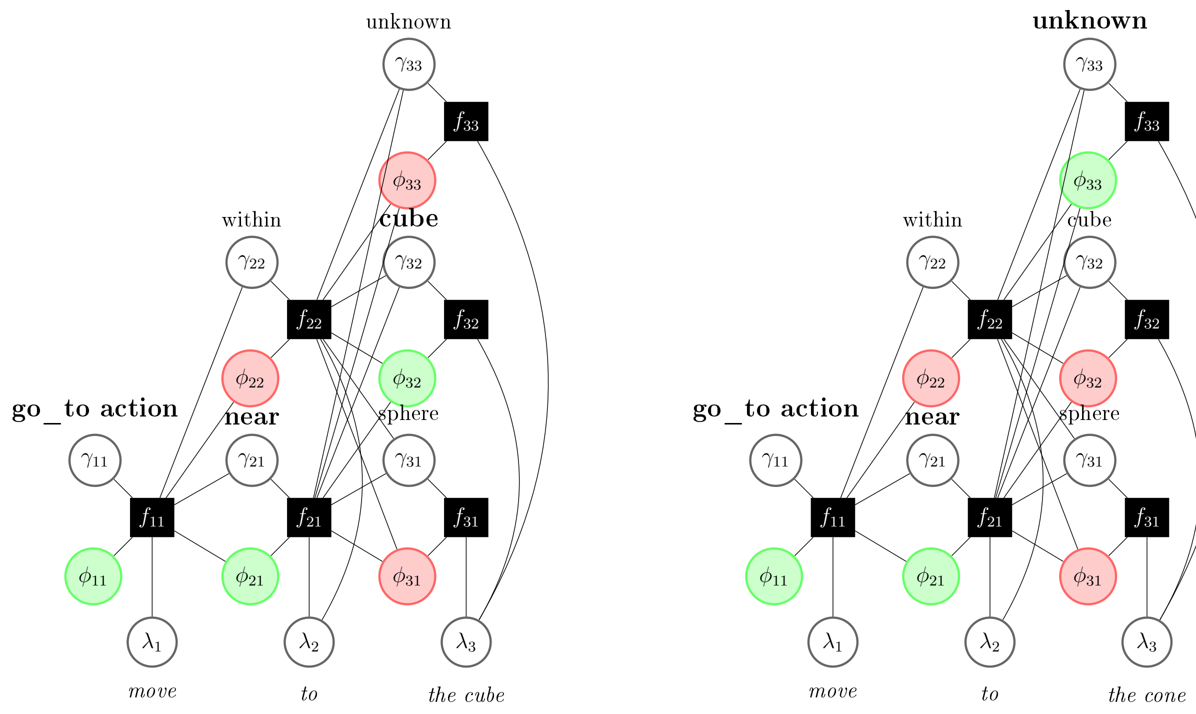


Figure 3-3: DCG-UPUP solved for both known and unknown phrases

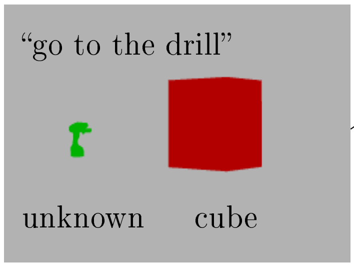
erate new training examples, with the relevant new phrase and a new object type, and create a new LLM using the original and new training files. (It may be possible to efficiently update the LLM rather than rebuilding it from scratch, but that would only change the runtime of the program, not its overall behavior.) Generating the training examples, too, is relatively straightforward: the output of DCG-UPUP, supplemented with a description of the full environment, exactly matches the format of a training file. In other words, DCG-UPUP can autonomously generate new training files that will permanently teach it new symbols.

There are, of course, several subtleties to take care of. On the one hand, the object recognizer must be retrained to correctly classify the new object, but that can be achieved by saving a snapshot of each object and relabeling and retraining whenever a new type is created. On the other hand, DCG-UPUP must allow for a growing set of object types, but that can be implemented by storing all object types in a growing set. The new object type, and the new phrase, must finally be incorporated into the feature set so that the LLM can reason about the new symbol, but adding features automatically is extremely straightforward. The rest of the learning (e.g. updating known_words to contain the new phrase) occurs automatically when the LLM is retrained.

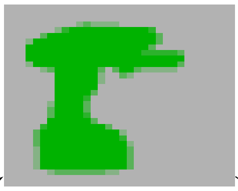
In order to improve the robustness of both the language grounding and object recognition programs, this unsupervised learning technique is used after every natural language command. (The LLM and object recognizer are retrained only when manually commanded.) This delayed retraining is particularly useful for the object recognizer, which often struggles to correctly classify objects based on only a single sample image of an object type. Instead, if the robot saves several snapshots of an object as it drives towards it, the robot may generate a more complete and robust image training set.

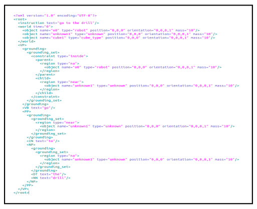
Finally, it is important to note there are dangers with this unsupervised learning approach. As with any fully unsupervised learning technique, concept drift risks worsening the performance of the system over time. For example, if DCG-UPUP has been trained to recognize cubes, spheres, and cylinders and hears the command "go to the fire hydrant," the unknown phrase "fire hydrant" will ground to the *first* unknown object that the robot sees. If that object is not a fire hydrant but a cone, after retraining, the object recognizer will subsequently perceive cones as fire hydrant types. This error will only worsen as the robot continues to ground phrases since "fire hydrant" will no longer be an unknown phrase, and cones will be perceived as fire hydrant types.

The obvious mitigation for concept drift is transforming the unsupervised learning into semi-supervised learning. A human user, for example, could confirm or deny proposed new training examples. In fact, significant research in Information-Theoretic Questions has already been conducted and would fit well with this work (TODO cite). Such research, however, must wait for future work.



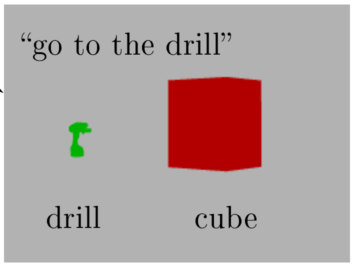
(a) DCG-UPUP, trained on cubes, spheres, and cylinders, is told to “go to the drill” and recognizes the “drill” and the physical drill as unknown.





```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <instruction text="go to the drill"/>
  <world time="0">
    <object name="o0" type="robot" position="0,0,0" orientation="0,0,0,1" mass="10"/>
    <object name="unknown1" type="unknown" position="0,0,0" orientation="0,0,0,1" mass="10"/>
    <object name="cube1" type="cube_type" position="0,0,0" orientation="0,0,0,1" mass="10"/>
  </world>
  <VP>
    <grounding>
      <grounding_set>
        <constraint type="inside">
          <parent>
            <region type="na">
              <object name="o0" type="robot" position="0,0,0" orientation="0,0,0,1" mass="10"/>
            </region>
          </parent>
          <child>
            <region type="near">
              <object name="unknown1" type="unknown" position="0,0,0" orientation="0,0,0,1" mass="10"/>
            </region>
          </child>
        </constraint>
      </grounding_set>
    </grounding>
    <VB text="go"/>
    <PP>
      <grounding>
        <grounding_set>
          <region type="near">
            <object name="unknown1" type="unknown" position="0,0,0" orientation="0,0,0,1" mass="10"/>
          </region>
        </grounding_set>
      </grounding>
      <IN text="to"/>
      <NP>
        <grounding>
          <grounding_set>
            <region type="na">
              <object name="unknown1" type="unknown" position="0,0,0" orientation="0,0,0,1" mass="10"/>
            </region>
          </grounding_set>
        </grounding>
        <DT text="the"/>
        <NN text="drill"/>
      </NP>
    </PP>
  </VP>
</root>
```

(b) A snapshot of the drill is saved for retraining the object classifier, and a new training example is written from the grounding output of DCG-UPUP. TODO add border



(c) After retraining, DCG-UPUP has learned to recognize drills and that “drill” grounds to drills.

Figure 3-4: Unsupervised learning allows DCG-UPUP to first recognize drills as unknown and later as their own type

3.5 Synonyms

In its current proposed form, DCG-UPUP may be easily adapted to yield one other important result: by assuming that phrases only refer to objects the robot can currently perceive, DCG-UPUP can learn synonyms. Unfortunately, this assumption will be violated in Chapter 4, so this work will be mostly discarded, but it is still important to note, if only for indicating an area for future work.

Synonyms easily fit into the grounding framework already presented. When grounding a synonym, the object type is already known, but the phrase is different. If the robot may assume that phrases only refer to objects it can currently perceive, one must only consider two possible cases: 1) the robot perceives one or more unknown object or 2) the robot does not perceive any unknown objects. In the first case, DCG-UPUP favors grounding unknown phrases to unknown objects, so it will try to choose the best candidate unknown object. In the second case, DCG-UPUP must ground an unknown phrase to a known object, so, similarly, must choose the best candidate known object. Heuristics used for breaking such ties are presented in Chapter 5.

The behavior in the second case is exactly what is necessary for learning synonyms. An unknown noun (e.g. "ball") is associated with a known object (e.g. a sphere). By using the same unsupervised learning trick of retraining on grounding outputs, the new phrase may be permanently learned. In fact, early experiments with this technique showed promising results: the first time DCG-UPUP grounds "ball" to a sphere, it has relatively low confidence, but after two or three groundings (and retraining) to a sphere, confidence for grounding "ball" rapidly approaches that of grounding "sphere."

Unfortunately, as stated previously, the assumption that made synonym-learning possible (i.e. that phrases only refer to objects seen by the robot) forced synonym capabilities to be set aside for future development.

Chapter 4

Grounding out of Perception

Regardless of how well the robot may ground to objects it perceives, performance of DCG-UPUP is limited unless the robot can ground to objects that it cannot perceive. For example, if a robot is told "go to the staircase behind the wall," if the robot can see a wall but cannot see behind it, the robot should deduce roughly where the staircase probably is. In an even simpler scenario, imagine that a robot that has been trained to ground cubes and spheres is facing only cubes and is told "go to the sphere." Once again, the robot should be able to deduce that in such a scenario, it must explore outside its current field of view rather than finding the most sphere-like cube. A new version of DCG-UPUP, called DCG-UPUP-Away (for Distributed Correspondence Graph, Unknown Phrase Unknown Percept, Away) is presented that explicitly allows a robot to ground to objects outside its field of view.

The majority of this chapter will discuss various approaches initially developed to adapt DCG-UPUP for out-of-world groundings. Early model changes, although potentially promising for future research, were discarded for a more elegant and robust approach that involves "hypothesizing" objects outside the field of view. Finally, this chapter concludes by explaining how natural language feedback from the robot to the human could be integrated for more robust performance.

4.1 Alternate Approaches

4.1.1 Default Out-Of-World Groundings

The first approach developed for deciding when to ground to out-of-world objects was based on a sort of default behavior: if the confidence of the best grounding falls below a certain threshold, the robot should explore the world. This simple method has the advantage of not changing the DCG-UPUP model and therefore scarcely affecting runtime. Furthermore, it appears to reflect how people may decide to look for objects; if one is told to find a cube yet sees nothing that looks like a cube, say, one might walk around until finding a good match.

Although such behavior may seem to reflect how humans think, in fact, upon further consideration, this appearance disintegrates. Default behavior and human behavior diverge when phrases either match several objects very well, or no objects very well. Consider, for example, telling the robot to "pick up the cube" when it is facing 20 cubes. If every cube is equally likely to be the correctly grounded object, then the probability of the most likely grounding will be around 5%. If the threshold for an out-of-world grounding is greater than 5%, therefore, the robot will erroneously decide to look for a cube other than the 20 it currently sees. Clearly, a similarly difficult environment may be constructed for any threshold, leading to failure.

In addition, this approach is unattractive simply because it is not a learned behavior. Ideally, DCG-UPUP-Away would inherently decide where to ground to not through a set of pre-defined rules, but rather through seeing several training examples and generating its own rules. Having a default threshold violates this principle.

4.1.2 Symbolic Representation of Out-Of-World Groundings

Another approach explored in early development was to have a symbol, *out_ of_ world* that represents all possible out-of-world groundings. Therefore, if DCG-UPUP-Away grounds to *out_ of_ world*, the robot decides to explore a new part of the environment.

This appears to be a satisfactory approach that matches the thresholding idea in the previous section, but with the theoretical advantage of being a learned behavior. In order to teach DCG-UPUP-Away when to ground out of the world, training examples are constructed in which nouns are grounded to the *out_ of_ world* symbol.

There remain, however, two problems with this approach. First, using a single symbol to represent all possible object types outside the current field of view seems like an inaccurate representation of how people decide whether or not a phrase refers to something they cannot see. Secondly, after some efforts retraining the system, performance remains frustratingly poor. Perhaps investing more time into choosing the right training examples could close this gap, but that must be postponed to future work.

4.2 "Hypothetical" Approach

The solution used for the final version of DCG-UPUP-Away addresses the problems in the previous two approaches. The main idea is to create a hypothetical object for each possible object type. If DCG-UPUP-Away grounds to a hypothetical object rather than an actual object that the robot perceives, the robot decides to explore new areas.

More specifically, DCG-UPUP is changed before any inference is actually conducted. First, DCG-UPUP-Away populates a world model Υ based on what objects are currently perceived. This is exactly what happens in DCG and DCG-UPUP. Second, DCG-UPUP-Away iterates through each possible object type (maintained as a growing set, as explained in Chapter 3) and adds a single hypothetical instance of each object to the world model. For the sake of computational speed and simplicity, only a single hypothetical instance of each object type is added, but that assumes that natural language commands will not refer to several out-of-world objects. This assumption can be relaxed by creating several hypothetical objects of the same type, but is outside the scope of this experiment.

After adding the hypothetical objects to the world model, inference proceeds as in DCG-UPUP, but with the new world model Υ'. If the grounded object is a hypothetical object, the robot must explore its surroundings; otherwise it can see the grounded object. Currently, "exploring" surroundings consists of slowly rotating in place. Future work could easily yield more sophisticated behavior by adding exploratory heuristics or even using language to guide search (as in [2]).

Intuitively, this approach finally matches how people decide whether or not a phrase refers to something they cannot currently see. Upon hearing a phrase, it is natural to compare how well that phrase matches each object currently seen (grounding within the world) or each object not seen (grounding outside the world). For example, when facing a cube and hearing the phrase "go to the cube," the phrase "cube" matches the cube better than, say, an imaginary sphere outside the field of view. Conversely, when facing a cube and hearing the phrase "go to the sphere," the phrase "sphere" does not match the cube but would match a sphere very well if it existed outside the field of view, so the natural decision is to look elsewhere for the sphere.

There is, however, one subtlety worth considering: a phrase may refer to an object that is currently outside the field of view, but within the field of view there is an object of that same type. Imagine, for example, a robot located between two cubes, only looking at one. When told "go to the cube," what should the robot do? On the one hand, the cube in front of it is a high probability grounding for the phrase "cube." At the same time, a hypothetical cube would match "cube" with equally high probability!

One way to resolve this subtlety is to introduce an additional feature and training examples into DCG-UPUP-Away. The feature – *feature_hypothetical_object* – fires when the grounded object is hypothetical, but does not fire if the object is perceived. Only a handful of training examples demonstrating tie-breaking between real and hypothetical objects are necessary to train DCG-UPUP-Away to favor real objects.

4.2.1 Redone Equations

Just as when comparing DCG-UPUP to DCG, one may compare DCG-UPUP-Away to DCG by examining the domains of variables used in solving the grounding problem. Once again, the underlying equation used for finding the optimal grounding remains the same.

DCG	$\vec{\phi} = \arg\max_{\phi_{ij} \epsilon \Phi} \prod_{i=1}^{n} P(\phi_{ij} \mid \gamma_{ij}, \lambda_{ij}, \Gamma_{c_{ij}}, \Upsilon)$
DCG-UPUP	$\vec{\phi} = \arg\max_{\phi_{ij} \epsilon \Phi} \prod_{i=1}^{n} P(\phi_{ij} \mid \gamma_{ij}, \lambda_{ij}, \Gamma_{c_{ij}}, \Upsilon)$
DCG-UPUP-Away	$\vec{\phi} = \arg\max_{\phi_{ij} \epsilon \Phi} \prod_{i=1}^{n} P(\phi_{ij} \mid \gamma_{ij}, \lambda_{ij}, \Gamma_{c_{ij}}, \Upsilon)$

Table 4.1: Maximization equations for all DCG, DCG-UPUP, and DCG-UPUP-Away

This time, the domain of γ has grown still larger, however, to reflect the set of possible hypothetical groundings. In essence, Γ now includes all perceived objects as well as all possible hypothetical objects.

symbol	meaning	example
Λ_K	set of known phrases	{“cube”, “sphere”}
Λ_U	set of unknown phrases	{“cone”, “thingy”}
Γ_{KP}	set of known perceived groundings	{symbolic perceived cube, symbolic perceived sphere}
Γ_{UP}	set of size one for unknown perceived grounding	{unknown perceived object}
Γ_{KH}	set of known hypothetical groundings	{symbolic hypothetical cube, symbolic hypothetical sphere}
Γ_{UH}	set of size one for unknown hypothetical grounding	{unknown hypothetical object}
Φ	set of possible correspondence variables	{True, False}
Υ	set of percepts from perceived world	{RGB image of cube, sonar readings of wall }

Table 4.2: You should know these domains

symbol	domain	meaning	example1	e
λ	$\Lambda_K \cup \Lambda_U$	a particular phrase (known or unknown)	"cube"	
γ	$\Gamma_{KP} \cup \Gamma_{UP} \cup \Gamma_{KH} \cup \Gamma_{UH}$	a particular grounding	symbolic cube	symb
ϕ	Φ	a particular correspondence variable	True	

Table 4.3: And here are variables

Finally, one may assemble the variables and domains into one large graphical model representing DCG-UPUP-Away, show in Figure 4-1. Note just how large the last column has grown in order to represent known and unknown objects that are either perceived or hypothetical. Following Figure 4-1, 4 solved DCG-UPUP-Away graphs are shown covering the 4 possible cases: grounding to a known, perceived object, to an unknown, perceived object, to a known, hypothetical object, and to an unknown, hypothetical object. In all cases, the model has been trained only with cubes, spheres, and cylinders. Here is the DCG-UPUP-Away graph.

Here is my new version of the DCG-UPUP-Away solved for a known, perceived phrase:

Here it is for an unknown, perceived phrase:

Here it is for a known, unperceived phrase:

Here it is for a known, unperceived phrase:

4.3 Potential Improvement through Natural Language Feedback

As mentioned at the end of Chapter 3, DCG-UPUP originally allowed for learning synonyms, but only under the assumption that phrases only refer to objects in the current field of view. That assumption is relaxed in DCG-UPUP-Away, and a new problem arises: how can one distinguish between a synonym for a known object that is present in the world and a new word referring to a new object that is out of the field of view of the robot. In fact, without outside information, it is impossible to distinguish between the two cases.

One possible way to resolve this problem is through natural language feedback from human users. For example, instead of finding the single most likely grounding, DCG-UPUP-Away could propose two groundings: the most likely grounding to a perceived object, and the most likely grounding to a hypothetical object. The human could then specify which of the two groundings to choose.

In fact, a crude version of this system was briefly implemented (users could input 0 or 1 to distinguish between the perceived or hypothetical grounding). Later iterations of DCG-UPUP-Away, however, rolled back this feature for design simplicity. Future versions could easily re-introduce the feedback feature and even add heuristics for improved behavior. For example, the robot could consider the relative likelihoods of the perceived grounding and a hypothetical grounding before asking a question (e.g. if the robot is quite confident that the correct grounding is hypothetical because there are no perceived objects, it should not ask the user a question).

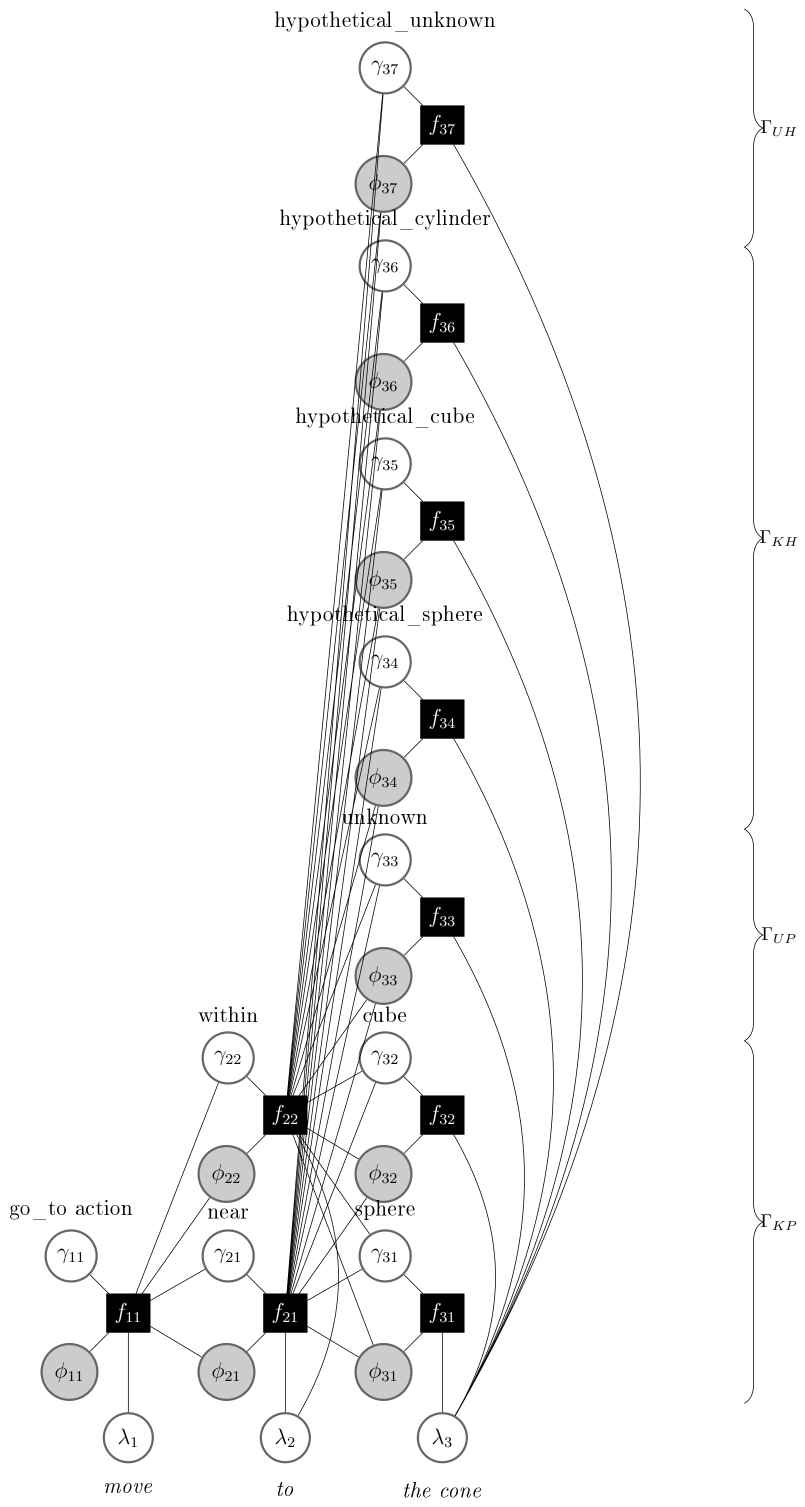


Figure 4-1: General version of DCG-UPUP-Away diagram. Note the hypothetical objects.

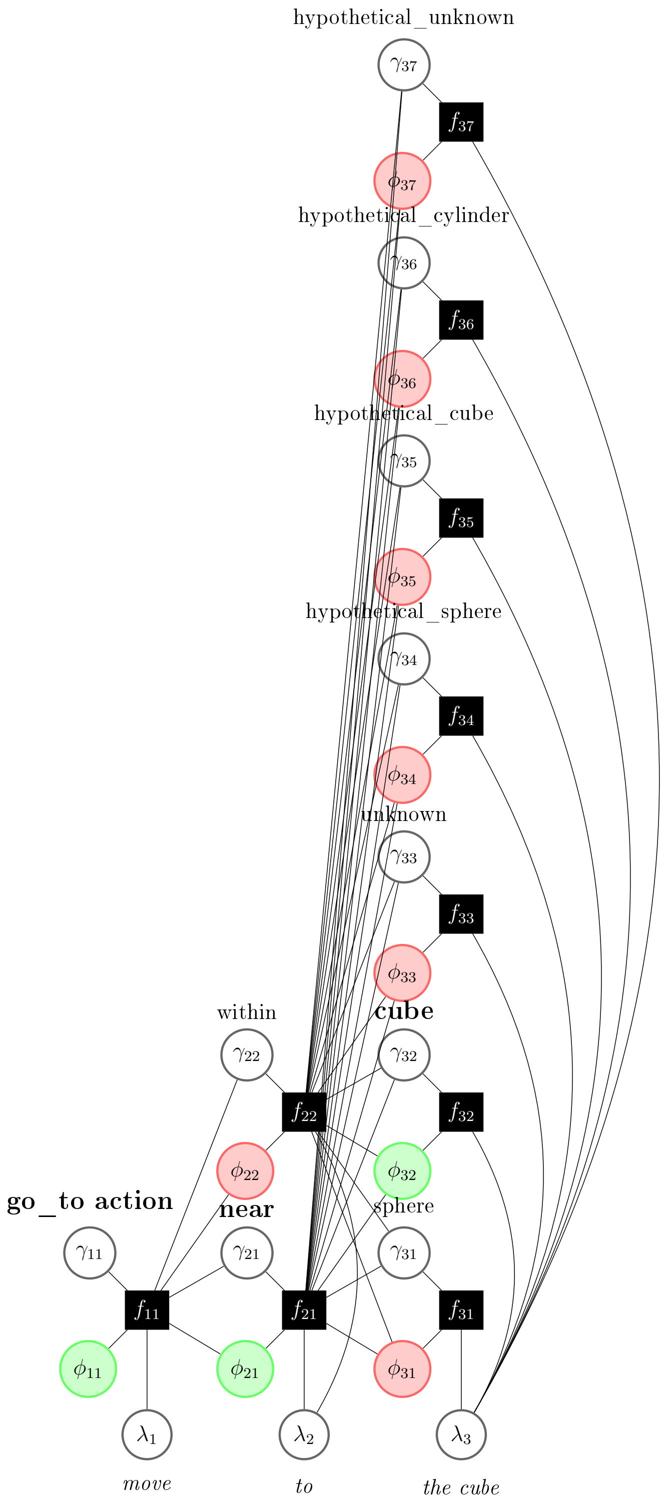


Figure 4-2: DCG-UPUP-Away model solved for a phrase that is known ("cube") when a cube, sphere, and an unknown object are perceived

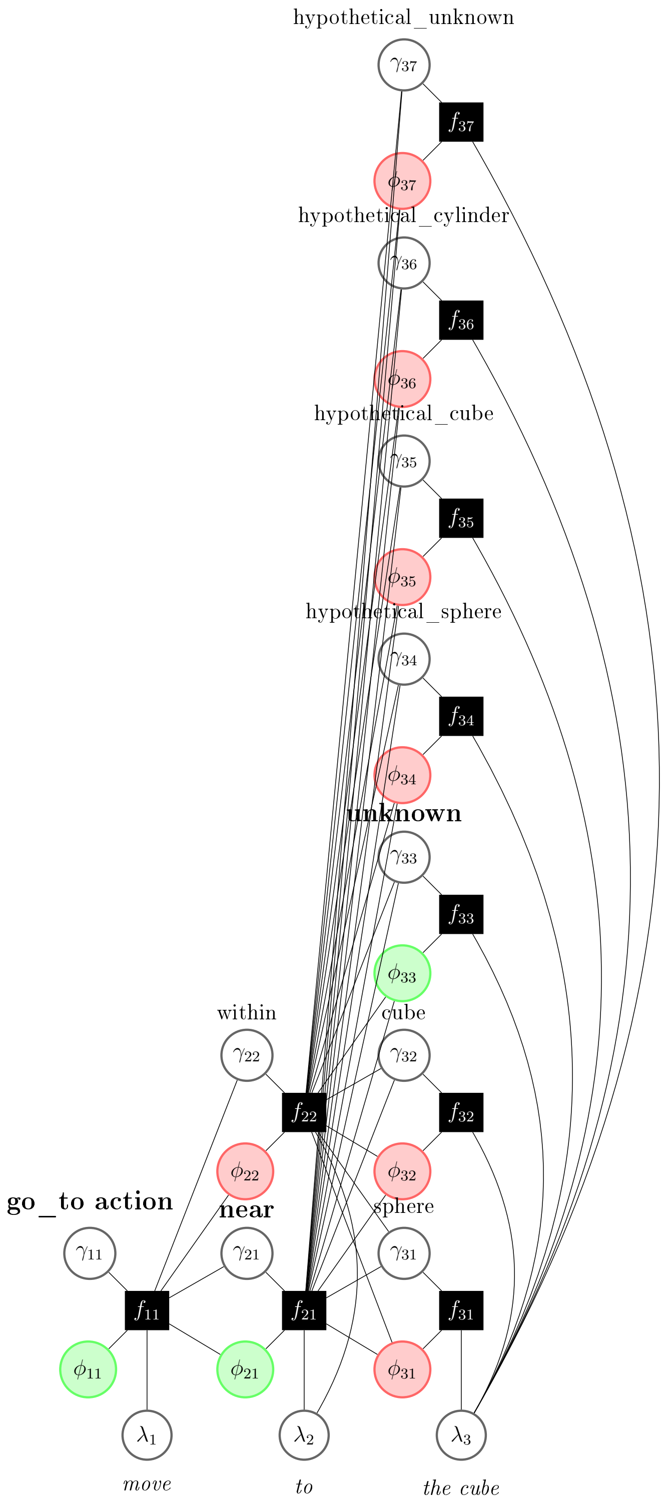


Figure 4-3: DCG-UPUP-Away model solved for a phrase that is unknown ("cone") when a cube, sphere, and an unknown object are perceived

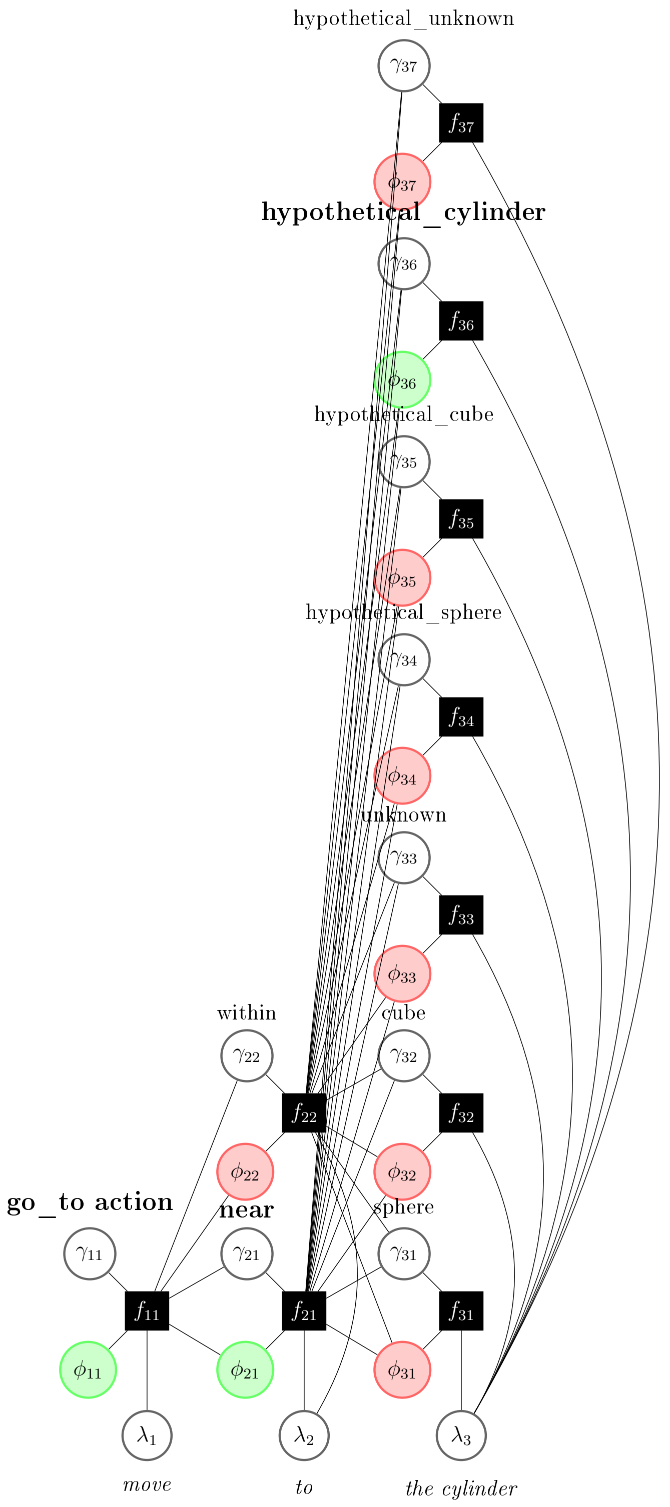


Figure 4-4: DCG-UPUP-Away model solved for a phrase that is known (“cylinder”) when a cube, sphere, and an unknown object are perceived.

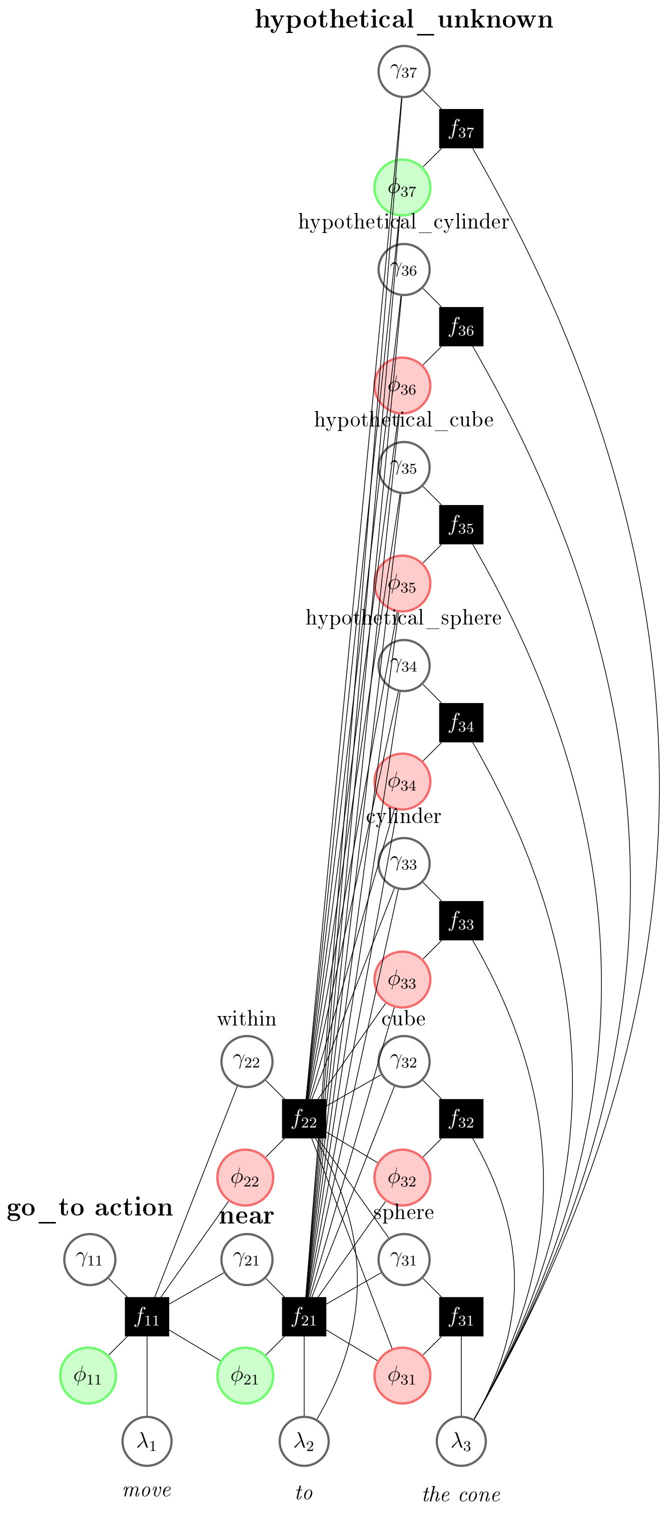


Figure 4-5: DCG-UPUP-Away model solved for a phrase that is unknown ("cone") when a cube, sphere, and cylinder are perceived. (If an unknown object were perceived, since perceived objects are favored over hypothetical objects, the perceived unknown would be grounded to instead of the hypothetical unknown.)

Chapter 5

Resolving Ambiguity through Heuristics

DCG-UPUP-Away, as developed so far, solves the problems initially described in the introduction: a robot may now reason about unknown objects and phrases, autonomously learn new symbols, and even reason about objects that it cannot perceive. In that sense, the required work is done. Nevertheless, in order to improve robustness of the robot for experimental validation and to indicate a broad area for future work, DCG-UPUP-Away has been modified to use the heuristic feature of color to determine the correct grounding.

5.1 Form of Color Heuristic

The role of the color heuristic is to differentiate between two objects of the same type: for example, a red cube and a blue cube. More generally, any heuristic relating an object's properties to language (e.g. "small" and and object's size) assists in disambiguating among objects. For a deployable robot in the real world, many such heuristics may be necessary because natural language routinely references object properties, not just their type.

Nearly all heuristics considered for DCG-UPUP-Away match the form of the color heuristic: an association is learned between natural language (e.g. "red") and an object property (e.g. its RGB value). Any heuristic that ignores natural language but favors certain groundings over others makes that assumption that those objects are generally more likely to be grounded to. The perfect example of this is the hypothetical heuristic, introduced in Chapter 4. In general, when deciding between two identical objects, one of which is perceived and the other of which is hypothetical, it is more likely that the language grounds to the perceived object. Therefore, the hypothetical heuristic only requires one feature that detects whether or not an object is hypothetical. More generally, though, heuristic features that rely on both language and object properties require at least two features: one to detect the language part, and one the object property part. By placing these two features in separate grounding sets, it is possible to learn their association.

5.2 Color Heuristic Features

The remainder of this chapter will explain how the color heuristic is worked into DCG-UPUP-Away. Only three main changes are necessary: 1) a *feature_word* must be added for each word that contains color information (e.g. "red"), 2) a new feature, *feature_color*, must be developed to fire if an object's color matches the feature's argument (e.g. red), and 3) a *feature_color* must be added for each color that an object can take on.

5.2.1 Color Language Feature

On the language detection side, minimal changes are required. *feature_word*, developed in DCG, has already been developed to fire when a phrase contains the word passed in as an argument to *feature_word*. Therefore, all that is required is to add features to detect what color phrases appear. In this particular development, that meant adding *feature_ word* with the arguments “red,” “green,” and “blue.”

There is one minor necessary for a sentence containing color adjectives to be parseable in the first place: the grammar file must be updated to allow for adjectives. It is relatively easy to add an adjective or even multiple adjectives in front of nouns, however, by introducing a recursive parse tree element with a base case of an adjective followed by a noun. This small modification is all that is necessary to parse phrases with potentially infinite adjectives.

5.2.2 Color Object Feature

Just as language features are added to detect phrases like “red,” object property features must be added to detect what color objects are. This requires two changes: 1) objects must contain a color property and 2) a feature must be added that accepts a color as an argument and fires if a grounded object matches the color.

Updating objects to keep track of color is relatively simple. There are myriad ways to represent color (e.g. RGB, HSV, enums, etc.), but for simplicity, in this instance, development was limited to representing color with a string that could take on four possible values: “red,” “green,” “blue,” and “na.” Updating the world model to reflect objects’ colors is similarly straightforward. The object recognizer introduced in Chapter 3 is modified to also detect the most prevalent RGB value within the bounding box of each object. If no value is noticeably greater than the others, the object color is set to “na.”

Given that objects maintain information about their color, it is easy to develop a feature that fires when the grounded object matches a specific color. The feature, called *feature_ color*, is almost exactly like *feature_ object_ type*, which checks what type of object the grounded object is. Instead of checking the object-type property, however, *feature_ color* checks the color property. Therefore, three *feature_ color*’s were added to the feature set with arguments “red,” “green,” and “blue.” By supplying several training examples in which phrases like “go to the red cube” grounds to a red cube instead of a blue cube, DCG-UPUP-Away can learn the association between color adjectives and color properties.

Although *feature_ color* as described so far accomplishes most of the desired goals of disambiguating between objects of the same type, one key problem remains: given that DCG-UPUP-Away hypothesizes objects in order to ground to objects it cannot perceive, and given that color now influences groundings, it appears as if it is necessary to hypothesize every object in every color. Such a procedure could be computationally prohibitively expensive, since it involves searching over the cross of the object type space and the color space. Instead, only a minor modification is made to *feature_ color*. Previously, *feature_ color* returned true if and only if the grounded object’s color matches the feature’s argument (e.g. *feature_ color*(“blue”) firing only if the grounded object is blue). Now, *feature_ color* returns true if the grounded object’s color matches the feature’s argument OR the object’s color is “na.” As a result, if hypothetical objects are created with colors set to “na,” every feature color will fire – as if the object has the correct color to match the language, whatever that may be. In practice, this approach works quite well and efficiently solves the problem of simultaneously solving for color and object type.

Another possible way to solve this problem would be to first solve for color (independent of object type) and then solve for object type. This would simultaneously solve the computational problem of searching jointly over color and object type while matching human intuition. In many ways, solving for color and type seem like independent problems, which bolsters the argument for this alternate approach. Theoretically, this should not be too difficult to implement, but it will involve introducing an additional type of symbol representing color. Such work is left for future research.

Chapter 6

Evaluation

DCG-UPUP-Away, augmented by a color heuristic, is ready for evaluation. This section will demonstrate that the DCG-UPUP-Away model behaves as expected in proof-of-concept trials (in simulation and on hardware) and, more generally, matches expected theoretical performance in randomly generated simulation trials.

The main metric of success is whether or not DCG-UPUP-Away grounds to the correct object in a scene. One additional datum worth considering, however, is how many symbols DCG-UPUP-Away correctly knows at any given time (e.g. it was trained to ground cubes, spheres, and cylinders, but now it also knows how to ground cones). Both metrics will be considered, although the primary focus is grounding accuracy.

6.1 Experimental Design

Although the proof-of-concept results from the previous section demonstrate the new capabilities developed in DCG-UPUP-Away, two questions remain unanswered: 1) do these capabilities apply to real-world scenarios and 2) how does DCG-UPUP-Away perform in such scenarios? To answer these questions, an experiment was designed specifically to test DCG-UPUP-Away using language generated by the general population in randomly generated environments.

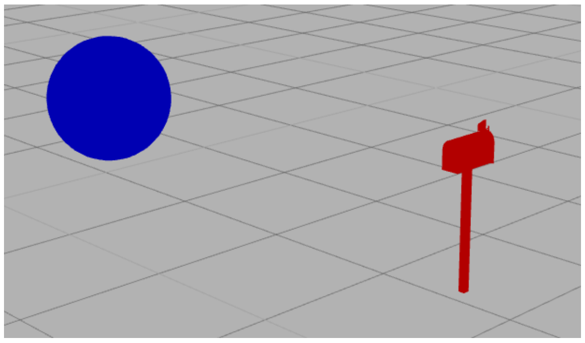
The overarching idea of the experiment falls into four phases:

1. Randomized environment generation
2. Natural language command generation
3. DCG-UPUP-Away training
4. DCG-UPUP-Away evaluation

6.1.1 Environment Generation

In the first phase, 10 random environments are generated by selecting a subset of objects from a library of 24 possible objects. These 24 objects are generated by combining one of 8 object types (cube, sphere, cylinder, cone, hydrant, handle, mailbox, or drill) with one of three possible colors (red, green, or blue). Each object is selected to be added to the environment with a 15% probability. Each added object is placed at a random location generated from a uniform distribution over a 4 by 4 meter square around the robot (excluding a 2 by 2 meter square directly around the robot). Note that the orientations of objects are not randomized - given that the robot will be placed in the middle of the scene and that objects are randomly located, the orientation of an object relative to the robot will therefore automatically be randomized. Finally, environments are rejected if they contained zero or one objects or if an object is completely hidden behind another object when viewed from the origin. Sample environments are shown in Figure 6-1.

Generating environments in such a manner yields two important results. First, the robot cannot know ahead of time what objects are present in the world. Second, objects may be placed out of the field of view of the robot. These two facts, coupled with the fact the turtlebot will only be trained with cubes, spheres, and cylinders, means that both the unknown-phrase, unknown-percept and out-of-world grounding


(a) A generated random environment containing a blue sphere and a red mailbox.

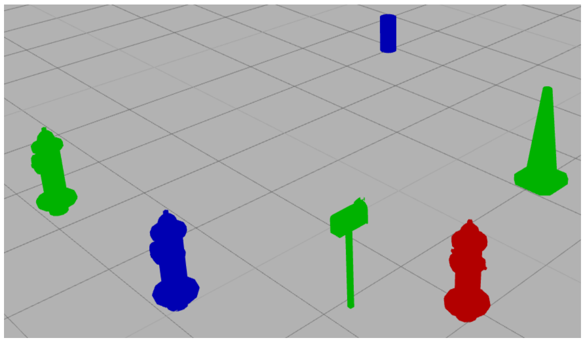

(b) A different generated random environment containing a green hydrant, blue hydrant, green mailbox, red hydrant, green cone, and blue cylinder.

Figure 6-1: Sample generated test environments

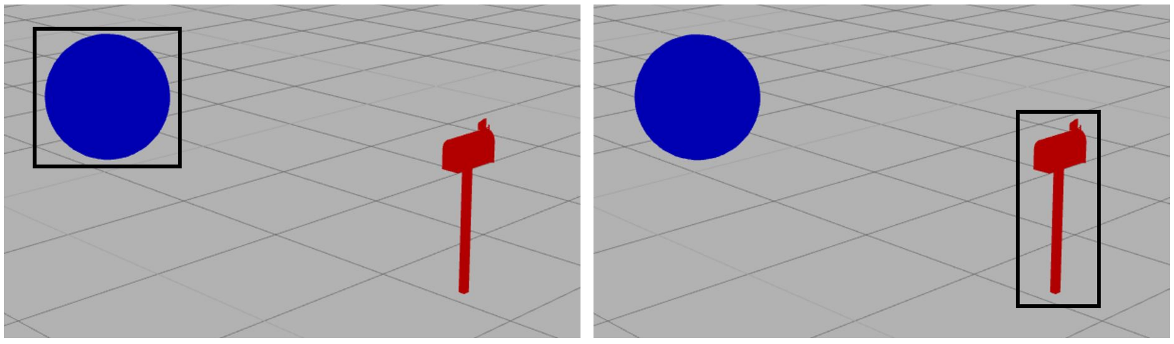
capabilities will be tested.

6.1.2 Natural Language Command Generation

After generating the environments, each object within each environment needs to be paired to a natural language command. Such commands can then be used to tell the robot how to approach any given object in any environment, and the response of the robot can later be assessed for accuracy.

Given the 10 particular generated environments, there are 39 objects total. Screenshots of each environment are taken and then modified to highlight one particular object. Figure 6-2 shows two sample images of the same environment with different objects highlighted.

These images with highlighted objects are then uploaded to a form on Amazon Mechanical Turk for natural language annotation. Online users are shown one such image and asked to "Please write the command for approaching the highlighted object." Users are also provided with a sample image of a green cylinder, red cube, and blue sphere in which the cylinder is highlighted. The sample image is accompanied



(a) Highlighting a blue sphere in a sample environment. (b) Highlighting a red mailbox in a sample environment.

Figure 6-2: Example environments with highlighted objects.

by the command “go to the green cylinder.”

In total, 390 annotations are collected: 10 for each of the 39 images. Allowing 10 annotations for each image mitigates the problem of improperly formed annotations. For example, one user routinely used spatial relations in annotations (e.g. saying “go to the object on the left”) despite the instructions explicitly prohibiting such phrases. In addition, because the true goal of collecting the natural language commands is to generate a series of consistently labeled images, ideally one user should label all 39 images. In fact, several users stopped labeling after only a handful of images, rendering their annotations useless. Ultimately, however, one user followed nearly all the instructions (although occasionally alternated between labeling spheres as “sphere” and “circle”) and labeled each of the 39 images. That user’s commands were used for the rest of the experiment.

6.1.3 DCG-UPUP-Away Initial Training

Many experiments designed to assess natural language understanding use a subset of their test data as a training set; in this experiment, however, a pre-defined corpus of 55 example groundings are used. If this experiment used a subset of the test data, two problems would arise: 1) more test data would need to be generated so that around 55 examples could be used for training while leaving some data for testing and 2) the likelihood of assessing performance when grounding with unknowns would be extremely small. The second point is particularly important.

The only way to guarantee that some phrases and percepts will remain unknown is by leaving them out of the training set. Therefore, an initial grounding data set is constructed that only grounds cubes, spheres, and cylinders (other nouns, such as "mallet" and "pipe" are also used, but only to train grounding to unknowns). The majority of the groundings are used for training how to ground "cube," "sphere," and "cylinder" as well as using color to disambiguate between objects. Eight of the examples, however, are used to introduce synonyms to the known types (e.g. "circle" grounds to a sphere type and "box" grounds to a cube type). These synonym training examples introduce a sort of vocabulary robustness that ended up being used to ground real users' phrases. Finally, similarly to the initial set of grounding training data, a set of images of cubes, spheres, and cylinders is set aside as the initial training for the object recognizer.

After training DCG-UPUP-Away with the initial training data, the system has learned the symbols for cubes, spheres, and cylinders. Subsequent trials could use any of 8 possible objects, meaning the system at first knows only 3/8 of symbols in the world.

6.1.4 DCG-UPUP-Away Evaluation

The simulated environments, natural language commands, and initial training data are all that is necessary for evaluating DCG-UPUP-Away. Two evaluation mechanisms are developed: one full end-to-end system with an automatic object recognizer, and one system in which object recognition is performed manually. Although the former experiment evaluates DCG-UPUP-Away in a more realistic setting, the inaccuracy of the object recognizer introduces errors that mask whether or not the grounding system DCG-UPUP-Away functions properly. Moreover, object recognition the object recognition module is fully separable from the grounding module, so future vision work may be easily incorporated into the overall system.

Both methods (named *automatic* and *manual* for the automatic object recognition and manual object recognition systems, respectively) are used to determine their grounding accuracy rate over a series of natural language commands. In other words, how likely is it that DCG-UPUP-Away will ground to the right object for many commands even as it performs unsupervised learning between commands.

The exact evaluation procedures for both methods (named *automatic* and *manual* for the automatic object recognition and manual object recognition systems, respectively) are nearly identical. First, for both methods, DCG-UPUP-Away is trained on the initial training set. The *automatic* method additionally trains an object recognizer based on a set of initial training images. Second, a series of environment, natural-language command pairs is uniformly randomly selected without replacement from the 39 possible commands. For the *automatic* method, which is considerably slower than the *manual* method, each series contains 5 environments; for the *manual* method, each series contains 10 environments. Third, a turtlebot is place in each of the simulated environments, in order, and given the corresponding natural language command. Between trials in a given series, DCG-UPUP-Away is retrained using the last generated unsupervised training example. For the *automatic* method, success for a given trial is determined by whether or not the last camera view of the robot contains exactly one object and it is the correct robot. For the *manual* method, success is determined by reading through the grounding output file. Alongside recording a trial's success rate, how many symbols DCG-UPUP-Away has correctly learned is also logged. After finishing a full series of trials, all unsupervised training examples are erased, and the entire evaluation process starts over with initial retraining.

6.2 Experiment Results and Analysis

The results have been divided into 4 groups. In the first, grounding accuracy rate using manual object recognition is calculated as a function of iteration number within a given series. In the second, the number of correctly known symbols (ranging from 3 to 8), is plotted as a function of iteration number as well. In the third, the symbol and accuracy data are combined to show how accuracy rates increase as a function of symbols learned. Finally, in the fourth, preliminary data is presented for accuracy rate using automatic object recognition as a function of iteration number.

6.2.1 Grounding Accuracy across a Series

The primary result shown in this experiment is that the accuracy of DCG-UPUP-Away remains relatively constant at 80% over series of 10 trials. The exact data showing this result is shown in Figure 6-3. The data used for this plot were generated using manual object recognition, for 30 runs of trials, each with ten assessments.

There are two main results shown in this plot. First, the baseline accuracy is quite high. Averaged across all iterations and all trials (for a total of 300 individual tests), 80% of trials result in a successful grounding. Folded within that 80% are a majority of trials in which the object being referred to is unknown (not a cube, sphere, or cylinder), out of the field of view of the robot, or both. This feeds nicely into the second main point: grounding accuracy remains high (and even grows slightly) as a function of iteration number. In other words, despite only using completely unsupervised learning within a series of trials, DCG-UPUP-Away is *as likely* to succeed after 9 trials then after only training on hand-picked data. Furthermore, during the course of retraining on unsupervised data, DCG-UPUP-Away learns new symbols.

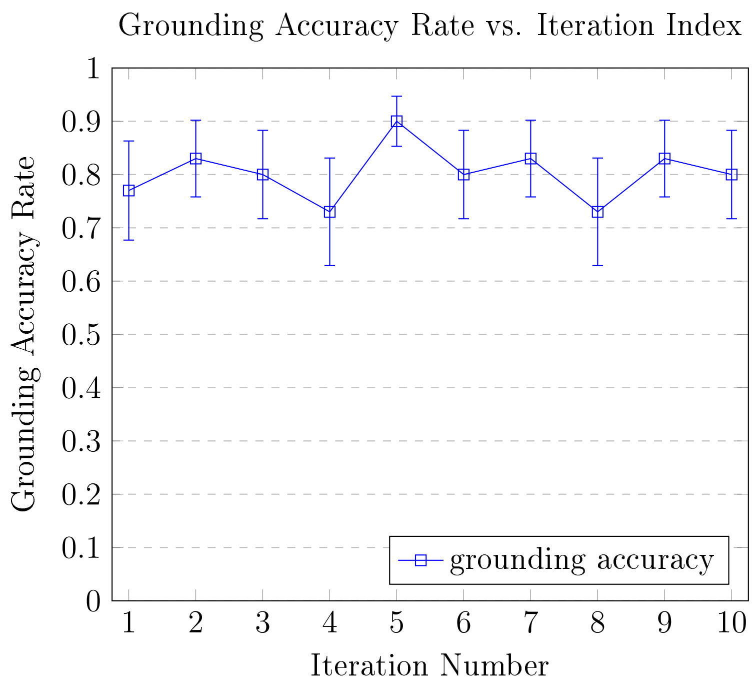


Figure 6-3: Relatively constant accuracy rate as a function of iteration number.

6.2.2 Symbol Learning across a Series

The next metric worth examining is exactly how many symbols DCG-UPUP-Away learns over the course of 10 tests. After all, if DCG-UPUP-Away is initially trained on cubes, spheres, and cylinders but only knows those three symbols at the end, maintaining the high grounding accuracy across trials is not that impressive. In fact, that is not the case. Figure 6-4 shows how, on average, the number of correctly learned symbols grows from 3 (cube, sphere, cylinder) in the first trial to over 6 by the last trial.

At the very least, the data in Figure 6-4 validate that DCG-UPUP-Away is behaving as expected. The blue line, showing actual data, tracks the red line, showing algebraically computed expected values, quite well. The data are admittedly heteroscedastic; variance in the number of correctly learned symbols increases as a function of iteration number. In fact, this trend of growing variance may be unavoidable. After all, all series start with exactly 3 known symbols and then randomly acquire

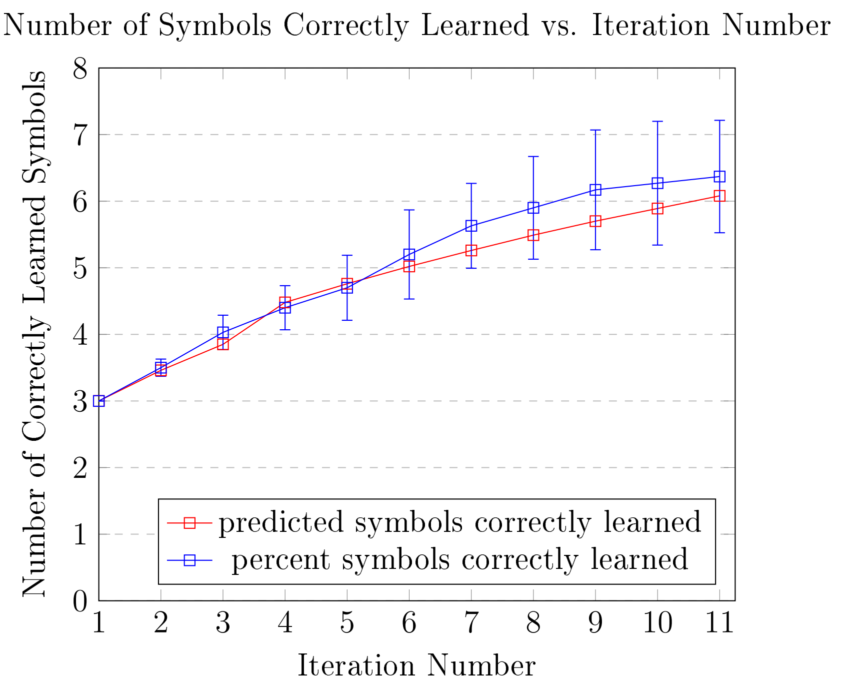


Figure 6-4: Increasing number of correctly learned symbols as a function of iteration number

new symbols.

The most important result from Figure 6-4, however, is that DCG-UPUP-Away reliably learns some symbols correctly. Despite only being initialized with knowledge of cubes, spheres, and cylinders, DCG-UPUP-Away may face scenes containing up to 5 other unknown symbols: fire hydrants, cones, door handles, mailboxes, and power drills. On average, after 10 trials, DCG-UPUP-Away has learned at least three of these symbols, completely autonomously. In fact, in 10% of the trials, DCG-UPUP-Away learned all five unknown symbols correctly.

Furthermore, if human feedback is ever incorporated into DCG-UPUP-Away, the average number of correctly learned symbols will likely increase dramatically. The most common failures, both in grounding accuracy and symbol learning, occurred when DCG-UPUP-Away had to pick between two unknown objects of the same color, grounded to the wrong one, and subsequently misclassified that object for all remaining trials.

6.2.3 Improved Accuracy through Symbol Acquisition

Having determined that DCG-UPUP-Away learns symbols, the natural next question is whether doing so improves grounding accuracy. Therefore, rather than plotting accuracy as a function of iteration number within a series, Figure 6-5 shows that knowing more symbols is positively correlated with improved accuracy ($r = 0.654$).

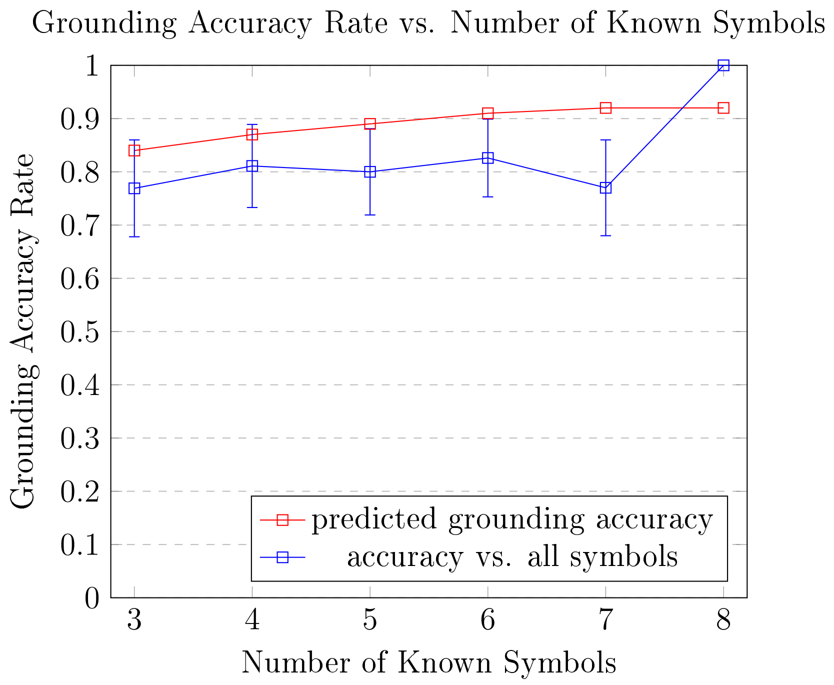


Figure 6-5: Grounding accuracy as a function of number of symbols

It should be emphasized that Figure 6-5 plots grounding accuracy as a function of total number of symbols learned, even if such symbols are learned incorrectly. For example, if DCG-UPUP-Away grounds the phrase “hydrant” to a cone because both hydrants and cones are unknown (and the robot sees the hydrant before it sees the cone), DCG-UPUP-Away will think that it has learned a symbol for hydrant. If one plots grounding accuracy as a function of *correctly* learned symbols, however, one generates Figure 6-6.

In this figure, the correlation between learning symbols correctly and grounding

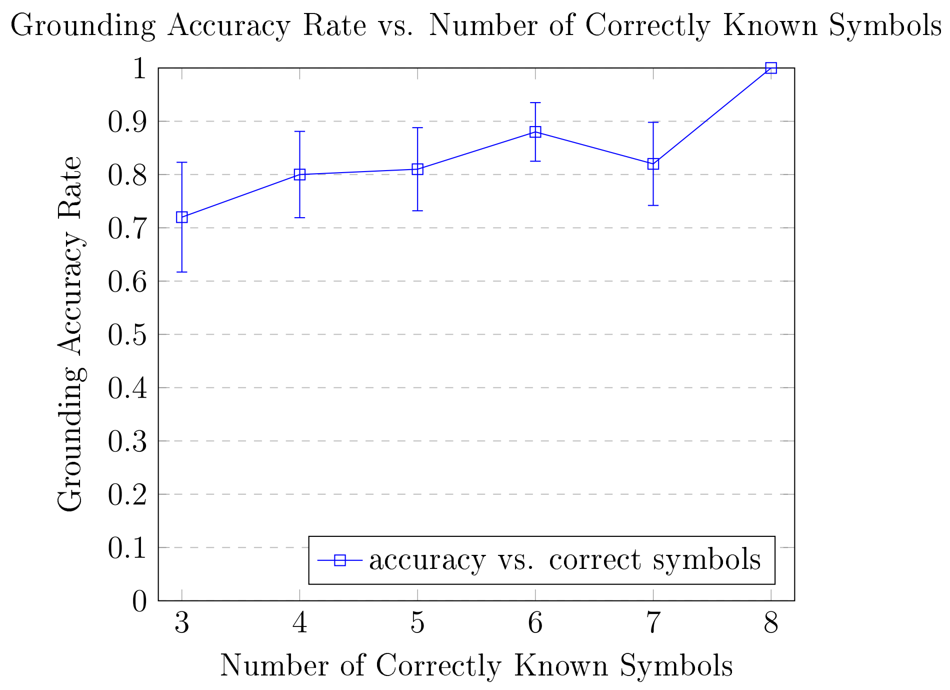


Figure 6-6: Grounding accuracy as a function of number of correctly known symbols

more accurately is even tighter. This substantial improvement in accuracy motivates still further future research in human feedback - if humans could tell robots that they learned a symbol incorrectly, the gap in performance between Figure 6-5 and Figure 6-6 would shrink.

6.2.4 Accuracy Using Automatic Object Recognition

One remaining question that must be addressed is how well the previous results apply in a fully autonomous system. All the data already presented were generated using manual object recognition; researchers viewed the camera output of a simulated turtlebot and listed the color and type of each visible object. In fact, incorporating an object recognizer, slightly, but nevertheless noticeably, worsens grounding accuracy. Figure 6-7 shows grounding accuracy, over 10 series, each with 5 tests, for runs in simulation using an object recognizer.

The blue line, representing grounding accuracy for the automatic object recog-

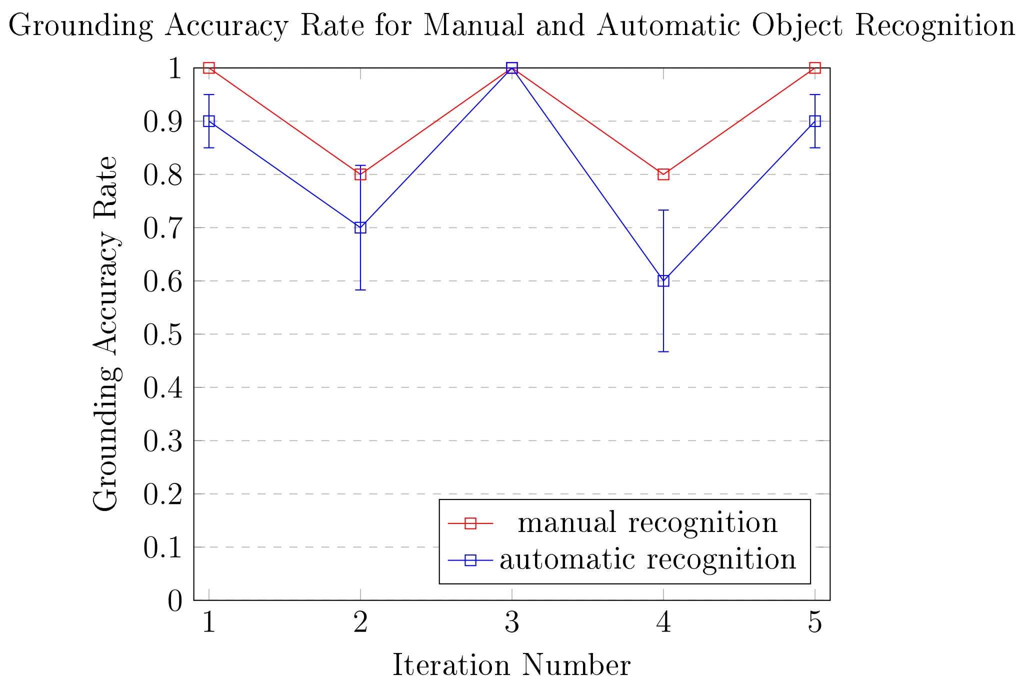


Figure 6-7: Grounding accuracy as a function of iteration number using automatic object recognition

nizer, routinely falls below the red line, which was generated using manual object recognition for the same scenes. This graph clearly demonstrates that imperfect object recognition worsens DCG-UPUP-Away's performance. Fortunately, as previously stated, the object recognition module of DCG-UPUP-Away is completely separable, so a better object recognizer could easily be integrated into the overall system.

6.2.5 Hardware Demonstration

One final way to demonstrate the improvements offered by DCG-UPUP-Away is on a physical platform: a real turtlebot driving around a laboratory. Given the high fidelity of turtlebot simulations in Gazebo, porting DCG-UPUP-Away code from a simulation environment to a real turtlebot environment is trivial. The only real modification necessary is retraining the object recognizer to recognize real world objects instead of simulated blocks. A trial run was divided into 3 steps, each demonstrating a unique capability. In the trial run, in order to simplify the job of the object recognizer, the known object types were boxes, helmets, and soap bottles. Both the image recognizer and natural language training examples were modified for the new types.

Hardware DCG

First, the robot demonstrates that the original DCG capabilities remain intact. The robot is placed in a scene with a box, a helmet, a bottle of soap, and a cone (the cone is unknown). After receiving the command "move towards the box," the turtlebot successfully drives to the box. This demonstrates the capability to ground known phrases to known types.

Hardware DCG-UPUP

Second, the robot demonstrates the ability to associate unknown phrases with unknown percepts. The robot is placed in a scene with a box, a helmet, and a cone (the cone is still unknown). The turtlebot is given the command "move towards the cone." Because the cone is perceived as an unknown object and the phrase "the cone" is unknown, "cone" is grounded to the cone, and the robot drives to the cone. This demonstrates that DCG-UPUP has learned how to ground to unknowns.

Hardware DCG-UPUP Learning

Third, the robot shows that it can permanently learn new symbols. By retraining after the second trial run (in which it grounded "cone" to a cone), the robot learns the symbol for a cone. In fact, when placed in a scene with a ball (which is unknown) and a cone (which is now known), the robot drives to the ball when it hears an unknown phrase such as "move to the ball." Conversely, in the same scene, if the robot gets the command "go to the cone," the robot drives to the cone. Thus, the robot has clearly learned what a cone is.

Chapter 7

Conclusion

7.1 Contributions

There are two main contributions from the work presented in this thesis: 1) proposing a model, DCG-UPUP-Away for automatic symbol acquisition by grounding to unknowns, and 2) experimental validation of the model. The key ideas behind the expanded model are that explicitly recognizing phrases or percepts as unknown allows for autonomous learning. In fact, the concept of recognizing unknowns could easily apply to many domains within robotics to increase robustness and learning capabilities in unknown environments.

Although such domains are left for future research, within the field of Natural Language grounding, DCG-UPUP-Away succeeds in two remarkable ways. First, in randomly generated environments in which more than half of all objects are unknown to DCG-UPUP-Away ahead of time, DCG-UPUP-Away grounds to the correct object around 80% of the time. Second, by using naive unsupervised learning between trials, DCG-UPUP-Away completely autonomously learns new symbols. This second point, in particular, appears important. One could imagine mass producing robots, training them with initial data that will be useful nearly everywhere, and then shipping off the robots to remarkably different environments. For example, one robot could go to an oil rig, while another robot, initially identical to the first, could work as a shop assistant in a tobacco store. Over time, however, the two robots will learn completely different meanings of the words “pipe” (one for pumping oil, the other for smoking out of) precisely because the phrases take on different meanings in the two contexts. This automatic divergence of knowledge because of separate experiences seems like precisely what will be necessary in robots of the future.

7.2 Recommendations for Future Work

There remains, of course, much work to be done before such robots arrive on everyone’s doorstep.
here’s what else I should do.
thesaurus + google images
build up physical database alone before groundings
build up linguistic database alone before grounding

Bibliography

[1] David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. pages 859–865, 2011.

[2] Felix Duvallet, Matthew Walter, Thomas Howard, Sachithra Hemachandra, Jean Hyaejin Oh , Seth Teller, Nicholas Roy, and Anthony (Tony) Stentz . Inferring maps and behaviors from natural language instructions. In *International Symposium on Experimental Robotics*, June 2014.

[3] Terrence W Fong, Chuck Thorpe, and Charles Baur. Robot, asker of questions. *Robotics and Autonomous Systems*, 2003.

[4] T. Howard, S. Tellex, and N. Roy. A natural language planner interface for mobile manipulators. In *International Conference on Robotics and Automation*, June 2014.

[5] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE International Conference on Human-robot Interaction*, HRI '10, pages 259–266. IEEE Press, 2010.

[6] Cynthia Matuszek, Nicholas FitzGerald, Luke S. Zettlemoyer, Liefeng Bo, and Dieter Fox. A joint model of language and perception for grounded attribute learning. In *ICML*. icml.cc / Omnipress, 2012.

[7] H. Moll and M. Tomasello. Twelve- and 18-month-old infants follow gaze to spaces behind barriers. In *British Journal of Developmental Psychology*, 2004.

[8] H. Moll and M. Tomasello. Level 1 perspective-taking at 24 months of age. In *Developmental Science*, 2006.

[9] R. Ros, S. Lemaignan, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken. Which one? grounding the referent based on efficient human-robot interaction. In *19th International Symposium in Robot and Human Interactive Communication*, pages 570–575, Sept 2010.

[10] N. Roy, J. Pineau, and S. Thrun. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, Hong Kong, 2000.

[11] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *National Conference on Artificial Intelligence*, 2011.

[12] Stefanie Tellex, Pratiksha Thaker, Dimitar Simeonov, and Nicholas Roy. Clarifying commands with information-theoretic human-robot dialog, 2013.

[13] J. G. Trafton, A. C. Schultz, M. Bugajska, and F. Mintz. Perspective-taking with robots: experiments and models. In *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, pages 580–584, Aug 2005.

[14] J. Gregory Trafton, Nicholas L. Cassimatis, Magdalena D. Bugajska, Derek P. Brock, Farilee E. Mintz, and Alan C. Schultz. Enabling effective human-robot interaction using perspective-taking in robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 35:460–470, 2005.

[15] Matthew R. Walter, Sachithra Hemach, Bianca Homberg, Stefanie Tellex, and Seth Teller. Learning semantic maps from natural language descriptions.

[16] Thomas Williams, Rehj Cantrell, Gordon Briggs, Paul Schermerhorn, and Matthias Scheutz. Grounding natural language references to unvisited and hypothetical locations, 2013.

[17] C. Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*. European Conference on Computer Vision, September 2014.