Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Open-World Reasoning for Service Robots

Yuqian Jiang,[*1] **Nick Walker,**[*2] **Justin Hart,**[1] **Peter Stone**[1]
[1]Department of Computer Science, University of Texas at Austin, Austin, TX, USA
[2]Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA, USA
jiangyuqian@utexas.edu, nswalker@cs.washington.edu, {hart, pstone}@cs.utexas.edu

Abstract

A service robot accepting verbal commands from a human operator is likely to encounter requests that reference objects not currently represented in its knowledge base. In domestic or office settings, the construction of a complete knowledge base would be cumbersome and unlikely to succeed in most real-world deployments. The world that such a robot operates in is thus "open" in the sense that some objects that it must act on in the real world are not described in its internal representation. However, when an operator gives a command referencing an object that the robot has not yet observed (and thus not incorporated into its knowledge base), we can think of the object as being hypothetical to the robot. This paper presents a novel method for closing the robot's world model for planning purposes by introducing hypothetical objects into the robot's knowledge base, reasoning about these hypothetical objects, and acting on these hypotheses in the real world. We use our implementation of this method on a domestic service robot as an illustrative demonstration to explore how it works in practice.

Introduction

Service robots assist human users in day-to-day tasks. A user may give a command such as "Get me the apple from the kitchen" to a service robot. Reasoning about the steps of this task is a classical type of planning problem. The classical approach requires adopting a "closed-world assumption," presupposing that everything the robot could possibly need to reason about is already represented in its knowledge base. In simulations and laboratory experiments, it is possible to enumerate all of the objects that a robot could interact with in its environment, but in most real-world service robot scenarios – like those that arise when interacting with people in offices or homes – such a complete enumeration is impossible. The real world for a service robot is "open" in the sense that, at any time, the next command it receives may refer to an object that is not currently in its knowledge base.

Planning approaches that work in open-world scenarios often find a way to close the world (Talamadupula et al. 2010b; Hanheide et al. 2017). This paper presents an approach to closing the world wherein the robot leverages the user's command to cope with the openness of the real world that it inhabits. The novel contribution of this paper is a process by which a system: 1) closes its world by grounding the object referred to by a human operator as "hypothetical;" 2) leverages domain knowledge represented as "conceptual" knowledge in its knowledge base in order to create predicates about the hypothetical object which can used for planning; 3) uses novel "realizer" actions which a) explicitly represent the robot's search for the object, b) allow feasible plans involving "hypothetical" objects, c) find concrete objects that fit the description of the "hypothetical" objects using the robot's perceptual system; and 4) provides diagnostic reasoning to falsify hypotheses that have been determined to be incorrect based on observations made during plan execution. This falsification of the robot's hypotheses is used both to guide the robot's search for hypothetical objects and, when necessary, to report back to the user that their command cannot be completed. At a higher level, we can view the novel contribution of this paper as a method of formulating and reasoning about these hypotheses, and the design of a world model that allows their formulation.

We have implemented this approach on a domestic service robot, and will use this implementation to discuss the method described in this paper with an illustrative demonstration. In this demonstration, a human operator gives a verbal command to the robot, requesting that the robot perform an object retrieval task. During execution, the robot may or may not perceive the object referenced in the command, validating or invalidating the hypotheses generated when the user gave the command to the robot. The robot finally retrieves the desired object, or reports its failure to do so back to the operator. This method could easily be implemented in other scenarios where such hypothetical reasoning about a human operator's commands could be an enabling technique.

Related Work

Service robots are envisioned as devices that interact with people and assist them in day-to-day tasks. There have been research platforms, such as CoBot (Veloso et al. 2015), the BWIBot (Khandelwal et al. 2017), and the Care-O-bot 3 (Reiser et al. 2009), designed wholly or in part by university researchers for experiments in service-robot scenarios, as well as a number of commercial platforms designed

[*]Equal contribution.

for hotels, hospitals, and homes. These systems are human-interactive, autonomous, and designed to perform mundane tasks and allow people to focus their attention elsewhere.

Several projects have engineered knowledge representation (KR) systems into service robots to support execution of human-specified tasks. Lemaignan et al. (2017) combine Open Robot Ontology server (ORO) (Lemaignan et al. 2012), a relational knowledge base, with an architecture for executing tasks. The system models knowledge perspectives for each agent in its environment, facilitating interactive grounding for human-robot collaboration tasks. The use of general-purpose planners to perform task-level control of intelligent service robots is also well-established. The KeJia robot leverages a symbolic planner and provides high-level functions to understand and execute natural language commands (Chen et al. 2010; 2013). Chen et al. (2016) extend the system to plan to acquire task-oriented knowledge by modeling possible states in an action language. The integration of planning and execution for service robot tasks has also been studied. Sanelli et al. (2017) integrate conditional planning and Petri Net Plans to plan and execute human-robot interactions. The system presented in this paper is similar to these systems, in that it combines a KR system with a planner for task-level planning and execution.

Puigbo et al. (2015) deployed the Soar cognitive architecture (Laird 2012) to support understanding and executing human-specified commands in a home setting, but did not address the challenge of as-yet ungroundable object references. Mininger and Laird (2016) used a Soar-based interactive task-learning system to learn strategies to handle references to unseen objects. Their approach defines a "find" subtask with a special postcondition so the system can succeed in planning for tasks requiring direct interaction with unseen objects. Our system represents the concept of hypothetical objects in its knowledge base and models perceptual actions in a planning domain.

Most planning algorithms rely on the closed-world assumption, which assumes that a logical statement is false if it is not known to be true (Etzioni, Golden, and Weld 1997; Talamadupula et al. 2010b). Open-world planning has been investigated in the context of visual object search (Aydemir et al. 2013; Hanheide et al. 2017) by leveraging assumptive actions with probabilistic effects to make hypotheses about object existence. Hanheide et al. (2017) extend this method to explain failures by planning over explicitly modeled additional action effects and assumptive actions. To plan in the open world for human-robot teams in search and rescue tasks, Talamadupula et al. (2010a; 2010b) developed an approach that uses open-world quantified goals and partial satisfaction planning. This paper provides an open-world task planning approach for service robots by forming hypotheses implied by commands of operators.

Some systems have focused on KR for specific task classes or functions. Work on KnowRob integrates semantics with robot control and a cloud datastore to enable a robot to execute complicated manipulation tasks (Tenorth and Beetz 2013; Beetz et al. 2018). Chernova et al. (2017) use an abstract knowledge base modeled as a Bayesian Logic Network to enable a robot to make better inferences about aspects of its environment. In contrast, probabilistic models or cloud resources are complimentary to our representation, but not necessary.

RoboCup@Home is a competition of domestic service robots performing benchmark tasks in simulated home environments (Wisspeintner et al. 2009), and the work presented in this paper was initially motivated by attempts to develop a general solution to a RoboCup@Home task called Enhanced-Endurance General Purpose Service Robot (EEGPSR). We distributed a survey to RoboCup@Home participants, asking other teams how their systems would handle the command, "Could you navigate to the kitchen, spot the apple, and give it to Patricia at the couch." Three teams responded, saying that their systems did not rely on a symbolic task planner, favoring pre-specified representations of what actions the robot should take in this and other conditions. Two respondents described current or past research efforts to integrate general purpose planning technology into their systems, but ultimately did not deploy their techniques in the competition. Though anecdotal, we believe that most RoboCup participants took similar approaches. We believe that our system is the only one to use a general purpose planner to solve problems involving ungrounded objects that has been entered into this competition.

The most related systems to the best of our knowledge are compared in Table 1. The table assesses the features of each system based on a representative corresponding publication and further notes characteristics present in any robot demonstrations. Systems that represent and update information in a graph of entities and their relations during execution are denoted as having a dynamic semantic network. Systems that are capable of generating plans that reference objects unknown to the robot are considered to have open-world planning capabilities. Systems that are able to identify and report inconsistencies in their knowledge to a human operator as well as systems which produce plans deterministically are also noted. In Table 1, systems that have been demonstrated on a robot platform performing tasks that require numerous capabilities - including at least object manipulation, navigation, and human interaction - are considered to support general purpose capabilities. Systems with evaluations showing that the robot does not remain idle for more than thirty seconds at a time are considered to be responsive.

Algorithm

The essence of this approach is to generate hypotheses of unseen objects that a general, closed-world planner can reason over. The planner uses *realizer actions* that look for evidence to support the hypotheses. At execution time, perception modules report concrete objects. When the existence and relations of the hypothetical object are realized, replanning uses the concrete object. If no object is found, the robot's diagnostic rules engage to attempt to reason about which hypotheses were invalid.

Task Representation

A robot task planning problem is defined by the tuple (S_0, G, A), where S_0 is the initial state, G is the goal condition,

	This work	Chen, Yang, and Chen	Hanheide et al.	Lemaignan et al.	Mininger and Laird	Puigbo et al.
Functionality						
Dynamic semantic network	✓	-	✓	✓	✓	✓
Open-world planning	✓	✓	✓	-	✓	✓
Reports inconsistent information	✓	-	✓	✓	-	-
Deterministic planning	✓	✓	-	✓	✓	✓
Demonstration						
General purpose capabilities	✓	✓	-	-	-	✓
Responsiveness	✓	✓	-	✓	-	-

Table 1: Comparison of Approaches to Task Planning and Execution on a Mobile Robot

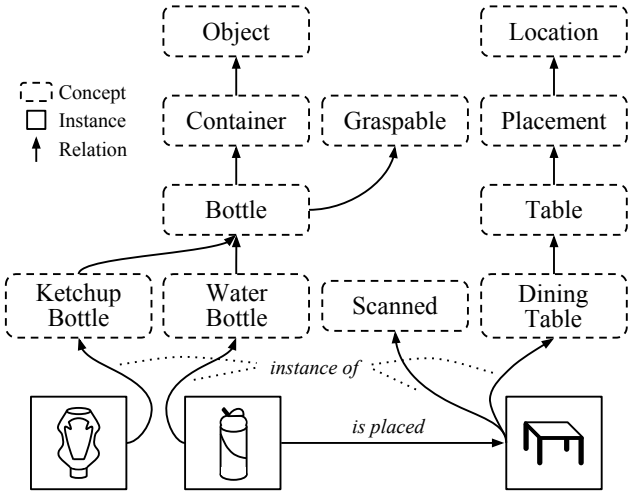

Figure 1: An illustration of how a water bottle sitting on a dining table is represented in our knowledge base after perception. Arrows represent relations between two entities, and their labels give the name of the relation. Unlabeled arrows refer to "is a" relations. The labels on each concept depict the value of their "name" attribute.

and A is a set of rules defining the preconditions and effects of available actions. A plan Π is a sequence of actions that satisfy G. Each state s is represented as (C, I, P), where C is a set of concepts, I is a set of object instances, and P is a set of predicates describing the attributes and relations of concepts and instances. The state is derived from the knowledge K in the robot's knowledge base, which is a semantic network made up of entities, attributes and relations. Entities can be either *concepts* – intangible, general ideas – or *instances* – things that are at least partially tangible. *Relations* describe the connections between entities, defining, for example, which concepts an instance inherits from. *Attributes* provide a means of attaching data to entities, and can be used to store information such as the name of a concept or instance. For example, Figure 1 visualizes part of a state describing a water bottle and a ketchup bottle sitting on a dining table. This representation provides a unified store of situated and general knowledge. Using it, a robot may leverage specific knowledge about its environment as well as its ontology, enabling inference across types of knowledge. For example, by representing a water bottle instance perceived on a dining table, the system can draw inferences such as "there is a container on a table." This is required for a general purpose reasoner to handle commands such as "clean all containers on the table."

Closing the World with Hypotheses

When all the objects, concepts, and predicates are assumed to be known a priori, the planning problem is considered to be a closed-world problem, and this is the regime under which most off-the-shelf planners are designed to operate. We present Algorithm 1 as an approach to closing the robot's planning scenario by encoding hypotheses derived from the operator's instruction. As input to Algorithm 1, a set of unknown objects referenced by the operator, O^H,[1] is extracted from the instruction, as is a set of hypotheses, P^H,[2] and the goal condition, G. We assume that the necessary concepts to describe the unknown objects are modeled in K. For instance, the concept of a "water bottle" is modeled, but the concrete instance of the water bottle is not represented in K.

The algorithm first incorporates concepts, instances, attributes, and relations from the current knowledge base in the initial state S_0 (Line 2-4). Then, a new instance is created for each unknown object referenced by the operator (Line 6). The instance is represented by adding an inheritance relation with its corresponding concept,[3] and an inheritance relation with the concept of "hypothetical" (Line 7-9). For each hypothesis that references the unknown object, the reference is replaced by the new hypothetical instance (Line 10). For

[1] For example, *there is a "water bottle"*

[2] For example, the "water bottle" *is on the kitchen table*

[3] For example, the concept of a "water bottle"

each goal condition that references the unknown object, the reference is replaced by its corresponding concept (Line 11). Referring to the object by its concept allows the planner to use a known object of the same class if it can satisfy the goal conditions. The modified hypotheses are then incorporated into the predicates in the initial state (Line 13). At that point, a closed-world planner can be used to generate an executable plan. This method adds $\left\|O^H\right\|$ objects and $\left\|P^H\right\|$ predicates representing hypotheses to the domain. The complexity of the final problem depends on the specific problem and encoding, and off-the-shelf solvers can be leveraged.

Algorithm 1 Initial State and Goal Construction

Input: Current knowledge K, unknown objects O^H, hypotheses P^H, goal conditions G
Output: Initial state S_0, goal conditions G'

1: $S_0 = (C, I, P)$, $G' = G$
2: $C = \{e : e \in \text{K}, e \text{ is a concept}\}$
3: $I = \{e : e \in \text{K}, e \text{ is an instance}\}$
4: $P = \{p : p \in \text{K}, p \text{ is an attribute OR relation}\}$
5: **for each** object name $o \in O^H$ **do**
6: add new instance i to I
7: $c_o =$ concept with name o
8: $c_h =$ concept with name hypothetical
9: add inheritance relation of i with c_o and c_h to P
10: replace references to o in P^H with i
11: replace references to o in G' with c_o
12: **end for**
13: $P = P \cup P^H$
14: **return** S_0, G'

Realizer Actions

In order to make physical observations of hypothetical objects and their attributes and relations, we introduce *realizer actions*. These actions have preconditions that rely on the hypotheses produced by Algorithm 1, and produce predicates that concretely describe the attributes or relations of the object. For example, if the operator thinks there can be a water bottle on the kitchen counter, a realizer action `perceive_surface` has the effect that the hypothetical water bottle is on the kitchen counter. Note that the realizer actions correspond to physical perceptual behaviors that the robot performs. Thus, the robot may not actually find a concrete instance that matches the hypothetical object, and at the same time it may find other types of objects. If execution fails because the hypotheses are wrong, a set of diagnostic rules D derive negations of hypotheses from the robot's current knowledge base to facilitate error reporting. The diagnostic rules are further illustrated in the next section.

Executing Plans with Hypotheses

Given the initial state and goal, Algorithm 2 shows the replanning execution strategy for accomplishing the goal while detecting if the hypotheses are invalid. Until the goal is satisfied, a plan is generated. Each action in the plan is dequeued and executed, resulting in updates to the state, S. After each action, the reasoner is queried to verify that the remainder of the current plan Π applied to the current state would still accomplish the goal. This *monitor query* will report failure when, for example, the action did not actually result in the effects necessary to continue executing the plan.

Critically, this case will occur after executing any realizer action. Because realizers are only applied to hypothetical instances (as enforced by line 10 of Algorithm 2) – and because an executor will never actually produce observations of the hypothetical instance – the effects of the realizer are not achieved. The observations made by the realizer are of concrete instances, which are incorporated into the robot's knowledge base. For example, executing `perceive_surface` will not result in a new state where the hypothesized target object is on the surface, but it may result in a new state where a real object that fits the description of the hypothetical object is on the surface. Thus, upon re-planning in this scenario, the robot now has concrete knowledge of a real-world instance of the object, and a plan is constructed using this knowledge rather than a formulated hypothesis in this subsequent planning attempt.[4]

Algorithm 2 Plan Execution

Input: Initial state S_0, Goal conditions G, action rules A, diagnostic rules D

1: $S = S_0$
2: **while** G not achieved **do**
3: $\Pi =$ plan(S, G, A)
4: **if** planning failed **then**
5: diagnose(S, S_0, D)
6: **return**
7: **end if**
8: **while** Π not empty **do**
9: $a =$ dequeue Π
10: $S =$ execute(a)
11: **if** not monitor_query(S, Π, G, A) **then**
12: **break**
13: **end if**
14: **end while**
15: **end while**

System Design

This section introduces the components of our fully implemented knowledge representation and reasoning system. Given a task description, these components carry out the task and enable the robot to report inconsistent information that it identifies during execution.

The system is underpinned by an expressive knowledge base, which combines an ontology capable of describing conceptual information as well as of representing execution-time knowledge and the current robot state.

[4]Note that there is also a body of work on plan repair that could further optimize the system. However, while it may seem that the plan could be repaired by simply swapping the ID of the hypothetical object for the observed object, it may not be so straightforward. For example, if a bottle is open when perceived, plan repair must consider inserting an action to close the bottle to satisfy the preconditions of picking up.

The *composer* takes simplified, parametric descriptions of tasks, such as `bring_me(object="water bottle", location="dining table")`, and leverages the knowledge base to construct instances of the "hypothetical" concept for referenced objects and produce a goal for use with a symbolic task planner. An executor carries out the goal and replans as the robot gathers additional knowledge about the environment. If no feasible plan can be found, a diagnostic reasoning process is used to identify whether the failure can be attributed to inconsistent hypotheses, and if so, which ones.

Knowledge Base

The knowledge base is implemented in a SQL database that stores the semantic network in tables of entities, attributes, and relations. To enable importing hand-specified knowledge from existing ontologies, it supports a subset of OWL2 EL corresponding to the features present in our semantic network. Custom parsers are also provided in order to load annotated map information. All of this knowledge is available to programs via C++ and Python interfaces for SQL access. An additional interface enables retrieving stored information in the form of facts usable by the planner.

Goal and Hypothesis Composer

Given a description of a task, the system needs to produce a planning goal in the representation of the planner. The system component that implements Algorithm 1 is the goal and hypothesis composer. Because the robot is operating in an open world, where it may not have full knowledge of all the objects in the environment, the composer creates hypothetical instances to represent the unknown objects, and add all hypothetical relationships that can be inferred from the command. For instance, the command parser may return the task description `bring_me(object="water bottle", location="dining table")`, for which the composer would create an instance entity that inherits from the "water bottle" and "hypothetical" concepts, and the predicate `can_be_placed(<new instance ID>, <dining table ID>)`. Finally, the composer generates a goal describing the task. For this scenario, the goal is `is_delivered(<concept ID>, <operator ID>)`. The goal and hypotheses are task dependent, and a template is pre-specified for each task. Currently the system assumes all locations are known, but the algorithm has no such restriction and the system can be extended to handle unknown locations.

Planning

The planning module takes the current state of the knowledge base and finds a sequence of symbolic actions that satisfy the goal. The action knowledge is encoded in the Answer Set Programming (ASP) language (Lifschitz 2002). The incremental solving mode of the ASP solver CLINGO is used as the reasoner and task planner (Gebser et al. 2011).

Table 2 lists the preconditions and effects of a subset of actions in the system, demonstrating navigation, manipulation, and human-robot interaction. Each precondition or effect is described as a rule in ASP semantics.[5] For example, rule (1) describes that the precondition of navigating to L_1 from L_2 at timestep n is to be near L_2 at timestep $n-1$.

$$\bot \leftarrow navigate\_to(L_1, n), is\_near(self, L_2, n-1). \quad (1)$$

For compactness, preconditions that restrict variables to be instances of certain concepts as well as the timestep variable "n" in fluents are omitted here.

Besides the action rules, the planning domain also models static and dynamic reasoning rules. Static rules make inferences on facts in the knowledge base and planning time hypotheses. For example, the following rule allows the reasoner to derive the new hypothesis that an object can be at any placement in the room if it is assumed to be in the room.

$$\begin{aligned} can\_be\_placed(O, L) \leftarrow\ & can\_be\_placed(O, R), \\ & is\_in(L, R), has\_concept(R, \text{“room”}), \\ & has\_concept(L, \text{“placement”}). \end{aligned} \quad (2)$$

Dynamic rules derive facts at a later time step from earlier time steps. For example, the following rule defines the inertia that the robot remains near the same locations unless it is explicitly moved.

$$\begin{aligned} is\_near(self, L, n) \leftarrow\ & is\_near(self, L, n-1), \\ & \text{not } \neg is\_near(self, L, n). \end{aligned} \quad (3)$$

Plan Execution and Monitoring

The system integrates a replanning executor that implements Algorithm 2. Given a goal, the executor creates an initial plan. After executing a step, the executor integrates any observations collected into the knowledge base and formulates an ASP query to verify that the remaining plan can still satisfy the goal. This infusion of knowledge permits the system to gracefully incorporate information that was not available when the executor initially formulated a plan and is key to enabling the system to plan and reason with hypothetical instances. The initial plan may include actions on hypothetical instances, but at execution time, when the robot observes (or observes the absence of) real instances, plan verification fails because the preconditions for acting on the hypothetical instance are not actually satisfied.

For example, the robot has planned to `perceive_surface(<hypothetical instance ID>, <dining table ID>)` and then `pick_up(<hypothetical instance ID>, <dining table ID>)`, but after executing perceive surface, plan verification will fail because `is_placed(<hypothetical instance ID>, <dining table ID>)` is not true. Replanning commences, and if the robot did in fact observe a real instance of the concept specified in the goal, a new plan would be generated using its instance ID. Otherwise, the robot fails to generate a plan, reasons about which hypotheses were invalid, and returns to the operator.

[5] ASP encodings in this paper follow the style of the planning examples in the CLINGO guide: https://github.com/potassco/guide

Action	Preconditions	Effects
$navigate\_to(L_1, L_2)$	$is\_connected(L_1, L_2)$ $is\_near(self, L_2)$	$is\_near(self, L_1)$ $is\_facing(self, L_1)$ $\neg is\_near(self, L_2)$
$pick\_up(O, L)$	$is\_placed(O, L)$ $is\_facing(self, L)$ $hand\_empty(self)$	$is\_holding(self, O)$ $\neg hand\_empty(self)$ $\neg is\_placed(O, L)$
$put\_down(O, P)$	$is\_holding(self, O)$ $is\_facing(self, P)$	$is\_placed(O, P)$ $\neg is\_holding(self, O)$ $hand\_empty(self)$
$hand\_over(O, P)$	$is\_holding(self, O)$ $is\_facing(self, P)$	$is\_delivered(O, P)$ $\neg is\_holding(self, O)$ $hand\_empty(self)$
$perceive\_surface(O, L)$	$is\_facing(L)$ not $scanned(L)$ $can\_be\_placed(O, L)$	$scanned(L)$ $is\_placed(O, L)$
$find\_person(P, L)$	$is\_near(self, L)$ $can\_be\_located(P, L)$	$is\_facing(self, P)$ $is\_located(P, L)$

Table 2: Action knowledge in our system, organized into preconditions and effects.

Diagnostics

In the case that the robot fails to generate a plan, the system engages a diagnostic reasoning process to assess whether the failure can be attributed to false hypotheses. This is accomplished by formulating an ASP query that attempts to deduce the negation of the hypothesis based on the robot's current knowledge. For example, if the robot had hypothesized that a water bottle existed and was placed on the dining table, the diagnostic program uses rule (4), specifying that a hypothetical instance cannot be placed on a surface if the surface has been scanned and no instances that descend from the hypothetical's other concepts were detected.

$$\neg can\_be\_placed(O_1, L) \leftarrow scanned(L),$$
$$has\_concept(O_1, \text{“hypothetical”}),$$
$$has\_concept(L, \text{“placement”}), \#count\{is\_placed(O_2, L) :$$
$$instance\_of(O_1, C), instance\_of(O_2, C)\} = 0. \quad (4)$$

Because the operator generally has perception and knowledge that are more reliable than the robot's, hypotheses implied by human commands are given the benefit of the doubt, and the system formulates the diagnostic as an optimization query which seeks to maximize the number of valid hypotheses. If the diagnostic derives a set of inconsistent hypotheses, the robot can then use this information to provide an explanation for why it could not achieve the task.

Evaluation

The system is evaluated as implemented on a Toyota Human Support Robot (HSR). The system also underpinned our entry in the General Purpose Service Robot (GPSR) and Enhanced Endurance General Purpose Service Robot (EEGPSR) tests of the RoboCup@Home competition in 2018. During these tests, robots are given verbal commands and must carry out tasks combining numerous skills, including navigation, human-robot interaction, object recognition and manipulation. Since the challenges this paper aims to address—such as partial and erroneous task information—are present in GPSR and EEGPSR tests, the system is evaluated in conditions that resemble these tests.

The system is portable and has been identically implemented on the BWIBot platform (Khandelwal et al. 2017) in an office environment. The physical capabilities of the platform are different, however the capabilities of the knowledge system and plan execution are the same.

Demonstration

This demonstration shows how the system supports the execution of tasks in the home environment by examining it running a command under varying conditions. Here, we expand on the implementation of the system, then show that the robot can receive the command, plan its actions, and smoothly execute the task.[6]

Platform Our evaluation uses a Toyota Human Support Robot (HSR), a domestic service robot with an omnidirectional base, an arm, and an RGB-D sensor. To provide fast object recognition, an external laptop with an Nvidia GTX 1080 GPU is mounted to the back of the robot. The laptop runs an instance of YOLO (Redmon and Farhadi 2017), and the detections are used to populate a 3D map with labeled objects and their centroids. The robot's various software modules are implemented and integrated with ROS (Quigley et al. 2009).

[6]A video of the demonstration is available at https://youtu.be/TLXGQDTAZvA.

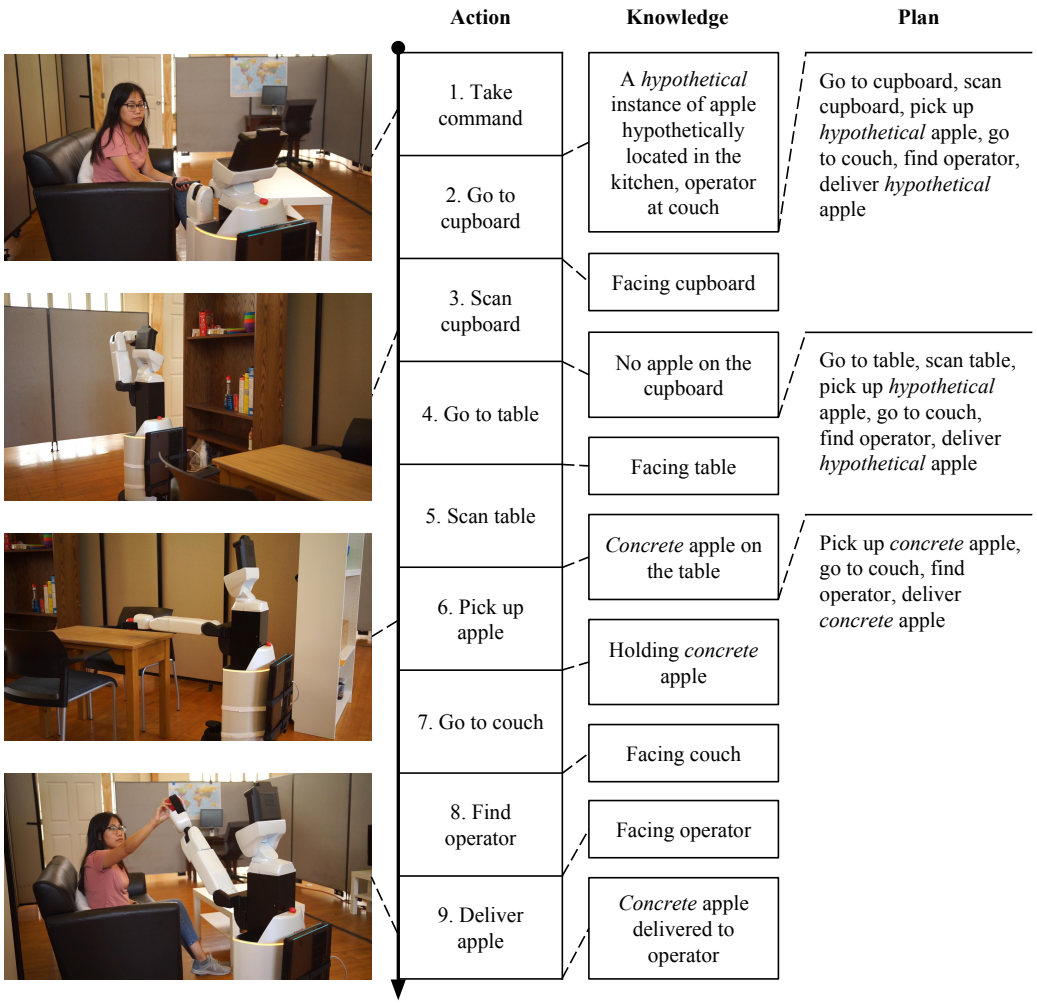

Figure 2: A timeline showing the execution for the command "Bring me the apple from the kitchen." The "action" column describes the primary actions undertaken by the top-level state machine (action 1) and the plan exectuor (actions 2-9). The information that is entered into the knowledge base stemming from observations after action execution are depicted in the "knowledge" column. The executor's current plan, which may change as a result of knowledge updates, is shown in the "plan" column.

System Implementation Top-level control of the system is implemented as hierarchical state machines in SMACH.[7] At the start of a task, a dialog module is triggered to take a verbal command from an operator. For these demonstrations, a parser which outputs parameterized tasks based on the RoboCup@Home GPSR command grammar is used. The command-taking state machine uses a pre-specified template to generate a confirmation sentence from the task description. After the operator confirms the task, the robot also makes sure there are no ambiguities in the task parameters by checking if the concepts have more specific child concepts in the knowledge base. For example, if the command mentions "bottle", and there are child concepts "ketchup bottle" and "water bottle" as shown in Figure 1, the robot would ask the operator to clarify the bottle type.

[7]http://wiki.ros.org/smach

Based on the task type, the state machine that implements the composer formulates the hypotheses and goal in an ASP query. The hypotheses and goal specifications are sent to the plan executor using the ROS action interface. The plan executor calls the ASP solver, sequences the actions, and monitors the execution. If no feasible plan can be found during execution, the plan executor triggers the diagnostic reasoning process, and reports the inconsistent hypotheses (if any) to the task-control state machine. If plan execution cannot make progress because the same action fails to achieve the planned effects multiple times, the executor also reports the failure through the ROS action interface.

If the goal is not achieved, the task-control state machine either tries to recover from the error or formulates a report to the operator, depending on the type of error. For example, if the "pick up" action fails, the recovery state attempts to ask a nearby person to place the object directly into the

robot's gripper. If the recovery is successful, the task control reissues the task goal. If the error is irrecoverable, the robot delivers a verbal report to the operator stating the failed action or inconsistent hypotheses. [8]

Object Retrieval Task This section details the behavior of the system as it executes a task in two scenarios. Both scenarios are shown in the supplemental video, and Figure 2 shows robot actions, important knowledge updates and plan changes of the first scenario along with frames in the video.

The task is initiated by a person asking the robot to "Bring me the fruit from the kitchen." The dialog module resolves the ambiguous reference to "fruit" by asking a clarifying question, then transforms the parsed command into the parametric task description `bring_me(object="apple", location="kitchen")`.

The composer encodes the implied existence of an apple somewhere in the kitchen by creating a hypothetical apple instance and a predicate `can_be_placed(<hypothetical apple ID>, "kitchen")`. The composer then produces the goal `is_delivered("apple", <operator ID>)`.

The replanning executor receives the goal and creates an initial plan to deliver the hypothetical apple from the cupboard in the kitchen. This planning leverages the static rule shown in (2) to infer that the hypothetical instance can be placed on any surface in the kitchen.

Once the robot finishes navigating to and perceiving the cupboard, the replanning executor discovers that the remainder of the plan can no longer be executed because the precondition that the hypothetical apple be placed on the cupboard is not satisfied. The executor replans to deliver the hypothetical from the table. After perceiving the table, the executor once more finds that the hypothetical instance is not placed on the table. This time however, the planner is able to make use of a concrete instance of an apple that was perceived on the table. The robot plans to deliver the concrete instance. This plan is executed without any incident, so no replanning is required.

If the apple had not been on the table, the robot would have replanned to deliver the hypothetical apple from the last remaining unscanned surface in the kitchen, the bar. When the robot does not observe an apple on the bar, replanning is unable to generate any plans for the goal. Diagnostic reasoning, leveraging the rule shown in (4), deduces that, because the robot scanned every surface in the kitchen and did not observe an instance of apple, the hypothetical apple could not be placed in the kitchen. The robot uses the dialog module to report the inconsistent hypothesis to the operator.

Summary The robot demonstrations show that the system achieves our goal of servicing commands which involve unseen objects and hypotheses about such objects by representing hypotheses, replanning with knowledge of concrete objects, and identifying hypotheses that are inconsistent with knowledge.

[8] Our implementation of knowledge representation and plan execution can be found at http://doi.org/10.5281/zenodo.2629308

Conclusion

We have presented a novel approach that reasons and plans in the open-world for service robots by leveraging an ontology to represent an operator's hypotheses. Our demonstration highlights our fully implemented system's ability to handle a realistic request that requires the robot to plan with ungrounded objects, and to identify inconsistent hypotheses in the case of planning failure, all while maintaining responsiveness. In the future, we intend to extend the planning domain to model additional aspects of human-robot interaction and incorporate probabilistic models to plan under uncertainties of hypotheses.

Acknowledgements

We are grateful to RoboCup@Home teams Homer, SPQReL and anonymous team X for their responses to our survey. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (IIS-1637736, IIS-1651089, IIS-1724157), ONR (N00014-18-2243), FLI (RFP2-000), ARL, DARPA, Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Aydemir, A.; Pronobis, A.; Göbelbecker, M.; and Jensfelt, P. 2013. Active visual object search in unknown environments using uncertain semantics. *IEEE Transactions on Robotics* 29(4):986–1002.

Beetz, M.; Beßler, D.; Haidu, A.; Pomarlan, M.; Bozcuoglu, A. K.; and Bartels, G. 2018. KnowRob 2.0 – A 2nd Generation Knowledge Processing Framework for Cognition-enabled Robotic Agents. In *International Conference on Robotics and Automation (ICRA)*.

Chen, X.; Ji, J.; Jiang, J.; Jin, G.; Wang, F.; and Xie, J. 2010. Developing High-level Cognitive Functions for Service Robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, 989–996. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Chen, X.; Xie, J.; Ji, J.; and Sui, Z. 2013. Toward Open Knowledge Enabling for Human-robot Interaction. *Journal of Human-Robot Interaction (JHRI)* 1(2):100–117.

Chen, K.; Yang, F.; and Chen, X. 2016. Planning with Task-oriented Knowledge Acquisition for a Service Robot. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 812–818. AAAI Press.

Chernova, S.; Chu, V.; Daruna, A.; Garrison, H.; Hahn, M.; Khante, P.; Liu, W.; and Thomaz, A. 2017. Situated Bayesian Reasoning Framework for Robots Operating in Diverse Everyday Environments. In *International Symposium on Robotics Research (ISRR)*.

Etzioni, O.; Golden, K.; and Weld, D. S. 1997. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence* 89(1):113 – 148.

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The Potsdam answer set solving collection. *AI Communications* 24(2):107–124.

Hanheide, M.; Göbelbecker, M.; Horn, G. S.; Pronobis, A.; Sjöö, K.; Aydemir, A.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janicek, M.; Zender, H.; Kruijff, G. J.; Hawes, N.; and Wyatt, J. L. 2017. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*.

Khandelwal, P.; Zhang, S.; Sinapov, J.; Leonetti, M.; Thomason, J.; Yang, F.; Gori, I.; Svetlik, M.; Khante, P.; Lifschitz, V.; and Others. 2017. BWIBots: A platform for bridging the gap between AI and human–robot interaction research. *The International Journal of Robotics Research (IJRR)* 36(5-7):635–659.

Laird, J. E. 2012. *The Soar Cognitive Architecture*. MIT Press.

Lemaignan, S.; Ros, R.; Sisbot, E. A.; Alami, R.; and Beetz, M. 2012. Grounding the Interaction: Anchoring Situated Discourse in Everyday Human-Robot Interaction. *International Journal of Social Robotics (IJSR)* 4(2):181–199.

Lemaignan, S.; Warnier, M.; Sisbot, E. A.; Clodic, A.; and Alami, R. 2017. Artificial cognition for social human–robot interaction: An implementation. *Artificial Intelligence* 247:45–69.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138(1-2):39–54.

Mininger, A., and Laird, J. E. 2016. Interactively Learning Strategies for Handling References to Unseen or Unknown Objects. *Advances in Cognitive Systems* 4:1–16.

Puigbo, J.-Y.; Pumarola, A.; Angulo, C.; and Tellez, R. 2015. Using a cognitive architecture for general purpose service robot control. *Connection Science* 27(2):105–117.

Quigley, M.; Conley, K.; Gerkey, B. P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Redmon, J., and Farhadi, A. 2017. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517–6525. IEEE.

Reiser, U.; Connette, C.; Fischer, J.; Kubacki, J.; Bubeck, A.; Weisshardt, F.; Jacobs, T.; Parlitz, C.; Hägele, M.; and Verl, A. 2009. Care-O-bot®3 - creating a product vision for service robot applications by integrating design and technology. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1992–1998.

Sanelli, V.; Cashmore, M.; Magazzeni, D.; and Iocchi, L. 2017. Short-term human robot interaction through conditional planning and execution. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010a. Planning for Human-robot Teaming in Open Worlds. *ACM Transactions on Intelligent Systems and Technology (TIST)* 1(2):14:1—-14:24.

Talamadupula, K.; Benton, J.; Schermerhorn, P.; Kambhampati, S.; and Scheutz, M. 2010b. Integrating a Closed World Planner with an Open World Robot: A Case Study. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, 1561–1566. AAAI Press.

Tenorth, M., and Beetz, M. 2013. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research (IJRR)* 32(5):566–590.

Veloso, M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. CoBots: Robust Symbiotic Autonomous Mobile Service Robots. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 4423–4429. AAAI Press.

Wisspeintner, T.; Van Der Zant, T.; Iocchi, L.; and Schiffer, S. 2009. RoboCup@ Home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies* 10(3):392–426.