RoboBrain:
Large-Scale Knowledge Engine for Robots

Ashutosh Saxena, Ashesh Jain, Ozan Sener, Aditya Jami, Dipendra K Misra, Hema S Koppula
Department of Computer Science, Cornell University and Stanford University.
Email: {asaxena,ashesh,ozansener,adityaj,dkm,hema}@cs.stanford.edu

***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 WWW and learned representations from leading 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.

I. Introduction

Over the last decade, we have seen many applications that have redefined machine intelligence by successfully combining information at a large-scale. Examples include Google knowledge graph [15], IBM Watson [19], Wikipedia, Apple Siri, and many others. The very fact they know answers to many of our day-to-day questions, and not crafted for a specific task, makes them valuable for humans. Inspired by them, researchers have aggregated domain specific knowledge by mining data [4, 6], processing natural language [10], images [13] and speech [45]. The knowledge available from these sources is centered around humans, however their symbolic nature 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).

Contrary to humans, for whom incomplete and ambiguous instructions may suffice, robots require access to a large variety of information with finer details for performing perception, planning, control and natural language understanding. Specifically, the robot would need access to knowledge for grounding the language symbols into physical entities, knowledge that sweet tea can either be on table or in fridge, and 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 robotics.

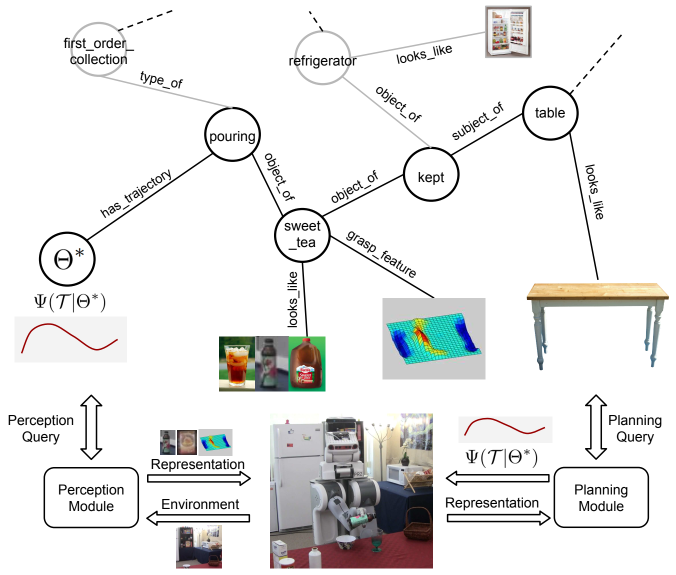


Fig. 1: **An example application showing a robot using RoboBrain for performing tasks.** The robot is asked "Bring me sweet tea from the kitchen", where it needs to translate the instruction into the perceived state of the environment. Our RoboBrain provides useful knowledge to the robot for performing 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.

In this paper, we present RoboBrain, a knowledge engine that allows robots to learn and share such knowledge. We learn these knowledge representations from a variety of sources, including interactions that robots may have while performing perception, planning and control, as well as natural language and other unstructured knowledge and visual data from the Internet. Our representation considers several modalities including symbols, natural language, visual or shape features, haptic properties, and so on. We believe that by learning and sharing such large-scale knowledge, different robots can become adept faster at performing a variety of tasks in new situations.

We mention a few challenges that we need to address while designing RoboBrain:

- *Multi-modal data.* Robots have a variety of sensors, ranging from visual (such as images, 3D point-clouds, and videos) to haptic. Our representations should be

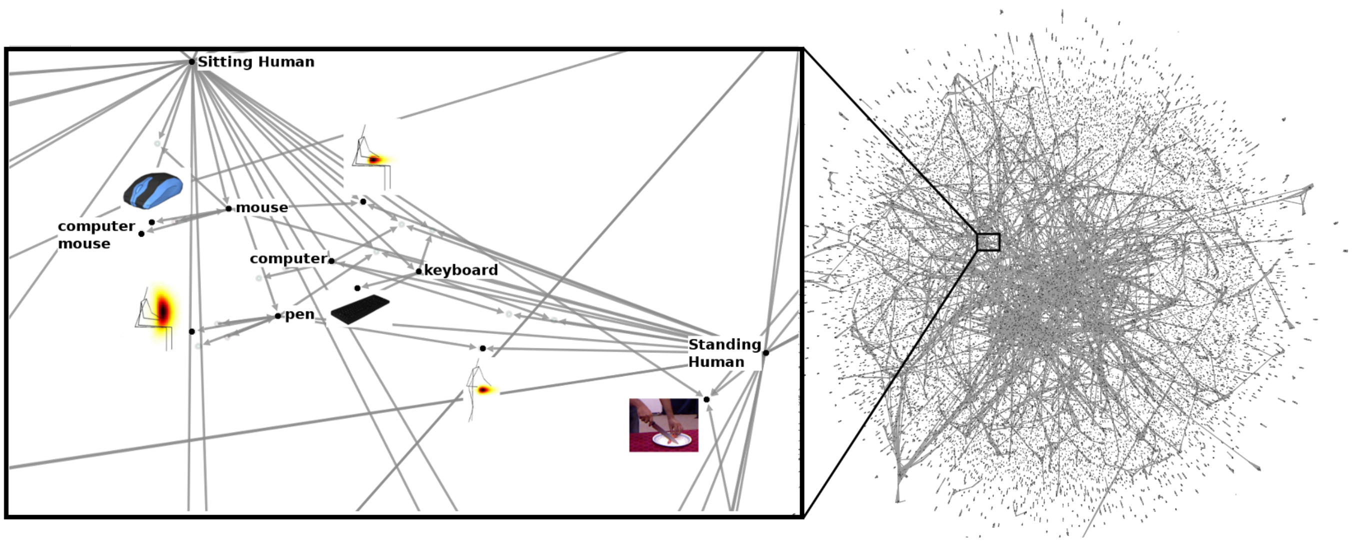


Fig. 2: **A visualization of the RoboBrain graph** on Nov 2014, showing about 50K nodes and 100K directed edges. The left inset shows a zoomed-in view of a small region of the graph with rendered media. This illustrates the relations between multiple modalities namely images, heatmaps, words and human poses. *For high-definition graph visualization, see: http://robobrain.me/graphNov26_2014.pdf*

capable of encoding these different modalities.

- *Beliefs in the concepts.* The concepts learned from the data come from a variety of noisy sources. RoboBrain should be able to represent the belief in the correctness of learned concepts.
- *Never ending learning.* Our knowledge engine should be able to incorporate new knowledge and update beliefs. The system should be scalable, allow real-time updates to existing knowledge, and allow knowledge sharing across robots and users.
- *Ability to answer queries.* Many robotic tasks such as perception, reasoning and grounding natural language should be able to make use of the knowledge in RoboBrain. We provide an interface to RoboBrain as a query library for building various applications.
- *Knowledge acquisition.* RoboBrain acquires knowledge from different sources, including physical robot interactions and partner projects. Thus we need an architecture that is compatible with most of the sources. Currently, the partners include Tell Me Dave [44], Tellex's group [56], PlanIt [26] and affordances [29].
- *Crowd-sourcing.* Incorporating knowledge from different sources may lead to significant inconsistencies. Therefore, we need to have some supervisory signal for learning correct concepts in RoboBrain. Hence, our system comprises several forms of feedback that allows users to give feedback at different levels of engagement.

In order to address the aforementioned challenges, we present an architecture that can store the large-scale knowledge, in a way that is efficient to retrieve and update during learning. Our knowledge is multi-modal and primarily contains relational information, therefore we use a graph representation of the knowledge (see Figure 1 and 2). We also present a query library that robots can use for performing different tasks.

We present use of RoboBrain on three applications in the area of grounding natural language, perception and planning. In first, we show how we can ground natural language commands into an action plan for robot controllers. In second, we consider human activity anticipation. Many perception tasks require different contextual information such as spatial context, affordances and others. In third, we consider path planning for mobile manipulators in context-rich environments. When planning a path, a robot needs to have the knowledge of the usage of different objects and human preferences. We show how we use RoboBrain to formulate a cost function for the planner to plan paths.

One nice property that emerges because RoboBrain being an integrated knowledge engine is that representations are now shared *across* research areas. For example, knowledge about how cups are placed on a table (upright, especially when they contain liquids) could have been learned from perception, but it would help in manipulation as well.

As more and more researchers contribute knowledge to RoboBrain, it will not only make their robots perform better but we also believe this will be beneficial for the robotics community at large. RoboBrain is under open, Creative Commons Attribution license (aka CC-BY),[1] and is avalable at: `http://robobrain.me`

II. Related Work

Knowledge bases. Collecting and representing a large amount of information in the form of a machine-readable knowledge base (KB) has been widely studied in the areas of artificial intelligence, data mining, natural language processing and machine learning. Early seminal works have manually created knowledge bases (KBs) with the study of common

[1]http://creativecommons.org/licenses/by/2.5, Extra conditions may apply depending on the data sources, see robobrain.me for detailed information.

sense knowledge (Cyc [40]) and lexical knowledge of english language (WordNet [18]). With the rapid growth of Wikipedia, KBs started to use crowdsourcing (DBPedia [4], Freebase [6]) and automatic information extraction methods (Yago [55, 24]) for mining knowledge.

Recently, several approaches to KB generation from web data and other unstructured sources have been developed. Examples include NELL [10] and NEIL [11]. 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 [13] enriched the wordnet synsets with images obtained from large-scale internet search and a crowdsourced approach was used to obtain the object labels. These object labels were further extended to object affordances [61].

In industry, we have seen many successful applications of the existing KBs within the modalities they covered, for example, Google Knowledge Graph and IBM Watson Jeopardy Challenge [20]. However, the existing KBs do not apply directly to robotics problems where the knowledge needs to be about entities in the physical world and of various modalities.

Robotics Works. For robots to operate autonomously they should perceive our environments, plan paths, manipulate objects and interact with humans. This is very challenging because solution to each sub-problem varies with the task, human preference and the environment context. We now briefly describe previous work in each of these areas.

Perceiving the environment. Perception is a key element of many robotic tasks. It has been applied to object labeling [37, 2, 60], scene segmentation [23], robot localization [43, 46], feature extraction for planning and manipulation [30], understanding environment constraints [28] and object affordances [12, 34]. Sharing representations from visual information not only improve the performance of each of the perception tasks, but also significantly help various applications such as autonomous or assistive driving [14, 8], anticipation [31, 35, 59], planning sociable paths [36] and for various household chores such as grasping and cutting [41]. Sharing representations from other modalities such as sound [52] and haptics [21] would also improve perception.

Path planning and manipulation. There exist a large class of algorithms which allow robots to move around and modify the environment. Broadly the planning algorithms can be categorized as motion planning [62, 53], task planning [1, 7] and symbolic planning [17, 51]. Bakebot [7], towel-folding [54], IkeaBot [32] and robots preparing pancakes [5] are few of the many successful planning applications. In order to execute complex manipulation tasks, cost functions have been learned in a data-driven manner [50, 26]. Most planning algorithms abstract out the perception details, however as explained earlier, access to perception and manipulation knowledge can allow robots to plan in dynamic real world environments.

Interacting with humans. Another important aspect is human-robot interaction. Previous works have focused on various aspects, such as human-robot collaboration for task completion [48, 33], generating safe robot motion near humans [42, 39], obeying user preferences [9], generating human like and legible motions [22, 16], interaction through natural language [57, 44], etc. All these applications require access to perception, manipulation, language understanding, etc., further demonstrating the need for large scale multi-modal data.

Previous efforts on connecting robots range from creating a common operating system (ROS) for developing robot applications [49] to sharing data acquired by various robots in the cloud [58, 3]. For example, RoboEarth [58] provides a platform for the robots to store and off-load computation to the cloud and communicate to other robots; and KIVA systems [3] use the cloud to coordinate motion and update tracking data for hundreds of mobile platforms. However, RoboBrain knowledge engine provides the knowledge representation layer on top of data storing, sharing and communication.

III. Overview

We represent the knowledge in RoboBrain as a graph (see Section IV, Figure 1 and Figure 2). The concepts are the nodes in the graph, and the edges represent relations between them.

RoboBrain is a never ending learning system in that it continuously incorporates new knowledge from different sources and modifies the graph. In other words, we look at every additional knowledge acquisition as a measurement event that dynamically modifies the graph. In the next section, we will describe the formal definition of the RoboBrain graph, how we update it, and the concept of *latent* graph.

Before we describe the technical specification of RoboBrain, we mention a few of the technical components:

Knowledge acquisition. RoboBrain acquires knowledge from various sources, including physical robot interactions, knowledge bases from WWW, and from partner projects. One key knowledge resource for RoboBrain is by crawling knowledge sources on the Internet such as WordNet, ImageNet, Freebase and OpenCyc. These knowledge sources provide lexical knowledge, grounding of concepts into images, and common sense facts about the world. In detail,

- *WordNet* provides the required lexical knowledge that allows us to group nouns and verbs into distinct cognitive concepts.
- *ImageNet* grounds the noun category from WordNet into images. This allows our knowledge graph to associate an image with every noun.
- *OpenCyc* is an ontology for everyday common sense knowledge.

We are also crawling many other sources such as Wikipedia to add knowledge to RoboBrain. The technical details of adding knowledge to RoboBrain is described in Section IV.

Disambiguation. Incorporating new knowledge into the graph is challenging. We need to decide whether the incoming knowledge should create new nodes and edges or instead add information to the existing nodes and edges. Since our knowledge graph carries semantic meaning, it should resolve polysemy using the context associated with nodes. For example, a 'plant' could mean a 'tree' or an 'industrial plant' and merging them together will create errors in the graph. Therefore, RoboBrain needs to disambiguate the incoming information using the associated context and decide how to add it to the existing knowledge graph.

Beliefs. Knowledge that we incorporate in the RoboBrain may not be fully reliable and there could be inconsistencies in the knowledge coming from different sources. We need a way to indicate beliefs in the correctness of the concepts and relations in the RoboBrain.

Online crowd-sourced interaction. We take inspiration from large-scale crowd-sourced projects such as Wikipedia. While previous works have focussed on using crowd-sourcing for data collection and labeling (e.g., [13]), our work focuses on taking crowd-sourced feedback at many levels:

- *Weak feedback.* We have found that it is easiest for users to provide a binary "Approve"/"Disapprove" feedback while they are browsing online. Our RoboBrain system encourages users to provide such feedback, which is then used for several purposes such as estimating belief on the nodes and edges.
- *Feedback on graph.* We present the graph to the users on the Internet and smartphones, where they can give feedback on the correctness of a node or an edge in the graph. Such feedback is stored for inferring the latent graph. The key novelty here is that the system shows the proposed "learned" concepts to the users — thus getting feedback not at the data level but at the knowledge level.
- *Feedback from partner projects.* There are several partner projects to RoboBrain, such as Tell Me Dave [44], PlanIt [26], and so on. These projects have a crowd-sourced system where they take feedback, demonstrations or interaction data from users. Such data as well as the learned knowledge is shared with the RoboBrain.

We not only need to figure out how to present knowledge in a consumable form to users but also need to figure out how to store and use the user feedback in improving the quality and correctness of knowledge in the RoboBrain.

Queries. The primary purpose of the RoboBrain is to allow queries by the users and robots. We present a RoboBrain Query Library (RaQueL) that is used to interact with the RoboBrain knowledge base. It comprises of functionalities for retrieving sub-graph, ranking relations between nodes, using beliefs, functions to work with multi-modal media and other application tailored functionalities that allow its use by the users and robots. Details of RaQueL along with usage examples are provided in Section VI.

IV. Knowledge Engine: Formal Definition

In this section, we present the formal definition of RoboBrain (RB) Knowledge Engine.

Our RB is expressed as a directed graph $\mathcal{G} = (V, E)$, see Figure 2 for an example. The vertices V can be of a variety of types such as image, text, video, haptic data, or a learned entity such as affordances, deep features, parameters, etc. The edges $E \subseteq V \times V \times C$ link two nodes with a direction and a type, where C is the set of edge types. See Table I and II for a few examples.

We further define a set of auxiliary functions $X^v(\cdot), X^e(\cdot), b^v(\cdot)$ and $b^e(\cdot)$ where $X^v(v)$ and $X^e(e)$ is the raw data associated with the vertex v and the edge e, and $b^v(v)$ and $b^e(e)$ are the beliefs that indicate the correctness of the node v and the edge e respectively.

An edge (v_1, v_2, ℓ) is an ordered set of two nodes v_1 and v_2 of type ℓ. Few examples of such edges are: (StandingHuman, Shoe, *CanUse*), (StandingHuman, $\mathcal{N}(\mu, \Sigma)$, *SpatiallyDistributedAs*) and (Grasping, DeepFeature23, *UsesFeature*). These edges can be considered as (*subject*, *object*, *predicate*) triplets. Although we do not impose any constraints over the nodes in terms of content or modality, we require the edges to be consistent with the RB edge set.

A. 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 infer the latent information in the aggregated data. These two stages are explained in more detail below.

Knowledge acquisition: RoboBrain accepts new information in the form of an edge set. This information can either be from an automated algorithm or from our research collaborators. We call this edge set a *feed*.

We add the new information to the existing graph through a graph union of the existing graph and the *feed* followed by an inference. Specifically, given a new *feed* consisting of N edges $(v_1^1, v_2^1, \ell^1) \ldots (v_1^N, v_2^N, \ell^N)$ and an existing graph $G = (V, E)$, graph union stage gives us $G' = (V', E')$ as:

$$\begin{aligned} V' &= v_1^1 \cup v_2^1 \cup \ldots \cup v_1^N \cup v_2^N \cup V \\ E' &= (v_1^1, v_2^1, \ell^1) \cup \ldots \cup (v_1^N, v_2^N, \ell^N) \cup E \end{aligned} \tag{1}$$

Inferring the Latent Graph: We treat the acquired knowledge as a noisy observation of the real knowledge-base that *should* contain the proper information about the real physical world. We may not be able to observe the real graph directly, therefore we call it the *latent graph*. For example, latent information *"coffee is typically in a container"* is partially observed through many images of a *container* in media corresponding to *coffee*. This construction can be explained in a generative setting as having a latent graph $\mathcal{G}^\star$ of all the knowledge about physical word. The measurements through various channels of limited expressiveness (e.g., natural language, trajectories, etc.), in the form of *feeds*, create an ambiguity over the underlying latent graph.

In order to obtain the latent graph $\mathcal{G}^\star(V^\star, E^\star)$, we can apply several operations over the aggregated data. We describe two of them here: *split* and *merge*. Split operation is defined as splitting a node into a set of two nodes. Edges having end point in the split node are connected to one of the

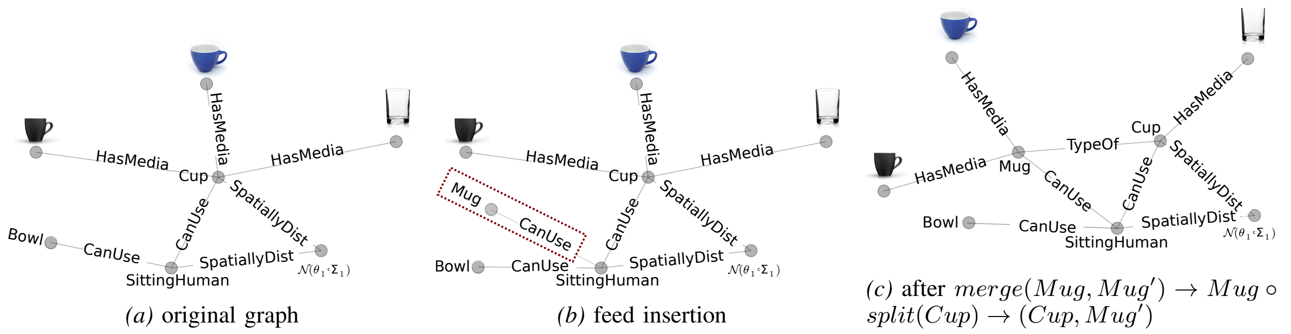

HasMedia
Cup
HasMedia
SpatiallyDist
CanUse
Bowl
CanUse
SittingHuman
$\mathcal{N}(\theta_1,\Sigma_1)$
Mug
TypeOf
(a) original graph
(b) feed insertion
(c) after $merge(Mug, Mug') \rightarrow Mug \circ split(Cup) \rightarrow (Cup, Mug')$

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

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

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

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.

resultant nodes using the disambiguation algorithm. A merge operation is defined as merging two nodes into a single node, while updated the connected edges with the merged node. These two operations create latent concepts by splitting nodes to semantically different subsets or inferring common latent concepts that exist in multiple nodes via merge operation. An example of such an update is shown in Figure 3. When a new information *"sitting human can use a mug"* is added, it causes the split of *Cup* node into *Cup* and *Mug* nodes.

Thus we can define the inferred latent RoboBrain graph as:

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

Resulting Latent Graph: Since we do not expect the acquired knowledge to be complete, we treat the information probabilistically. In other words, we express the uncertainty over the missing information as a set of beliefs. These beliefs represent the confidence over the created nodes and edges. In other words, we apply an *update* operation on the belief functions as new knowledge is added to the RoboBrain. Ideally, the beliefs should converge to the latent graph of the physical world with more information and feedback. Practically, however, we never expect full convergence, and we design probabilistic query algorithms that use the beliefs for answering the input queries.

V. System Architecture

We now describe the system architecture of RoboBrain, shown in Figure 4. The system consists of four interconnected

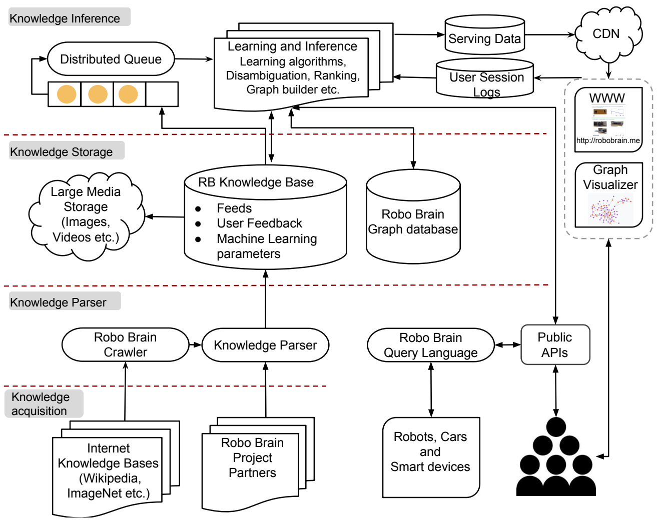

Knowledge Inference
Distributed Queue
Learning and Inference
Learning algorithms, Disambiguation, Ranking, Graph builder etc.
Serving Data
CDN
User Session Logs
WWW
http://robobrain.me
Graph Visualizer
Knowledge Storage
Large Media Storage (Images, Videos etc.)
RB Knowledge Base
Feeds
User Feedback
Machine Learning parameters
Robo Brain Graph database
Knowledge Parser
Robo Brain Crawler
Knowledge Parser
Robo Brain Query Language
Public APIs
Knowledge acquisition
Internet Knowledge Bases (Wikipedia, ImageNet etc.)
Robo Brain Project Partners
Robots, Cars and Smart devices

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

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 structured RoboBrain graph. In addition, our design also supports various mechanisms for users and robots to interact with the brain. Through these interactions users retrieve knowledge and also improve RoboBrain through feedback. Below we discuss each of the components.

Knowledge acquisition layer is the interface between the RoboBrain and the different sources of multi-modal data. Through this layer RoboBrain gets access to new information which the upper layers process (see Figure 4). RoboBrain primarily collects knowledge through its partner projects and by crawling the existing knowledge bases such as Freebase, ImageNet and WordNet, etc., as well as unstructured sources such as Wikipedia and Youtube.

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. Further, since 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 – RB Knowledge Base (RB-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. RB-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 (RB-GD) which stores the structured knowledge. The data from RB-KB is refined through multiple learning algorithms and its graph representation is stored in RB-GD. One of the primary goal behind this design is to keep RB-KB as RoboBrain's single source of truth (SSOT). SSOT centric design allow us to re-build RB-GD in case of failures or malicious knowledge sources.

Knowledge inference layer contains the key processing and machine learning components of RoboBrain. All the new and recently updated feeds go through a persistent replicated distributed queuing system which are consumed by some of our machine learning plugins (disambiguation, graph builder, etc.) and populates the graph database. These plugins along with other learning algorithms (that operates on entire knowledge graph) constitutes our Learning and Inference framework. This framework comprises a bunch of fault tolerant restful application servers that are scaled automatically based on load. Every application server also bookmarks the progress information (metadata) so that a new server can catch up instantly if one of the systems die.

RoboBrain also support various *interaction mechanisms* to enable robots, cars and users to communicate with RoboBrain knowledge base. RoboBrain query library is the primary method for robots to interact with the RoboBrain. A set of public APIs allow information to be presented on WWW for enabling online learning mechanisms (crowd-sourcing).

Our RoboBrain system is designed with several other desired properties such as security, overall throughput, monitoring and alerting. Our front end and API servers (querying and crowd-sourcing systems) are auto-scaled based on load and most of the data is served using a commercial content delivery network to reduce the end user latency. The detailed description of the system design is beyond the scope of this paper, and will be described in future publications.

VI. RoboBrain Query Library ($RaQueL$)

For robots to perform tasks (see Section VII for some examples), they not only need to retrieve the stored knowledge (e.g., trajectories, heatmaps, object affordance, natural language lexicons, etc.) but also need to perform certain types of operations specific to the applications. This situation is not usually encountered in traditional databases, and thus we need to design a RoboBrain Query Library ($RaQueL$) for making RoboBrain easily usable.

$RaQueL$ can be used for diverse tasks such as semantic labeling, cost functions and features for grasping and manipulation, grounding language, anticipating activities, and fetching trajectory representations. Typically, the queries involve finding a sub-graph matching a given pattern, retrieving beliefs over nodes and edges, and then performing reasoning on the extracted information. The structure of $RaQueL$ admits the following three types of functions:

- **Graph retrieval function**: `fetch` function for querying a sub-graph of RoboBrain that satisfies a given pattern. A pattern defines relations between node and edge variables. E.g., the pattern given below defines two node variables u,v in parenthesis and one edge variable e in square brackets. The arrows represent the edge direction.
$$(u) \rightarrow [e] \rightarrow (v)$$
This pattern is satisfied by all paths of length 1 and `fetch` function can then be used to retrieve all of them.
- **Integrating programming constructs**: $RaQueL$ integrates programming language constructs such as `map`, `filter`, `find`, etc.
- **Task specific functions**: Functions for performing a specific type of task, such as ranking trajectories, semantic labeling of an image, grounding natural language, functions to work with heatmaps, etc.

We describe these functions in detail in the following.

A. Graph retrieval function

The `fetch` function is used to retrieve a sub-graph of RoboBrain that matches an input pattern. The function `fetch` takes as input a pattern $Pattern(arg_1, arg_2, \cdots)$ where $\{arg_1, arg_2, \cdots\}$ are the pattern variables and returns a list of tuples with instantiated values of variables.

$$\texttt{fetch} : Pattern(arg_1, arg_2, \cdots) \rightarrow [(arg_1^{(i)}, arg_2^{(i)}, \cdots)]_i$$

This pattern is matched against sub-graphs of RoboBrain and these matched sub-graphs are used to create initialized entries for the pattern variables. $RaQueL$ uses Neo4j Cypher [47] to implement the `fetch` functionality. We now explain the usage of `fetch` in the following examples.

Example 1: Retrieve all the activity nodes connected to nodes with name "cup".

$$\texttt{fetch}(\{\texttt{name} : \text{`}\texttt{cup}'\}) \rightarrow [\{\texttt{istype} : \text{`}\texttt{IsActivity}'\}] \rightarrow (\texttt{v})$$

The parenthesis describes a node variable whose attribute specifications are given inside the curly brackets. In this query, the $name$ attribute of the node should be '`cup`'. The edge variable also has a `istype` attribute specification. The pattern will match against paths of length one $\{u, e, v\}$; where node u has name '`cup`' and edge e has type '`IsActivity`'. Since there is only one exposed variable v, the `fetch` will return a list of tuple with instantiated values of variable v only.

For brevity, we will directly write the attribute value for edges when attribute name is `istype`.

We can also fetch paths between nodes using the extension of the above example. These paths can be unbounded or bounded by a given length.

***Example** 2:* Return the path between the nodes with name "cup" and "chair".

$$\texttt{fetch}(\{\texttt{name}: \text{`}\texttt{cup}'\}) \rightarrow [\texttt{r}*] \rightarrow (\{\texttt{name}: \text{`}\texttt{chair}'\})$$
$$\texttt{fetch}(\{\texttt{name}: \text{`}\texttt{cup}'\}) \rightarrow [\texttt{r}*5] \rightarrow (\{\texttt{name}: \text{`}\texttt{chair}'\})$$

The first query returns edges of all the paths between nodes with name '*cup*' and '*chair*'. The second one only returns edges of all paths of length less than or equal to 5.

The `fetch` function in general can accept any pattern string allowed by Cypher. For more details we refer the readers to Cypher's documentation [47].

B. Integrating Programming Constructs

While `fetch` function is sufficient for queries where the pattern is known in advance. For other type of queries, we want to make a sequence of `fetch` queries depending upon the results from the previous queries. This is specially useful for greedy exploration of the graph e.g, by fetching only on those nodes, returned by a fetch query, which have a high belief. For this reason, $RaQueL$ supports functional programming constructs such as `map`, `filter`, `find`, etc.[2]

We now present how these constructs can be combined with `fetch` to do a greedy exploration of RoboBrain. Consider the problem of exploring the sub-graph rooted at a given node u and finding its relations to all other nodes such that these relations have a certain minimum belief.

Formally, by relation R_u of a given node u we refer to a tuple (u, P, v), where P is a path from node u to node v.

In such cases, exploring the entire graph is impractical. Hence, several greedy search heuristics can be employed. We now take up a specific example, to explain this further:

***Example** 3:* Find at least N number of relations of nodes named '*cup*' which have a belief of atleast b.

$$\texttt{query t} := \texttt{fetch}\ (\texttt{u}\{\texttt{name}: \text{`}\texttt{cup}'\}) \rightarrow [\texttt{P}: * \cdots \texttt{t}] \rightarrow (\texttt{v})$$
$$\texttt{trustedPath t} := \texttt{filter}\,(\lambda\,(\texttt{u},\texttt{P},\texttt{v})\ \rightarrow\ \texttt{belief P}\ \geq\ \texttt{b})\ \texttt{query t}$$
$$\texttt{find}\,(\lambda\texttt{rel} \rightarrow \texttt{len rel}\ \geq\ \texttt{N})\ \texttt{map}\,(\lambda\,\texttt{t}\ \rightarrow\ \texttt{trustedPath t})\,[\texttt{1}\cdots]$$

In this query, we begin by defining two functions `query` and `trustedPath`. The `query` function takes a parameter t and returns all relations (u, P, v) such that u is a node with name '*cup*' and P is a path of length less than t. The `trustedPath` function takes a parameter t and filters in only those relations (u, P, v) returned by calling `query t`, such that path P has a belief of at least b. Here, the function `belief P` is an user-defined function which returns a belief for path P. This could be as simple as addition/multiplication of beliefs over nodes and edges in path P. The last line gradually calls the `trustedPath` function on increasing values of $t \in \{1, 2 \cdots\}$ and each time returning a set of relations whose belief is at least b. The `find` function then returns the first element (which itself is a list) in this list whose length is at least N. Handling the corner case where there does not exist a set of N relations of belief at least b, is not shown in this example for brevity. Note that since these programming constructs are lazy-evaluated, the last line terminates as soon as it explores the sub-graphs rooted at '*cup*' nodes until the depth required for finding N relations with belief at least b.

[2]`map` applies a function to list elements, `filter` returns list elements which satisfy a predicate, `find` returns the first element that satisfies the predicate.

C. Task Specific Functions

$RaQueL$ provides functions for special tasks such as semantic labeling of an image, ranking a trajectory based on human preference, finding a set of affordances for a given object, grounding natural language expressions, etc. $RaQueL$ also provides functionalities to work with media files of RoboBrain, such as functions to generate and work with heatmaps, trajectories, videos, etc. For full list of functions supported by $RaQueL$ and more examples, please refer to the RoboBrain website.

VII. Applications

In this section, we show use of RoboBrain in following three applications: (a) grounding natural language into controllers, (b) anticipating human actions, and (c) planning in presence of humans.

A. Grounding Natural Language

In the context of a given environment and natural language discourse, we define the problem of grounding a natural language command as coming up with an action plan for a robot that accomplishes close to the given command.

It is challenging to ground natural language for an open set of tasks. For example, grounding the sentence *"get me a cup of hot water"* requires that the robot must know the appearance of *cup*, it should be able to infer that *cup* has the *containable* affordance. The robot must be able to classify if the *cup* has water or not and in case the cup is empty then the robot must know that objects such as *tap, fridge dispenser, etc.* can potentially fill it with water. It must also know that objects such as *stove, microwave* can heat an object. Finally, robot should be able to manipulate these devices to accomplish the required sub-goals. Specifically, this consists of knowledge of environment such as manipulation features of objects, object affordances, prior constraints over state-action space etc.; of language such as synset relations, POS tags and knowledge that requires both such as grounding of noun-phrases like "red mug". We now take specific examples to show how some of these queries are translated to a graph-query on RoboBrain.

Use of RoboBrain. A common approach to language grounding problem is to maximize the likelihood $P(\mathcal{I}|E, L)$ of the grounded instruction sequence $\mathcal{I}$ given the environment E and the natural language command L. Figure 5 shows such a setting for the task of making recipes. The instruction sequences typically consist of action primitives such as *pour, grasp, keep* etc. that are suitable for high-level planning. For example, the instruction sequence corresponding the command *"fill a cup with water"* could be:

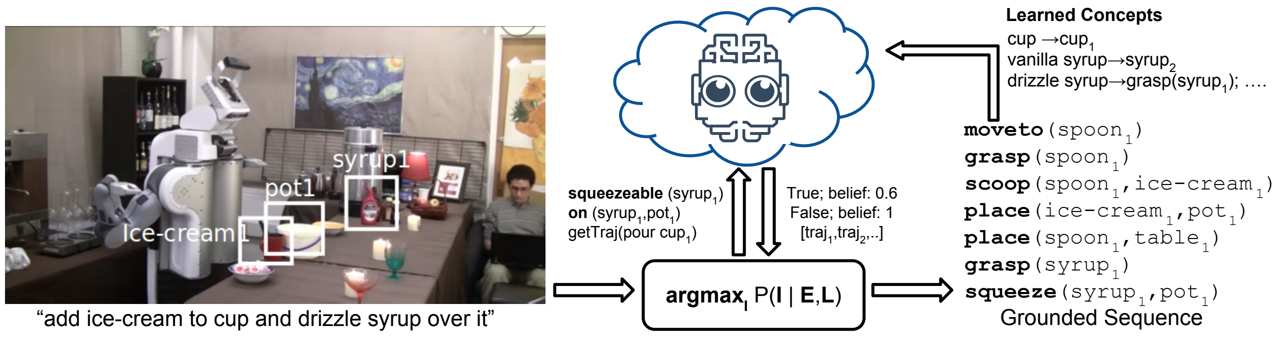


Fig. 5: **Grounding a natural language sentence** for a given environment requires several pieces of knowledge. Robot should know the appearance of each object, it should know the affordances and possible action related to each object and it should know the trajectories and manipulation features of those actions. These are all translated as queries to the RoboBrain graph.

moveto(cup_{01}); grasp(cup_{01}); moveto($sink_{01}$); keep(cup_{01}, on, $sink_{01}$); toggle($sink\_knob_{01}$); wait(); toggle($sink\_knob_{01}$);

However not all instructions are meaningful (trying to pour from a book), and not all instruction sequences are valid (keeping cup inside microwave without opening the door). In order to come up with meaningful instruction sequence $\mathcal{I}$ requires knowing what actions can be performed with a given object and what sequences of instructions are permitted. Misra et al. [44] used strips language to represent pre-conditions for a given instruction.

For example, in Figure 5 the action primitive *squeeze($syrup_1$,pot_1)* is meaningful only if it satisfies the following predicate:

- *grasping pr2 $syrup_1$*: Robot is grasping the syrup.
- *squeezeable $syrup_1$*: Syrup bottle should be squeezeable.
- *on $syrup_1$ pot_1*: Syrup bottle should be placed directly above the pot.

RoboBrain is used to compute satisfiability of these predicates which are challenging to compute for an arbitrary object. For example, the satisfiability of the second predicate, for text modality, is written as the following $RaQueL$ query:

$$\begin{aligned}\textit{squeezeable syrup}_1 \;=\; & \texttt{len fetch } (\texttt{u}\{\texttt{name} : \text{`}\texttt{syrup}'\}) \rightarrow \\ & [\text{`}\texttt{HasAffordance}'] \rightarrow (\texttt{v}\{\texttt{name} : \text{`}\texttt{squeezeable}'\}) \;\; > 0\end{aligned}$$

Once the robot finds the optimum instruction sequence, it needs to know how to translate action primitives such as *pour(cup_{01})* into trajectories. This is challenging since the trajectory would depend upon the object parameter (consider pour(cup_{01}) and pour($kettle_{01}$)). These are translated to queries to RoboBrain as follows:

$$\begin{aligned}\textit{pour cup}_{01} \;=\; & \texttt{fetch } (\{\texttt{name} : \text{`}\texttt{pour}'\}) \rightarrow \\ & (\texttt{v}\{\texttt{name} : \text{`}\texttt{HasTrajectory}'\}) \leftarrow (\{\texttt{name} : \text{`}\texttt{cup}'\})\end{aligned}$$

The above example returns all the trajectory nodes(containing trajectory parameters) that are connected to nodes with name *cup* and nodes with name *pour*. Note that since RoboBrain returns the result with beliefs that are continuously updated by user feedback, we further order them by rank and use the highest belief node or apply more specific filters, such as safety.

B. Anticipating Human actions

Assistive robots working with humans need to reactively respond to the changes in their environments. In addition to understanding what can be done in the environment and detect the human activities, the robots also need to anticipate which activities will a human do next (and how). This enables an assistive robot to plan ahead for reactive responses [35].

In this application, our goal is to predict the future activities as well as the details of how a human is going to perform them in short-term (e.g., 1-10 seconds). For example, if a robot has seen a person move his hand to a coffee mug, it is possible he would move the coffee mug to a few potential places such as his mouth, to a kitchen sink or to a different location on the table. In order to predict this, we model the activities as a conditional random field which captures the rich spatial-temporal relations through object affordances [35]. However, the challenge is that there are a large number of objects and affordances that need to be stored and updated as new affordances are discovered.

Use of RoboBrain. In this work, we use RoboBrain to store the learned activity, affordance and trajectory parameters and query for the appropriate knowledge as and when required by the anticipation algorithm. More specifically, there are two sets of queries issued to the RoboBrain. We first query for the RoboBrain for node and edge parameters of the conditional random field model as shown below:

$$\begin{aligned}&\textit{parents}\ \texttt{n} := \texttt{fetch}\ (\texttt{v}) \rightarrow [\text{`}\texttt{HasParameters}'] \rightarrow (\{\texttt{handle} : \texttt{n}\})\\ &\textit{parameters}\ \texttt{n} := \texttt{fetch}\ (\{\texttt{name} : \texttt{n}\}) \rightarrow [\text{`}\texttt{HasParameters}'] \rightarrow\\ &\qquad\qquad (\texttt{v}\{\texttt{src} : \text{`}\texttt{Activity}'\})\\ &\textit{node\_parameters}\ \texttt{a} = \texttt{filter}(\lambda \texttt{u} \rightarrow \textit{len parents}\ \texttt{u} = 1)\textit{parameters}\ \texttt{a}\\ &\textit{edge\_parameters}\ \texttt{a}_1\ \texttt{a}_2\ = \texttt{filter}(\lambda \texttt{u} \rightarrow \textit{len parents}\ \texttt{u} = 2\ \texttt{and}\\ &\qquad\qquad \texttt{u in}\ \textit{parameters}\ \texttt{a}_2)\ \textit{parameters}\ \texttt{a}_1\end{aligned}$$

The queries *node_parameters* and *edge_parameters* are issued with all activities of interest and pairs of them as parameters, respectively. The returned parameters are then used by the robot to infer activities and affordances in its environment. Once the activities of the past are predicted, the second set of queries, shown below, are issued to the RoboBrain for generating possible futures.

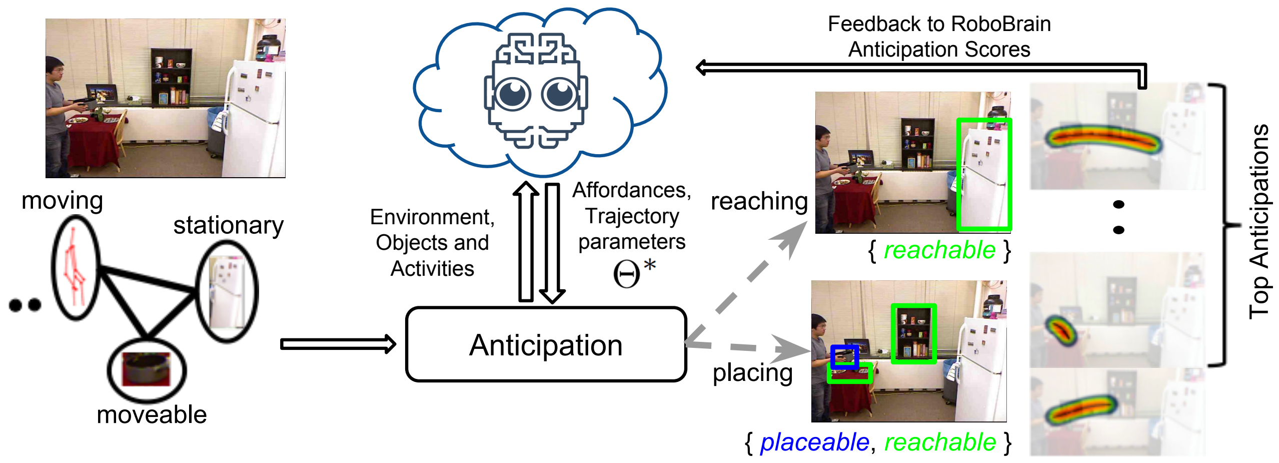


Fig. 6: **RoboBrain for anticipating human activities.** Robot queries RoboBrain, for the activity, affordance and trajectory parameters in order to generate and rank the possible future activities in a given environment. Feedback from the scored anticipations are then provided to the RoboBrain for updating the beliefs over the affordance and activity nodes.

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

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

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

The *affordances* query is issued for each object of interest in the environment in order to obtain the possible future object affordances and hence the actions that can be performed with them. For each retrieved object affordance, the *trajectories* query is issued to fetch the corresponding motion trajectory parameters, using which the anticipation algorithm generates the most likely futures and ranks them (see Figure 6).

Querying the RoboBrain for activity and affordance parameters allows the anticipation algorithm to scale to new activities and affordances. For example, as other works on learning activities and affordances [38, 61] use RoboBrain, these will become available to the anticipation application as well.

Feedback to RoboBrain. In addition to querying RoboBrain for affordance and trajectory parameters for predicting the future, the anticipation algorithm can also provide feedback on the quality of the information returned by the query. For example, when queried for the possible affordances of the object *cup*, if an incorrect affordance is returned, say *writable*. The anticipation algorithm gives the possibility of writing with a cup a very low score. This information is communicated back to RoboBrain which is useful for updating the beliefs in the knowledge graph.

C. Planning in presence of humans

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 valid from a geometric standpoint (i.e., feasible and obstacle-free), but it also needs to satisfy the user preferences [26, 27]. For example, a household robot should move a glass of water in an upright position while maintaining a safe distance from nearby electronic devices. Similarly, robot should move sharp objects such as knife strictly away from nearby humans [25].

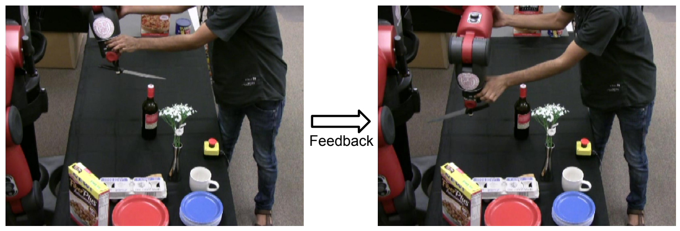
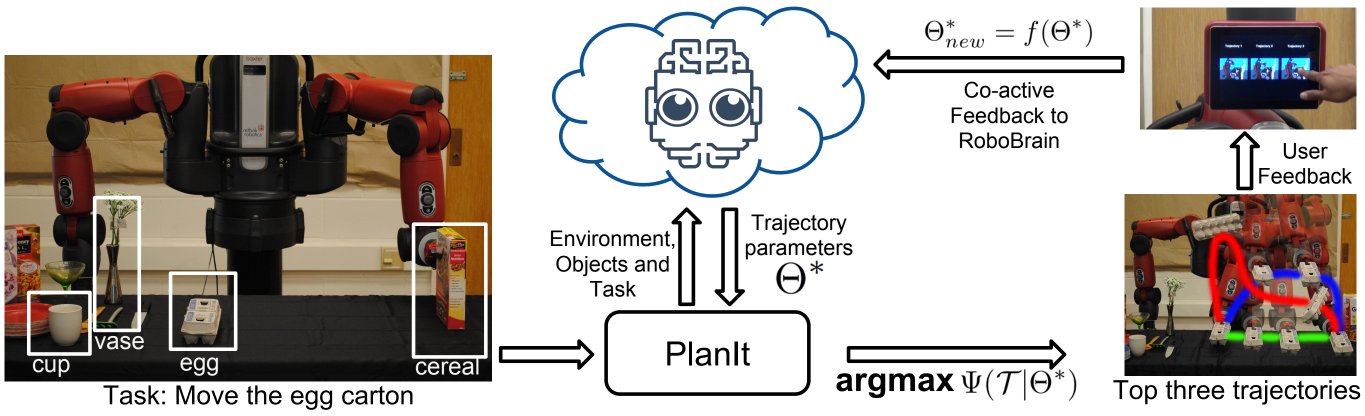


Fig. 8: **Feedback to RoboBrain.** RoboBrain also improves with feedback provided on physical robots. In this figure user provides feedback on robot by improving the trajectory proposed by the RoboBrain. As feedback user moves the knife away from himself.

These preferences are commonly represented as cost functions, parameters of which are learned in a data-driven manner. The cost functions for generating good trajectories jointly model the environment, task and trajectories. This joint modeling is expensive in terms of resource requirement and data collection, and typically research groups have independently learned different cost functions over the years [26, 36, 31] that are not shared across the research groups.

Use of RoboBrain. We now demonstrate use of RoboBrain for planning. Through RoboBrain, robots can share cost function parameters for different planning problems, such as indoor 2D-navigation, manipulation planning, high-level symbolic planning etc., as shown in Figure 7. In addition to this, robots can also query RoboBrain for expressive trajectories features to instantiate cost functions for predicting good trajectories. This kind of sharing will allow research groups to share their learned cost functions and trajectory features.

Figure 7 shows how PlanIt planning system [27] queries RoboBrain for trajectory parameters for the task at hand. The robot's goal is to move an egg carton to the other end of



cup
vase
egg
cereal
Task: Move the egg carton
$\Theta^*_{new} = f(\Theta^*)$
Co-active Feedback to RoboBrain
User Feedback
Environment, Objects and Task
Trajectory parameters Θ^*
PlanIt
argmax $\Psi(\mathcal{T}|\Theta^*)$
Top three trajectories

Fig. 7: **RoboBrain for planning trajectory.** Robot queries RoboBrain, through PlanIt planning system [27], for trajectory parameters for planning fragile objects such as egg cartons. **(Right)** If robot is uncertain about optimal behaviour it elicits feedback from users. Feedback from users are directly registered in RoboBrain and trajectory parameters are updated for future iterations.

table. Since eggs are fragile users would prefer to move them slowly and close to table surface. In order to complete the task, robot first queries RoboBrain for labeling of objects in the environment. For this it uses task specific *semanticLabeling* function described in Section VI-C.

$$\texttt{objects} = \texttt{semanticLabeling}\ \text{“}\texttt{environment.png}\text{”}$$

After object labels are obtained it locates the egg carton and queries for its attributes, which the RoboBrain returns as *fragile*. Finally it queries for the trajectory parameters of *fragile* objects. Below we show the $RaQueL$ queries.

$$\textit{attributes}\ \texttt{n} := \texttt{fetch}\ (\{\texttt{name} : \texttt{n}\}) \rightarrow [\text{‘}\texttt{HasAttribute}\text{’}] \rightarrow (\texttt{v})$$
$$\textit{trajectories}\ \texttt{a} := \texttt{fetch}\,(\{\texttt{handle} : \texttt{a}\}) \rightarrow [\text{‘}\texttt{HasTrajectory}\text{’}] \rightarrow (\texttt{v})$$
$$\textit{trajectory\_parameters} = \texttt{map}(\lambda\texttt{a} \rightarrow \textit{trajectories}\ \texttt{a})\ \textit{attributes}\ \text{‘}\texttt{egg}\text{’}$$

After getting the cost function parameters robot samples trajectories and executes the top-ranked trajectory.

Feedback to RoboBrain. Sometimes the robot may not be certain about the optimal trajectory, when it can ask the users for feedback. This feedback is directly registered in RoboBrain and trajectory parameters are updated for future queries, as shown in Figure 7. A user can provide various kinds of trajectory feedback. For example, in Figure 7 user provides co-active feedback: the robot displays top three ranked trajectories and user selects the best one [26]. In another example, RoboBrain also improves with feedback provided physically on the robot. In Figure 8, the user directly improves a trajectory by correcting one of the trajectory waypoints. Such feedback can thus help improve all the robots that use RoboBrain.

VIII. Discussion and Conclusion

RoboBrain is a collaborative effort to create a knowledge engine that acquires and integrates knowledge about the physical world the robots live in from several sources with multiple modalities. We described some technical challenges we have addressed, such as the multi-modality of knowledge, never-ending learning, large-scale nature of the system, use of crowd-sourcing, and allowing queries with $RaQueL$ for building applications with RoboBrain. We have shown how to use RoboBrain in three different areas: perception, planning and natural language grounding. We showed that sharing knowledge across different robotic tasks allows them to scale their learning to new situations.

RoboBrain is an ongoing effort, where we are constantly improving different aspects of the work. We are improving the system architecture and expanding $RaQueL$ to support scaling to even larger knowledge sources (e.g., millions of videos). Furthermore, we have several ongoing research efforts that include achieving better disambiguation and improving never-ending learning abilities. More importantly, we are constantly expanding the set of our RoboBrain research partners. This will not only improve the abilities of their robots, but also their contribution of knowledge to RoboBrain will help other researchers in the robotics community at large.

Acknowledgments

We thank Arzav Jain for his help in building the graph infrastructure for the RoboBrain. We thank Bart Selman, Jitendra Malik and Ken Goldberg for their visionary ideas. We thank Emin Gun Sirer for help with databases, and thank Silvio Savarese, Stefanie Tellex, Fei-Fei Li, and Thorsten Joachims for useful discussions. We also thank Yun Jiang, Ian Lenz, Jaeyong Sung, Chenxia Wu, and Ayush Dubey for their contributions to the knowledge for the RoboBrain. We also thank Michela Meister, Hope Casey-Allen, and Gabriel Kho for their help and involvement with the RoboBrain project. We are also very thankful to Debarghya Das and Kevin Lee for their help with developing the front-end for the RoboBrain.

This work was supported in part by Army Research Office (ARO) award W911NF-12-1-0267, Office of Naval Research (ONR) award N00014-14-1-0156, NSF National Robotics Initiative (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.

References

[1] R. Alami, A. Clodic, V. Montreuil, E. A. Sisbot, and R. Chatila. Toward human-aware robot task planning. In *AAAI Spring Symposium: To Boldly Go Where No Human-Robot Team Has Gone Before*, 2006.

[2] A. Anand, H. S. Koppula, T. Joachims, and A. Saxena. Contextually guided semantic labeling and search for 3d point clouds. *IJRR*, 2012.

[3] 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.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. *Dbpedia: A nucleus for a web of open data*. Springer, 2007.

[5] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth. Robotic roommates making pancakes. In *Humanoids*, 2011.

[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] W. Burgard, D. Fox, and S. Thrun. Probabilistic state estimation techniques for autonomous and decision support systems. *Informatik Spektrum*, 34(5), 2011.

[9] 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.

[10] 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.

[11] X. Chen, A. Shrivastava, and A. Gupta. NEIL: Extracting Visual Knowledge from Web Data. In *ICCV*, 2013.

[12] 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.

[13] 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.

[14] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *IJRR*, 29(5), 2010.

[15] X.L. Dong, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.

[16] A. Dragan and S. Srinivasa. Generating legible motion. In *RSS*, June 2013.

[17] S. Edelkamp and P. Kissmann. Optimal symbolic planning with action costs and preferences. In *IJCAI*, 2009.

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

[19] 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.

[20] D. A. Ferrucci. Introduction to this is watson. *IBM J. of RnD*, 56(3.4): 1–1, 2012.

[21] M. Gemici and A. Saxena. Learning haptic representation for manipulating deformable food objects. In *IROS*, 2014.

[22] M. J. Gielniak, C. Karen Liu, and A. L. Thomaz. Generating human-like motion for robots. *IJRR*, 32(11), 2013.

[23] 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.

[24] 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.

[25] A. Jain, S. Sharma, and A. Saxena. Beyond geometric path planning: Learning context-driven trajectory preferences via sub-optimal feedback. In *ISRR*, 2013.

[26] A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *NIPS*, 2013.

[27] 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.

[28] Z. Jia, A. Gallagher, A. Saxena, and T. Chen. 3d reasoning from blocks to stability. *IEEE PAMI*, 2014.

[29] Y. Jiang, H. Koppula, and A. Saxena. Hallucinated humans as the hidden context for labeling 3d scenes. In *CVPR*, 2013.

[30] 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.

[31] K. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert. Activity forecasting. In *Proc. ECCV*, 2012.

[32] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *ICRA*, 2013.

[33] H. Koppula, A. Jain, and A. Saxena. Anticipatory planning for human-robot teams. *ISER*, 2014.

[34] H.S. Koppula and A. Saxena. Physically grounded spatio-temporal object affordances. In *Proc. ECCV*, 2013.

[35] H.S. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013.

[36] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *RSS*, 2012.

[37] K. Lai, L. Bo, X. Ren, and D. Fox. A Large-Scale Hierarchical Multi-View RGB-D Object Dataset. In *ICRA*, 2011.

[38] T. Lan, T.C. Chen, and S. Savarese. A hierarchical representation for future action prediction. In *Proc. ECCV*, 2014.

[39] P. A. Lasota, G. F. Rossano, and J. A. Shah. Toward safe close-proximity human-robot interaction with standard industrial robots. In *CASE*, 2014.

[40] D. B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[41] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. In *RSS*, 2013.

[42] J. Mainprice and D. Berenson. Human-robot collaborative manipulation planning using early prediction of human motion. In *IROS*, 2013.

[43] C. McManus, B. Upcroft, and P. Newmann. Scene signatures: Localised and point-less features for localisation. In *RSS*, 2014.

[44] 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.

[45] 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.

[46] T. Naseer, L. Spinello, W. Burgard, and C. Stachniss. Robust visual robot localization across seasons using network flows. In *AAAI*, 2014.

[47] neo4j. Cypher. ''http://neo4j.com/docs/stable/cypher-query-lang.html".

[48] S. Nikolaidis, P. A. Lasota, G. F. Rossano, C. Martinez, T. A. Fuhlbrigge, and J. A. Shah. Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action. In *ISR*, 2013.

[49] 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.

[50] N. D. Ratliff, J. A. Bagnell, and S. S. Srinivasa. Imitation learning for locomotion and manipulation. In *Int. Conf. on Humanoid Robots*, 2007.

[51] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[52] A. Saxena and A.Y. Ng. Learning sound location from a single microphone. In *ICRA*, 2009.

[53] 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.

[54] 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.

[55] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[56] 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.

[57] S. Tellex, R. Knepper, A. Li, D. Rus, and N. Roy. Asking for help using inverse semantics. In *RSS*, 2014.

[58] M. Waibel, M. Beetz, R. D'Andrea, R. Janssen, M. Tenorth, J. Civera, J. Elfring, D. Gálvez-López, K. Häussermann, J. Montiel, A. Perzylo, B. Schießle, A. Zweigle, and R. van de Molengraft. Roboearth: A world wide web for robots. *IEEE Robotics & Automation Magazine*, 2011.

[59] J. Walker, A. Gupta, and M. Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, 2014.

[60] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *RSS*, 2014.

[61] Y. Zhu, A. Fathi, and Li Fei-Fei. Reasoning about object affordances in a knowledge base representation. In *Proc. ECCV*, 2014.

[62] 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.