RoboBrain:
Large-Scale Knowledge Engine for Robots

Ashutosh Saxena, Ashesh Jain, Ozan Sener, Aditya Jami, Dipendra K Misra, Hema S Koppula

Abstract In this paper we introduce a knowledge engine, which learns and shares knowledge representations, for robots to carry out a variety of tasks. Building such an engine brings with it the challenge of dealing with multiple data modalities including symbols, natural language, haptic senses, robot trajectories, visual features and many others. The *knowledge* stored in the engine comes from multiple sources including physical interactions that robots have while performing tasks (perception, planning and control), knowledge bases from the Internet and learned representations from several robotics research groups.
We discuss various technical aspects and associated challenges such as modeling the correctness of knowledge, inferring latent information and formulating different robotic tasks as queries to the knowledge engine. We describe the system architecture and how it supports different mechanisms for users and robots to interact with the engine. Finally, we demonstrate its use in three important research areas: grounding natural language, perception, and planning, which are the key building blocks for many robotic tasks. This knowledge engine is a collaborative effort and we call it RoboBrain: `http://www.robobrain.me`

Keywords—Cloud Robotics, Robot Learning, Systems, knowledge bases.

1 Introduction

Over the last decade, we have seen many successful applications of large-scale knowledge systems. Examples include Google knowledge graph [13], IBM Watson [16], Wikipedia, and many others. These systems know answers to many of our day-to-day questions, and not crafted for a specific task, which makes them valuable for humans. Inspired by them, researchers have aggregated domain specific knowledge by mining data [3, 6], and processing natural language [9], images [12] and speech [41]. These sources of knowledge are specifically designed for humans, and

Authors are with the Department of Computer Science, Cornell University and Stanford University.
A. Saxena is also with Brain Of Things Inc.
Email: {asaxena, ashesh, ozansener, adityaj, dipendra, hema}@cs.stanford.edu

their human centric design makes them of limited use for robots—for example, imagine a robot querying a search engine for how to "bring sweet tea from the kitchen" (Figure 1).

In order to perform a task, robots require access to a large variety of information with finer details for performing perception, planning, control and natural language understanding. When asked to bring sweet tea, as shown in Figure 1, the robot would need access to the knowledge for grounding the language symbols into physical entities, the knowledge that sweet tea can either be on a table or in a fridge, and the knowledge for inferring the appropriate plans for grasping and manipulating objects. Efficiently handling this joint knowledge representation across different tasks and modalities is still an open problem.

In this paper we present RoboBrain that allows robots to learn and share such representations of knowledge. We learn these knowledge representations from a variety of sources, including interactions that robots have while performing perception, planning and control, as well as natural language and visual data from the Internet. Our representation considers several modalities including symbols, natural language, visual or shape features, haptic properties, and so on. RoboBrain connects this knowledge from various sources and allow robots to perform diverse tasks by jointly reasoning over multiple data modalities.

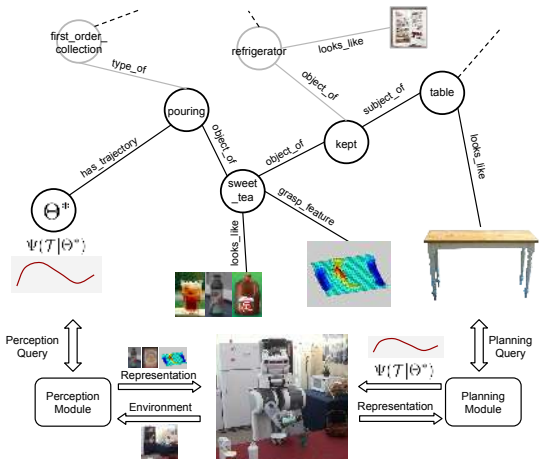

Fig. 1 An example showing a robot using RoboBrain for performing tasks. The robot is asked "Bring me sweet tea from the kitchen", where it needs to translate the instruction into the perceived state of the environment. RoboBrain provides useful knowledge to the robot for performing the task: (a) sweet tea can be kept on a table or inside a refrigerator, (b) bottle can be grasped in certain ways, (c) opened sweet tea bottle needs to be kept upright, (d) the pouring trajectory should obey user preferences of moving slowly to pour, and so on.

RoboBrain enables sharing from multiple sources by representing the knowledge in a graph structure. Traversals on the RoboBrain graph allow robots to gather the specific information they need for a task. This includes the semantic information, such as different grasps of the same object, as well as the functional knowledge, such as spatial constraints (e.g., a bottle is kept on the table and not the other way around). The key challenge lies in building this graph from a variety of knowledge sources while ensuring dense connectivity across nodes. Furthermore, there are several challenges in building a system that allows concurrent and distributed update, and retrieval operations.

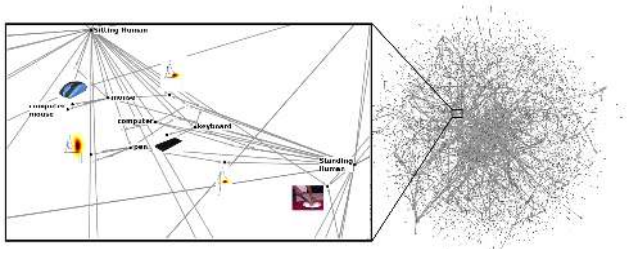

Fig. 2: **A visualization of the RoboBrain graph** on Nov 2014, showing about 45K nodes and 100K directed edges. The left inset shows a zoomed-in view of a small region of the graph with rendered media. This illustrates the relations between multiple modalities namely images, heatmaps, words and human poses. For high-definition graph visualization, see:
`http://pr.cs.cornell.edu/robobrain/graph.pdf`

We present use of RoboBrain on three robotics applications in the area of grounding natural language, perception and planning. For each application we show usage of RoboBrain *as-a-service*, which allow researchers to effortlessly use the state-of-the-art algorithms. We also present experiments to show that sharing knowledge representations through RoboBrain improves existing language grounding and path planning algorithms.

RoboBrain is a collaborative project that we support by designing a large-scale cloud architecture. In the current state, RoboBrain stores and shares knowledge across several research projects [53, 40, 23, 25, 26, 31, 57, 35] and Internet knowledge sources [34, 15]. We believe as more research projects contribute knowledge to RoboBrain, it will not only improve the concerned project but will also be beneficial for the robotics community at large. RoboBrain knowledge graph is available at `http://www.robobrain.me`.

This is our first paper introducing RoboBrain. It summarizes the key ideas and challenges in building a knowledge engine for robots. The goal of the paper is to present an overall view of the RoboBrain, its architecture, functionalities, and demonstrate its application to robotics. In Section 4 we formally define the RoboBrain graph and describe its system architecture in Section 5. In order for robots to use RoboBrain we propose the Robot Query Library in Section 6. In Section 7 we present different robotic applications using RoboBrain.

2 Related Work

We now describe some works related to RoboBrain. We first give an overview of the existing knowledge bases and describe how RoboBrain differs from them. We then describe some works in robotics that can benefit from RoboBrain, and also discuss some of the related on-going efforts.

Knowledge bases. Collecting and representing a large amount of information in a knowledge base (KB) has been widely studied in the areas of data mining, natural language processing and machine learning. Early seminal works have manually created KBs for the study of common sense knowledge (Cyc [34]) and lexical knowledge (WordNet [15]). With the growth of Wikipedia, KBs started to use crowd-sourcing (DBPedia [3], Freebase [6]) and automatic information extraction (Yago [51, 21], NELL [9]) for mining knowledge.

One of the limitations of these KBs is their strong dependence on a single modality that is the text modality. There have been few successful attempts to combine multiple modalities. ImageNet [12] and NEIL [10] enriched text with images obtained from Internet search. They used crowd-sourcing and unsupervised learning to get the object labels.

We have seen successful applications of the existing KBs within the modalities they covered, such as IBM Watson Jeopardy Challenge [17]. However, the existing KBs are human centric and do not directly apply to robotics. The robots need finer details about the physical world, e.g., how to manipulate objects, how to move in an environment, etc. In RoboBrain we combine knowledge from the Internet sources with finer details about the physical world, from RoboBrain project partners, to get an overall rich graph representation.

Robot Learning. For robots to operate autonomously they should perceive their environments, plan paths, manipulate objects and interact with humans. We describe previous work in each of these areas and how RoboBrain complements them.

Perceiving the environment. Perception is a key element of many robotic tasks. It has been applied to object labeling [33, 1, 57], scene understanding [28, 20], robot localization [39, 42], path planning [27], and object affordances [11, 30]. RoboBrain stores perception related knowledge in the form of 3D point clouds, grasping features, images and videos. It also connects this knowledge to human understandable concepts from the Internet knowledge sources.

Path planning and manipulation. Planning algorithms formulate action plans which are used by robots to move around and modify its environment. Planning algorithms have been proposed for the problems of motion planning [58, 49], task planning [7] and symbolic planning [46]. Some planning applications include robots baking cookies [7], folding towels [50], assembling furniture [29], and preparing pancakes [4]. The previous works have also learned planning parameters using Inverse Optimal Control [45, 23]. RoboBrain stores the planning parameters learned by previous works and allow the robots to query for the parameters.

Interacting with humans. Human-robot interaction includes collaborative tasks between humans and robots [43], generating safe and human-like robot motion [36, 18, 14, 8], interaction through natural language [54, 40], etc. These applications require joint treatment of perception, manipulation and natural language understanding. RoboBrain stores different data modalities required by these applications.

Previous efforts on connecting robots range from creating a common operating system (ROS) for robots [44] to sharing data acquired by various robots via cloud [56, 2]. For example, the RoboEarth [56] provides a platform for the robots to store and off-load computation to the cloud and communicate with other robots; and the KIVA systems [2] use the cloud to coordinate motion for hundreds of mobile platforms. On the other hand, RoboBrain provides a knowledge representation layer on top of data storing, sharing and communication.

Open-Ease [5] is a related on-going effort towards building a knowledge engine for robots. Open-Ease and RoboBrain differ in the way they learn and represent knowledge. In Open-Ease the knowledge is represented as formal statements using predefined templates. On the other hand, the knowledge in RoboBrain is represented as a graph. The nodes of the RoboBrain graph have no pre-defined templates and they can be any robotic concept like grasping features, trajectory parameters, and visual

data. This graph representation allows partner projects to easily integrate their learned concepts in RoboBrain. The semantic meaning of concepts in the RoboBrain graph are represented by their connectivity patterns in the graph.

3 Overview

RoboBrain is a never ending learning system that continuously incorporates new knowledge from its partner projects and from different Internet sources. One of the functions of RoboBrain is to represent the knowledge from various sources as a graph, as shown in Figure 2. The nodes of the graph represent concepts and edges represent the relations between them. The connectivity of the graph is increased through a set of graph operations that allow additions, deletions and updates to the graph. As of the date of this submission, RoboBrain has successfully connected knowledge from sources like WordNet, ImageNet, Freebase, OpenCyc, parts of Wikipedia and other partner projects. These knowledge sources provide lexical knowledge, grounding of concepts into images and common sense facts about the world.

The knowledge from the partner projects and Internet sources can sometimes be erroneous. RoboBrain handles inaccuracies in knowledge by maintaining beliefs over the correctness of the concepts and relations. These beliefs depend on how much RoboBrain trusts a given source of knowledge, and also the feedback it receives from crowd-sourcing (described below). For every incoming knowledge, RoboBrain also makes a sequence of decisions on whether to form new nodes, or edges, or both. Since the knowledge carries semantic meaning RoboBrain makes many of these decisions based on the contextual information that it gathers from nearby nodes and edges. For example, RoboBrain resolves polysemy using the context associated with nodes. Resolving polysemy is important because a 'plant' could mean a 'tree' or an 'industrial plant' and merging the nodes together will create errors in the graph.

RoboBrain incorporates supervisory signals from humans in the form of crowd-sourcing feedback. This feedback allows RoboBrain to update its beliefs over the correctness of the knowledge, and to modify the graph structure if required. While crowd-sourcing feedback was used in some previous works as means for data collection (e.g., [12, 48]), in RoboBrain they serve as supervisory signals that improve the knowledge engine. RoboBrain allows user interactions at multiple levels: (i) Coarse feedback: these are binary feedback where a user can "Approve" or "Disapprove" a concept in RoboBrain through its online web interface; (ii) Graph feedback: these feedback are elicited on RoboBrain *graph visualizer*, where a user modifies the graph by adding/deleting nodes or edges; (iii) Robot feedback: these are the physical feedback given by users directly on the robot.

In this paper we discuss different aspects of RoboBrain, and show how RoboBrain serves as a knowledge layer for the robots. In order to support knowledge sharing, learning, and crowd-sourcing feedback we develop a large-scale distributed system. We describe the architecture of our system in Section 5. In Section 6 we describe the robot query library, which allow robots to interact with RoboBrain. Through experiments we show that robots can use RoboBrain *as-a-service* and that knowledge sharing through RoboBrain improves existing robotic applications. We now present a formal definition of our Robot Knowledge Engine and the graph.

(a) original graph *(b)* feed insertion *(c)* after $merge(Mug, Mug') \rightarrow Mug \circ split(Cup) \rightarrow (Cup, Mug')$

Fig. 3: **Visualization of inserting new information.** We insert '*Sitting human can use a mug*' and RoboBrain infers the necessary split and merge operations on the graph. In (a) we show the original sub-graph, In (b) information about a *Mug* is seen for the first time and the corresponding node and edge are inserted, In (c) inference algorithm infers that previously connected cup node and cup images are not valid any more, and it splits the *Cup* node into two nodes as *Cup* and Mug' and then merges Mug' and *Mug* nodes.

4 Knowledge Engine: Formal Definition

In this section we present the formal definition of RoboBrain. RoboBrain represents knowledge as a directed graph $\mathscr{G} = (V, E)$. The vertices V of the graph stores concepts that can be of a variety of types such as images, text, videos, haptic data, or learned entities such as affordances, deep learning features, parameters, etc. The edges $E \subseteq V \times V \times C$ are directed and represents the relations between concepts. Each edge has an edge-type from a set C of possible edge-types.

An edge (v_1, v_2, ℓ) is an ordered set of two nodes v_1 and v_2 and an edge-type ℓ. Few examples of such edges are: (StandingHuman, Shoe, *CanUse*), (StandingHuman, $\mathscr{N}(\mu, \Sigma)$, *SpatiallyDistributedAs*) and (Grasping, DeepFeature23, *UsesFeature*). We do not impose any constraints on the type of data that nodes can represent. However, we require the edges to be consistent with RoboBrain edge set C. We further associate each node and edge in the graph with *a feature vector representation* and a *belief*. The feature vector representation of nodes and edges depend on their local connections in the graph, and their belief is a scalar probability over the accuracy of the information that the node or an edge represents. Tables 1 and 2 show few examples of nodes and edge-types. A snapshot of the graph is shown in Figure 2.

Creating the Graph. Graph creation consists of never ending cycle of two stages namely, knowledge acquisition and inference. Within the knowledge acquisition stage, we collect data from various sources and during the inference stage we apply statistical techniques to update the graph structure based on the aggregated data. We explain these two stages below.

1. Knowledge acquisition: RoboBrain accepts new information in the form of set of edges, which we call a *feed*. A *feed* can either be from an automated algorithm crawling the Internet sources or from one of RoboBrain's partner projects. We add a new *feed* to the existing graph through a sequence of union operations performed on the graph. These union operations are then followed by an inference algorithm. More specifically, given a new *feed* consisting of a set of N edges $\{(v_1^1, v_2^1, \ell^1) \ldots (v_1^N, v_2^N, \ell^N)\}$, and the existing graph $G = (V, E)$. The graph union operations give a graph $G' = (V', E')$ as follows:

$$V' = v_1^1 \cup v_2^1 \cup \ldots \cup v_1^N \cup v_2^N \cup V$$
$$E' = (v_1^1, v_2^1, \ell^1) \cup \ldots \cup (v_1^N, v_2^N, \ell^N) \cup E \quad (1)$$

2. Inference on the Graph: After adding the *feed* to the graph using equation (1), we perform inference to update the graph based on this new knowledge. The inference outputs a sequence of graph operations which are then performed on the graph. These graph operations modify the graph by adding new nodes or edges to the graph, deleting nodes or edges from the graph, merging or splitting nodes, etc.

We mention two graph operations here: *split* and *merge*. The split operation is defined as splitting a node into a set of two nodes. The edges having end points in the split node are connected to one of the resultant nodes using the inference algorithm. A merge operation is defined as merging two nodes into a single node, while updating the edges connected to the merged nodes. An example of such an update is shown in Figure 3. When a new information *"sitting human can use a mug"* is added to the graph, it causes the *split* of the *Cup* node into two nodes: a *Cup* and a *Mug* node. These two are then connected by an edge-type *TypeOf*. The graph update can be expressed through the following equation:

Word	an English word represented as an ASCII string
DeepFeature	feature function trained with a Deep Neural Network
Image	2D RGB Image
PointCloud	3D point cloud
Heatmap	heatmap parameter vector

Table 1: Some examples of different node types in our RoboBrain graph. For full-list, please see the code documentation.

$$G^\star = split_{v_{s_1}} \circ merge_{v_{m_1}, v_{m_2}} \circ \ldots \circ split_{v_{s_M}} \circ G'$$

In the above equation $G^\star$ is the graph obtained after the inference. The goal of the inference steps is to modify the graph G' in a way that best explains the physical world. However, the graph that captures the real physical world is a *latent graph*, i.e., it is not directly observable. For example, the latent information that *"coffee is typically in a container"* is partially observed through many edges between the *coffee* node and the nodes with *container* images. Our graph construction can also be explained in a generative setting of having a latent graph with all the knowledge about physical word, and we only observe noisy measurements in form of *feeds*. In this paper, we abstract the algorithmic details of inference and focus on the overall ideas involved in RoboBrain, its architecture, and its application to robotics.

IsTypeOf	human *IsTypeOf* a mammal
HasAppearance	floor *HasAppearance* as follows (this image)
CanPerformAction	human *CanPerformAction* cutting
SpatiallyDistributedAs	location of human is *SpatiallyDistributedAs*
IsHolonym	tree *IsHolonym* of leaf

Table 2: Some examples of different edge types in our RoboBrain graph. For full-list, please see the code documentation.

5 System Architecture

We now describe the system architecture of RoboBrain, shown in Figure 4. The system consists of four interconnected layers: (a) knowledge acquisition, (b) knowledge parser, (c) knowledge storage, and (d) knowledge inference. The principle behind our design is to efficiently process large amount of unstructured multi-modal knowledge

and represent it using the structured RoboBrain graph. In addition, our design also supports various mechanisms for users and robots to interact with RoboBrain. Below we discuss each of the components.

Knowledge acquisition layer is the interface between RoboBrain and different sources of multi-modal data. Through this layer RoboBrain gets access to new information which the other layers process. RoboBrain primarily collects knowledge through partner projects, by crawling existing knowledge bases such as Freebase, ImageNet, WordNet, etc., and from unstructured sources such as Wikipedia.

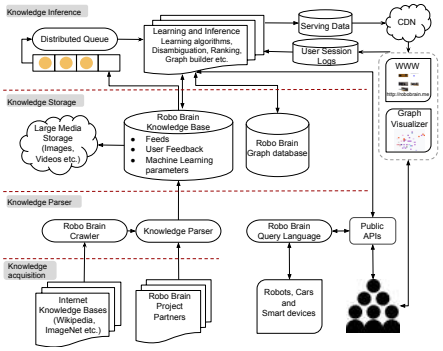

Fig. 4: **RoboBrain system architecture.** It consists of four interconnected knowledge layers and supports various mechanisms for users and robots to interact with RoboBrain.

Knowledge parser layer of RoboBrain processes the data acquired by the acquisition layer and converts it to a consistent format for the storage layer. It also marks the incoming data with appropriate meta- data such as timestamps, source version number etc., for scheduling and managing future data processing. Moreover, since the knowledge bases might change with time, it adds a back pointer to the original source.

Knowledge storage layer of RoboBrain is responsible for storing different representations of the data. In particular it consists of a NoSQL document storage database cluster – RoboBrain Knowledge Base (RoboBrain-KB) – to store "feeds" parsed by the knowledge parser, crowd-sourcing feedback from users, and parameters of different machine learning algorithms provided by RoboBrain project partners. RoboBrain-KB offloads large media content such as images, videos and 3D point clouds to a distributed object storage system built using Amazon Simple Storage Service (S3). The real power of RoboBrain comes through its graph database (RoboBrain-GD) which stores the structured knowledge. The data from RoboBrain-KB is refined through multiple learning algorithms and its graph representation is stored in RoboBrain-GD. The purpose behind this design is to keep RoboBrain-KB as the RoboBrain's single source of truth (SSOT). SSOT centric design allows us to re-build RoboBrain-GD in case of failures or malicious knowledge sources.

Knowledge inference layer contains the key processing and machine learning components of RoboBrain. New and recently updated feeds go through a persistent replicated distributed queuing system (Amazon SQS), which are then consumed by some of our machine learning plugins (inference algorithm, graph builder, etc.) and populates the graph database. These plugins along with other learning algorithms (operating on the entire graph) constitute our learning and inference framework.

RoboBrain supports various *interaction mechanisms* to enable robots and users to communicate with the knowledge engine. We develop a Robot Query Library as a primary method for robots to interact with RoboBrain. We also make available a set of public APIs to allow information to be presented on the WWW for online learning mechanisms (eg., crowd-sourcing). RoboBrain serves all its data using a commercial content delivery network (CDN) to reduce the end user latency.

6 Robot Query Library (RQL)

In this section we present the RQL query language, through which the robots use RoboBrain for various robotic applications. The RQL provides a rich set of *retrieval functions* and *programming constructs* to perform complex traversals on the RoboBrain graph. An example of such a query is finding the possible ways for humans to use a cup. This query requires traversing paths from the human node to the cup node in the RoboBrain graph.

RQL allows expressing both the *pattern* of sub-graphs to match and the *operations* to perform on the retrieved information. An example of such an operation is *ranking* the paths from the human to the cup node in the order of relevance. The RQL admits following two types of functions: (i) graph retrieval functions; and (ii) programming construct functions.

Graph retrieval function. The graph retrieval function is used to find sub-graphs matching a given *template* of the form: Template: $(u) \rightarrow [e] \rightarrow (v)$

In the template above, the variables u and v are nodes in the graph and the variable e is a directed edge from u to v. We represent the graph retrieval function with the keyword `fetch` and the corresponding RQL query takes the form: `fetch`(Template) This RQL query finds the sub-graphs matching the template. It instantiates the variables in the template to match the sub-graph and returns the list of instantiated variables. We now give a few use cases of the retrieval function for RoboBrain.

Example 1. RQL query to retrieve all the objects that a human can use

$$\text{Query: } \texttt{fetch}((\{\texttt{name} : \text{`}\texttt{Human}'\}) \rightarrow [\text{`}\texttt{CanUse}'] \rightarrow (\texttt{v}))$$

The above query returns a list of nodes that are connected to the node with name `Human` and with an edge of type `CanUse`.Using the RQL we can also express several *operations* to perform on the retrieved results. The operations can be of type `SortBy`, `Len`, `Belief` and `ArgMax`. We explain it with an example.

Example 2. RQL query to sort possible paths from the `Human` node to the `Cup` node.

$$\texttt{paths} := \texttt{fetch}(\{\texttt{name} : \text{`}\texttt{Human}'\}) \rightarrow [\texttt{r}*] \rightarrow (\{\texttt{name} : \text{`}\texttt{Cup}'\})$$
$$\texttt{SortBy}(\lambda \texttt{P} \rightarrow \texttt{Belief P})\,\texttt{paths}$$

In the example above, we first define a function `paths` which returns all the paths from the node `Human` to the node `Cup` in the form of a list. The `SortBy` query first runs the `paths` function and then sorts, in decreasing order, all paths in the returned list using their beliefs.

Programming construct functions. The programming construct functions serve to process the sub-graphs retrieved by the graph retrieval function `fetch`. In order to define these functions we make use of functional programming constructs like `map`, `filter` and `find`. We now explain the use of some of these constructs in RQL.

Example 3. RQL query to retrieve affordances of all the objects usable by a human.

$$\texttt{objects} := \texttt{fetch}(\{\texttt{name} : \text{`}\texttt{Human}'\}) \rightarrow [\text{`}\texttt{CanUse}'] \rightarrow (\texttt{v})$$
$$\texttt{affordances n} := \texttt{fetch}(\{\texttt{name} : \texttt{n}\}) \rightarrow [\text{`}\texttt{HasAffordance}'] \rightarrow (\texttt{v})$$
$$\texttt{map}(\lambda \texttt{u} \rightarrow \texttt{affordances u})\,\texttt{objects}$$

In this example, we illustrate the use of `map` construct. The `map` takes as input a function and a list, and then applies the function to every element of the list. More specifically, in the example above, the function `objects` retrieves the list of objects that the human can use. The `affordances` function takes as input an object and returns its affordances. In the last RQL query, the `map` applies the function `affordances` to the list returned by the function `objects`.

We now conclude this section with an expressive RQL query for retrieving *joint parameters* shared among nodes. Parameters are one of the many concepts we store in RoboBrain and they represent learned knowledge about nodes. The algorithms use joint parameters to relate multiple concepts and here we show how to retrieve joint parameters shared by multiple nodes. In the example below, we describe the queries for parameter of a single node and parameter shared by two nodes.

Example 4. Retrieve the joint parameters shared between a set of nodes.

$\texttt{parents n} := \texttt{fetch}\ (\texttt{v}) \rightarrow [\texttt{`HasParameters}'] \rightarrow (\{\texttt{handle : n}\})$

$\texttt{parameters n} := \texttt{fetch}\ (\{\texttt{name : n}\}) \rightarrow [\texttt{`HasParameters}'] \rightarrow (v)$

$\texttt{find\_parameters a} := \texttt{filter}(\lambda \texttt{u} \rightarrow \texttt{Len parents u} = 1)\texttt{parameters a}$

$\texttt{joint\_param}\ \texttt{a}_1\ \texttt{a}_2 := \texttt{filter}(\lambda \texttt{u} \rightarrow \texttt{Len parents u} = 2\ \texttt{andu in parameters}\ \texttt{a}_2)\ \texttt{parameters}\ \texttt{a}_1$

The query above uses the `filter` construct function and `Len` operation. The `filter` takes as input a list and a check condition, and returns only those items from the list that satisfies the input condition. The `Len` takes as input a list and returns the number of items in the list. In the query above, we first define a function `parents` which for a given input node returns its parent nodes. Then we define a function `parameters` which for a given input node returns its parameters. The third and the fourth queries are functions accepting one and two input nodes, respectively, and return the (joint) parameters that share an edge with every input node and not with any other node.

7 Applications

In this section we first show how RoboBrain can be used *as-a-service* by the robots for several robotics problems. Specifically, we explain the usage of RoboBrain in anticipating human activities, grounding of natural language sentences, and path planning. We then show how RoboBrain can help robotics projects by sharing knowledge within the projects and throughout the Internet.

7.1 RoboBrain as-a-service

Our goal with providing RoboBrain *as-a-service* is to allow robots to use the representations learned by different partner projects. This allows RoboBrain to effortlessly address many robotics applications. In the following we demonstrate RoboBrain as-a-service feature for three robotics applications that deal with different data modalities of perception, natural language and trajectories.

7.1.1 Anticipating human actions

The assistive robots working with humans should be able to understand human activities and also anticipate the future actions that the human can perform. In order to anticipate, the robot should reason over the action possibilities in the environment, i.e., *object affordances*, and how the actions can be performed, i.e., *trajectories*. Several works in robotics have addressed the problem of anticipation [28, 31, 32].

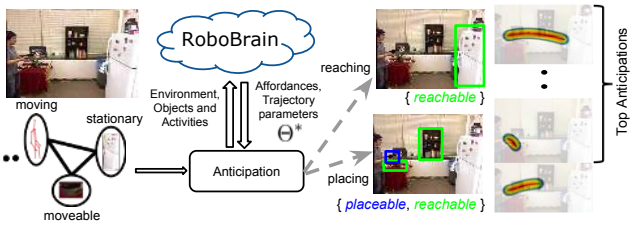

Fig. 5: **RoboBrain for anticipating human activities.** Robot using anticipation algorithm of Koppula and Saxena [31] queries RoboBrain, for the activity, affordance and trajectory parameters in order to generate and rank the possible future activities in a given environment.

We now show how robots can query RoboBrain and use the algorithm of Koppula et al. [31] for anticipating human actions. In order to anticipate the future human actions, the authors [31] learn parameters using their anticipatory algorithm, and using the learned parameters they anticipate the most likely *future* object affordances and human trajectories. RoboBrain serves anticipation *as-a-service* by storing those learned parameters, object affordances and trajectories as concepts in its graph. Figure 5 illustrates a robot retrieving relevant information for anticipation. The robot first uses the following queries to retrieve the possible trajectories of an object:

$$\texttt{affordances n} := \texttt{fetch}\ (\{\texttt{name} : \texttt{n}\}) \rightarrow [\text{`}\texttt{HasAffordance}'] \rightarrow (\texttt{v}\{\texttt{src} : \text{`}\texttt{Affordance}'\})$$

$$\texttt{trajectories a} := \texttt{fetch}\ (\{\texttt{handle} : \texttt{a}\}) \rightarrow [\text{`}\texttt{HasParameters}'] \rightarrow (\texttt{v}\{\texttt{src} : \text{`}\texttt{Affordance}', \texttt{type} : \text{`}\texttt{Trajectory}'\})$$

$$\texttt{trajectory\_parameters o} := \texttt{map}(\lambda \texttt{a} \rightarrow \texttt{trajectories a})\ \texttt{affordances o}$$

In the queries above, the robot first queries for the affordances of the object and then for each affordance it queries RoboBrain for the trajectory parameters. Having retrieved all possible trajectories, the robot uses the learned parameters [31] to anticipate the future human actions. Since the learned parameters are also stored in the RoboBrain graph, the robot retrieves them using the following RQL queries:

$$\texttt{parents n} := \texttt{fetch}\ (\texttt{v}) \rightarrow [\text{`}\texttt{HasParameters}'] \rightarrow (\{\texttt{handle} : \texttt{n}\})$$

$$\texttt{parameters n} := \texttt{fetch}\ (\{\texttt{name} : \texttt{n}\}) \rightarrow [\text{`}\texttt{HasParameters}'] \rightarrow (\texttt{v}\{\texttt{src} : \text{`}\texttt{Activity}'\})$$

$$\texttt{find\_parameters a} := \texttt{filter}(\lambda \texttt{u} \rightarrow \texttt{Len parents u} = 1)\texttt{parameters a}$$

$$\texttt{joint\_param}\ \texttt{a}_1\ \texttt{a}_2\ := \texttt{filter}(\lambda \texttt{u} \rightarrow \texttt{Len parents u} = 2\ \texttt{and u in parameters}\ \texttt{a}_2)\ \texttt{parameters}\ \texttt{a}_1$$

The queries above retrieves both independent and joint parameters for anticipating the object affordances and human activities. Detailed explanation of the query is given in Example 4 of Section 6

7.1.2 Grounding natural language

The problem of grounding a natural language instruction in an environment requires the robot to formulate an action sequence that accomplish the semantics of the instruction [53, 40, 38]. In order to do this, the robot needs a variety of information. Starting with finding action verbs and objects in the instruction, the robot has to discover those objects and their affordances in the environment.

We now show the previous work by Misra et al. [40] using RoboBrain *as-a-service* in their algorithm. In order to ground a natural language instruction the robot has to check for the satisfiability of the actions it generates in the given environment. For example, an action which pours water on a book should be deemed unsatisfiable. In

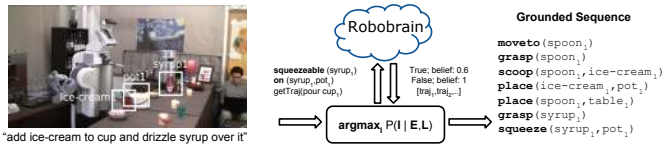

Fig. 6: **Grounding natural language sentence.** The robot grounds natural language by using the algorithm by Misra et al. [40] and querying RoboBrain to check for the satisfiability of actions. In another application, Tellex at al. [52] at Brown University query RoboBrain for placing mugs.

the previous work [40], the authors manually define many pre-conditions to check the satisfiability of actions. For example, they define manually that a *syrup bottle* is *squeezable*. Such satisfiability depends on the object's affordances in the given environment, which can be retrieved from RoboBrain.

Figure 6 illustrates a robot querying RoboBrain to check the satisfiability of actions that it can perform in the given environment. Below is the RQL query for retrieving the satisfiability of *squeezable* action:

$$\texttt{squeezable syrup} := \texttt{Len fetch}\,(\texttt{u}\{\texttt{name} : \text{`}\texttt{syrup}'\}) \rightarrow [\text{`}\texttt{HasAffordance}'] \rightarrow (\texttt{v}\{\texttt{name} : \text{`}\texttt{squeezable}'\}) > 0$$

7.1.3 Path planning using RoboBrain

One key problem robots face in performing tasks in human environments is identifying trajectories desirable to the users. An appropriate trajectory not only needs to be geometrically valid (i.e., feasible and obstacle-free), but it also needs to satisfy the user preferences [23, 24]. For example, a robot should move sharp objects such as knife strictly away from nearby humans [22]. Such preferences are commonly represented as cost functions which jointly model the environment, the task, and trajectories. Typically research groups have independently learned different cost functions [23, 32, 28], which are not shared across these groups. Here we show RoboBrain *as-a-service* for a robot to store and retrieve the planning parameters.

In Figure 8 we illustrate the robot planning for an egg carton by querying RoboBrain. Since eggs are *fragile*, users prefer to move them slowly and close to the surface of the table. In order to complete the task, the robot queries RoboBrain and retrieves the attributes of the egg carton and also the trajectory parameters learned in the previous work by Jain et al. [24]. Using the retrieved attributes and the parameters, the robot samples trajectories and executes the top-ranked trajectory. Below we show the RQL queries.

$$\texttt{attributes n} := \texttt{fetch}\,(\{\texttt{name} : \texttt{n}\}) \rightarrow [\text{`}\texttt{HasAttribute}'] \rightarrow (\texttt{v})$$

$$\texttt{trajectories a} := \texttt{fetch}\,(\{\texttt{handle} : \texttt{a}\}) \rightarrow [\text{`}\texttt{HasTrajectory}'] \rightarrow (\texttt{v})$$

$$\texttt{trajectory\_parameters} := \texttt{map}(\lambda \texttt{a} \rightarrow \texttt{trajectories a})\ \texttt{attributes}\ \text{`}\texttt{egg}'$$

7.2 RoboBrain for sharing knowledge

RoboBrain allows sharing the knowledge learned by different research groups as well as knowledge obtained from various internet sources. In this section we show with experiments how sharing knowledge improves existing robotic applications:

Sharing knowledge from the Internet. In this experiment we show that sharing knowledge from several Internet sources using RoboBrain improves robotic applications such as path planning. Knowledge from the Internet sources has been shown to help robots in planing better paths [55], understand natural language [47, 53], and

also recently in object retrieval [19]. However, for certain robotic tasks a single Internet source does not cover many of the real world situations that the robot may encounter. In such situations it is desired to use multiple sources to get richer representation. The RoboBrain graph is designed to acquire and connect information from multiple Internet sources and make it accessible to robots.

In this experiment we build upon work by Jain et al. [23] for planning trajectories that follow user preferences. The work relied on object attributes in order to plan desirable trajectories. These attributes convey properties such as whether an object is sharp, heavy, electronic etc. The attributes were manually defined by the authors [23]. In practice this is very challenging and time-consuming because there are many objects and many attributes for each object. Instead of manually defining the attributes, we can retrieve many of them from the Internet knowledge sources such as OpenCyc, Wikipedia, etc. However, a single knowledge source might not have attributes for all objects. The RoboBrain graph connects many attributes obtained from multiple Internet sources to their respective objects.

Figure 7 illustrates the planning results when the robot does not use any attributes, when it uses attributes from a single source (OpenCyc), and when it use attributes from RoboBrain. The planning performance is best when using RoboBrain since it covers more attributes than the OpenCyc alone. Most importantly all these attributes are retrieved from the RoboBrain graph with a single RQL query as explained in Section 7.1.3.

Sharing learned representations. New algorithms are commonly proposed for a problem to address the shortcomings of previous methods. These algorithms have their own learned representations. For example, different representations have been learned for grounding natural language [53, 40, 38]. It is usually hard for practitioners to select a representation since there could be inputs where one representation fails but some others work. In this experiment we show that a robot can query RoboBrain for the best representation while being agnostic to the algorithmic details of the learned representations.

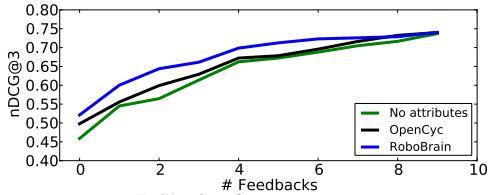

Fig. 7: Sharing from Internet sources.

Algorithm	IED	EED
Algorithm A	31.7	16.3
Algorithm B	23.7	**27.0**
RoboBrain (A+B)	**34.2**	24.2

Table 3: RoboBrain allows sharing learned representations.

Left: The plot shows performance of the algorithm by Jain et al. [23] for three settings of attributes. This is an online algorithm that learns a good trajectory from the user feedback. The performance is measured using the nDCG metric [37], which represents the quality of the ranked list of trajectories. RoboBrain combines information from multiple sources and hence its richer in attributes as compared to retrieving attributes from OpenCyc alone.

Right: It allows the robot to query RoboBrain for a representation given an input natural language command. In this table the Algorithm *A* is a greedy algorithm based on Misra et al. [40], and Algorithm *B* is their full model. The *IED* metric measures the string-edit distance and the *EED* metric measures the semantic distance between the ground-truth and the inferred output instruction sequences. The metrics are normalized to 100 such that higher numbers are better.

Simulating the above setting, we present an experiment for sharing multiple learned representations on a natural language grounding problem. Here the goal is

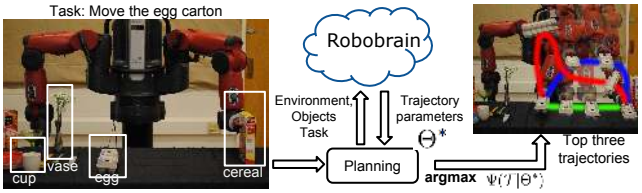

Fig. 8: **RoboBrain for planning trajectory.** The robot queries RoboBrain for the trajectory parameters (learned by Jain et al. [23]) to plan paths for the fragile objects like an egg carton.

to output a sequence of instructions for the robot to follow, given an input natural language command and an environment. Following the work by Misra et al. [40], we train a baseline algorithm for the task of *making ramen* (Algorithm A), and train their full algorithm for the task of *making affogato* (Algorithm B). These algorithms assign a confidence score (i.e., probability) to the output sequence of instructions. We store these learned representations as concepts in the RoboBrain graph, along with a prior belief over the correctness of the algorithms. The robot queries RoboBrain for a representation as follows:

$$\texttt{algParam} := \texttt{fetch}(\texttt{u}\{\texttt{type} :' \texttt{GroundingAlgorithm}'\}) \rightarrow [\text{`}\texttt{HasParameters}'] \rightarrow (\texttt{v})$$

$$\texttt{prior n} := \texttt{fetch}(\{\texttt{name} : \texttt{n}\}) \rightarrow [\text{`}\texttt{HasPriorProb}'] \rightarrow (\texttt{v})$$

$$\texttt{groundings L,E} := \texttt{argMaxBy}(\lambda(\texttt{u},\texttt{v}) \rightarrow \texttt{v})\texttt{map}(\lambda(\texttt{u},\texttt{v}) \rightarrow \texttt{u}(\texttt{L},\texttt{E},\texttt{v}) * \texttt{prior u})\ \texttt{algParam}$$

In the `algParam` function, we retrieve all natural language grounding algorithms from the RoboBrain graph with their parameters. This returns a list in which each element is a tuple of algorithm u and its parameters v. The `prior` function retrieves the prior belief over the correctness of an algorithm. In order to ground a given natural language command `L` in environment `E`, the `grounding` function evaluates the likelihood score for each algorithm using their parameters as $\texttt{u}(\texttt{L},\texttt{E},\texttt{v})$. It further incorporates the prior belief over the algorithms, and returns the representation with the highest likelihood score. These set of queries corresponds to the following likelihood maximization equation: $\mathscr{I}^* = \arg\max_{\mathscr{I},m' \in \{A,B\}} P(\mathscr{I}|E,L,w^*_{m'},m')P(m')$. As shown in the Table 3, choosing a representation by querying the RoboBrain achieves better performance than the individual algorithms.

8 Discussion and Conclusion

RoboBrain graph currently has 44347 nodes (concepts) and 98465 edges (relations). The knowledge in the graph is obtained from the Internet sources and through the project partners. For the success of many robotics application it is important to relate and connect the concepts from these different knowledge sources. In Figure 9 we plot the degree distribution of the RoboBrain graph and compare it with the degree distribution of independent knowledge sources. The graph of independent knowledge sources is the union of each knowledge source, which have nodes from all the projects and the edges only between the nodes from the same project. RoboBrain successfully connects projects and increases the average degree per-node by 0.8. RoboBrain graph has fifteen thousand nodes with degree one. Most of these nodes come from Wikipedia and WordNet. These nodes are not directly related to the physical world and represent concepts like political ideas, art categories, etc.

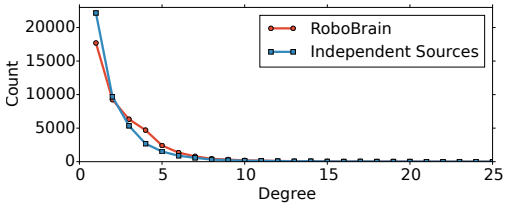

Fig. 9: Degree distribution of RoboBrain and the union of independent knowledge sources. For the case of independent sources, we only consider the edges between nodes from the same source. RoboBrain connects different projects successfully: number of nodes with degree 1 and 2 decrease and nodes with degree 3 and more increase.

In this paper we described different aspects and technical challenges in building a knowledge engine for robots. RoboBrain represents multiple data modalities from various sources, and connects them to get an overall rich graph representation. We presented an overview of the large-scale system architecture and developed the Robot Query Library (RQL) for robots to use RoboBrain. We illustrated robotics applications of anticipation, natural language grounding, and path planning as simple RQL queries to RoboBrain. We also showed in experiments that sharing knowledge through RoboBrain improves existing path planning and natural language grounding algorithms. RoboBrain is an ongoing effort where we are collaborating with different research groups. We are working on improving different aspects such as learning from crowdsourcing feedback, inference methods over the graph for discovering new relations between concepts, and expanding RoboBrain to new robotics applications.

Acknowledgement. This work was supported in part by ARO award W911NF-12-1-0267, ONR award N00014-14-1-0156, NRI award IIS-1426744, Google Faculty Research award (to Saxena), and Qualcomm research award. This was also supported in part by Google PhD Fellowship to Koppula, and by Microsoft Faculty Fellowship, NSF CAREER Award and Sloan Fellowship to Saxena.

[1] A. Anand, H. S. Koppula, T. Joachims, and A. Saxena. Contextually guided semantic labeling and search for 3d point clouds. *IJRR*, 2012.
[2] R. D. Andrea. Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead. *IEEE Tran. on Automation Science and Engineering (T-ASE)*, 9(4), 2012.
[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.
[4] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth. Robotic roommates making pancakes. In *Humanoids*, pages 529–536. IEEE, 2011.
[5] M. Beetz, M. Tenorth, and J. Winkler. open-ease a knowledge processing service for robots and robotics/ai researchers. *TZI Technical Report*, 74, 2014.
[6] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proc. ACM SIGMOD*, pages 1247–1250, 2008.
[7] M. Bollini, S. Tellex, T. Thompson, M. Roy, and D. Rus. Interpreting and executing recipes with a cooking robot. In *ISER*, 2012.
[8] M. Cakmak, S. S. Srinivasa, M. K. Lee, J. Forlizzi, and S.B. Kiesler. Human preferences for robot-human hand-over configurations. In *IROS*, 2011.
[9] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.
[10] X. Chen, A. Shrivastava, and A. Gupta. NEIL: Extracting Visual Knowledge from Web Data. In *ICCV*, 2013.
[11] V. Delaitre, D. Fouhey, I. Laptev, J. Sivic, A. Gupta, and A. Efros. Scene semantics from long-term observation of people. In *Proc. ECCV*, 2012.
[12] J. Deng, W. Dong, R. Socher, L-J Li, K. Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.
[13] X.L. Dong, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.
[14] A. Dragan and S. Srinivasa. Generating legible motion. In *RSS*, June 2013.
[15] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.
[16] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.
[17] D. A. Ferrucci. Introduction to this is watson. *IBM J. of RnD*, 56(3.4):1–1, 2012.

[18] M. J. Gielniak, C. Karen Liu, and A. L. Thomaz. Generating human-like motion for robots. *IJRR*, 32(11), 2013.
[19] S. Guadarrama, E. Rodner, K. Saenko, N. Zhang, R. Farrell, J. Donahue, and T. Darrell. Open-vocabulary object retrieval. RSS, 2014.
[20] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik. Learning rich features from RGB-D images for object detection and segmentation. In *Proc. ECCV*, 2014.
[21] J. Hoffart, F. M Suchanek, K. Berberich, and G. Weikum. Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.
[22] A. Jain, S. Sharma, and A. Saxena. Beyond geometric path planning: Learning context-driven trajectory preferences via sub-optimal feedback. In *ISRR*, 2013.
[23] A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *NIPS*, 2013.
[24] A. Jain, D. Das, J. K. Gupta, and A. Saxena. Planit: A crowdsourced approach for learning to plan paths from large scale preference feedback. *arXiv preprint arXiv:1406.2616*, 2014.
[25] Y. Jiang, H. Koppula, and A. Saxena. Hallucinated humans as the hidden context for labeling 3d scenes. In *CVPR*, 2013.
[26] Yun Jiang, Marcus Lim, and Ashutosh Saxena. Learning object arrangements in 3d scenes using human context. In *ICML*, 2012.
[27] D. Katz, A. Venkatraman, M. Kazemi, J. A. Bagnell, and A. Stentz. Perceiving, learning, and exploiting object affordances for autonomous pile manipulation. *Autonomous Robots*, 37(4), 2014.
[28] K. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert. Activity forecasting. In *Proc. ECCV*, 2012.
[29] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *ICRA*, 2013.
[30] H.S. Koppula and A. Saxena. Physically grounded spatio-temporal object affordances. In *Proc. ECCV*, 2013.
[31] H.S. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013.
[32] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *RSS*, 2012.
[33] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *ICRA*, 2011.
[34] D. B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, 1995.
[35] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. In *RSS*, 2013.
[36] J. Mainprice and D. Berenson. Human-robot collaborative manipulation planning using early prediction of human motion. In *IROS*, 2013.
[37] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.
[38] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *Proc. of the 13th International Symposium on Experimental Robotics (ISER)*, June 2012.
[39] C. McManus, B. Upcroft, and P. Newman. Scene signatures: Localised and point-less features for localisation. In *RSS*, 2014.
[40] D.K. Misra, J. Sung, K. Lee, and A. Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *RSS*, 2014.
[41] A. R. Mohamed, T. N. Sainath, G. Dahl, B. Ramabhadran, G. E. Hinton, and M. A. Picheny. Deep belief networks using discriminative features for phone recognition. In *(ICASSP)*, pages 5060–5063, 2011.
[42] T. Naseer, L. Spinello, W. Burgard, and C. Stachniss. Robust visual robot localization across seasons using network flows. In *AAAI*, 2014.
[43] S. Nikolaidis, P. Lasota, G. Rossano, C. Martinez, T. Fuhlbrigge, and J. Shah. Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action. In *Intl. Sym. on Robotics*, 2013.
[44] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
[45] N. Ratliff, J. A. Bagnell, and M. Zinkevich. Maximum margin planning. In *ICML*, 2006.
[46] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.
[47] M. Rouhizadeh, D. Bauer, R. E. Coyne, O. C. Rambow, and R. Sproat. Collecting spatial information for locations in a text-to-scene conversion system. *Computational Models for Spatial Languages*, 2011.
[48] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *IJCV*, 77(1-3), 2008.
[49] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *RSS*, 2013.
[50] J.M. Shepard, M.C. Towner, J. Lei, and P. Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *ICRA*, 2010.
[51] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.
[52] S. Tellex and J. Oberlin. Placing mugs with robobrain. `http://h2r.cs.brown.edu/placing-mugs-on-pedestals-with-robobrain/`.
[53] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.
[54] S. Tellex, R. Knepper, A. Li, D. Rus, and N. Roy. Asking for help using inverse semantics. In *RSS*, 2014.
[55] M. Tenorth, D. Nyga, and M. Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *ICRA*, 2010.
[56] M. Waibel, M. Beetz, R. D'Andrea, et al. Roboearth: A world wide web for robots. *IEEE R & A. Magz.*, 2011.
[57] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *RSS*, 2014.
[58] M. Zucker, N. D. Ratliff, A. D. Dragan, M. Pivtoraiko, M. K., C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. CHOMP: covariant hamiltonian optimization for motion planning. *IJRR*, 32(9-10), 2013.