2015-09-10

Collaborative Learning of Hierarchical Task Networks from Demonstration and Instruction

Anahita Mohseni-Kabir
Worcester Polytechnic Institute

Collaborative Learning of Hierarchical Task Networks from Demonstration and Instruction

by

Anahita Mohseni-Kabir - amohsenikabir@wpi.edu

A Thesis

Submitted to the Faculty

of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

September 2015

Approved

Professor Charles Rich, Thesis Advisor

Professor Sonia Chernova, Thesis Advisor

Professor Joseph Beck, Thesis Committee Member

Abstract

This thesis presents learning and interaction algorithms to support a human teaching hierarchical task models to a robot using a single or multiple examples in the context of a mixed-initiative interaction with bi-directional communication. Our first contribution is an approach for learning a high level task from a single example using the bottom-up style. In particular, we have identified and implemented two important heuristics for suggesting task groupings and repetitions based on the data flow between tasks and on the physical structure of the manipulated artifact. We have evaluated our heuristics with users in a simulated environment and shown that the suggestions significantly improve the learning and interaction. For our second contribution, we extended this interaction by enabling users to teaching tasks using the top-down teaching style in addition to the bottom-up teaching style. Results obtained in a pilot study show that users utilize both the bottom-up and the top-down teaching styles to teach tasks. Our third contribution is an algorithm that merges multiple examples when there are alternative ways of doing a task. The merging algorithm is still under evaluation.

Acknowledgements

I would like to express the deepest appreciation to my advisors, Professor Charles Rich and Professor Sonia Chernova, who have been tremendous mentors for me. Besides my advisors, I would like to thank my thesis reader, Professor Joseph Beck, for his insightful comments. I would like to also thank Professor Candace Sidner for sharing expertise and guidance throughout the project and Professor Dmitry Berenson for his insights into the robotics part of the project. Furthermore, I would like to thank fellow postdoc student Jim Mainprice for his work on simulated tasks' execution, and fellow graduate students Daniel Miller for the development of Gazebo simulation and primitive action detection using Vicon data, and Artem Gritsenko for extending Jim's work and applying it on the PR2 robot. This work is supported in part by ONR contract N00014-13-1-0735. The opinions expressed in this document are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

Contents

List of Figures

List of Tables

1 Introduction

In this section, we first explain the problem we aim to address. Then, we will focus on the contributions of our work in resolving that problem.

1.1 Problems statement

Robots have finally made their way into our homes, offices, museums and other public spaces as service robots. In order to realize the dream of robot assistants performing human-like tasks together with humans in a seamless fashion, we need to provide robots with the fundamental capability of understanding complex, dynamic and unstructured environments and tasks. Equally importantly, to permit natural cooperation we need to enable robots to share our understanding of the tasks and the environment. Robotics and artificial intelligence (more specifically the field of human-robot interaction) are working together to make this dream possible.

The ability to learn, both from direct experience and from other agents, such as humans, is at the core of artificial intelligence. This work focuses on the problem of how a robot can efficiently learn complex procedural tasks from a human teacher who is an expert in the task domain, but not in robot programming. Our approach integrates learning from demonstration (LfD) with hierarchical task networks (HTNs). Since our target application domain is learning complex procedural tasks, such as manufacturing, equipment inspection and maintenance, we use HTNs as the representation of the learned knowledge. HTNs are a powerful framework for representing and organizing task knowledge that is both efficient for computers and natural for humans. The benefits of HTNs over flat representation are discussed in detail in Section 2.1.

Most LfD research has focused on learning low-level manipulation behaviors and has treated the human quite narrowly as a "source of data" for the machine learning algorithms employed. Our approach is to view the interaction between the human teacher and the learning agent as a mixed-initiative *collaboration*, in which both parties are committed to

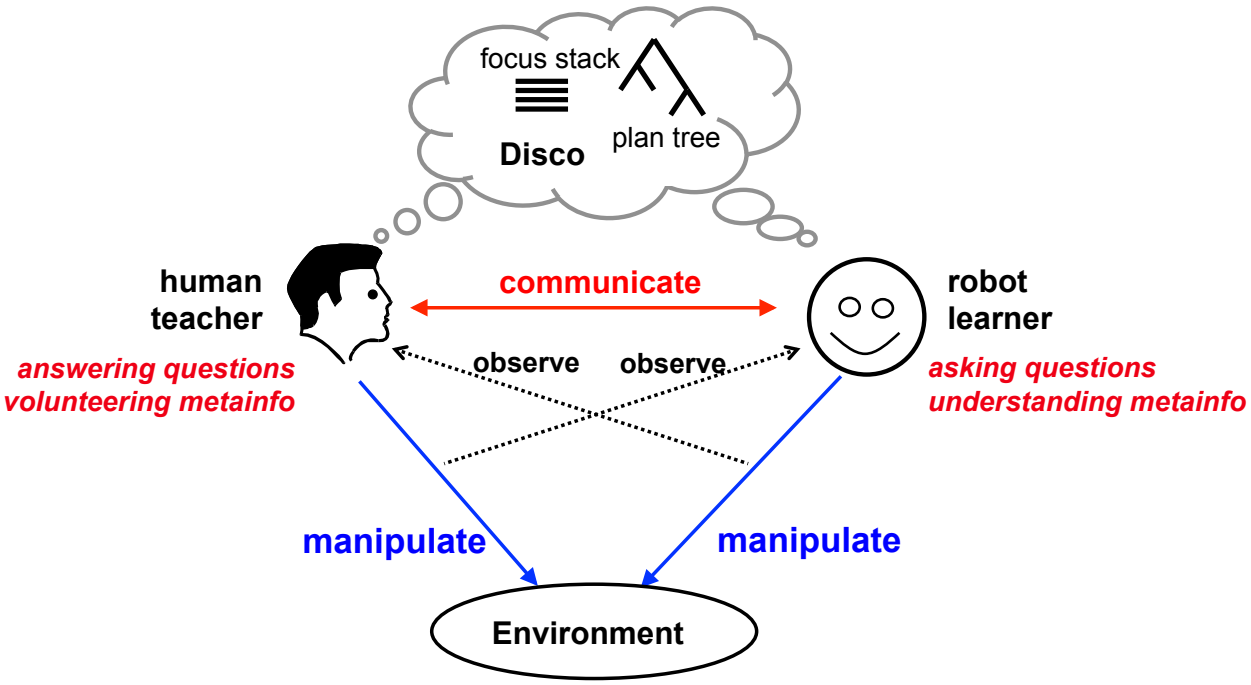


Figure 1: LfD as a collaborative discourse.

the shared goal of successful learning, and in which both parties make contributions in the form of both actions and communication, including verbal instructions, making suggestions, and critiquing. For this, we draw heavily on collaborative discourse theory and tools. More specifically, in the situated interaction illustrated in Figure 1, which is our target, both the human and the robot can manipulate and observe the other's manipulations of artifacts in the shared environment, and there is bi-directional communication between them. Collaborative discourse theory [5] provides a foundation for both the algorithms and the implementation of our system.

In this work, we present algorithms that will allow a robot to learn complex tasks from a human teacher. Our approach is based on the conjecture that it is often easier for people to generate and discuss examples of how to accomplish tasks than it is to deal directly with task model abstractions. Our approach is thus for the robot to iteratively learn tasks from human demonstrations and instructions. We use the term *instruction* to mean that the teacher tells the robot the steps of a task without any execution of these steps, and *demonstration* to mean that the robot (or teacher) also performs the desired action. The robot actively

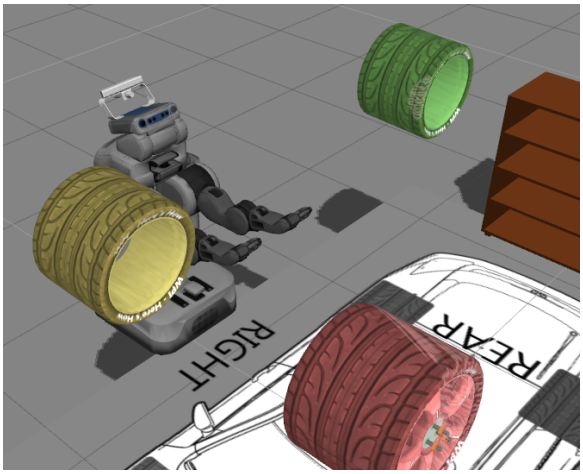


Figure 2: Simulated environment with PR2.

participates in the learning process by asking questions and making suggestions, to which human responds. By engaging the robot as an active partner in the learning process, and by using the hierarchical structures, we believe that complex tasks can be naturally taught by non-expert in robot programming users.

The example domain we have chosen for this research is maintenance tasks and specifically car maintenance, because it is challenging yet familiar to most readers. We have started with tire rotation and expect in the future to teach a robot how to check the oil, change filters and belts, and so on.

This thesis is organized around the running example of a tire rotation HTN, which is introduced in Section 4.1. The intended application of this work is with a real robot, such as the PR2, in a shared physical environment. As a first step, however, we have implemented and evaluated our techniques using Gazebo (robot simulation tool) in the simulated environment shown in Figure 2.

1.2 Contributions

In summary, our work makes the following contributions:

1. A unified system that integrates hierarchical task networks and collaborative discourse theory with the learning from demonstration;

2. A novel approach for learning task structure from a single demonstration, including the task hierarchy, temporal constraints and inputs/outputs of a task;
3. Integration of mixed-initiative interaction into the learning process through suggestion generation;
4. A novel approach that integrates demonstrations and instructions to make the interaction more natural;
5. Novel generalization techniques that reduce the number of demonstrations required to learn the task through merging multiple examples.

In the following sections, we begin by discussing related work in the areas of collaborative discourse theory and robot learning from demonstration. We then present an overview of our system in Section 3 and describe our algorithms in details in Section 4 and 5. Section 4.2 presents the algorithms implemented to learn from a single example using the bottom-up teaching style, and our user study. The results of our first user study described in Section 4.2.3 show that the robot's suggestions significantly improve the learning and interaction. Section 4.4 then describes an extension of our system that integrates both the bottom-up and the top-down teaching styles based on the lessons learned from the first user study, as well as a pilot study evaluating that system. We conclude this work by discussing the future direction of this work.

2 Background and Related Work

In this section, we give an overview of HTNs. We then discuss related work in collaborative discourse theory and robot learning from demonstration.

2.1 Hierachical Task Networks

HTNs are widely used in artificial intelligence, especially in interactive systems. An HTN is essentially a tree in which the fringe nodes denote *primitive* tasks, i.e., tasks that can be directly executed (e.g., by a robot), and the other nodes denote *abstract* tasks, which must

be decomposed in order to be executed. As an example, consider the HTN for tire rotation shown in Figure 3 (page 8). This is a hand-coded version of the kind of HTN that our system learned from the participants in our user study.

The abstract tasks in an HTN are not absolutely necessary to execute the toplevel task—RotateTire could simply be represented as a sequence of 64 primitives. However, there is much evidence that humans naturally think about complex tasks hierarchically. For example, collaborative discourse theory (described below) is based on a hierarchy of goals.

The abstract tasks in an HTN provide an important vocabulary for communication between the human and robot in the context of a collaboration. This shared vocabulary is needed for discussing the partial completion status of the task, for delegating subtasks, for discussing potential problems with execution, and so on. Furthemore, the abstract tasks can often be reused in other situations. For example, after learning the HTN in Figure 3, the robot not only knows how to rotate tires, but it has also learned two abstract tasks (UnscrewHub and UnhangHub) that are useful for fixing a flat tire.

HTNs are also commonly used as an alternative to symbolic planning in complex real-world applications where a complete logical formalization is difficult or infeasible. Although tire rotation is simple enough that it could easily be formalized for a symbolic planner, our target is applications in which this is not true. The learning algorithms in this paper therefore require only the name and input types of a task.

There are many different formalisms available for representing HTNs. In this work, we use the ANSI/CEA-2018 standard [12] because, among other things, it explicitly represents data flow between tasks, which is the basis for one of the grouping heuristics described in 4.2.2.

2.2 Collaborative Discourse Theory

Discourse is the technical term for an extended communication between two or more participants in a shared context, such as collaboration. Collaborative discourse theory refers to a body of empirical and computational research about how people collaborate [5], that

uses the same hierarchical task abstraction notions expressed in HTNs. We are applying the principles of collaborative discourse theory in this work to improve the effectiveness of the interaction between the human teacher and robot learner (such as to help them jointly manage their focus of attention) using a software tool called Disco, which is the open-source successor to Collagen [13].

2.3 Learning from Demonstration

There is extensive research (involving both robots and virtual agents) on learning from demonstration [1], hierarchical task learning [4, 6, 9] and interactive task learning [2, 10, 9, 6]. However, other than Rybski et al., discussed below, we do not know of any other work that combines all of these aspects, as our present work does.

Research on hierarchical task learning includes both learning from demonstration and learning from instructions approaches. LfD approaches generally focus on learning from multiple demonstrations, rather than learning from a single demonstration. For example, in robotics research, Nicolescu and Mataric [10] implemented a system for LfD that enables a robot to learn and refine representations of complex tasks, and to generalize multiple demonstrations into a single graph-like representation. In [16], Veeraraghavan and Veloso presented a demonstration-based approach for teaching a robot sequential tasks with repetitions. Our approach differs from the above methods in representation, and in that it integrates learning through demonstration, instruction and making suggestions.

Niekum et al. [11] presented a method for automatically segmenting low-level trajectory demonstrations, recognizing repeated skills, and generalizing complex tasks from unstructured demonstrations. In contrast, our work focuses on high-level task learning, in which the task structure is more intuitively clear to users, allowing us to leverage human knowledge to learn from fewer demonstrations. In other related work, Hayes and Scassellati [6] learn HTNs through multi-agent coordination with humans in the loop. In contrary to their work, we focus on learning temporal constraints, and parametrization as well as task hierarchy from a single example.

Garland et al. [4] developed an offline, purely software system that generalizes an HTN from a set of execution traces (demonstrations). A novel aspect of their system was support for a domain expert to refine past demonstrations. In addition to relying on multiple demonstrations, this system does not ask questions—the user must take all of the initiative in building the HTN.

Most research on interactive task learning goes under the rubric of "active learning" and involves the system asking questions of the user. We prefer the term "interactive" to emphasize the bidirectional nature of the communication. For example, Chernova and Veloso [3] developed a policy learning algorithm in which the robot asks the human to provide labels for states in which the robot has a high uncertainty. Focusing more on social interaction and broader question types, Cakmak and Thomaz [2] have formalized three question types—label, demonstration and feature requests—and studied their use in LfD. None of these systems generate questions that suggest repetitions or grouping.

Several other works have focused on learning from instruction. Huffman and Laird [7] explored the requirements for instructable agent design, and demonstrated that a simulated robotic agent could learn useful instruction in natural language. In [8], Mohan and Laird extended this work by instantiating their design in the Soar cognitive architecture. In this work, they also used a hierarchical structure for their task representation. Our approach differs from theirs in our focus on learning ordering constraints, parametrization, generalization and independence from a particular cognitive architecture. Our additional distinction is that our robot will make suggestions about the structure of the tasks, for example concerning repeating or grouping the actions, whereas in their approach, the questions are focused on learning the steps of the tasks.

She et al. [15] presented a method for learning new actions through situated human-robot dialogue in a blocks world. They explored connections between symbolic representation of natural language and continuous sensorimotor representations of the robot. Different from their approach, here we focus on learning ordering constraints, parametrization and generalization and we use HTNs instead of symbolic planning. Additionally, in contrast to

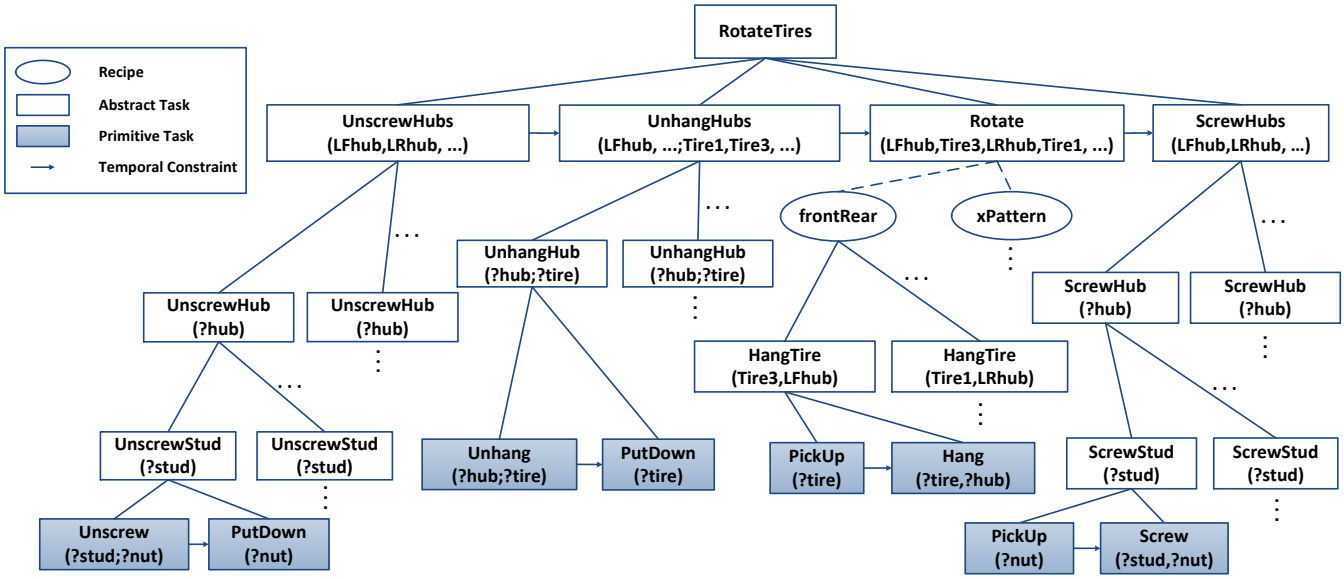


Figure 3: A hand-coded hierarchical task network for tire rotation.

their work, we are not learning from natural language dialogue.

The closest work to ours is by Rybski et al. [14], who developed an algorithm that combines spoken language understanding, dialog and physical demonstration to learn complex plans. Their task representation allows abstract tasks to be constructed from simpler subtasks in order to create hierarchical structures. Their robot can also verify the HTN with the human by asking questions and allowing the human to add additional conditional cases. However, their approach for learning task structure differs from ours in that it does not leverage ordering and data flow constraints. Furthermore, the robot's questions are limited to filling in unspecified conditions (e.g., missing else statements) while our suggestions are aimed at improving the plan quality and teaching efficiency.

3 Overview of the System

Figure 4 presents an overview of our system architecture. The key collaborative learning components are highlighted in gray and are discussed in detail in Section 3.1. The architecture consists of two subsystems, learning and execution. The inputs to the system we have built are the human's demonstrations, instructions and answers to suggestions. In Section

4.4, we describe an example learning scenario for the `RotateTires` task that utilizes all three types of interaction – demonstrations, instructions and suggestions. The goal of our system is to learn a task representation similar to the hand-coded HTN shown in Figure 3. In the current implementation, the user interacts with the robot via a graphical user interface (GUI) connected to Gazebo (see Figure 7). This graphical user interface is completely menu-based and does not involve any natural language processing.

The graphical details of GUI's design are not the main contribution of the this thesis and the primary purpose of the GUI is to evaluate our algorithms. We are currently working on making the interaction between the user and system as natural as possible by replacing the menu-based GUI with the human's physical demonstrations and using simple speech recognition to enable the user to give instructions more naturally.

3.1 Learning Subsystem

The Learning subsystem learns tasks from a single or multiple examples. The Task Structure Learning component is responsible for learning the hierarchical task structure, ordering constraints, inputs/outputs of tasks, and applying relations. After learning this general structure, we apply the Generalization module, which merges multiple examples of a task if they are available. The Suggestion Generation module is responsible for generating suggestions based on the heuristics discussed in Section 4.2.2.1. We will explain these modules

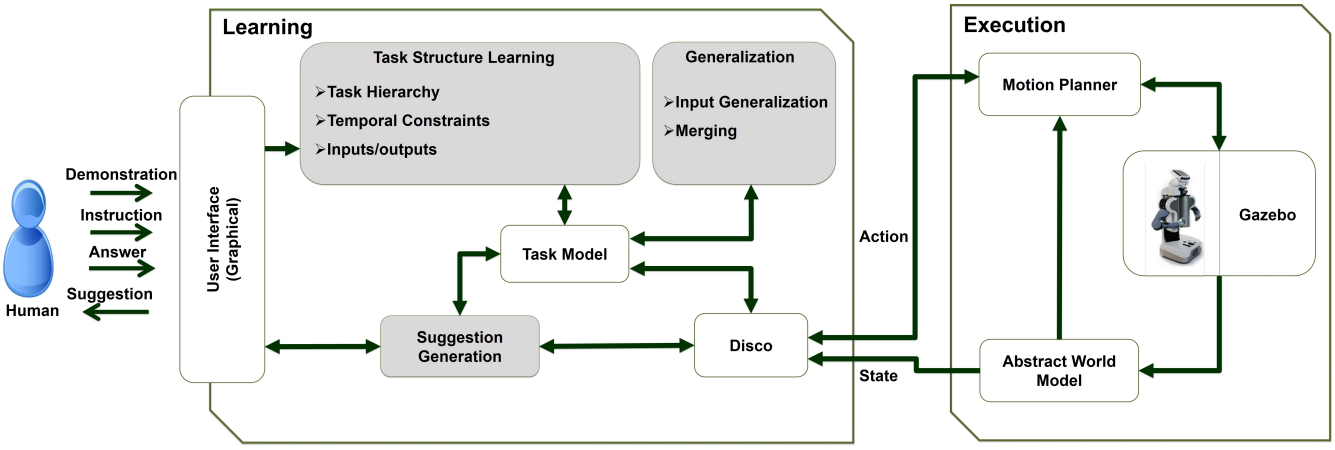


Figure 4: System architecture

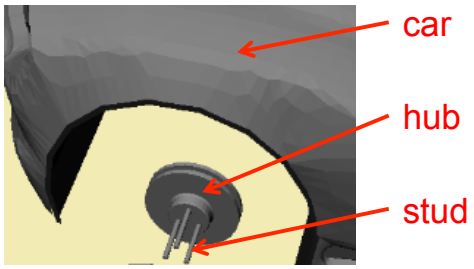


Figure 5: Terminology for car parts.

in more details in Section 4 and 5. The entire learning process is supported by Disco, an implementation of collaborative discourse theory and ANSI/CEA-2018. Disco's dialogue management capabilities are used to maintain a focus stack which keeps track of the current topic and has expectations for what needs to be said or done next. During the execution of a learned task, the Disco planner decomposes each non-primitive task in the HTN into its subtasks. When the planner reaches a primitive task, the primitive task is sent as an action command to the execution subsystem.

3.2 Execution Subsystem

The execution subsystem includes the Motion Planner, Abstract World Model (AWM), and the robot embodied either as a physical PR2 or as a Gazebo simulation. The Motion Planner receives primitive task commands, generates plans for their execution, and sends the plans to the robot. The AWM module tracks the current state of the world based on the simulated or the real-world data. For real world experiments we use the Vicon motion capture system to track the position of world objects and the user. Failures in the execution of the task are propagated back to Disco both through the Motion Planner (planning failure) and AWM (execution failure). Handling failures is not the focus of this work and is part of our future work.

4 Learning from a Single Example

In this section, we first discuss the system's capabilities to learn an HTN from the user using the bottom-up teaching style. We then show that these capabilities are not sufficient for learning from a naive user and we explain how semantics are used to make learning possible. Before getting into the details of the system, we will describe the task and the terminology that we are using in the next subsection.

4.1 Target Domain and Task Description

Since the HTN in Figure 3 (page 8) will be used as a running example in this document, and it was the subject of our user studies, it is worthwhile to examine it here in some detail. First, Figure 5 shows the terminology we are using for the relevant parts of a car: a tire goes on a hub and is secured with three nuts (one on each stud). The hubs are identified as LF, LR, RF, RR, standing for the left-front, left-rear, etc., and each hub has three studs, identified as "LFhub.stud1", etc. The tires are named Tire1, Tire2, etc., and in the user study have distinct colors for convenience of identification.

In ANSI/CEA-2018, both abstract and primitive tasks have typed inputs and outputs, and preconditions and postconditions (specified as as JavaScript expressions). The tire rotation HTN uses six primitive task types: Screw, Unscrew, Hang, Unhang, PickUp, and PutDown, with their respective input and output types shown. For example Unscrew (see Figure 6) takes a stud as input and returns a nut as output. There are 64 primitive tasks in the fringe of the tire rotation tree (some of the repetitive structure has been elided in the figure to save space).

Figure 3 (page 8) defines 11 abstract task types, such as UnscrewHubs, which are used 45 times in the interior of the tree. This particular way of decomposing the tire rotation task is not the only possible way, but the other reasonable decompositions all have a similar number of abstract types and total number of nodes.

In ANSI/CEA-2018, the subtasks of a given task are by default *unordered*. An explicit

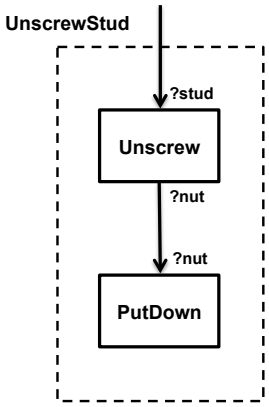


Figure 6: Data flow.

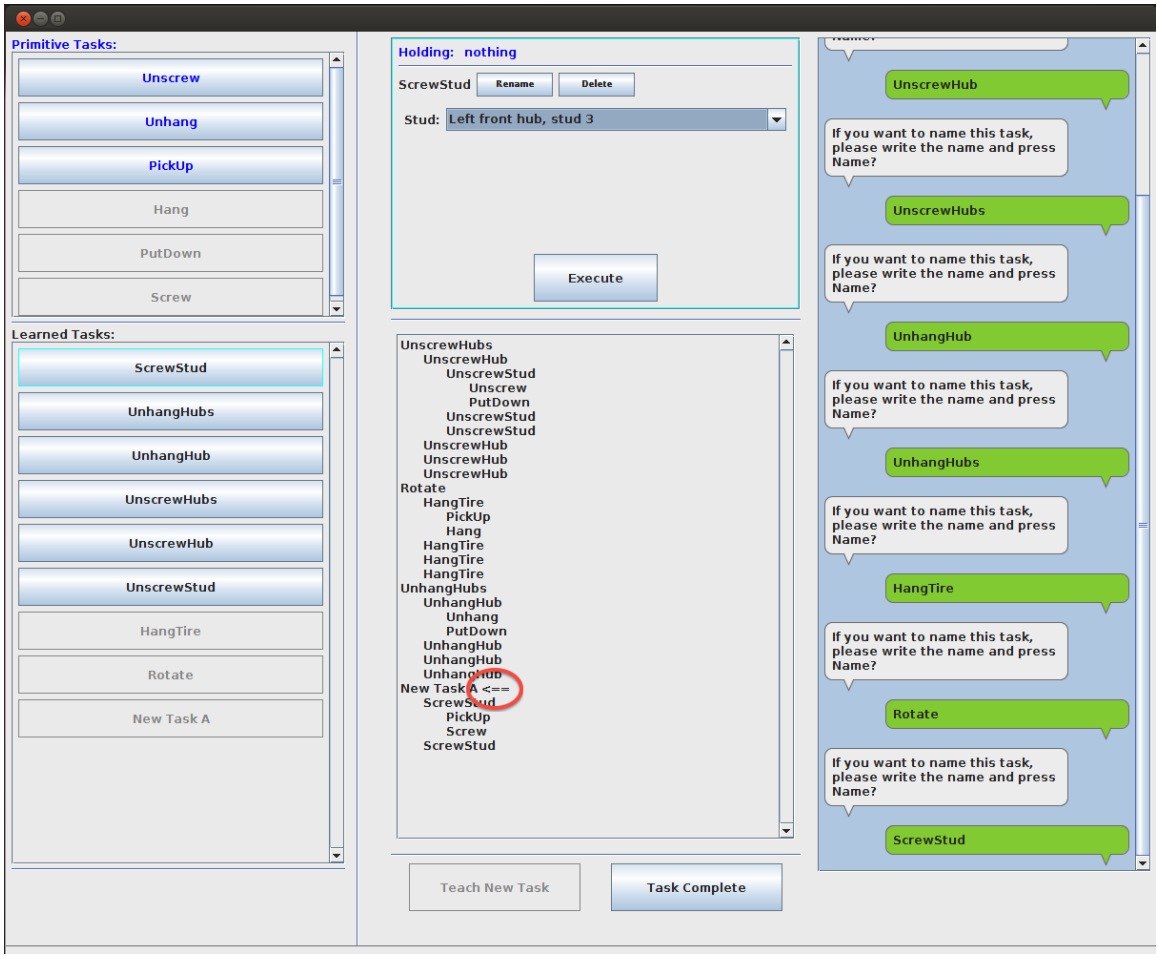


Figure 7: Graphical user interface for teacher.

temporal constraint can be added between two tasks to specify a partial ordering. For example, the four toplevel steps of RotateTires are totally ordered, but the steps of Unscrew-Hubs are unordered, since it does not matter in what order you unscrew the nuts.

Data flow in ANSI/CEA-2018 is an identity constraint between a task's output and an input of a following task, i.e., another subtask of the same parent task. Figure 6 shows an example of data flow in the tire rotation HTN.

Finally, a key feature of HTN's in general is the ability to specify alternate decompositions, or "recipes." Figure 3 (page 8) provides two recipes for the Rotate step of tire rotation corresponding to two different rotation patterns: front-to-rear and corner-to-corner ("X"). Learning alternative recipes is one of those circumstances in which multiple examples are unavoidable.

4.2 Learning from a Single Example Using the Bottom-Up Style

In this section we will explain, using two example interactions, how our system learns the hierarchical structure of the tire rotation HTN in Figure 3 (page 8). The first example interaction, shown in 4.2.1, is a hypothetical "ideal" interaction in which the robot provides no helpful suggestions, but the user teaches exactly the hand-coded hierarchical structure anyway. While this interaction is fully supported by our system, in our user study, no participants came even close to this ideal behavior when the robot provided no suggestions, instead resulting in far less optimal task structures. In the second interaction, shown in 4.2.2, the same HTN is learned *with* suggestions from the robot that guide the user in structuring the task. This interaction is in fact typical of what we saw with our users (see Section 4.2.3) who received and accepted suggestions from the robot.

Our focus in this work is on learning the grouping structure of the HTN. The names chosen for abstract tasks and some details of the inputs and outputs may vary from our hand-coded HTN. Furthermore, the first version of our system (described in this section) only supports a bottom-up style of teaching, in which primitives are first combined into small abstract tasks, which are then combined into bigger abstract tasks, and so on. This

style lends itself well to demonstration. In 4.3, we expand this approach to also support the top-down teaching style using instructions, in which the higher level tasks are defined before the lower level tasks. The second version of our system supports both the top-down and the bottom-up styles of teaching using both instructions and demonstrations.

Figure 7 shows the graphical user interface we have developed for our first user study with a simulated robot and environment. In use, the human teacher will be interacting with this interface while simultaneously seeing the effects of actions on a side-by-side simulation screen, similar to Figure 2 (page 3). The screen shot in Figure 7 corresponds to the state of the GUI during line 32 of Figure 8.

Before we describe its operation in detail, we also need to emphasize that this GUI is *not* a research goal in its own right, but rather a means to test our learning algorithms and overall interaction design while still in simulation. As we discuss in Section 6, most or all of this GUI will disappear once we move into a physical shared human-robot environment. After our first user study with the GUI, we found that the system requires some other capabilities to make the interaction between the user and the robot more natural so we designed another GUI which is closer to the interaction that we want to have in a shared human-robot environment. We will explain this user interface in more detail in Section 4.3.

The left side of the GUI contains buttons that select a primitive (top left) or abstract (bottom left) task to execute. The primitive task types are predefined and do not changed. However, new abstract task buttons are added whenever a new abstract task type is learned. Furthermore, to help the user maintain focus of attention, buttons for tasks that are inapplicable in the current environment are disabled (greyed out). A primitive task is inapplicable if its precondition is false; an abstract task is inapplicable if all possible first subtasks (according to the temporal constraints) are inapplicable.

The top middle area of the GUI is mainly for specifying inputs, if any, as needed before executing a task. Each input can be selected from a drop-down list of objects in the environment of the correct type. When the user presses the Execute button, the robot executes the selected primitive or abstract task.

The Rename and Delete buttons at the top of this area are for renaming and deleting learned abstract tasks. The Holding line above these buttons is a small informational display that keeps track of the object, if any, that the robot is currently holding.

The middle area of the GUI below the input selection area is the main informational display, which shows the current hierarchical structure of the learned tasks. The small arrow next to New Task A (red circle added here for visibility) indicates the abstract task currently being learned. This area is also where the highlighting appears for tasks suggested for grouping, as described in Section 4.2.2. The Teach New Task and Task Complete buttons below this area respectively start and end the demonstration of a new abstract task.

Finally, the right side of the GUI contains a "chat" window in which the robot can ask the user questions and the user can reply. For example, whenever the user presses the Task Complete button, the robot prompts the user for a name for the newly learned abstract task. This is also where the robot's helpful suggestions appear and where the user can accept or reject them.

4.2.1 Learning Without Suggestions

Figure 8 shows a hypothetical interaction that results in the system learning the hierarchical structure of Figure 3 (page 8). The user starts by teaching a new abstract task (which will be named `UnscrewStud`), demonstrating its two primitive subtasks: `Unscrew` (applied to `stud1` of the `LFhub`) and `PutDown` (applied to the nut it is holding as a result).

Then, in lines 5–8, the user applies this new abstract task to the other two studs of the `LFhub`. Notice that when the user completes this demonstration and names this new higher-level abstract task `UnscrewHub`, the system includes the immediately preceding demonstration of `UnscrewStud` (lines 1–4) in the abstraction, so that `UnscrewHub` has *three* subtasks, one for each stud.

Continuing, in lines 9–13, the user similarly applies this new abstract task to the other three hubs and names this new yet higher-level abstract task `UnscrewHubs` (plural). As before, the system includes the immediately preceding demonstration (lines 5–8) in the

abstraction, so that `UnscrewHubs` has four subtasks, one for each tire. At this point all the nuts are off the car.

Next, in lines 14–22, the user similarly teaches a high-level abstract task, `UnhangHubs`, which takes off all the tires and leaves them on the ground. The heart of tire rotation is in teaching the Rotate abstract task in lines 23–31, where the user hangs the tires back on in the rotated pattern (`Tire3` on `LFhub`, etc.).

To save space, 13 button presses (including 7 Execute buttons) are elided at line 32, culminating in the learning of `ScrewHubs`. The learning process for `ScrewHubs` is entirely parallel to the learning of `UnscrewHubs` in lines 1–13. Finally, in lines 33–38, the four highest-level abstract tasks are put together to make `RotateTires`.

In summary, in this interaction, the user provided 28 execution commands and the robot learned 11 new abstract task types, including the toplevel `RotateTires`. (The robot, of course, executed more than 28 actions because some of the tasks instructed by the user were abstract.)

Unfortunately, as mentioned earlier, no users in our study came even close to this ideal behavior. The best anyone achieved was 6 new abstract task types. So, the question is, how can we help users get closer to this ideal behavior? The next section describes our approach, which is to use general heuristics to generate helpful suggestions.

Teach New Task
  Execute Unscrew(LFhub.stud1)
  Execute PutDown(the nut)
Task Complete: UnscrewStud
Teach New Task
  Execute UnscrewStud(LFhub.stud2)
  Execute UnscrewStud(LFhub.stud3)
Task Complete: UnscrewHub
Teach New Task
  Execute UnscrewHub(LRhub)
  Execute UnscrewHub(RFhub)
  Execute UnscrewHub(RRhub)
Task Complete: UnscrewHubs
Teach New Task
  Execute Unhang(LFhub)
  Execute PutDown(Tire1)
Task Complete: UnhangHub
Teach New Task
  Execute UnhangHub(LRhub)
  Execute UnhangHub(RFhub)
  Execute UnhangHub(RRhub)
Task Complete: UnhangHubs
Teach New Task
  Execute PickUp(Tire3)
  Execute Hang(Tire3,LFhub)
Task Complete: HangTire
Teach New Task
  Execute HangTire(Tire1,LRhub)
  Execute HangTire(Tire4,RFhub)
  Execute HangTire(Tire2,RRhub)
Task Complete: Rotate
...
Teach New Task
  Execute UnscrewHubs(LFhub,LRhub,RFhub,RRhub)
  Execute UnhangHubs(LFhub,LRhub,RFhub,RRhub)
  Execute Rotate(LFhub,Tire3,LRhub,Tire1, ...)
  Execute ScrewHubs(LFhub,LRhub,RFhub,RRhub)
Task Complete: RotateTires

Figure 8: An hypothetical ideal interaction to teach tire rotation without robot's suggestions.

4.2.2 Learning With Suggestions

The robot should act as an active participant in the collaboration to help the human teacher in providing useful information about the task. To provide this useful information, the robot requires some knowledge about the task and the environment. The advancements in robotics, more especially the advancements in perception along reasoning about the environment to find the spatial semantics, provides us with this knowledge. Acquiring this knowledge (more specifically, we are using the *Part-whole* relation in the following subsection and the *On* relation in Subsection 4.3.1) from the sensory input is not the focus of this work. This information is used as an input to the system to make repetition suggestions in the following subsection and handling inputs in Section 4.3.1.

4.2.2.1 Heuristics

Reflecting on the ideal interaction in Figure 8, we identified two general heuristics to help the robot learn HTNs with useful abstract tasks. Because these are heuristics, which means that they are not always right, we communicate the robot's suggestions as questions to which the user can say yes or no (to accept or reject the suggestion).

The first heuristic, called the *parts* heuristic, is inspired by the grouping of repeated tasks in the ideal interaction, which is driven by the part-whole structure of the domain, as shown in Figure 10. The heuristic rule suggests repeating the last executed task on the other parts of the same type that share the same physical parent (e.g., the other two studs of a hub). If this suggestion is accepted, the system then groups the last task together with repetitions for the other parts into a new abstract task.

The second heuristic, called the *data flow* heuristic, is inspired by the grouping that the ideal user does in lines 1–4 of Figure 8. The heuristic rule suggests grouping two consecutive tasks whenever an output of the first is an input of the second (e.g., the data flow shown in Figure 6 on Page 12, the output of `Unscrew` is the input of `PutDown`).

Figure 9 shows pseudocode for these two heuristics, which are applied in order after the execution of each primitive or abstract task. Notice also that both of these heuristics are

$last \leftarrow$ last task executed

if $|inputs(last)| \neq 1$ **then return**

$input \leftarrow first(inputs(last))$; $whole \leftarrow parent(input)$

$parts \leftarrow \varnothing$

foreach $other \in type(input)$

  if $other \neq input \wedge parent(other) = whole$

    then add $other$ to $parts$

if $parts \neq \varnothing$

  then ask 'Should I execute '+$last$+' on other '+$type$+'s of '+$whole$+'?'

(a) Parts heuristic

$last \leftarrow$ last task executed

$previous \leftarrow$ task executed immediately before $last$

foreach $input$ of $last$

  if $source(input) = previous$

    then ask 'Do you want to group the highlighted steps ('+$previous$+' and'+$last$+')

      into a new subtask?'

      return

(b) Data flow heuristic

Figure 9: Pseudocode for suggestion heuristics.

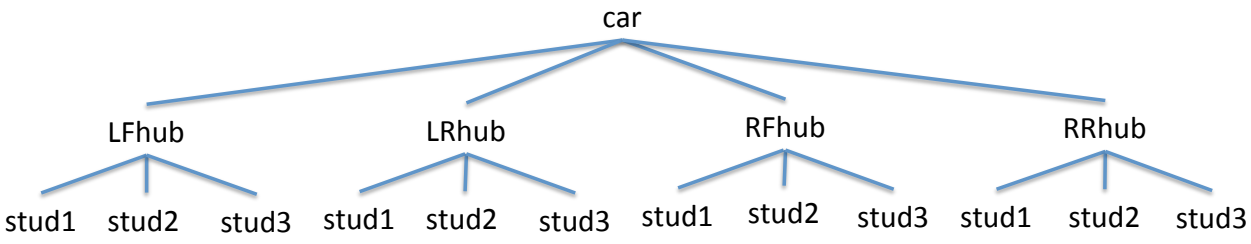


Figure 10: Part-whole knowledge.

quite general. They may be more or less useful in different domains, but they are expressed entirely in terms of mathematical concepts, such as data flow and *Part-whole* relationships. There is nothing specifically about car maintenance in the statement of the rules.

4.2.2.2 Ideal Interaction Using Suggestions

Now let's see how these heuristics function in the typical interaction shown in Figure 11. The first question, based on the data flow heuristic, appears on line 4. The user accepts the suggestion, leading to the creation of a new abstract task type called `Unscrew/PutDown`. Notice that system automatically generates a convenient default name for new abstract task types created in response to suggestions. The user can rename these using the Rename button at the top middle of the GUI.

The second question, based on the parts heuristic, appears on line 6 and results in the new abstract task `Unscrew/Putdown*3`, which corresponds to `UnscrewHub` in Figure 8. If the user accepts the suggestion, the three studs (LFhub.stud1,LFhub.stud3,LFhub.stud3) will also be generalized to one hub (LFhub).

The interaction proceeds now in a parallel fashion to Figure 8, with the user answering yes to the next three questions. Then, on line 20, the user first answers no. This suggestion is generated by the data flow heuristic (in this case applied to abstract tasks). Because the user has just executed the first step of the front-back tire rotation pattern, which is to put `Tire3` (that was originally on the `LRhub`) onto the `LFhub`, it seems plausible that she already has in mind the idea of grouping this with the next three steps, not the previous task.

The next question, appearing on line 25, suggests in fact grouping these steps into a new abstract task, which corresponds to Rotate in Figure 8. The user accepts this suggestion.

The second instance in which the user answers no appears on line 27, in answer to another suggestion generated by the grouping heuristic. This is a case in which a yes answer would also be reasonable. All it would do is add an extra layer of grouping near the top of the hierarchy, grouping together what are called `UnhangHubs` and `Rotate` in Figure 3 (Page 8).

The remaining three questions follow the same pattern as the first three questions, just

as the elided lines in Figure 8 follow the same pattern as lines 1–9 in that interaction.

In summary, notice that in this interaction the user executed only 11 tasks, as compared to 28 in Figure 8 (Page 17). Even with answering 12 yes/no questions, the user in this interaction is expending less effort to achieve the same quality plan. This effect is borne out by the user study described in the next section.

Importantly, we note that following the suggestions does not always lead to desired results, which is why we rely on them as heuristics instead of rules. As an example, consider teaching the robot to cook an omelet following a recipe that requires three eggs. After breaking the second egg in a carton, the grouping heuristic will make the suggestion to break all the remaining eggs, which is not the intended behavior. By presenting the suggestion in the form of a question, our system provides the user with a simple way of ignoring or overriding the suggested behavior when it is not desired.

1 Teach New Task
2 **Execute** Unscrew(LFhub.stud1)
3 **Execute** PutDown(the nut)
4 *Robot: "Do you want to group the highlighted steps (Unscrew and PutDown) into a new subtask?"*
5 "Yes"
6 *Robot: "Should I execute Unscrew/PutDown on other studs of left front hub?"*
7 "Yes"
8 *Robot: "Should I execute Unscrew/PutDown*3 on other hubs of the car?"*
9 "Yes"
10 **Execute** Unhang(LFhub)
11 **Execute** PutDown(Tire1)
12 *Robot: "Do you want to group the highlighted steps (Unhang and PutDown) into a new subtask?"*
13 "Yes"
14 *Robot: "Should I execute Unhang/PutDown on other hubs of the car?"*
15 "Yes"
16 **Execute** PickUp(Tire3)
17 **Execute** Hang(Tire3,LFhub)
18 *Robot: "Do you want to group the highlighted steps (PickUp and Hang) into a new subtask?"*
19 "Yes"
20 *Robot: "Do you want to group the highlighted steps (Unhang/PutDown*4 and PickUp/Hang) into a new subtask?"*
21 "No"
22 **Execute** PickUp/Hang(Tire1,LRhub)
23 **Execute** PickUp/Hang(Tire4,RFhub)
24 **Execute** PickUp/Hang(Tire2,RRhub)
25 *Robot: "Do you want to group the highlighted steps (four HangTire's) into a new subtask?"*
26 "Yes"
27 *Robot: "Do you want to group the highlighted steps (Unhang/PutDown*4 and HangTire*4) into a new subtask?"*
28 "No"
29 ...
30 Task Complete: RotateTires

Figure 11: A typical interaction to teach tire rotation with robot's suggestions.

4.2.3 Evaluation

We conducted a study with non-expert users to evaluate the usability of our algorithms discussed in this section, and more specifically, the contribution of the suggestions toward improving the interaction and learning. In this user study, we focus on "one shot" learning with active agent, which means that the complete task is only demonstrated once and also the agent makes suggestions based on the heuristics discussed in Section 4.2.2.1. We used

the graphical user interface described in Section 4.2 to test our algorithms.

4.2.3.1 Study Design

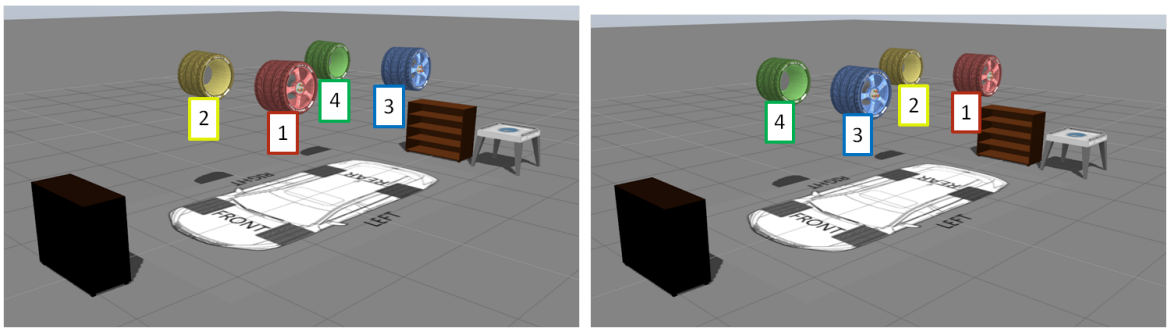
We conducted a between-subjects study to evaluate two conditions: a *No-Suggestions* condition, in which the robot provided no repetition or grouping suggestions, and a *Suggestions* condition, in which these suggestions were made when appropriate. Participants in both conditions were able to use the GUI to add hierarchy to their task themselves, as in Figure 8 (Page 17).

A total of 32 participants (18 female and 14 male) were recruited from the local community. Most of the participants (15 female and 8 male) did not have programming experience. 15 participants were assigned to the No-Suggestions condition and 17 to the Suggestions condition. The two conditions were balanced for gender and programming experience.

4.2.3.2 Task

The study consisted of a training activity (in a blocks world domain) followed by the main study based on tire rotation. All participants were given the same training activity to develop familiarity with the GUI and the underlying learning algorithms. Specifically, participants were given step-by-step written instructions for building a tower using six colored blocks and two primitive actions: Pickup and PutOn. Participants were led through a series of steps, beginning with simply moving blocks around, to teaching a new abstract task, to reusing already learned abstract tasks to build a tower. The training process did not include any robot suggestions, to avoid biasing participants in the No-Suggestions condition. All of the participants successfully completed the training steps and did not have any questions for the experimenter afterwards.

After training, participants performed the main study activity in which they were asked to teach the robot how to rotate the tires from the start configuration to the goal configuration shown in Figure 12. Participants were given detailed written descriptions of the six primitive actions described in Section 4.1 and a terminology picture similar to Figure 5 (Page 10), but



(a) Start configuration (b) Goal configuration

Figure 12: Start and goal configurations

no step-by-step instructions on how to perform tire rotation. The GUI used by participants in both conditions was identical. The only difference between the two conditions was whether or not repetition and grouping suggestions were provided.

4.2.3.3 Procedure

Upon arrival, participants were asked to sign an informed consent form and were surveyed about their programming experience. Following this, participants were given the written instructions for the training activity; the experimenter started the learning system and left the room until training was complete. Following training, the experimenter provided the written tire rotation instructions, started the system and again waited outside the room. Upon completion of the tire rotation task, participants were asked to fill out a questionnaire about their experience. Participants were then asked to perform the tire rotation task a second time (under the same study condition), followed by the same questionnaire.

We had participants perform two trials of the main study task because we anticipated that improved performance would occur as they gained experience with the GUI. As we will see in the results below, we did observe this effect.

4.2.3.4 Measures and Analysis

We used the following three objective measures to evaluate participant performance:

- *Teaching Effort:* The effort expended by the human, measured as the number of times the teacher pressed the Execute button during the interaction (for both primitive and abstract tasks). We think of this as the number of task demonstrations communicated by the human to the robot.

- *Plan Quality:* The quality of the learned tire rotation plan, measured as the number of abstract tasks learned (and used). We choose this measure because hierarchy is a valuable property in complex plans for both communication and reuse.

- *Teaching Efficiency:* The efficiency of the interaction, measured by dividing the teaching effort by the plan quality (using the definitions above).

The two-tailed Kolmogorov-Smirnov test was used for the evaluation of these measures.

4.2.3.5 Hypotheses

We formed the following five hypotheses about the effect of suggestions on the learning interaction based on the models we presented earlier and findings from human-computer and human-robot interaction studies.

- **Hypothesis 1:** Participants will expend less teaching effort in the Suggestions condition than in the No-Suggestions condition.

- **Hypothesis 2:** The quality of the learned plans will be higher in the Suggestions condition than in the No-Suggestions condition.

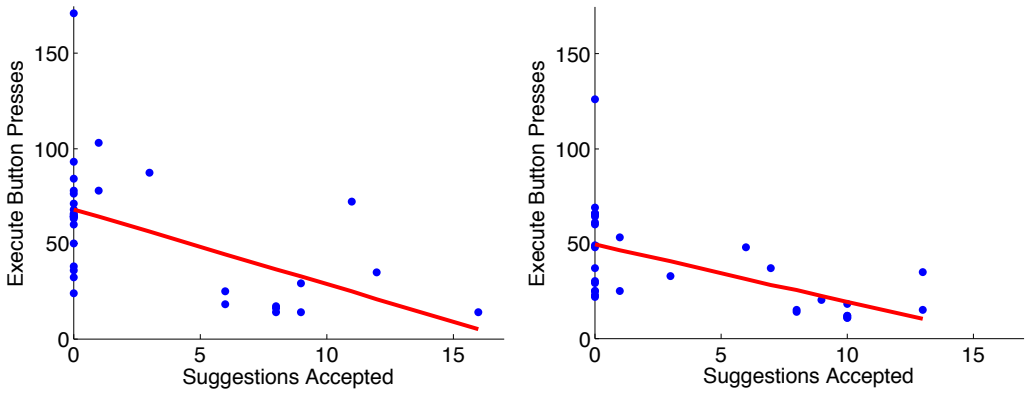
- **Hypothesis 3:** The teaching efficiency will be greater in the Suggestions condition than in the No-Suggestions condition.

- **Hypothesis 4:** In the Suggestions condition, participants who accepted more suggestions will have better measures of teaching effort, plan quality and teaching efficiency.

- **Hypothesis 5:** All participants in both conditions will be able to successfully teach the tire rotation task by the second trial.

$Mean \pm SD$	Execute Button Presses (First Trial)	Execute Button Presses (Second Trial)	Abstract Tasks Learned (First Trial)	Abstract Tasks Learned (Second Trial)	Efficiency (First Trial)	Efficiency (Second Trial)
No-suggestions	62.07 ± 19.10	51.40 ± 27.22	2.87 ± 2.83	2.47 ± 1.77	0.04 ± 0.03	0.06 ± 0.06
Suggestions	50.47 ± 42.68	26.65 ± 16.42	7.94 ± 5.48	8.53 ± 3.92	0.31 ± 0.31	0.41 ± 0.30
p-value	0.05	0.01	0.02	$\ll 0.001$	$\ll 0.001$	0.001

Table 1: Comparison of objective measures between the two conditions.



(a) First trial

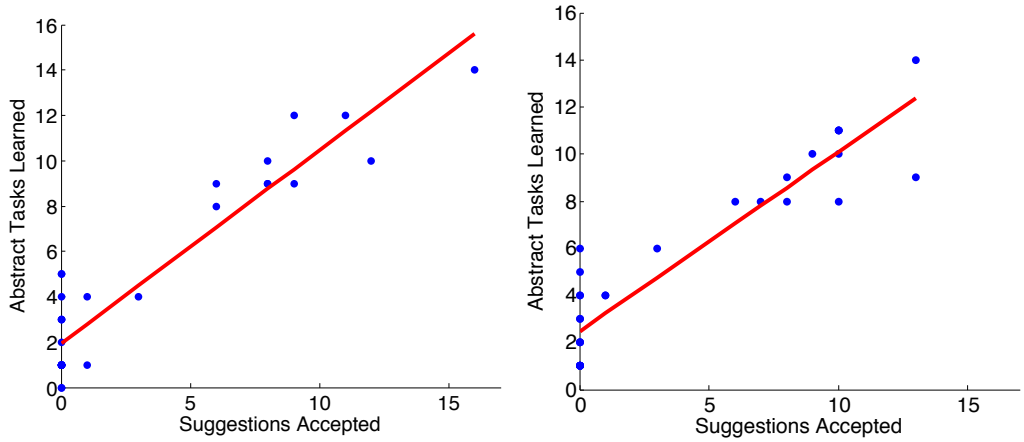
(b) Second trial

Figure 13: Teaching effort as function of suggestions.

4.2.3.6 Results and Discussion

Table 1 presents a summary of the results of our study, which are discussed in detail below.

Teaching Effort: Analysis of the number of Execute button presses during the training process shows that the teaching effort was significantly higher in the No-Suggestions than in the Suggestions condition, supporting Hypothesis 1. Notice that the p-value in the second trial is lower than in the first trial due to the large reduction in Execute button presses by participants in the Suggestions condition during the second trial. We attribute this difference to participants adapting to the robot's suggestions. A reduction in effort was also observed in the No-Suggestions condition, but the change is much smaller. To further highlight the impact of suggestions on teaching effort, Figure 13 shows via linear regression how the number of Execute button presses decreases with the number of robot suggestions accepted by the user, supporting Hypothesis 4.



(a) First trial

(b) Second trial

Figure 14: Plan quality as function of suggestions.

Plan Quality: Analysis of the number of abstract tasks learned shows that the quality of plans was significantly higher in the Suggestions than in the No-Suggestions condition, supporting Hypothesis 2. Additionally, Figure 14 shows via linear regression how the number of abstract tasks learned increases with the number of robot suggestions accepted by the uses, further supporting Hypothesis 4.

Teaching Efficiency: Analysis of the efficiency measure provides insight into how teaching effort and plan quality vary in combination between conditions. Teaching efficiency (see last column of Table 1) is significantly higher in the Suggestions than in the No-Suggestions condition, supporting Hypothesis 3. Additionally, we observe greater improvement in efficiency between the first trial and the second trial for participants in the Suggestions condition.

Hypothesis 5 predicted that by the second trial all participants would successfully teach the tire rotation task (i.e., teach a plan that achieves the goal configuration starting from the start configuration specified in Figure 12 on Page 24). Unfortunately, this hypothesis was not supported by our study. We only found that most, but not all, plans learned from participants were correct. Several participants in the first trial (5/15 in No-Suggestions and 3/17 in Suggestions) and in the second trial (3/15 in No-Suggestions and 2/17 in Suggestions) left the world state in the goal configuration at the end of their teaching session, but the

learned plan would not successfully achieve the goal configuration starting in the specified start configuration.

Finally, it is interesting to note that in informal post-study debriefing several participants in the Suggestions condition commented that they initially rejected the robot's suggestions because they were worried about the effect of saying yes would be. This was an unintended side effect of our study design, in which we sought to eliminate bias by following the same training procedure in both conditions. Nevertheless, the Suggestions condition showed clear advantages over No-Suggestions despite this limitation, and we expect better results could have been achieved with a training process tailored for the Suggestions condition.

In summary, Hypotheses 1–4 were supported, and Hypothesis 5 was not.

4.3 Learning from a Single Example Using the Top-Down Style

The user study described in Section 4.2.3 shows that the plan quality and the teaching effort increased significantly using robot's suggestions. Although the results of this user study show a significant improvement, interacting with our system was not very easy for some users. We speculate that one of the reasons for this problem was that the system supported the bottom-up teaching style only. Therefore, enabling the system to support the top-down teaching style might improves the system's intractability. To address this, we enable the users to teach tasks using the top-down teaching style.

In this section we will explain how our system learns a hierarchical structure of the tire rotation HTN similar to the HTN in Figure 3 (Page 8) using only instructions. This ideal interaction is based on a top-down approach, in which abstract tasks are first defined and primitive actions are used in the last stages to define lowest level abstract tasks. In Section 4.4, we give an explanation of the hybrid approach that takes advantage of both the bottom-up teaching style and the top-down teaching style. We will clarify why this hybrid approach is more natural by giving an example that utilizes both teaching styles.

Figure 15 shows the graphical user interface we developed that supports both the top-down and the bottom-up teaching styles. In order to make this user interface closer to the

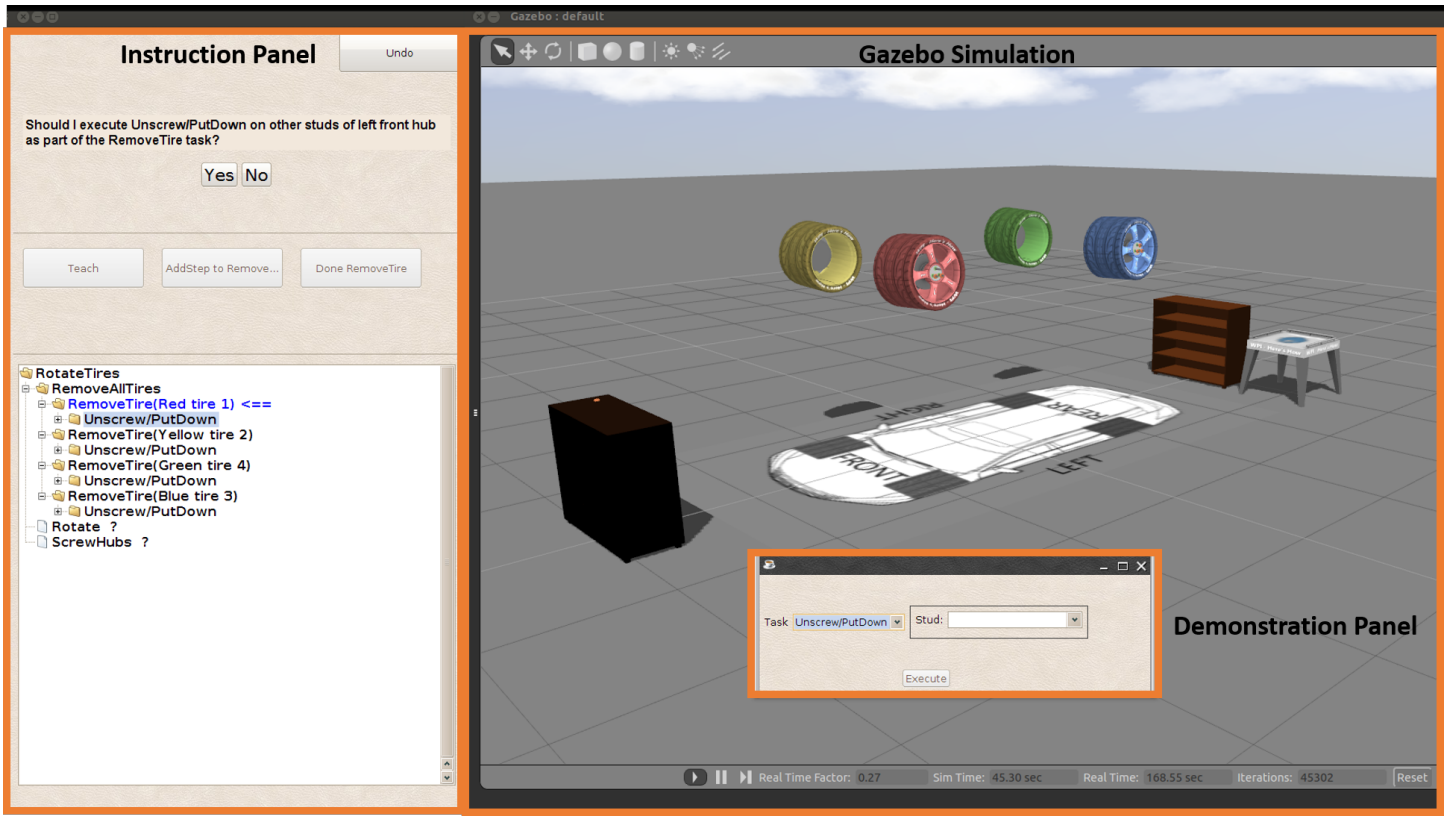


Figure 15: User interface for both the bottom-up and the top-down teaching styles.

situated interaction, we have separated the instruction and demonstration part of the user interface. Note that in this section we only use the instruction part of the GUI and the entire GUI will be used in Section 4.4. The panel on the left (Instruction Panel), shows the instructions that can be given to the system using language. The panel on the right (Demonstration Panel) illustrates the demonstrations that can be given to the system by the user executing the primitive tasks.

The user can teach the hierarchy of tasks to the robot by pressing the Teach button in the middle area of the instruction panel to start teaching a new task and the Done button to end the started teaching session. After starting a new teaching session, the user can add a new step to the task by pressing the AddStep button and then adding a new step using available demonstrated tasks or defining a new task. The user can also add inputs to the new defined task using the AddInput button. Once an abstract task is defined with an input, this abstract task can be used multiple times with other inputs of the same type.

The top part of the instruction panel contains a "chat" window in which the robot asks questions and the user can reply. This is where the robot's helpful suggestions appear and where the user can accept or reject them.

The bottom area of the instruction panel is the main informational display, which shows the current hierarchical structure of the learned tasks. Similar to the previous user interface, the small arrow next to New Task A indicates the abstract task currently being learned. The question mark next to the task shows the task is not yet fully defined, because the task was given as an instruction without any body. If the user right clicks on any of the tasks in this area, another menu will appear with the option of deleting or renaming the tasks.

To execute a task, the user can select the appropriate task from the demonstration panel, specify its inputs and press Execute. When a primitive task is executed using the interface buttons or demonstrated in the real-world, it will be added to the information display part of the user interface.

4.3.1 An Example Using the Top-Down Style

Figure 16 shows an ideal interaction using the top-down teaching style. The user starts by teaching the `RotateTires` task by defining its three steps: `RemoveAllTires`, `Rotate`, and `ScrewHubs`. Then, the user teaches the `RemoveAllTires` task by adding `RemoveTire` as its first step. The system asks the user to add other `RemoveTire` tasks with different tires; the user accepts the suggestion and the system learns `RemoveAllTires` task.

On line 16, the user starts teaching the `UnscrewTire(Tire1)` task. The user first adds `UnscrewStud` with `LFhub.stud1` as its input. Guided by the *Part-whole* heuristic, the system asks to add `UnscrewStud` with other studs of left front hub to `UnscrewTire` task. The user accepts the suggestion and completes the task. This interaction continues similarly until the user teaches all the abstract tasks.

```
1   Teach RotateTires
2     AddStep RemoveAllTires()
3     AddStep Rotate()
4     AddStep ScrewHubs()
5   Task Complete
6   Teach RemoveAllTires
7     AddStep RemoveTire(Tire1)
8     Robot: "Should I execute RemoveTire on other tires   of the car?"
9     "Yes"
10  Task Complete
11  Teach RemoveTire
12    AddStep UnscrewTire(Tire1)
13    AddStep UnhangTire(Tire1)
14  Task Complete
15  Teach UnscrewTire
16    AddStep UnscrewStud(LFhub.stud1)
17    Robot: "Should I execute UnscrewStud on other studs   of left front hub?"
18    "Yes"
19  Task Complete
20  Teach UnscrewStud
21    AddStep Unscrew(LFhub.stud1)
22    AddStep PutDown(the nut)
23  Task Complete
24    Robot: "Do you want to continue executing UnscrewTire?"
25    "Yes"
26  Teach UnhangTire
27    AddStep Unhang(LFhub,Tire1)
28    AddStep PutDown(Tire1)
29  Task Complete
30    Robot: "Do you want to continue executing RemoveAllTires?"
31    "Yes"
32  ...
33  Teach ScrewStud
34    AddStep PickUp(a nut)
35    AddStep Screw(LFhub.stud1,the nut)
36  Task Complete
37    Robot: "Do you want to continue executing ScrewTires?"
38    "Yes"
```

Figure 16: A typical interaction to teach tire rotation using only the top-down teaching style.

The above example highlights another important property of our system related to the definition of task inputs. Note that in the example, the abstract `RemoveTire` task takes tires

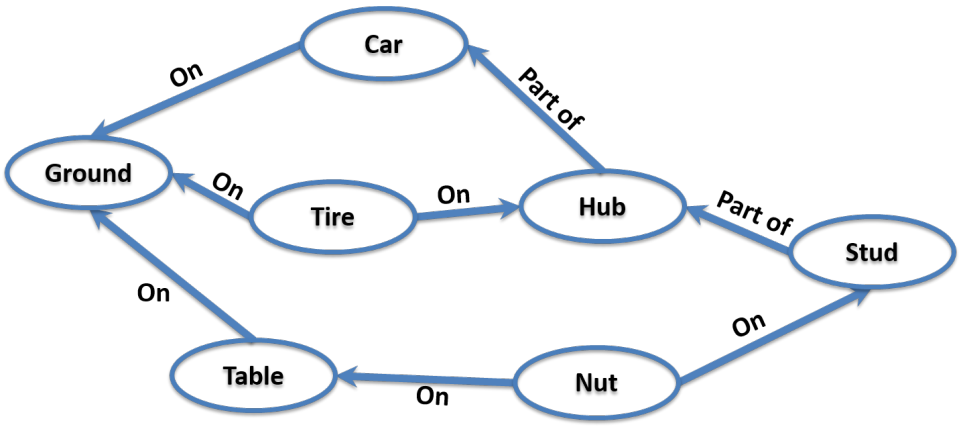


Figure 17: Semantic Relations between Object Types of Car Maintenance Domain.

as inputs, yet the primitive task used within the removal, `Unhang`, takes hubs as inputs. This issue is the result of the differences between the fixed primitive tasks and how people think about abstract tasks. We allow for this to provide a natural interaction for the users of our system. In the case that we only allow the bottom-up teaching style in Section 4.2, this was not a problem since the inputs would propagate up the HTN and the user was forced to think about the abstract tasks in terms of the propagated inputs; the input of `Unhang` would propagate up so the input of the `RemoveTire` task would be a hub as well. However, since we have the top-down teaching style here, there is no information about the inputs and the user should use her understanding of the task to define them. In the ideal interaction described above, the system is using semantic relations (described below) to find the relations between the inputs of abstract tasks and the inputs of primitive tasks and to learn the task definition.

4.3.2 Semantic Relations

In addition to the *Part-whole* relation which determines whether one object is a part of another object (e.g. `stud1` is part of `LFhub`, or studs are part of hubs) that we used in Section 4.2.2.1 to generate suggestions, we have identified another useful semantic relation: *On*. The *On* relation determines whether an instance of an object is on another instance of another object, or in the case that we do not have this information about the instances, it determines whether an object type is on another object type. For example, in the former we know that

`Tire1` is on `LFhub` (this information might not be available in a different world state) but in the latter we just know that tires go on hubs (this information is always available). Figure 17 shows the semantic relations between object types in the car maintenance domain. Given these relations, our system infers the connection between the inputs that the system inferred (we refer to these inputs as system inferred inputs) and the inputs defined by the user.

When the user is done teaching a task, the system checks if the inputs that the user has specified differ from the system inferred inputs. In the case that they are different, if the user inputs are inferred by a combination of the *On* and the *Part-whole* relations, the relations are considered as the mapping between the the user inputs and system inferred inputs. If there is no mapping between the inputs, or there is more than one sequence of relations that results in the same mapping, the system reports failure. Handling failures in the inference will be analyzed in more details in our future work.

An example of the described inference process can be seen in the `UnscrewTire` task, which has 3 inputs (`LFhub.stud1`, `LFhub.stud2`, and `LFhub.stud3`) that differ from the `Tire1` input provided by the user. The system recognizes the difference between the user input and the system inferred inputs and applies the semantic relations on the system inferred inputs. Applying *Part-whole* relation on `LFhub.stud1`, `LFhub.stud2`, and `LFhub.stud3` inputs results in `LFhub`, and then applying the *On* relation on the `LFhub` results in an object of type Tire. Since this is the only relation found between the user inputs and the system inferred inputs, the system successfully learns the task definition.

4.4 Integration of the Bottom-Up and the Top-Down Styles

Now, it is the time to consider mixing the bottom-up and the top-down teaching styles. We believe that a natural human-robot interaction is neither purely bottom-up nor top-down, since humans naturally switch between these modes of interaction while teaching other humans. Below, we will see an example of a natural interaction scenario using both the top-down and the bottom-up teaching styles. This interaction is just one of many natural interactions that uses both instructions and demonstrations.

4.4.1 An Example Using Both the Bottom-Up and the Top-Down Styles

Figure 18 shows an example interaction that results in the system learning the hierarchical structure similar to Figure 3 (Page 8). In this interaction, higher level abstract tasks are taught using the top-down style and lower level abstract tasks are taught using the bottom-up style. We hypothesize that to teach the tire rotation HTN, humans prefer to explain higher level tasks by giving instructions; however, they prefer to demonstrate lower level tasks.

Similar to the typical interaction explained in Section 4.3.1, in lines 1–10 the user teaches the tasks using only instructions. Then, in lines 11–22, the user uses a similar approach as 4.2.2 and teaches a previously defined task called `RemoveTire` using demonstrations. This interaction continues until all steps of tire rotation task are defined.

1 Teach RotateTires
2 **AddStep** RemoveAllTires()
3 **AddStep** Rotate()
4 **AddStep** ScrewHubs()
5 Task Complete
6 Teach RemoveAllTires
7 **AddStep** RemoveTire(Tire1)
8 *Robot: "Should I execute RemoveTire on other tires of the car?"*
9 "Yes"
10 Task Complete
11 Teach RemoveTire(Tire1)
12 **Execute** Unscrew(LFhub.stud1)
13 **Execute** PutDown(the nut)
14 *Robot: "Do you want to group the highlighted steps (Unscrew and PutDown) into a new subtask?"*
15 "Yes"
16 *Robot: "Should I execute Unscrew/PutDown on other studs of left front hub?"*
17 "Yes"
18 **Execute** Unhang(LFhub,Tire1)
19 **Execute** PutDown(Tire1)
20 *Robot: "Do you want to group the highlighted steps (Unhang and PutDown) into a new subtask?"*
21 "Yes"
22 Task Complete
23 *Robot: "Do you want to continue executing RemoveAllTires?"*
24 "Yes"
25 ...
26 Teach ScrewHubs()
27 **Execute** PickUp(a nut)
28 **Execute** Screw(LFhub.stud1,the nut)
29 *Robot: "Do you want to group the highlighted steps (PickUp and Screw) into a new subtask?"*
30 "Yes"
31 *Robot: "Should I execute PickUp/Screw on other studs of left front hub?"*
32 "Yes"
33 *Robot: "Should I execute PickUp/Screw*3 on other hubs of the car?"*
34 "Yes"
35 Task Complete

Figure 18: A typical interaction to teach tire rotation using both the bottom-up and the top-down teaching styles.

4.4.2 Evaluation

We conducted a pilot study with 4 computer science students to evaluate the algorithms discussed in in this section. For this pilot study, we used the graphical user interface described in Section 4.3.

4.4.2.1 Procedure

Upon arrival, participants were given the written instructions for the training activity. The training activity for this pilot study is similar to the training activity of the previous user study with one difference, users were also taught to teach the tasks using the top-down teaching style. Similar to the previous training activity, participants were led through a series of steps, beginning with simply moving blocks around, to providing instructions for the `Make Tower` task using two instances of `Stack Block` task and then they demonstrated the `Stack Block` task using `PickUp` and `PutOn` primitive actions.

4.4.2.2 Results and Discussion

The participants in the first two pilot studies could not teach the tire rotation task completely. In the first pilot study with participant1 (p1), p1 reported some bugs in the user interface. In the second pilot study, participant2 reported another bug related to the algorithms, caused by the fact that we did not differentiate between nuts in the world. Both participants used the top-down teaching style as well as the bottom-up teaching style. We fixed the user interface's bugs and we continued the pilot study with two more students.

Participant3 could teach the tire rotation task completely, but he didn't take advantage of the top-down teaching style. He preferred to see that the actions are being executed in the world to ensure that he is teaching the task in a right way. He also mentioned that the robot should be more active in this process to assure the teacher that it is learning the tasks (i.e. the robot should give more feedbacks to the user)

Participant4 was also able to teach the tire rotation task but he reported that providing inputs for the execution instructions is a really complicated process and at some points he

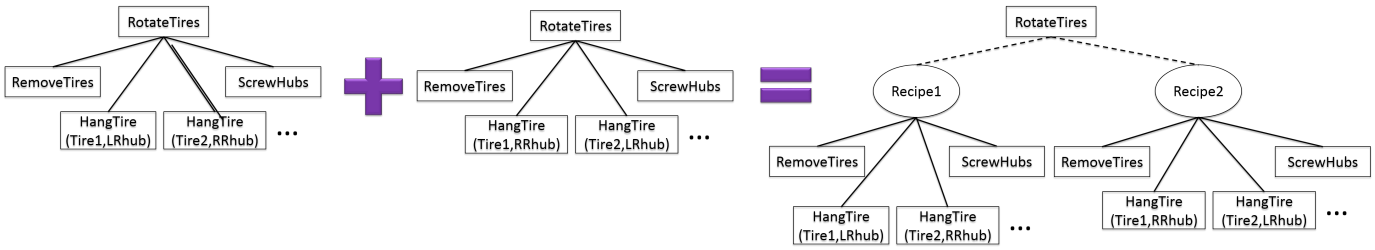


Figure 19: An example of how alternative recipes are added to an HTN if we do not use the merging algorithm described in Section 5.3 (for simplicity, temporal constraints' arrows are not shown).

was asking if he should also add the nuts as the inputs of the task. He started to teach the tire rotation task using instructions by defining one abstract task for each tire without noticing that he can use the same task for other tires. In addition, he forgot that he was teaching RemoveTire task and he demonstrated the whole tire rotation task as the steps of RemoveTire task.

In conclusion, the users tend to use the top-down teaching style in addition to the bottom-up teaching style. However, our current approach for learning tasks using the top-down teaching style is very complicated and not natural for the users. Using voice commands instead of the current GUI might provide a more natural interaction for the users. Furthermore, giving more feedbacks to the users assures them that the robot understands their commands. We will address these issues in our future work.

5 Learning from Multiple Examples

In this section, we first discuss why dealing with multiple examples is necessary by giving an example in the tire rotation task. We then explain our algorithm to merge multiple examples.

5.1 Why learning from multiple examples is necessary?

Having alternative recipes for tasks are inevitable in any task domain. This necessitate dealing with multiple examples of a task execution. In this section, we provide an algorithm

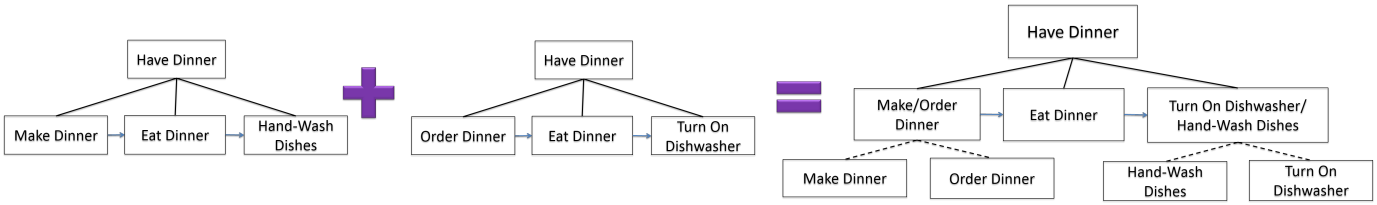


Figure 20: An example of how alternative recipes are added to an HTN using the merging algorithm (alternative recipes are shown with dashed lines).

to merge multiple demonstrations for the same task to allow for generalization across the two or more examples. This approach has three advantages. First, this avoids adding a separate recipe for each tiny difference between the new demonstration for a specific task and the previously learned model. Second, by merging different demonstrations and factoring the common steps between them, we are postponing the system's needs to choose between the recipes, resulting in a more robust system. Third, this reduces the number of demonstrations required to learn the task. In the following subsection, we will explain with an example why merging algorithms are necessary.

5.2 An Example

Assume that we have a user who wants to teach `RotateTires` task. The user demonstrates `RemoveAllTires` task as before, then the user demonstrates and executes four `HangTire` tasks to hang the tires using front-to-rear rotation pattern and then the user demonstrates `ScrewHubs` task. Now, the `RotateTires` task includes six tasks: `RemoveAllTires`, four `HangTire`, and `ScrewHubs` tasks. After teaching the `RotateTires` task based on front-to-rear pattern, the user decides to also demonstrate `RotateTires` task using xPattern. The user again demonstrates `RemoveAllTires` task, then she demonstrates four `HangTire` tasks, and then she demonstrates `ScrewHubs` task. The second demonstration of `RotateTires` task also has six task executions. Figure 19 shows how these two demonstrations are added to an HTN if we do not use the merging algorithm described in the following subsection, and Figure 21 shows the results of applying the merging algorithm. If you compare the HTN in Figure 19 with the HTN in Figure 21, the HTN's definition in Figure 21 does not have

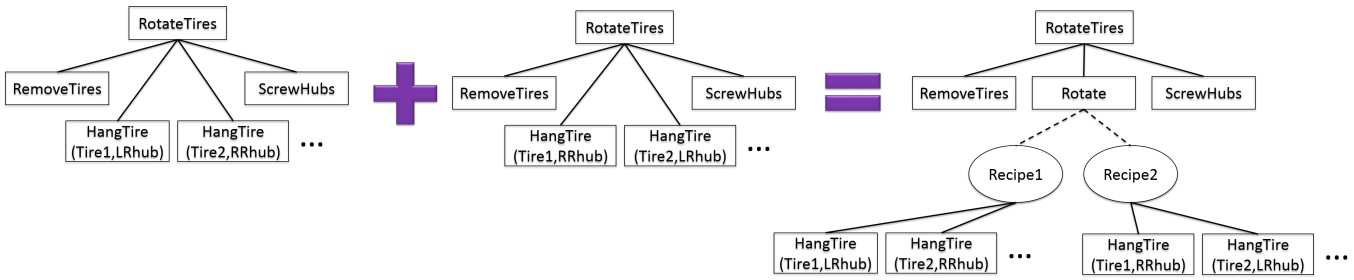


Figure 21: An example of applying the merge algorithm to RotateTires task (for simplicity, temporal constraints' arrows are not shown).

two redundant `RemoveAllTires` and `ScrewHubs` tasks since four `HangTire` executions are grouped into a higher level task (hypothetically, called `Rotate`). Furthermore, at the time of execution, the HTN planner will ask for the preferred rotation pattern after removing the tires (at `Rotate` task, not `RotateTires` task).

This example, illustrates why it is important for our system to merge multiple demonstrations. Please note that this example does not illustrate how our merging algorithm reduces the number of demonstrations required to learn the task. Assume that we have two demonstrations for `Have Dinner` task, the first demonstration (D1) includes: `Make Dinner`, `Eat Dinner`, and `Hand-Wash Dishes` and the second demonstration (D2) includes: `Order Dinner`, `Eat Dinner`, and `Turn On Dishwasher`. Merging these two sequence of tasks results in an HTN depicted in Figure 20. In addition to the previous demonstrations (D1 and D2), we can also infer two more demonstrations from this HTN, D3: `Make Dinner`, `Eat Dinner`, and `Turn On Dishwasher` and D4: `Order Dinner`, `Eat Dinner`, and `Hand-Wash Dishes`. Therefore, the proposed merging algorithm reduces the number of demonstrations required from the human teacher. In the following subsection, we explain in more details how our merging algorithm works. This merging algorithm can also be used for merging multiple instructions, or merging instructions with demonstrations.

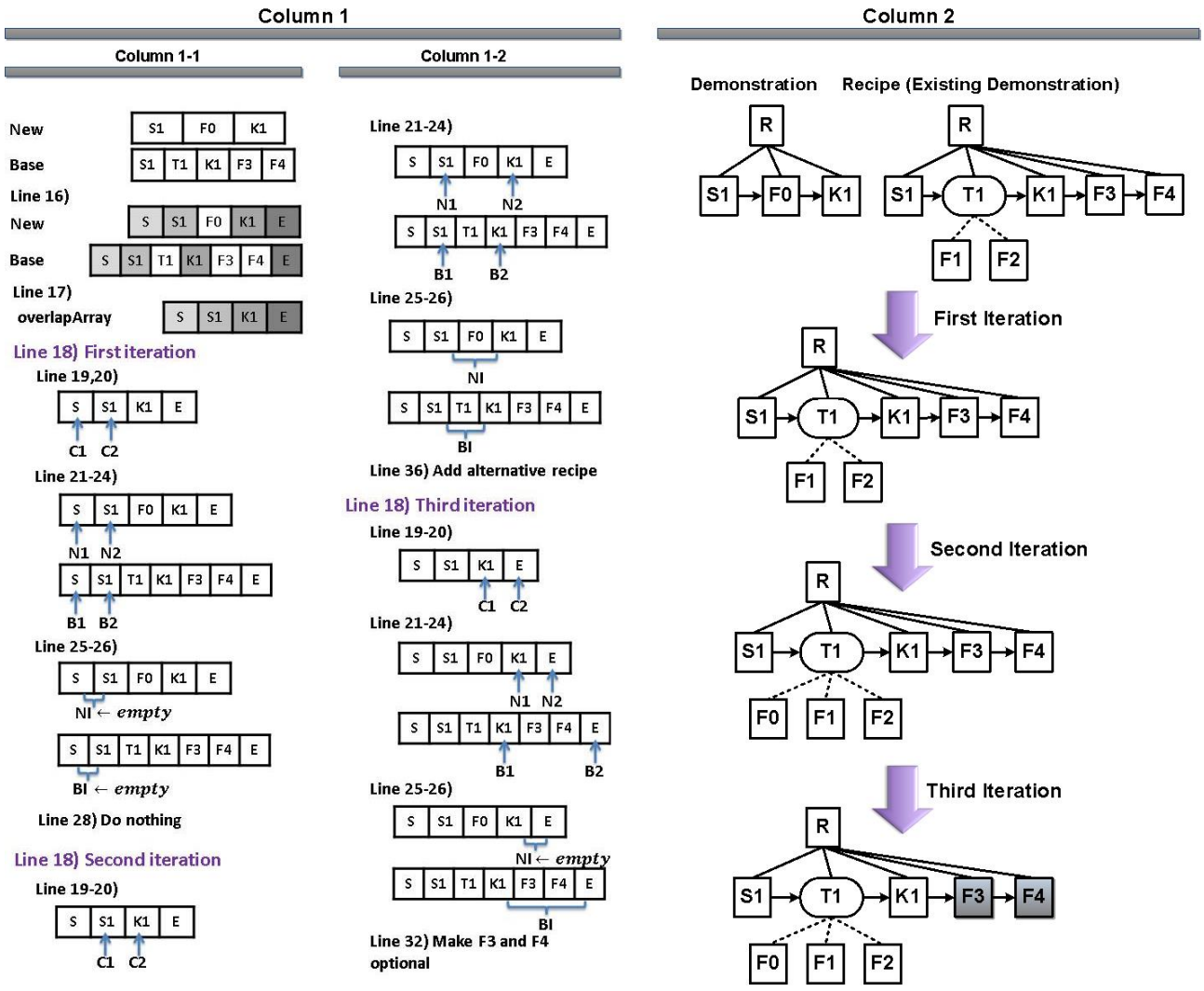


Figure 22: An example for the execution of the `MERGE` function in Algorithm 1 is shown in column 1. Column 2 shows the changed HTN after each iteration in column 1 (in column 2, the oval with dashed lines attached to it shows alternative recipes for that specific task and gray boxes show optional steps).

5.3 Merging Algorithm

In this algorithm, once the current demonstration sequence is encoded as an HTN using the above methods, we check whether any previous demonstrations exist for this task. If this is the only demonstration, then the HTN remains unchanged and the new task will be added to the task model. However, if there exists another demonstration for the demonstrated task,

we apply the merging algorithm on the previous task demonstration and the new one. Figure 21 shows an example (described in the previous subsection) of merging two demonstrations in tire rotation task. In this example, instead of having two recipes for `RotateTires` task, we capture their common steps by adding a new, higher level task `Rotate` (the task name is specified by the user).

The pseudocode for the merging algorithm is shown in Algorithm 1. First, the `MERGETASKS` function is called with current model (existing task) and the new demonstration for the task. If the new demonstration is satisfiable by the current model, there will be no changes in the model and the merge ends (lines 2-3). If the new demonstration is not mergeable with the current model, the new demonstration will be added as an alternative recipe of the current model (lines 6-7). Otherwise, if the models are mergeable, the new demonstration will be combined with current model using the `MERGE` function (lines 4-5).

Mergeability is determined by the `MERGEABLE` function through two steps: `GetRootRecipes` and `FindMaxLCS`. The `GetRootRecipes` function returns all possible recipes of the model (e.g., if this function is called on the `Rotate` task in Figure 3, it will return the steps of `xPattern` recipe, and the steps of `frontRear` recipe.). Then the `FindMaxLCS` function is called to determine which recipe shares more common steps with the new demonstration by using Longest Common Subsequence (LCS) algorithm (line 12). If there are no shared steps between any of the recipes and the new demonstration, the model and the demonstration are not mergeable. Otherwise the recipe with the most shared steps will be chosen to be merged with the new demonstration.

To merge the LCS recipe with the new demonstration, the `MERGE` function iteratively selects two pairs of shared steps in the two HTNs, and merges the steps between them. This process is illustrated in Figure 22. First, given the new demonstration and the base recipe, we add `start` and `end` steps to each (line 16). Next, we calculate the overlap (using LCS) between the `base` and the `new` (line 17). Then, for each pair of consecutive steps in the `overlapArray`, we find the corresponding steps in the `base` and `new` (lines 21-26) (note that `BI` and `NI` are the steps between pairs of equivalent consecutive steps in `base` and `new`,

Algorithm 1 Merge algorithm

```
1: function MergeTasks(model, demonstration)
2:     if Statisfiable(model, demonstration) then
3:         return
4:     else if (maxLCS ← Mergeable(model, demonstration)) ≠ NULL then
5:         Merge(maxLCS, demonstration)
6:     else
7:         AddAlternativeRecipe(model,demonstration)
8:     end if
9: end function

10: function Mergeable(model, demonstration)
11:     modelRootSeqs ← GetRootRecipes(model)
12:     maxLCS ← FindMaxLCS(modelRootSeqs,
    demonstration)
13:     return maxLCS
14: end function

15: function Merge(base, new)          ▷ base is the chosen recipe, and new is the new demonstration
16:     Add start and end to base and new
17:     overlapArray ← Overlap(base,new)
18:     for Pi ∈ overlapArray do
19:         C1 ← P_i in overlapArray
20:         C2 ← P_{i+1} in overlapArray
21:         B1 ← find pointer to C1 in base
22:         B2 ← find pointer to C2 in base
23:         N1 ← find pointer to C1 in new
24:         N2 ← find pointer to C2 in new
25:         BI ← steps between B1 and B2
26:         NI ← steps between N1 and N2
27:         if BI = empty ∧ NI = empty then
28:             Do nothing
29:         else if BI = empty ∧ NI ≠ empty then
30:             modelRecipe ← AddOptional(base,BI)
31:         else if BI ≠ empty ∧ NI = empty then
32:             modelRecipe ← MakeOptional(base,NI)
33:         else
34:             modelRecipe ← MakeNewRecipe(BI)
35:             demRecipe ← MakeNewRecipe(NI)
36:             AddAlternativeRecipe(modelRecipe,demRecipe)
37:         end if
38:     end for
39: end function
```

respectively). If `BI` and `NI` are empty, we will continue by choosing the next equivalent consecutive steps (lines 27-28). If `NI` is not empty and `BI` is empty, we will add `NI` steps to the model but we will mark them as optional steps (line 30), but if `BI` is not empty and `NI` is empty, we will make `BI` steps in model as optional (line 32). If neither of them is empty, we replace `BI` with a new task, and `BI` and `NI` become alternative recipes in the new task (lines 34-36).

Importantly, the merge operation is not commutative, and thus changing the order in which demonstrations are performed will change the hierarchical structure of the HTN. However, the execution of the primitive actions within the model is preserved, regardless of demonstration order, ensuring correct execution of the task regardless of demonstration order. The evaluation of this method is parts of our future work.

6 Future Work

Our main next step is to move the human-robot interaction from simulation into a shared physical space using a PR2 robot and an approximately life-size plastic and wood mockup of the relevant parts of a car. To circumvent the need for computer vision processing, we will locate the user, robot and mockup inside of a Vicon motion-capture cell and attach markers to all elements as needed. This change will make it possible to move away from the GUI and towards a more natural and situated interaction. If the GUI does not totally disappear in this context, it will be moved to a hand-held touch screen. To start, instead of pressing a primitive task button, selecting the input object(s) from a menu, and then pressing the Execute button, the user will be able to physically demonstrate a primitive task by picking up, putting down, screwing, unscrewing objects, etc. With only a small number of primitive task types, we are currently able to recognize which primitive action is being performed and its inputs.

We noticed in our user studies that it is not natural for the users to use our predefined primitive tasks. To address this problem, we are working on learning the primitive actions from scratch both at symbolic level (HTN) and motion planning level (learning from the

human trajectories). At the symbolic level, the system automatically segments reusable primitives from narrated demonstrations and sends the segmented human trajectory (just one segment) to a component at the motion planning level. At the motion planning level, the constraints of the human trajectory are extracted and generalized. These constraints are represented as Task Space Regions (TSRs) and are given to a robot planner to be executed by the robot.

For learned abstract actions, the GUI (or perhaps voice recognition) will need to be used to identify the action, but the selection of input(s) will be done by gesture, such as pointing to or touching the appropriate object(s). We will study how these changes affect the learning process, which will inform how other communication functions of the GUI, such as the Teach New Task and Task Complete buttons, the chat pane and the display of the current hierarchical structure should be handled. We speculate that voice commands might be good replacements for those buttons (we do not want to deal with natural language processing).

In a more theoretical vein, our next steps include exploring how to extend our algorithms to support the flexible combination of demonstrations and instructions using the bottom-up and the top-down teaching styles. We also plan to explore both improvements to our existing heuristic rules and new heuristics for making helpful suggestions, especially as we extend our task domain beyond tire rotation. Finding other useful relations and making suggestions (by applying heuristics) based on the output of relations is another area that we have not explored yet.

Finally, working with a physical robot will inevitably present us with the challenge and opportunity to explore the very broad issue of how uncertainty and failure in robotic systems affects the learning process.

7 Conclusion

In this paper, we presented an approach that integrates HTNs and collaborative discourse theory into learning from demonstration and instruction paradigm to enable robots to learn

complex procedural tasks from human users through mixed-initiative interaction. We have evaluated our algorithms to learn using the bottom-up teaching style with users in a simulated environment and shown both that the overall approach is usable and that the suggestions significantly improve the learning and interaction. We noticed that not having the top-down teaching style in the learning process degraded the efficiency and flexibility of the interaction. We evaluated our new system with four users using both the bottom-up and the top-down teaching styles in a pilot study and we noticed that the users are able to teach a task using both teaching styles but the interaction is not sufficiently natural for them. We speculate amending our system to leverage natural language in a situated interaction will greatly improve the users' ability to use the top-down teaching style. The ultimate goal of our project is to move the human-robot interaction from simulation into a shared physical space using a PR2 robot and we are currently working on making this happen. We also provided an algorithm for merging multiple examples of the same task; we will evaluate this algorithm in our future work.

References

[1] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[2] Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *ACM/IEEE International Conference on Human-Robot Interaction*, pages 17–24. ACM, 2012.

[3] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1, 2009.

[4] Andrew Garland, Kathy Ryall, and Charles Rich. Learning hierarchical task models by defining and refining examples. In *International Conference on Knowledge Capture*, pages 44–51, 2001.

[5] Barbara J. Grosz and Candace L. Sidner. Attention, intentions, and the structure of discourse. *Comput. Linguist.*, 12(3):175–204, July 1986.

[6] Bradley Hayes. Social hierarchical learning. In *Proceedings of the 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI 2013) Pioneers Workshop*.

[7] Scott B. Huffman and John E. Laird. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271–324, 1995.

[8] Shiwali Mohan and John E Laird. Towards situated, interactive, instructable agents in a cognitive architecture. In *AAAI Fall Symposium Series*, 2011.

[9] Shiwali Mohan and John E Laird. Exploring mixed-initiative interaction for learning with situated instruction in cognitive agents. In *AAAI*, 2012.

[10] Monica N Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *AAMAS*, pages 241–248, 2003.

[11] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 5239–5246, 2012.

[12] Charles Rich. Building task-based user interfaces with ANSI/CEA-2018. *IEEE Computer*, 42(8):20–27, 2009.

[13] Charles Rich, Candace L Sidner, and Neal Lesh. Collagen: applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15, 2001.

[14] Paul E Rybski, Kevin Yoon, Jeremy Stolarz, and Manuela M Veloso. Interactive robot task training through dialog and demonstration. In *ACM/IEEE Int. Conf. on Human-Robot Interaction*, pages 49–56, 2007.

[15] Lanbo She, Shaohua Yang, Yu Cheng, Yunyi Jia, Joyce Y Chai, and Ning Xi. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *15th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, page 89, 2014.

[16] Harini Veeraraghavan and Manuela Veloso. Learning task specific plans through sound and visually interpretable demonstrations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2599–2604, 2008.