University of Nevada, Reno

Generalized Task Structure Learning for Collaborative Multi-Robot/Human-Robot Task Allocation

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computer Science and Engineering

by
Janelle J. Blankenburg

Dr. David Feil-Seifer/Dissertation Advisor

May 2020

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

JANELLE JUNE BLANKENBURG

Entitled

Generalized Task Structure Learning for Collaborative Multi-Robot/Human-Robot Task Allocation

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

David Feil-Seifer, Advisor

Monica Nicolescu, Committee Member

Mircea Nicolescu, Committee Member

Hung La, Committee Member

Anna Panorska, Graduate School Representative

David W. Zeh, Ph. D., Dean, Graduate School
May, 2020

Abstract

by Janelle Blankenburg

The basis of this work is a control architecture for collaborative multi-robot systems focusing on the problem of task allocation under hierarchical constraints imposed upon a joint task. For these types of tasks, multiple robots need to dynamically coordinate their execution during the task execution. However, due to the different abilities between different robots, a single task definition is not sufficient to ensure the task can be completely without faults. In order to alleviate these concerns, we have developed a generalized task structure which is able to transfer skills of a learned task to teams of heterogeneous robots. This system uses a small number of human demonstrations to learn a hierarchical task structure on a single robot. This structure acts as a skeleton for the task which has been adapted to work on teams of heterogeneous robots through the development of a continuous-valued metric which is able to account for the robots' variable skills during the task execution. Additionally, to further emphasize the collaboration of the multi-robot team, our previous architecture is extended to include an interdependence constraint which requires explicit cooperation between agents to complete the task.

Furthermore, to allow the task execution as well as the learning of the task structure to be as generalizable as possible, several efforts were made to extend the previous architecture to work with human-robot teams. First, a system was created to learn the hierarchical tasks through verbal instruction. Second, a dialogue-based fault recovery

system was developed to allow for a more robust task execution. Lastly, an intent recognition system was incorporated into the architecture to allow for human-robot teams to work collaboratively on a task.

Each of these extensions were validated separately through several experiments utilizing either multi-robot or human-robot teams for pick and place tasks with hierarchical constraints. The experiments included different environmental conditions in order to show the robustness of the proposed extensions to the control architecture. Combining each of these extensions together results in a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for both multi-robot and human-robot teams.

Acknowledgements

I would like to thank Dr. David Feil-Seifer, Dr. Monica Nicolescu, Dr. Mircea Nicolesu, Dr. Hung La, and Dr. Anna Panorska for being on my committee, with special thanks to Dr. David Feil-Seifer for giving me the opportunity to complete this research as part of the Socially Assistive Robotics Group (SARG) in the Robotics Research Lab (RRL). I would also like to thank my lab-mates for their help with this research: Bashira Akter Anima, Natalie Arnold, Santosh Balajee Banisetty, Eloisa Burton, Muhammed Tawfiq Chowdhury, Luke Fraser, S. Pourya Hoseini A., Thor Monteverde, Andrew Palmer, Nathaniel Rose, Stephen Michael Simmons, Gabrielle Talavera. I would especially like to thank Mariya Zagainova and Matthew the teddy bear for their help with this research. Without the assistance from Mariya and comfort from Matthew, neither the results of this work nor my sanity would exist. Lastly, I would like to thank my friends and family for helping to support me through this hectic time in my life. Special thanks goes to Becky Blankenburg, Rachel Ventayen, Megan Dominguez, and Scott Tello for always being there for me when I needed it.

This material is based in part upon work supported by: The Office of Naval Research under grant number(s) #N00014-16-1-2312 and #N00014-14-1-0776. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

Contents

List of Figures

Chapter 1

Introduction

The basis of this work is a control architecture for collaborative multi-robot/human-robot systems, focusing on the problem of task allocation under hierarchical constraints imposed on a joint task. Real-world tasks are not only a series of sequential steps, but typically exhibit a combination of multiple types of constraints, where some parts of the task are sequential, some have no ordering constraints, and others allow for alternative paths of execution. Therefore, to enable multi-robot and human-robot teams to complete joint tasks in the real world, the design of a generalized hierarchical control architecture which is able to encompass all of these types of constraints is necessary. One primary example which illustrates this concept is a building task. In order to correctly build a piece of furniture, certain parts have to be connected first whereas others can be attached at various points in the process.

These tasks pose significant challenges even in the single robot domain, as enumerating all the possible ways in which the task can be performed can lead to very large representations and keeping track of the task constraints during execution is not trivial. In previous work [1, 2] we developed an architecture that provides a compact encoding of such tasks and validated it in a single robot domain. For more complex building tasks however, multiple agents are needed to attach parts together as they require more than one part to be manipulated at a time.

A central problem for task execution in multi-robot teams is the allocation of robots to task(s). For building tasks, each agent must be allocated tasks to ensure they can work together to build the entire structure. Finding an optimal solution to this problem is an instance of the MT-MR problem (Multi-Task robots performing Multi-Robot tasks) as defined in [3]. In order to tackle this problem, we extended our previous architecture to work in the multi-robot domain [4]. This architecture utilizes a distributed message passing system for communication between the robots where each robot maintains a compact task encoding based off the encoding used in the single robot domain. Extending this work to the multi-robot domain provided the following contributions: i) it allows for on-line, dynamic allocation of robots to various steps of the task, ii) it ensures that the collaborative robot system will obey all of the task constraints and iii) it allows for opportunistic, flexible task execution given different environmental conditions.

The aim of the proposed work is to develop a generalized task structure which enables

collaborative task allocation for complex, hierarchical tasks for both multi-robot and human-robot teams. Therefore, the proposed extensions of this work fall into two categories: multi-robot capabilities and human-robot capabilities. The relevance of these extensions to the field are discussed in Sections 1.1 and 1.2. The status of the field for generalized task structure learning is presented in Section 1.3. Lastly, the proposed contributions are provided in Section 1.4.

1.1 Multi-Robot Task Allocation

The proposed contributions for the multi-robot domain focuses on task allocation for MT-MR problems [3]. For these types of tasks, multiple robots need to dynamically coordinate their execution during the task. The robots must work together to complete the joint task to ensure that they are each working on separate parts and that the constraints of the task are upheld throughout the execution. However, due to the different abilities between different robots, a single task definition is not sufficient to ensure the task can be completely without faults. In the case of heterogeneous robot teams this becomes even more of a problem as the individual capabilities of each robot can affect the outcome of the task if the task structure does not take their respective capabilities into account. In order to alleviate these concerns, we developed a generalized task structure which is able to transfer skills of a learned task to teams of heterogeneous robots. This system uses a small number of human demonstrations

to learn a hierarchical task structure on a single robot. This structure acts as a skeleton for the task which has been adapted to work on teams of heterogeneous robots through the development of a continuous-valued metric which is able to account for the robots' variable skills during the task execution.

The collaborative control architecture in [4] assumes that the team of robots all had the same grasping capabilities. In a team made up of heterogeneous robots, each robot will have different capabilities. Therefore, the problem of task allocation for heterogeneous teams is more complex due to these varying capabilities of the robots. While numerous approaches have been developed for both *collaborative task execution* and *heterogeneous teams*, we addressed these problems from a different perspective with the aim of expanding the capabilities of heterogeneous multi-robot systems by extending our control architecture to include a continuous grasp suitability score to ensure the tasks are performed by the robots with the best suitability.

To best utilize the varying capabilities of heterogeneous robots, a interdependence task constraint was added to the proposed generalised task structure. This constraint requires explicit cooperation between agents in order to complete the task. Many tasks, such as building tasks, require robots with different capabilities to work together to complete them. This constraint enables this capability by allowing for cooperation between robots for manipulation tasks. The primary aim of this constraint is to allow one agent to hold a part in place, while another agent attaches another component. Thus, this constraint demonstrates the collaborative capabilities of the

proposed generalized task structure.

1.2 Human-Robot Task Allocation

To allow both the task execution and learning of the task structure to be as generalizeable as possible, several efforts were made to extend the previously developed control architecture to work with human-robot teams.

One focus for task allocation for human-robot teams is teaching the robot how to perform tasks. Existing research on teaching robots by demonstration or verbal instruction focuses on learning tasks that mainly involve sequential constraints, building representations that encode steps which have to be executed in order. In practice, robot tasks may require more complex dependencies. For instance, some parts of the task could be allowed to execute in any order (e.g., adding ingredients for making cookies), leading to multiple ways in which the task can be performed. Other parts of the task may have to be executed in a specific order (e.g., adding ingredients before doing the mixing). Furthermore, other parts of the task could be achieved through entirely different paths of execution (e.g.,could add either whole wheat, or white flour, or almond flour in a recipe). Such tasks are difficult for a human to teach by demonstration [5], as in order to capture the various different ways in which a task may be executed, a learning system may need to be provided with multiple demonstrations of the same task. In contrast, these types of complex dependencies

can be efficiently conveyed by use of conjunctions in verbal instructions in a single command. This work proposes algorithms that process such instructions and produce a hierarchical task representation that encapsulates the execution constraints and is directly executable by the robot.

Another central focus of task allocation methods is how to determine the ordering of sub-tasks that results in a completed task. In simple tasks, this focus is sufficient. However, in very complex or precise tasks, ensuring correct execution of all sub-tasks is just as important as identifying the order in which to complete them. In some cases, correct execution cannot be ensured. In these cases an additional fault recovery component is required to determine how to handle execution failures. For robotic systems, this kind of fault recovery can be done through human-robot collaboration. By allowing a robot to initiate a dialogue with a human, the robot can discuss how best to handle a failure with its human teammate before taking action. The goal of this capability is to extend the generalized task structure for hierarchical tasks to recover from faults during execution through dialogue and human-robot collaboration. The proposed architecture is cognizant of failures and upon failure initiates a dialogue to resolve the issue. The architecture maintains an extended dialogue between the robot and the human, rather than a single request for help, which allows for multiple ways of resolving a given fault. Failures are autonomously detected with the robot's on-board sensors through a combination of views from multiple cameras. The architecture ensures that the task constraints are adhered to throughout the entire task execution, even during failures. This allows for a robust execution of a hierarchical task with

multiple types of constraints such as sequential, non-ordering, and multiple paths of execution.

The final focus of this work is to extend the generalized task structure to allow for human-robot teams to work together on a joint task. While autonomy and the ability of robots to perform complex tasks have significantly improved, the challenges of operating in collaborative domains prevent current robotic systems from working effectively alongside people as collaborators and assistants. We propose a solution where the robot uses its own task structure (e.g., controller) both to plan its own future actions, and to keep track of its human teammate's current and future goals. Another challenge in this domain is the resolution of conflicts in task allocation between the human and the robot. To allow for a smooth collaboration, the human works independently alongside the robot on the available parts of the task. However, this can result in a conflict occurring in which both agents attempt the same portion of a task at the same time. Our proposed method resolves these conflicts through dialogue, similar to the resolution method in the previous extension for task failures.

1.3 Generalized Task Structure Learning

The space of generalized learning can be broken down into two primary areas: generalized policy learning and generalized task structure learning.

Generalized policy learning focuses on learning a policy which is able to perform a task in a generalized context. These types of policies tend to focus on low-level methods which define how to complete a given task, such as how to pick up a block and move it to a specified location [6, 7]. These types of policies are very good at generalizing to unseen configurations of the task environment. However, since these methods focus on learning a single policy for transfer to other agents, they are very limited in the types of skills that are transferred. These approaches cannot adapt these policies to robots with different capabilities, as is the case in heterogeneous teams. The low-level policies learned by these approaches only factor in how to move the robot arm to different locations, but do not know how to factor in the policy for a different arm or gripper configuration. By extending our previously developed control architecture, the proposed generalized task structure is able to factor in the different capabilities of a robot which helps our learned tasks to generalize not only to different environmental conditions but also to heterogeneous teams of robots.

In contrast to the policy learning work which learns a single policy, there has been work done which learns the task structure instead. Generalized task structure learning focuses on learning the underlying structure of a given task so that the task can be generalized to other instances of the task. These types of methods focus on what steps need to be completed in order for a task to be performed. Unlike generalized policy learning methods, these approaches do not look at the low-level policies of how to move the agent to perform a task, but instead focus on the bigger picture of the ordering of steps necessary to complete the task. For example, these methods might

look at a building task and identify the order in which the parts need to be moved to correctly build the given structure, instead of determining how exactly to move an individual block. In other words, these methods focus on a form of task allocation in which the task structure is being learned. However, most of this work focuses on learning a sequential task representation. These types of representations are very limited as they only allow a task to be completed in a single way. For collaborative tasks performed by heterogeneous teams of robots, a single way of completing a task may not be sufficient due to the varying capabilities of the different robots. One robot may be tasked with a step along the sequence which it is not able to perform, which then prevents the entire task from being completed. Therefore, by utilizing our proposed generalized task structure, we are able to account for multiple ways to perform the task, which allows us to allocate robots to different sub-tasks according to their capabilities.

In current approaches for multi-robot control, a robot's ability to perform certain actions or tasks (i.e., the team's heterogeneity) has been considered to be known in advance, represented as a binary choice of whether the robot can or cannot perform a task (without any other options in between), and with the assumption that these capabilities are not changing over time (i.e., the robot has a fixed set of skills throughout the entire task execution). In practical applications, it becomes apparent that the degree to which a robot may perform a task covers a continuous spectrum instead of just taking binary values. For instance, a robot with a dexterous hand may pick up objects better than a robot with a 2-dimensional gripper; both are able to perform

the task, though with varying degrees of effectiveness. Having a continuous-valued metric that encodes this information can be highly beneficial for task allocation in a multi-robot team, as it would allow the selection of the robot best suited for a given task. Furthermore, the value of this metric may vary continuously throughout a task execution, as different environmental conditions may allow a robot to perform actions that would otherwise be impossible. For example, a robot without a mobile base cannot grasp a cup that is on a shelf in another room, but could pick up that cup if it were placed on a table in front of it. The approach proposed in this work utilizes a metric that encodes a robot's ability to perform a particular task component over a continuous spectrum; the metric is updated continuously during task execution, allowing for dynamic task allocation that takes into account the most recent environmental conditions. Therefore, the proposed generalized task structure is able to transfer learned skills to teams of heterogeneous robots while accounting for their heterogeneity.

To enable generalization of the learned task, the structure of the task must be flexible to allow for these variances in grasping capabilities for heterogeneous teams of robots. Therefore, it is not sufficient to give a single, rigid task structure to a team of heterogeneous robots and assume that they will be able to complete the task without taking into account their own skill-sets. Instead of manually crafting a different task structure for each robot in the team, we propose a method in which we are able to learn the underlying task structure from a set of human demonstrations which can be transferred to a robot and modified online with its own suitability for each

part of the task through the utilization of our previous developed hierarchical control architecture.

1.4 Contributions of Proposed Approach

The aim of the proposed work is to develop **a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for both multi-robot and human-robot teams**. Therefore, the proposed contributions of this work fall into two categories: human-robot capabilities and multi-robot capabilities.

Human-robot capabilities:

- A novel approach to robot task learning from verbal instruction.
  - Development of a framework for *directly mapping a complex verbal instruction to an executable task representation*, from a single training experience.
  - Allows the generalized task structure to be utilized to teach robots through verbal instruction.
- A fault recovery system able to detect and inform users of failures and resolve them through dialogue.
  - Development of a hierarchical control architecture that *1) autonomously detects and is cognizant of task execution failures*, *2) initiates a dialogue*

with a human helper to obtain assistance, and *3) enables collaborative human-robot task execution through extended dialogue* in order to *4) ensure robust execution of hierarchical tasks with complex constraints, such as sequential, non-ordering, and multiple paths of execution.*

  - Allows the generalized task structure to be more robust to failures.

- An extension of our previously developed control architecture to facilitate collaboration by human-robot teams.

  - Development of an architecture with the ability for dynamic allocation of tasks in human-robot teams and opportunistic task execution given different environmental conditions.

  - Allows the generalized task structure to be utilized for human-robot teams performing collaborative tasks.

Multi-robot capabilities:

- An extension of our previously developed control architecture for incorporating the varying capabilities of a team of heterogeneous robots.

  - Development of an architecture that enables *collaborative execution of tasks with hierarchical representations and multiple types of execution constraints* by teams of robots with *variable heterogeneity.*

  - Allows the generalized task structure to incorporate the varying capabilities of robots into the task allocation scheme.

- An extension of our previously developed control architecture for an interdependence constraint which requires explicit coordination between agents.
  - A *WHILE* constraint is incorporated into the control architecture to enable explicit coordination between agents.
  - Allows the generalized task structure to represent tasks which require explicit coordination between agents.
- A novel method which is able to take sequences of demonstrations and learn a hierarchical task representation.
  - Development of a learning method which utilizes human demonstrations to generate a hierarchical representation that can be directly executed by a robot.
  - Allows the generalized task structure to learn tasks from a small number of human demonstrations.

Combining each of these major contributions together results in a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for both multi-robot and human-robot teams. By learning the underlying hierarchical structure of a task we allow for skill transfer of the task to collaborative heterogeneous teams through the utilization of our control architecture. Our extension of the control architecture for the interdependence constraint allows for a wider range of tasks which can be formulated in a manner capable of utilizing our control architecture. The task

structure can also be used to teach the robot to perform tasks both through human demonstration as well as verbal instruction. Lastly, the generalized structure can be used by human-robot teams for robust and collaborative execution of joint tasks.

1.5 Summary

The basis of this work is a control architecture for collaborative multi-robot/human-robot systems, focusing on the problem of task allocation under hierarchical constraints imposed on a joint task. Real-world tasks are not only a series of sequential steps, but typically exhibit a combination of multiple types of constraints, where some parts of the task are sequential, some have no ordering constraints, and others allow for alternative paths of execution. Therefore, to enable multi-robot and human-robot teams to complete joint tasks in the real world, the design of a generalized hierarchical control architecture which is able to encompass all of these types of constraints is necessary. One primary example which illustrates this concept is a building task. In order to correctly build a piece of furniture, certain parts have to be connected first whereas others can be attached at various points in the process. In previous work [1, 2] we developed an architecture that provides a compact encoding of such tasks and validated it in a single robot domain. We extended this architecture to work in the multi-robot domain [4].

The proposed contributions for the multi-robot domain focuses on task allocation for MT-MR problems. One of the major issues with this class of problems is that a new task structure must be defined for each new task as well as each robot with different capabilities. In order to alleviate these concerns, we developed a generalized task structure which is able to transfer skills of a learned task to teams of heterogeneous robots. To best utilize the varying capabilities of heterogeneous robots, a interdependence task constraint was added to the proposed generalised task structure. This constraint requires explicit cooperation between agents in order to complete the task. Thus, this constraint demonstrates the collaborative capabilities of the proposed generalized task structure.

To allow both the task execution and learning of the task structure to be as generalizable as possible, several efforts were made to extend the previous architecture to work with human-robot teams. Firstly, development of a system which takes verbal instructions, produces a hierarchical task representation that encapsulates the execution constraints, and is then directly executable by the robot. Secondly, extension of the generalized task structure for hierarchical tasks to recover from faults during execution through the use of dialogue and human-robot collaboration. Lastly, extension of the generalized task structure to allow for human-robot teams to work together on a joint task.

The aim of the proposed work is to develop **a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for**

both multi-robot and human-robot teams. Therefore, the proposed contributions of this work fall into two categories: human-robot capabilities and multi-robot capabilities.

Human-robot capabilities:

- A novel approach to robot task learning from verbal instruction.
  - Development of a framework for *directly mapping a complex verbal instruction to an executable task representation*, from a single training experience.
  - Allows the generalized task structure to be utilized to teach robots through verbal instruction.
- A fault recovery system able to detect and inform users of failures and resolve them through dialogue.
  - Development of a hierarchical control architecture that *1) autonomously detects and is cognizant of task execution failures*, *2) initiates a dialogue with a human helper to obtain assistance*, and *3) enables collaborative human-robot task execution through extended dialogue* in order to *4) ensure robust execution of hierarchical tasks with complex constraints, such as sequential, non-ordering, and multiple paths of execution.*
  - Allows the generalized task structure to be more robust to failures.
- An extension of our previously developed control architecture to facilitate collaboration by human-robot teams.

  - Development of an architecture with the ability for dynamic allocation of tasks in human-robot teams and opportunistic task execution given different environmental conditions.
  - Allows the generalized task structure to be utilized for human-robot teams performing collaborative tasks.

Multi-robot capabilities:

- An extension of our previously developed control architecture for incorporating the varying capabilities of a team of heterogeneous robots.
  - Development of an architecture that enables *collaborative execution of tasks with hierarchical representations and multiple types of execution constraints* by teams of robots with *variable heterogeneity.*
  - Allows the generalized task structure to incorporate the varying capabilities of robots into the task allocation scheme.
- An extension of our previously developed control architecture for an interdependence constraint which requires explicit coordination between agents.
  - A *WHILE* constraint is incorporated into the control architecture to enable explicit coordination between agents.
  - Allows the generalized task structure to represent tasks which require explicit coordination between agents.

- A novel method based on a genetic algorithm which is able to take sequences of demonstrations and learn a hierarchical task representation.
  - Development of a learning method based on a genetic algorithm which utilizes human demonstrations to generate a hierarchical representation that can be directly executed by a robot.
  - Allows the generalized task structure to learn tasks from a small number of human demonstrations.

Chapter 2

Background

The space of generalized learning can be broken down into two primary areas: generalized policy learning and generalized task structure learning. We briefly discuss the differences between these two areas as well as the previous work done in each area. More focus is given to the second category as our proposed work falls into this category. Additionally, we provide a brief overview of the prior work done in multi-robot task allocation in order to explain how our generalized task structure fits in with the field. Lastly, we provide a brief overview of the prior work done in each of the areas necessary to extend this generalized task structure to collaborative tasks for human-robot teams.

2.1 Generalized Policy Learning

Generalized policy learning focuses on learning a policy which is able to perform a task in a generalized context. These types of policies tend to focus on low-level methods which define how to complete a given task, such as how to pick up a block and move it to a specified location [6, 7]. These types of policies are very good at generalizing to unseen configurations of the task environment. For instance, the block may be placed anywhere in the robot's reachable space and the policy will be able to generalize from its trained examples in order to move the block to the desired location even though it was not trained to pick the block up from that specific location.

Recently there has been a lot of work done in the area of generalized policy learning. The majority of this work utilizes various machine learning algorithms in order to learn a generalized policy. Two of the most used methods are reinforcement learning (RL) and imitation learning. This section describes how RL and imitation learning have been applied to this area in more detail.

2.1.1 Reinforcement learning for generalized policies

In recent years, there has been a huge push for using reinforcement learning in robotics to learn policies that are able to complete simple tasks. Kober *et al.* discuss these methods in their survey paper [8]. The majority of these methods focus on using RL for single robot tasks. Some other recent notable work in the single robot domain

is the method called guided policy search proposed by Levine *et al.* in [9]. This method has been used to train policies capable of performing a wide range of tasks from arbitrary starting positions in several different application areas such as gait-learning tasks [9], object manipulation tasks [6, 10], and peg insertion tasks [11]. Guided policy search is a prime example of the types of methods that fall under the category of generalized policy learning. However, generalized policy learning does not just encompass methods which learn a single policy at a time. This area also involves methods which are able to generalize by learning multiple policies at the same time such as the method proposed by Reidmiller *et al.* which uses sparse reward signals to learn multiple auxiliary tasks simultaneously [7]. Nair *et al.* use a self-supervised "practice" phase in which agents learn a set of self-generated goals [12]. Very recent work in this area also illustrate how RL policies can be combined with model-based representations of a task to improve the policies. Zhang *et al.* use a probabilistic graphical model which infers the dynamics of the system in order to further improve the learned RL policies [13]. Furthermore, this area encompasses methods proposed for the multi-robot domain.

The concept of applying reinforcement learning to the multi-robot domain has not has not had as much of a recent push as that of the single-robot domain. The majority of these methods are over several years old, such as the methods described in the survey paper by Matarić *et al.* [14]. The more recent applications instead focus on agents in general, not specifically robotic agents. These methods fall under the term multi-agent reinforcement learning (MARL). By focusing on a general agent instead of on

teams of robots, these methods are able to learn a wide variety of policies. Forester *et al.* propose a method which learns communication policies between robots in order to understand how best to transfer knowledge between teammates [15]. Ghavamzadeh *et al.* propose a method in which agents learn three interrelated skills: how to perform each individual sub-task, the order in which to carry them out, and how to coordinate with other agents. However, the majority of works however focus on low-level motor polices [16–18].

Each of these works, in both the single robot and multi-robot domains, effectively utilize RL methods to generalize policies from a set of training configurations to unseen configurations. However, these methods are not able to generalize to different formulations of these learned tasks. By learning a specific policy, it restricts these methods to execute the task in a specific ordering which means these methods are not able to handle tasks which have multiple paths of execution. Additionally, because these methods learn low-level policies, they are not able to generalize to sets of agents with varying capabilities, such as a team of heterogeneous robots, as each agent would require a different policy to reflect its specific capabilities.

2.1.2 Imitation learning for generalized policies

Similar to reinforcement learning methods, there has recently been a push in imitation learning for generalized policy learning. Many of these methods focus on one-shot imitation learning in which only a single example is needed to generalize to unseen

examples of a trained policy [19, 20]. Other methods focus on improving the learned policy by gathering additional human demonstrations [21, 22]. All of these methods focus on imitation learning in the the single robot domain. Very little work has been done on generalized policy learning through imitation learning on multi-robot systems. Freelan *et al.* focus of learning policies for a multi-robot team through a set of hierarchical finite state automata [23]. Amato *et al.* uses finite state controllers as policies in order to coordinate a multi-robot team [24]. Chernova *et al.* propose a method that is able to teach multi-robot teams policies which are able to sort a set of balls [22].

These methods do well at generalizing from provided demonstrations to similar yet unseen scenarios. However, as in the RL work, these methods also restrict how a task can be executed. Since these methods are learning from demonstrations, they will only be able to learn to execute a task in a formulation which was represented in the demonstrations. Therefore, these methods cannot utilize formulations which would require multiple paths of execution to complete a task, unless every possible path was provided in the demonstrations. Furthermore, these methods cannot easily generalize to robots with different capabilities. These policies are trained using either a single agent or teams of homogeneous agents. Therefore, the tasks learned by these agents assume the agent performing the task will be able to execute the task the exact way that it has been taught, which is not the case for teams of heterogeneous agents with different capabilities.

These generalized policy learning methods focus on learning how to physically perform a given task. Although this type of generalization is important, the focus of the proposed work are the steps that need to be done in order to complete the task, i.e., how to determine the underlying structure of a given task. Therefore, we must examine the area of generalized task structure learning.

2.2 Generalized Task Structure Learning

Generalized task structure learning focuses on learning the underlying structure of a given task. This learned structure can then be utilized for similar tasks. These types of methods focus on what steps need to be completed in order for a task to be performed. Unlike generalized policy learning methods, these approaches do not look at the low-level policies of how to move the agent to perform a task, but instead focus on the bigger picture of the ordering of steps necessary to complete the task. For example, these methods might look at a building task and identify the order in which the parts need to be moved to correctly build the given structure, instead of determining how exactly to move an individual block. The ordering also reflects constraints of the task which might require a hierarchical component, such as in a building task where certain blocks must be placed first so others can be stacked on top. In other words, these methods focus on a form of task allocation in which the task structure is being learned.

There has been a lot of work done that focuses on task allocation. Many of these methods use an auction type of scheduling algorithm [25, 26]. Some work extends this auction type of scheduling to allow task allocation on multi-robot teams [27, 28]. These methods focus on learning how to allocate sub-tasks to robots in order to complete the overall tasks. Although these methods are simply learning when to complete a task, and not necessarily the underlying constraints behind a task, they still fall into the generalized task structure learning category. However, by simply learning to schedule tasks, these methods are learning a sequential ordering of tasks, which means that they are limited in the types of tasks that they can learn. For instance, these methods cannot deal with multiple choices within a task such as in cases where only sub-task A or sub-task B need to be completed but not both. For these types of problems, a sequential representation is not sufficient. For this reason, there has been a substantial amount of work done on learning hierarchical task representations.

Methods presented in many papers focused on learning a generalized hierarchical task representation use a hierarchical task network (HTN) for their task representation [29], [30]. Although these methods are able to handle more diverse constraints such as temporal constraints and alternate forms of tasks, the main limitation of these methods is that they are only designed for the single-robot domain and therefore do not take into account the possibility of varying capabilities on a robot. Other methods focus on using behavior network representations [31, 32]. This type of representation is the basis for our proposed generalized task structure. However, these methods

also are limited to the single-robot domain. The closest method to our proposed work is the work by Browne *et al.* [33]. This work uses a small number of task demonstrations in order to learn a finite state machine which encodes a wide variety of constraints between different parts of the task. Although this work is able to encode all of the types of constraints that we are looking at, including conditional constraints, this work is only formulated to work in the single robot domain and therefore does not take into account the possibility of varying capabilities of a robot. Since these methods only work in the single robot domain, it is unclear how they would perform in the multi-robot domain for teams of robots which have varying capabilities. Due to the formulations of these methods, it would be very difficult to encode any suitability metric on the constraints which would be necessary for a team of heterogeneous robots. This kind of metric greatly affects the flow of the learned task as it ensures robots only perform tasks which they are best at. However, as the methods are proposed in the single robot domain, they do not have a method to deal with this type of dynamic constraint for multi-robot teams. Therefore, one of the main contributions of the proposed work is that the learned task can be performed by a set of heterogeneous robots with a dynamic capability constraint, unlike the previous methods explored in this area.

2.3 Multi-Robot Task Allocation

Multi-robot systems gained momentum in the 80's and 90's when a series of projects were implemented successfully such as ACTRESS [34], ALLIANCE [35] and MURDOCH [36]. These projects proposed the efficient use of multi-robot systems over a single powerful robot. To date, a wide range of distributed approaches have been developed for task allocation in multi-robot systems.

Several approaches fall under the category of behavior-based systems [37]. These approaches perform computations on internal representations in order to decide what action to take. They consist of a collection of parallel, concurrently executing behaviors devoid of a centralized arbiter [38]. Our proposed architecture is such a behavior-based system, relying on activation spreading and peer-behavior communication for task allocation. Parker *et al.* [35] proposed one of the first behavior-based architectures for the multi-robot task allocation problem called ALLIANCE and a related L-ALLIANCE architecture [39]. These approaches focus on fault tolerant and efficient control. Werger [40] presented a distributed behavior based approach to the problem of Cooperative Multi-Robot Observation of Multiple Moving Targets (CMOMMT). The architecture used cross-inhibition and cross-subsumption between peer behaviors on each robot in order to determine allocation of robots to targets. Unlike these approaches, our architecture incorporates various types of ordering constraints and multiple paths of execution which allows for a more diverse application to multi-robot

collaboration tasks, such as building or manufacturing, instead of navigation-based tasks as in these earlier approaches.

Other approaches focus on a market-based architecture for allocating tasks distributively [41]. In these approaches, the team seeks to optimize an objective function based upon individual robot utilities for performing particular tasks [42]. Gerkey *et al.* [36] proposed a novel dynamic task allocation approach for a group of heterogeneous robots utilizing a publish/subscribe messaging system to carry out auctions called MURDOCH. Wang *et al.* [43] proposed a market-based task allocation algorithm which utilizes a task evaluation function based on distance fitness and urgency. [44] designed CeCoTA, a market based algorithm for simultaneous allocation of multiple tightly couple multi-robot tasks to coalitions of heterogeneous robots. The approach was validated in a simulated environment with simple (atomic) tasks. Trigui *et al.* [45] proposed two auction-based distributed algorithms for task allocation namely DMB and IDMB. Coalition formation is a prevalent approach for handling team heterogeneity, enabling multiple robots to build small teams that allow them to perform a larger overall task. An approach that uses a model to predict the time to execute a task is presented in [46]. Unlike these approaches, our control architecture defined in [4] does not use a complicated utility function or an explicit auction system with a coordinator and bidders. Our hierarchical architecture uses activation-spreading based on distance to the robots' grippers to identify which tasks to complete.

2.3.1 Heterogeneous Robot Teams

Coalition formation is a prevalent approach for handling team heterogeneity, enabling multiple robots to build small teams that allow them to perform a larger overall task. ASyMTRe enables the sharing of sensory and computational capabilities [47] in a navigation task in which only one of the robots has localization capabilities. This approach was extended in [48], demonstrating formation of coalitions in tightly coupled multi-robot tasks that need to maintain a set of given sensor constraints, in a domain in which robots need to navigate to various goals. Similar coalitions have been demonstrated in cooperative manipulation tasks: [49] demonstrate an approach based on two-robot leader/follower coalitions to carry a box. Furthermore, [50, 51] present CAMPOUT, a Control Architecture for Multi-robot Planetary Outposts validated on physical experiments of coordinated object transport and team cliff traverse. However, these types of coalition formation methods assume that the tasks are atomic behaviors which do not have any inter-task constraints. These inter-task constraints are the focus of our tasks which contain complex hierarchical constraints between the tasks.

Methods that aim to handle more complex task representations have been shown in [52, 53], which focus on the execution of tightly coupled tasks. In [52], the task allocation problem is modeled as a mixed integer linear programming (MILP) problem and a centralized anytime algorithm is developed to provide an optimal solution that handles the allocation, scheduling and path planning for a search and rescue

task with spatial constraints. Due to the centralized nature of the algorithm, the method is dependent on prior knowledge of a static environment and produces fixed allocations that do not change during the course of the task. The Petri Net Plan framework developed in [53] can represent multi-robot plans using sensing, loops, concurrency, non-instantaneous actions, action failures, and different types of action synchronization. This does *not* consider heterogeneity as a factor for task allocation. Furthermore, this method was tested on a homogeneous team of robots (AIBO's) with equivalent capabilities.

The proposed heterogeneity metric is highly similar to the utility functions computed by the above market-based approaches or to the motivation factors used in the ALLIANCE architecture [35]. It incorporates task specific utility (such as a distance to a target object) with both continuous utility (perceived grasp effectiveness) and discrete information about the robots' skills (ability/inability to grasp a given object). The proposed work contributes this metric for hierarchically structured tasks, that exhibit a combination of complex constraints such as sequential, non-ordering, and alternative paths of execution by a team of heterogeneous robots.

2.3.2 Interdependence Constraints

To allow for explicit cooperation between multiple robots, an interdependence constraint must be used which requires several parts of a task to be completed together. One type of task which requires explicit cooperation is building tasks. These tasks

require one agent to hold a part in place while another agent connects another piece. Our proposed work is focused on these types of manipulation tasks which require a holding behavior while another task component is completed.

Saeedvand *et al.* proposed a robust Multi-Objective Multi-Humanoid Robots Task Allocation (MO-MHTA) [54] algorithm which is a variant of the Multi-Robots Task Allocation (MRTA) problem. This work utilizes four objectives, namely energy consumption, total tasks' accomplishment time, robot's idle time, and fairness which were optimized in an evolutionary framework in MO-MHTA. However, this work focuses on a different application than our proposed work, namely rescue applications. The objectives used to allocate tasks are regarding the operation conditions which have interdependence due to time constraints and other factors. Our work focuses on pick and place tasks with a hierarchical nature which require explicit cooperation to complete the tasks.

Interdependence constraints have also been explored in task learning. Santucci *et al.* propose a robot control architecture which allows for autonomous open-ended learning of multiple tasks which may be interdependent [55]. This system is validated on a humanoid robot learning interdependent multiple reaching tasks. There are two major differences between this work and ours. The first is that this work is focused on learning whereas ours is focused on task execution. The second is that this work focuses on tasks which can be accomplished by a single robot whereas our work is primarily focused in the multi-robot domain with tasks which require multiple agents

to coordinate to complete the tasks.

Another common area of interdependence that is explored in multi-robot and human-robot teams is focused on creating frameworks which enable to robot agent to be interdependent on the other agents. Johnson *et al.* propose a coactive design to address the desire for robots to perform like interdependent teammates [56]. The major difference between this work and ours is that the interdependence on our work is focused on relations between the tasks themselves, not the roles of the agents in the task.

2.4 Collaboration and Dialogue for Human-Robot Teams

In order to extend the proposed generalized task structure to the human-robot domain, three major components are explored: 1) using verbal instruction to teach the robot to perform tasks, 2) development of a task verification system to detect and inform users of failures, and 3) a system to allow the human and robot to work alongside each other on a joint task is explored. Prior work done in each of these areas are discussed below.

2.4.1 Verbal Instruction

Numerous approaches have been designed for translating natural language instructions into control structures, focusing on various aspects of grounding linguistic information onto physical actions, objects or other relevant attributes. In this work, we focus on the two specific problems: 1) learning representations that encode complex execution constraints (provided through conjunctions) and 2) parameterizing learned tasks from information provided by adjectives and prepositions.

Despite the rich spectrum of instruction-based task learning approaches, existing methods encapsulate mostly sequential constraints, as a series of individual steps that have to be performed in a particular given order. Lauria *et al.* [57] show a first example of controlling a robot using instructions given in natural language. The system uses an explicitly defined grammar that is domain dependent, restricting the robot to understanding only instructions related to navigation. Similarly, [58] and [59] present systems that focus on guiding a robot through natural language to navigate in the environment, focusing on issues related to spatial representation of the world and navigation actions. Other approaches have focused on problems such as pick-and-place [60], grasping [61], blocks worlds [62], building of motion controllers [63], or navigation, delivery and validation tasks [64]. Chang [65] presents an approach to increase the flexibility of a speech-based interface, by having the system learn to associate complex desired configurations with particular simple instructions (such as what lights need to be turned on and to what level for "reading mode" or "TV

mode"). However, the configurations are specified by the user (by actively performing the operations needed for that configuration) and are next associated with a simpler command that is used to identify that situation. Arumugam *et al.* [66] present a method for grounding verbal instructions at varying degrees of specificity using a deep neural network language model that selects the appropriate level of a planning hierarchy. A related approach for one shot learning of actions and new objects from language instruction has been proposed in [67], in which new tasks are learned as sequences of individual actions. In this work we aim to use the flexibility of natural language that can concisely include multiple dependencies in a single command (for example, *"Do a THEN b OR c OR d"*) to enable the learning of more complex task representations.

Recent work closely related to our goals is presented in [68]. The approach relies on a library of verb-environment-instructions built from a data set of task descriptions, which represents all possible instructions for each verb in that environment. Relying on this, a model dependent on an energy function resolves ambiguities based on appropriate environment context and task constraints. A framework on Generalized Grounding Graphs (G3) is presented in [69], for both navigation and object manipulation. The framework allows for dynamic instantiation of a probabilistic graphical model for a given natural language command, taking into account the hierarchical and compositional semantic structure of the instruction. The method relies on a corpus of sentences specific to the manipulation task to infer the most likely meaning of the instruction. We propose a method that does not require training or a corpus

specific to a particular task: we use the argument information provided by a semantic parser [70] to automatically generate task controllers for an unrestricted set of action verbs. Furthermore, the focus of this work is not on handling the full spectrum of unstructured linguistic instructions and ambiguities, but rather on handling a more restricted domain that is typical for teaching by instruction in which the teacher aims to convey the information to the learner as clearly as possible, aiming to minimize ambiguity in order to facilitate the learning process.

2.4.2 Task Verification

Joint assembly tasks employ several elements in order for the system to acquire/learn a model of the task, to monitor its progress, and to repair the system when things do not go according to plan. Task construction is an essential part of this process. Dialogue has been used for task construction for human-robot collaboration systems [71]. Similarly, dialogue-based systems can be used for hierarchical task construction [72]. These systems used human dialogue to construct a model of a given task for the robot to then execute. Conversely, Hayes and Scassellati used human demonstrations in order to learn a hierarchical plan [30]. These systems made task training a one-time activity which then provided a robot a plan to execute. Finally, a task description can be manually specified, as was done in [73], by using a graphical user interface. However, failure resolution was not done when collisions occurred in these works; the robots would generally defer to what their partner wanted to do.

Allocating tasks among multiple agents is required in order to avoid conflicts over resources, such as space or tools. Tasks can be assigned implicitly based upon environmental conditions, capabilities, or what other agents are doing. One such example uses monitoring of the scene in order to implicitly assign tasks to agents [74]. Our own prior work has used a hierarchical task representation along with distributed communication to forestall conflicts and assign tasks based on jointly-optimal proximity to goals [4]. As in the above works however, these methods also did not incorporate failure resolution during the task execution.

Several studies have successfully had robots initiate communication with humans when a problem arose [75–77]. Fong et. al. had a robot explore a room via teleoperation and ask a remote human through a mobile device about how to proceed when confronted with uncertainty [75]. The control system framed the human's role as a limited resource. Extensions of this work had a team of robots conduct a surveillance task, leading the authors to conclude that dialogue improved the human's ability to deal with context switching [76]. Both [75, 76] focused on collaborative teleoperation based tasks. These studies primarily focused on how humans interact with robots asking questions. However, humans primarily offered additional information to the robot but were not capable of helping the robot complete the task, which our work does allow.

2.4.3 Task Allocation for Human-Robot Teams

Human-robot collaboration is becoming increasingly important as more robots are incorporated into daily activities and industry settings. The focus of this work is to enable robots to work safely alongside humans by developing an intent recognition system to monitor the human's actions. Intent recognition encompasses many domains, including: entertainment [78]; museum documents [79]; personal assistants [80]; health care [81]; space exploration [82]; police SWAT teams [83]; military robotics [83]; and rescue robotics [84]. The proposed work demonstrates the ability for dynamic allocation of tasks in human-robot teams based on intent recognition, while also observing hierarchical constraints.

Many approaches exist for recognizing human intent. In [85], weighted probabilistic state machines were used to complete a recognition task that was split into two categories: explicit intention communication and implicit intention communication. Wang *et al.* used Recurrent Convolutional Neural Networks (RCNNs) to recognize the intention of humans manipulating objects [86]. Li *et al.* used Neural Networks to develop an online estimation method to handle the nonlinear and time-varying properties of using a limb model for estimating human intention. Human-aware motion planning was examined in [87] and [88]. The ability of a robot to work with a human in close proximity without collisions was demonstrated in [89]. In [90], human motion was modelled using a Gaussian Mixture Model (GMM). Unlike these methods, our work focuses primarily on hand detection to infer the human's intent which is

then used by the robot to directly update the status of the joint task through our hierarchical architecture.

A collaborative robot should be able to execute complex tasks, be aware of its teammates' goals and intentions, as well as be able to make decisions for its actions based on this information. Recent work addresses these challenges using a probabilistic approach for predicting human actions and a cost based planner for the robot's response [91]. Tasks are represented as Bayes networks and prediction of human actions is performed using a forward-backward message passing algorithm in the network. However, this inference process is dependent on knowledge of the full conditional probability table for the task, which increases computational complexity for large tasks and limits adaptability to changes in the task at run-time. This approach has been extended in [92], with a new task representation that can encode tasks with multiple paths of execution. The initial representation for the task is a compact AND-OR tree structure, but for action prediction and planning, it has to be converted into an equivalent Bayes network, which has to explicitly enumerate all possible alternative paths. Our work is able to utilize a compact encoding of a task throughout the entire execution, thereby avoiding the overhead needed to enumerate all possibilities which is required by this work.

2.5 Summary

The space of generalized learning can be broken down into two primary areas, generalized policy learning and generalized task structure learning. We briefly discuss the differences between these two areas as well as the previous work done in each area. Additionally, we provide a brief overview of the prior work done in multi-robot task allocation in order to explain how our generalized task structure fits in with the field. Lastly, we provide a brief overview of the prior work done in each of the areas necessary to extend this generalized task structure to collaborative tasks for human-robot teams.

Generalized policy learning focuses on learning a policy which is able to perform a task in a generalized context. These types of policies tend to focus on low-level methods which define how to complete a given task, such as how to pick up a block and move it to a specified location [6, 7]. These types of policies are very good at generalizing to unseen configurations of the task environment.

Generalized policy learning is a prominent area of research focus. The majority of this work utilizes various machine learning algorithms in order to learn a generalized policy. Two of the most used methods are reinforcement learning (RL) and imitation learning. These generalized policy learning methods focus on learning how to physically perform a given task. Although this type of generalization is important, the focus of the proposed work is about what steps need to be done in order to complete

the task, i.e. how to determine the underlying structure of a given task. Therefore, we must examine the area of generalized task structure learning.

Generalized task structure learning focuses on learning the underlying structure of a given task so that the task can be generalized to other instances of the task. These types of methods focus on what steps need to be completed in order for a task to be performed. Unlike generalized policy learning methods, these approaches do not look at the low-level policies of how to move the agent to perform a task, but instead focus on the bigger of picture of the ordering of steps necessary to complete the task. For example, these methods might look at a building task and identify the order in which the parts need to be moved to correctly build the given structure, instead of determining how exactly to move an individual block. In other words, these methods focus on a form of task allocation in which the task structure is being learned.

There has been a lot of work done which focuses on task allocation. However, by simply learning to schedule tasks, these methods are learning a sequential ordering of tasks, which means that they are limited in the types of tasks that they can learn. Additionally, there has been substantial work done on learning hierarchical task representations. Although some methods are able to encode all of the types of constraints that we are looking at, these methods are formulated to work in the single robot domain and therefore do not take into account the possibility of varying capabilities on a robot, which is one of the primary contributions of the proposed work.

To date, a wide range of distributed approaches have been developed for task allocation in multi-robot systems. Several approaches fall under the category of behavior-based systems [37]. Other approaches focus on a market-based architecture for allocating tasks distributively [41]. Each of these methods treat tasks as atomic behaviors without complex representations or temporal constraints, which is the focus of our work.

The proposed work contributes a heterogeneity metric for hierarchically structured tasks, that exhibit a combination of complex constraints such as sequential, non-ordering, and alternative paths of execution by a team of heterogeneous robots. This metric is most similar to the utility functions computed by the above market-based approaches or to the motivation factors used in the ALLIANCE architecture [35].

To allow for explicit cooperation between multiple robots, an interdependence constraint must be used which requires several parts of a task to be completed together. One type of task which requires explicit cooperation is building tasks. These tasks require one agent to hold a part in place while another agent connects another piece. Our proposed work is focused on these types of manipulation tasks which require a holding behavior while another task component is completed.

Lastly, the prior work in three major components related to extending the proposed generalized task structure to the human-robot domain are discussed. First, using verbal instruction to teach the robot to perform tasks is discussed. Second, development of a task verification system to detect and inform users of failures is discussed. Third,

a system to allow the human and robot to work alongside each other on a joint task is discussed.

Chapter 3

Prior Work

The goal of the proposed work is to develop a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for both multi-robot and human-robot teams. In order to develop this generalized task structure, several major extensions to our existing control architecture are described in Chapters 4-9. Since the existing architecture is the backbone of the major contributions proposed in this work, the details of the architecture are described in this chapter. Many capabilities of the architecture are referenced and extended throughout the remaining chapters of this work.

3.1 Distributed Collaborative Task Allocation Architecture

Many of the proposed extensions are built upon a previously developed distributed multi-robot control architecture which allows for dynamic allocation of tasks between multiple robots as well as for opportunistic task execution given different environmental conditions [4]. The architecture uses a behavior-based paradigm [37] and it provides an efficient and compact encoding of tasks with various types of constraints (such as sequential, non-ordering, and alternative paths of execution), allowing the robots to dynamically decide which execution path to follow using an activation spreading mechanism that relies on environmental conditions. The execution constraints can be incorporated into a single behavior network task representation such as that presented in Figure 3.1, encoding a task for making a sandwich and tea.

A behavior network task is built of the following two types of nodes:

- **Goal Nodes:** These are the base goal control behaviors of the hierarchical task structure, and include the *THEN*, *AND*, and *OR* nodes that are used internally in the tree to encode the task constraints:
  - **THEN:** This is a n-ary node which is used to encode sequential constraints (the left child must execute before the children to its right can execute).

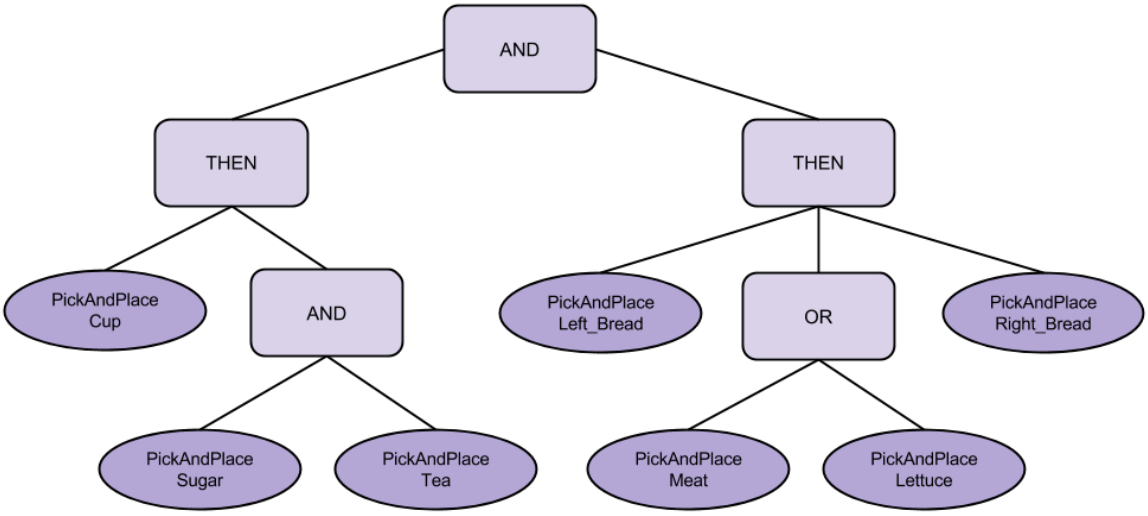


FIGURE 3.1: The full task structure of a tea-time task experiment. The lighter purple nodes represent the goal nodes of the task structure and the darker purple nodes represent the behavior nodes.

  - **AND:** This is a n-ary node which is used to encode non-ordering constraints (children can be executed in any order).
  - **OR:** This is a n-ary node which is used to encode alternative paths of execution (only one of the children will be executed).

- **Behavior Nodes:** These are the leaf nodes in the task tree structure and encode the physical behaviors that the robot can perform, e.g. a *PickAndPlace(Cup)* behavior will control the arm of the robot to pick up a cup from the table in front of it and place it in another location.

In order to maintain communication and connectivity between the nodes in a task tree, each node in the architecture maintains a state consisting of several components:

- **Activation Level:** a number provided by the node's parent and represents the priority placed on executing and finalizing a given node
- **Activation Potential:** a number representing the node's perceived efficiency, which is sent to the parent of the node,
- **Active:** a boolean variable that is set to true when the node's activation level exceeds a predefined threshold, indicating that the behavior is currently executing
- **Done:** a boolean variable that is set to true when the node has completed its required work.

The above state information is continuously maintained for each node and is used to perform top-down and bottom-up activation spreading that ensures the proper execution of the task given the constraints. To execute a task, *activation spreading messages* are sent from the root node of a task toward its children. These messages spread the *activation level* throughout the task tree in a top-down manner. At the same time, each node sends *status messages*, which encode a node's current state, to its parent node. These messages spread the *activation potential* throughout the tree in a bottom-up fashion. The state of each node in the task structure is maintained via an update loop which runs at each cycle. This loop performs a series of checks of the node's state and updates the various components of the state accordingly. The full details of this approach are presented in [1].

To enable cooperative execution of team tasks, each robot is equipped with its own instance of the task tree structure, identical to that of the other robots, which encodes the joint team task. Equivalent nodes in the task structures across robots are called *peers*. These peers are the means of communication between the robots and allow nodes to keep track of other robots' progress on the task. While the task hierarchy is uniform across robots, the *activation potential* and *activation levels* for each node in each tree are calculated individually by each robot. In addition to the state components used in the single robot case above, the multi-robot state of each node contains two new variables

- **peer_active**: a boolean variable that is true when either the node is active or the node's peer is active
- **peer_done**: a boolean variable that is true when either the node is done or the node's peer is done.

These additional state variables are required for collaboration between the robots because they allow each robot to identify if the node is currently being worked on or was already completed by another robot. This information is necessary to ensure there is no overlap in the sub tasks that the robots perform. By identifying what tasks are being worked on by its teammates and which tasks are already completed, each robot is able to determine the next step it should perform based on the activation spreading mechanism within its own task tree structure as well as its own state. This process allows the robots to maintain and communicate the states of all of the nodes

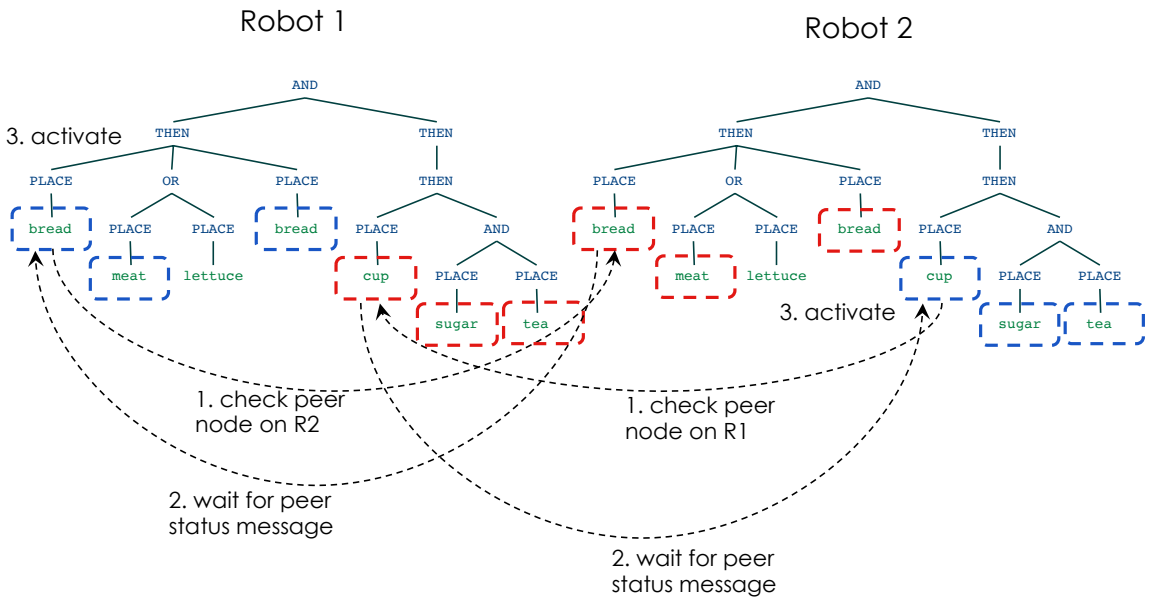


FIGURE 3.2: Example of the multi-robot decision making process for non-overlapping sub-tasks. Here robot 1 begins by choosing to place the bread for the sandwich, while robot 2 begins by choosing to place the cup for the tea. Initially, the nodes for PLACE-bread (on robot 1) and PLACE-cup (on robot 2) check the status of the peer nodes on the other robot (step 1) and wait for the peer status message (step 2). Since the peer nodes indicate that the other robot does not intend to activate the same node, each robot decides it can activate their respective nodes and begin the sub-task execution.

to their corresponding peer nodes on the other robots in order to ensure that the robots can work collaboratively to complete the task in a manner that follows its constraints.

Figure 3.2 shows the steps of node activation for situations in which the robots choose to work on different sub-tasks: robot 1 begins by choosing to place the bread for the sandwich, while robot 2 begins by choosing to place the cup for the tea. Initially, the nodes for PLACE-bread (on robot 1) and PLACE-cup (on robot 2) check the status of the peer nodes on the other robot (step 1) and wait for the peer status message (step 2). Since the peer nodes indicate that the other robot does not intend

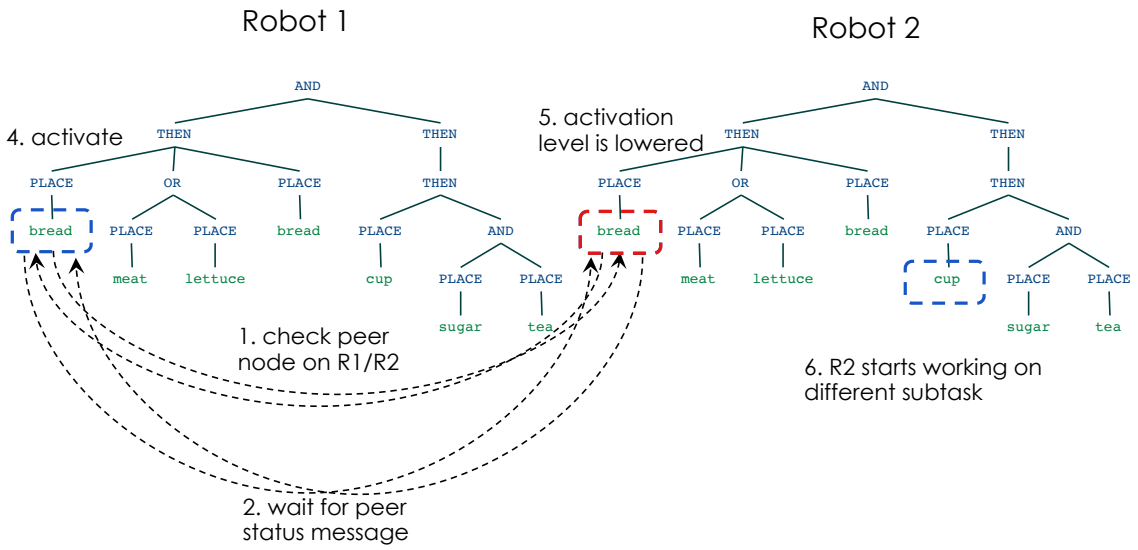


FIGURE 3.3: Example of the multi-robot decision making process for overlapping sub-tasks. Here both robots choose to work on placing the bread for the sandwich. Initially (step 1) the nodes for PLACE-bread on both robots check the status of the peer nodes and then wait for their status message (step 2). The response messages indicate that both robots plan to work on the same node, but have a timestamp indicating which robot first initiated the activation. The robot that has the earliest activation timestamp would then activate its node (steps 3-4), while the other robot lowers its activation for the same sub-task (step 5). This enables another node in robot 2's network (e.g. PLACE-cup) to get a higher activation level, and thus to begin working on another part of the task (step 6)

to activate the same node, each robot decides that it can activate their nodes and begin the sub-task execution (step 3).

Figure 3.3 shows the node activation process when the robots decide to work on the same sub-task: in this scenario both robots choose to work on placing the bread for the sandwich. Initially (step 1) the nodes for PLACE-bread on both robots check the status of the peer nodes and then wait for their status message (step 2). The response messages indicate that both robots plan to work on the same node, but have a timestamp indicating which robot first initiated the activation. The robot that has

the earliest activation timestamp would then activate its node (steps 3-4), while the other robot lowers its activation for the same sub-task (step 5). This enables another node in robot 2's network (e.g. PLACE-cup) to get a higher activation level, and thus to begin working on another part of the task (step 6).

From both the overlapping and non-overlapping examples, we see that the architecture is able to complete the tasks in a joint manner with minimal disruptions to the execution. Altogether, this architecture provides an efficient and compact encoding of tasks with various types of constraints (such as sequential, non-ordering, and alternative paths of execution) and allows the robots to dynamically decide which execution path to follow using an activation spreading mechanism that adapts to different environmental conditions.

3.2 Summary

The goal of the proposed work is to develop a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for both multi-robot and human-robot teams. In an effort to develop such a generalized task structure, several major extensions are proposed in this work which utilize our previously developed control architecture as their backbone. Many capabilities of the previously developed architecture are referenced and extended throughout the remaining chapters of this work.

The previously developed distributed multi-robot control architecture allows for dynamic allocation of tasks between multiple robots as well as for opportunistic task execution given different environmental conditions [4]. The architecture uses a behavior-based paradigm [37] and it provides an efficient and compact encoding of tasks with various types of constraints (such as sequential, non-ordering, and alternative paths of execution), allowing the robots to dynamically decide which execution path to follow using an activation spreading mechanism that relies on environmental conditions. Many execution constraints can be incorporated into a single, complex, hierarchical task representation. In order to maintain communication and connectivity between the nodes in a task tree, each node in the architecture maintains a state which is used to perform top-down and bottom-up activation spreading that ensures the proper execution of the task given the constraints.

To enable cooperative execution of team tasks, each robot is equipped with its own instance of the task tree structure, identical to that of the other robots, which encodes the joint team task. Equivalent nodes in the task structures across robots are called *peers*. These peers are the means of communication between the robots and allow nodes to keep track of other robots' progress on the task. By utilizing this setup, the robots are able to maintain and communicate the states of all of the nodes to their corresponding peer nodes on the other robots in order to ensure that the robots can work collaboratively to complete the task in a manner that follows its constraints.

Chapter 4

Learning of Complex-Structured Tasks from Language Instruction

This chapter presents a novel approach to robot task learning from language-based instructions, which focuses on increasing the complexity of task representations that can be taught through verbal instruction. The major proposed contribution is the development of a framework for *directly mapping a complex verbal instruction to an executable task representation*, from a single training experience. The executable task representation is based upon a previously developed framework which is described in Section 3.1. The verbal instruction method can handle the following types of complexities: 1) instructions that use conjunctions to *convey complex execution constraints* (such as alternative paths of execution, sequential or non-ordering constraints, as well as hierarchical representations) and 2) instructions that use prepositions and multiple

adjectives to *specify action/object parameters relevant for the task*. Specific algorithms have been developed for handling *conjunctions*, *adjectives*, and *prepositions* as well as for translating the parsed instructions into parameterized executable task representations. The work describes validation experiments with a PR2 humanoid robot learning new tasks from verbal instruction, as well as an additional range of utterances that can be parsed into executable controllers by the proposed system. The development of this verbal instruction system allows the proposed generalized task structure, modified from Section 3.1, to be utilized to teach robots to perform tasks through verbal instruction. This outcome is one of the main components needed to enable collaboration between humans and robots for complex, hierarchical tasks using a generalized task structure.

4.1 Learning of Task Controllers from Verbal Instruction

We assume that the robot is equipped with a set of basic skills (or behaviors), each of which has a mapping to the teacher's instruction vocabulary. In addition, the robot has knowledge of various objects (and their attributes) that can be appropriately recognized and manipulated. The teacher's instruction is parsed and mapped into an executable controller as outlined in Figure 4.1.

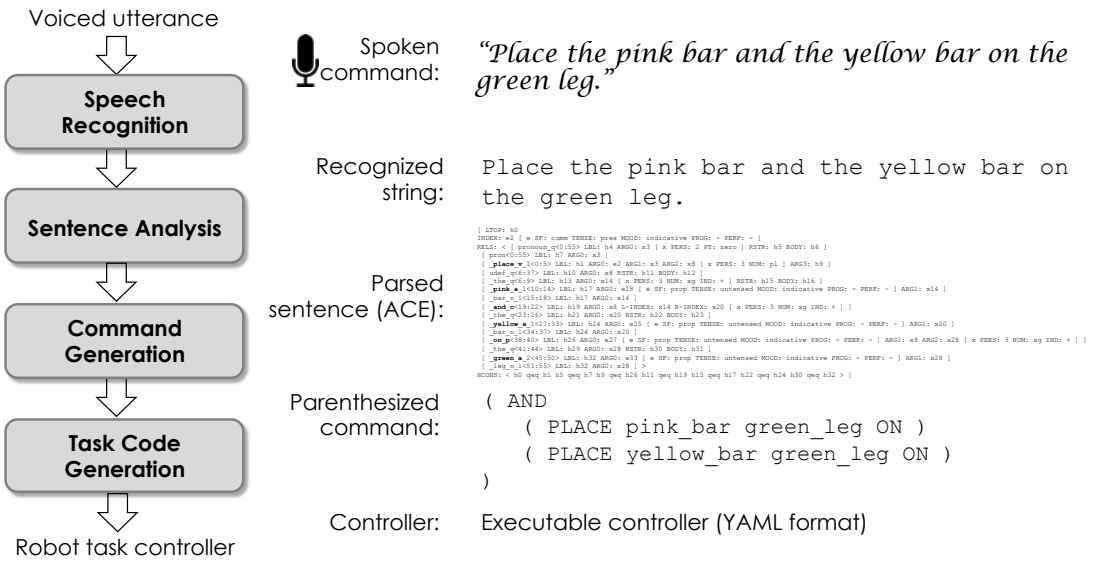


FIGURE 4.1: Stages of parsing verbal instructions to controllers.

The *speech recognition* module takes as input a voice command from the user through the PocketSphinx [93] package in ROS [94] and produces a string representing the user's command.

The *sentence analysis* module takes the command string and produces a parsed representation of the command. This is then used by the *command generation* module, which produces a parenthesized version of the command. In turn, this is next used by the *task code generation* module that produces the executable controller, in the form of a YAML file. These modules are described in more detail below.

For *sentence analysis*, in order to represent the semantic roles of each used utterance, we use *minimal recursion semantics* (MRS), which are based on the English Resource Grammar open source project [95, 96]. In MRS the links between meaningful words are shown through argument roles and handle links, which can capture some scope

ambiguity. To extract the MRS representations of the verbal command, we used the Answer Constraint Engine (ACE) tool available at [70]. The engine also tags each word with part-of-speech information, which will be used in the next step of our processing and will be described in detail below.

The *command generation* module takes as input the parsed sentence representation, and produces a parenthesized form of the command (Figure 4.1) as follows. The semantic representation produced by ACE is parsed to extract relational information for each of the words in the sentence, which is then organized in a dictionary of relations (RELS) with the following structure: *Handle: [category, word,[arguments]]*, as shown in Figure 4.2. The *Handle* is a unique identifier given to the word by the ACE analyzer consisting of a letter and a number (the *ARG0* of each relation, for example, *x10* corresponds to the noun *bar*). The *category* represents the part of speech of the word (e.g. noun, verb, conjunction, etc.) and the *arguments* is a list of relations (*ARG0-ARGn*) to other words in the sentence. Each part of speech has a different number of arguments, as follows. Conjunctions have two arguments, each pointing to the items that they connect. For example, the conjunction *'and'* has arguments *x10, x16* (representing the first, and respectively the second noun *'bar'* it connects). Verbs have three arguments, but only the second one (*ARG2*) is relevant for our purpose: this argument points to the handle of either a noun or a preposition that links several nouns. For example, the verb *'place'* has *ARG2=x4*, which is the conjunction *'and'* that links two nouns. Prepositions have two arguments, but only the second one (*ARG2*) is relevant, indicating the object of the preposition. For

instance, the preposition *'on'* has *ARG2=x24*, which is the noun *'leg'*. Adjectives have only one argument *ARG1*, which indicates the object they refer to (e.g., adjective *e21*, representing *'yellow'*, refers to relation *x16*, which is the noun *'bar'*). Nouns do not have any arguments, except for their *ARG0* name.

```
{'e2': ['verb', 'place', ['i3', 'x4', 'h5']],
'e14': ['adjective', 'pink', ['x10']],
'x10': ['noun', 'bar', []],
'x4':  ['conj', 'and', ['x10', 'x16']],
'e21': ['adjective', 'yellow', ['x16']],
'x16': ['noun', 'bar', []],
'e23': ['preposition', 'on', ['x4', 'x24']],
'e29': ['adjective', 'green', ['x24']],
'x24': ['noun', 'leg', []]}
```

FIGURE 4.2: Dictionary of relations (RELS) extracted for command generation.

Algorithm 1, *MRSCrawling(sentence, RELS)* takes as input the parsed sentence produced by ACE and the *RELS* dictionary and starts by finding the sentence index (*e2* in our case), and the semantic relation to which it corresponds. In our sentence this is represented by the verb *'place'*, which should correspond to one of the robot's basic behaviors (lines 1-3). Next, adjectives are appended to their corresponding nouns, indicated by their *REL1* argument, as shown in Algorithm 2, *ConnectAdjectives(RELS)*. After this processing, the *word* fields for the nouns in the dictionary become *'pink_bar'*,*'yellow_bar'*, and *'green_leg'*.

Algorithm 3, *HandlePrepositions(RELS)* processes all the prepositions in the dictionary to build relations of the type $<$ *subject object preposition* $>$. The object of the

Algorithm 1 $MRS\_Crawling(sentence, RELS)$

```
1: index = sentence.INDEX //chose sent. index
2: look for semantic relation (in RELS)
   with rel.ARG0 == index
3: verb = rel //(this is the action verb)
4: ConnectAdjectives(RELS)
5: HandlePrepositions(RELS)
6: command = BuildCommand(RELS, verb)
```

Algorithm 2 $ConnectAdjectives(RELS)$

```
1: for all rels in RELS do
2:    if rel.category == adjective then
3:       //get handle for noun
4:       noun_handle = rel.ARG1
5:       //append adjective to noun
6:       noun_handle.word += "_" + rel.word
7:    end if
8: end for
```

preposition is obtained from the preposition's *ARG2*. The subject of the preposition is found in *ARG0*, and can be either a single noun or a conjunction (as in the example: both the pink bar and yellow bar are the subjects placed on the green bar). Conjunction objects are recursively found by Algorithm 4, *FindAllSubjects(rel)*, which takes as input one of the relations in the *RELS* dictionary. After this processing stage, the *word* field for the preposition's subject nouns in the dictionary become *'pink_bar green_leg on'*, *'yellow_bar green_leg on'*.

The *controller construction* module takes as input the parenthesized form of the task representation and translates it into a robot controller that can automatically be executed by the robot, using the procedure shown in Algorithm 5. The input to the

Algorithm 3 *HandlePrepositions*(*RELS*)

```
1: for all rels in RELS do
2:   if rel.category == preposition then
3:     main_subject = rel.ARG1 //get source of preposition
4:     subjects = FindAllSubjects(main_subject)
5:     object = rel.ARG2 //get object of preposition
6:     for all sbj in subjects do
7:       //append preposition and subject noun
8:       sbj.word += "" + object.word + "" + rel.word
9:     end for
10:  end if
11: end for
```

Algorithm 4 *FindAllSubjects*(*rel*)

```
1: for all arg in rel.ARG0 do
2:   if rel.category == noun then
3:     Appendreltosubjects //append source noun
4:   else if rel.category == conjunction then
5:     //recursively find conjunction-connected subjects
6:     subjects_list = FindAllSubjects(rel.arg)
7:     Append subjects_list to subjects
8:   end if
9: end for
```

algorithm is a fully parenthesized string (Figure 4.1) and the output is a node with its corresponding list of children. The *GetCrtToken()* function just reads from the string the next relevant element (either a parenthesis, a node label such as *THEN*, *OR*, etc. or a parameter such as *cup*, *tea*, etc.). The *AdvanceToNextToken()* advances to the next token in the string command. The algorithm proceeds with extracting the opening parenthesis, then the node label, and initializes the list of children to be empty (lines 1-6). After this, the algorithm repeatedly processes the next tokens until reaching a closing parenthesis *')'*. The new tokens could be either new nodes

Algorithm 5 CreateNode(cmd)

```
 1: new node  // create new node object
 2: token = GetCrtToken(cmd)  // must be '('
 3: AdvanceToNextToken(cmd)
 4: node.Label = GetNextToken(cmd)
 5: AdvanceToNextToken(cmd)
 6: node.ChildrenList = emtpy
 7: repeat
 8:    token = GetCrtToken(cmd)
 9:    if token == '(' then
10:       // child is a new node
11:       child = CreateNode(cmd)
12:       node.AddToChildren(child)
13:    else if token == ')' then
14:       // the end, do nothing
15:    else
16:       // child is a parameter
17:       child = token
18:       node.AddToChildren(child)
19:    end if
20:    AdvanceToNextToken(cmd)
21: until token == ')'
22: return node
```

themselves (lines 8-12), or behavior parameters (lines 8, 15-18). When the token is a closing parenthesis *')'*, there is nothing to be done and the loop stops (lines 13-14, 21). The newly created node with the list of its children is returned (line 22).

4.2 Learning of Basic and High-Level Tasks

While single instructions can have a high-degree of complexity (as shown in Section 4.1), they may only represent a part of a larger task to be learned. We developed an approach to allow the robot to learn tasks composed of multiple instructions, as

well as to build up from those tasks in order to learn even more complex representations. In this work we differentiate between *basic* and *high-level* tasks. *Basic tasks* are built entirely from combinations of low-level skills (behaviors) of the robot. *High-level* tasks are built from combinations of already existing basic tasks, or other high-level tasks previously learned.

The process for learning *basic tasks* consists of three main steps that run in a loop until the teacher is done with teaching the task. In each iteration of the loop, the robot receives a verbal instruction from the teacher, which is next processed and converted into an executable controller, as described in Section 4.1. An instruction can be as simple as a basic command (e.g., *Place the bread*), or could have a higher degree of complexity (such as *Place the cup, then the sugar and the tea*). Third, the newly learned step is executed by the robot before the teacher provides the next instruction. When the teacher finishes the training, all the individual steps (if more than one) are combined into a single task representation, which consists of a *THEN* root node, whose children are the nodes representing each individual step of the task in the order in which they have been presented. To facilitate a flexible and natural interaction during learning, both the human and the robot use specific verbal cues to indicate the following: 1) by the human: when a training task begins, the name of the task, the end of the task, 2) by the robot: confirmation of proper instruction received, request of names for newly trained tasks, requests for new steps, or request to repeat the task if the command is not properly understood by the speech recognition module.

Examples of full dialogue sequences between the human and the robot during training are presented in Section 4.3.

To learn *high-level* tasks, the teacher provides instructions that combine tasks already existing in the robot's repertoire. The process runs in a loop in which verbal instructions are provided, then processed and converted into an executable controller, similar to the process for basic tasks. If the instruction includes reference to a task previously learned, the robot does not execute the individual command after it is received, but rather waits for the training to finish. This is an arbitrary choice we made to distinguish the two cases, but has no influence on the learning process. At the end of the training, to build the task representation for the entire task, all the individual steps are combined into a single task representation as in the case for the basic tasks.

4.3 Experimental Validation

4.3.1 Robot Experiments

We validated our approach with a PR2 humanoid robot in two scenarios: a household environment (in which we validate the learning of *basic* and *high-level* tasks) and an IKEA EKET base frame construction environment (in which we validate the use of prepositions and adjectives to parameterize the robot's behaviors).

FIGURE 4.3: Experimental setup. Left: household, Right: IKEA EKET base.

4.3.1.1 Household Environment

In the household environment, the robot was taught to perform the following tasks: 1) two *basic* tasks, one for making tea, and one for making a sandwich and 2) a *high-level* task called tea-time that consists of the two basic tasks for making tea and a sandwich, using toy objects that represent *bread*, *meat*, *lettuce*, *tea*, *cup*, and *sugar*. The robot is equipped with a *Place(Object)* behavior, that is parameterized for the different objects. Since the focus of this work is on the learning approach, for these experiments the initial and final locations of the objects are pre-defined. The experimental setup is shown in Figure 4.3 (left).

The first two training experiments were focused on teaching the two basic tasks. Table 4.1 shows the verbal communication between the human and the robot during the training experiment, as well as the robot's actions during this process. At the end of this training, the robot has a new task in its repertoire, called **sandwich** (left branch of the root *AND* node in Figure 4.4), which can be executed at any time or used as a part of a higher-level task. During the practice of the command *"Place*

the meat or the lettuce", the robot chose to select the *meat* out of the two possible options, due to the fact that it was closer to the robot's gripper than the *lettuce*.

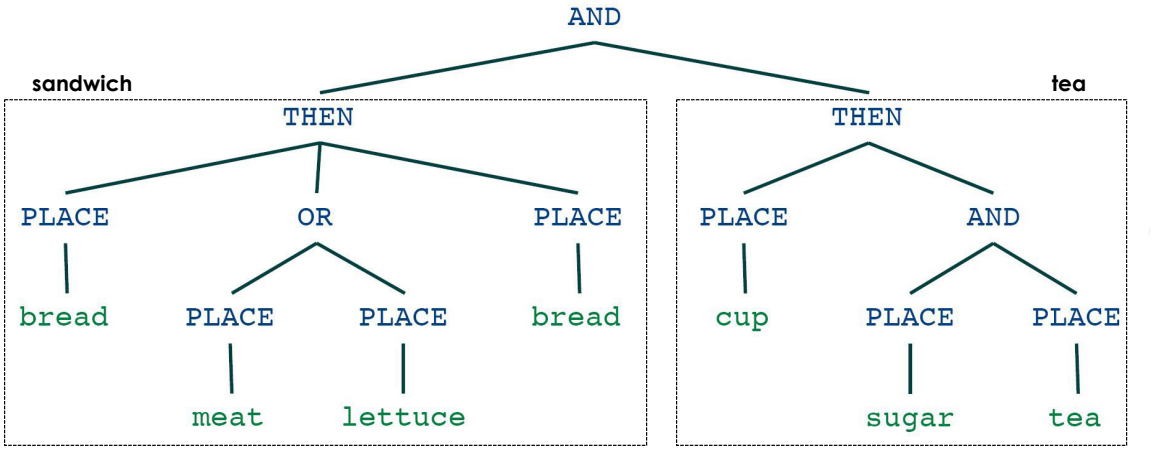


FIGURE 4.4: Hierarchical representation for the household tasks. The left sub-tree is the **sandwich** task. The right sub-tree is the **tea** task. These two tasks are combined into a *higher-level* **tea-time** task represented by the entire tree.

Using a similar process, in the second experiment the robot is trained a task for making **tea**, the representation of which is shown in Figure 4.4 (right branch). During the practice of this task, the ordering constraints have been enforced, with the robot placing the *cup* first and then proceeding to the next two objects. Since placing the *sugar* and *tea* do not have any ordering constraints, the robot chooses a path of execution based on the state of the environment, placing the *tea* first as it was closer.

The third experiment consists of teaching the robot a *higher-level* task, which combines tasks that are already known to the robot. Table 4.2 shows the verbal communication between the human and the robot during training. The task representation that is learned by the robot is shown in Figure 4.4. Since the teacher has provided a

Step	Dialogue and Actions
1.	**H:** *"Make recipe."*
2.	**R:** *"Ok, make recipe. What is the name of the recipe?"*
3.	**H:** *Make* ***sandwich.***
4.	**R:** *"Ok, recipe for* ***sandwich****. Please start with the instructions."*
5.	**H:** *"Place the left bread."*
6.	**R:** *"Ok, place the left bread."* Builds controller and executes the task: (PLACE Bread) **R:** *"What's the next step?"*
7.	**H:** *"Place the meat or the lettuce."*
8.	**R:** *"Ok, place the meat or the lettuce."* Builds controller and executes the task: (OR (PLACE Meat) (PLACE lettuce)) **R:** *"What's the next step?"*
9.	**H:** *"Place the right bread."*
10.	**R:** *"Ok, place the right bread."* Builds controller and executes the task: (PLACE Bread) **R:** *"What's the next step?"*
11.	**H:** *"Store recipe."*
12.	**R:** *"Ok, recipe for* ***sandwich*** *is complete and stored."* Final task representation stored (Fig. 4.4, left *THEN* branch).

TABLE 4.1: Human-robot dialogue and interaction during training of the **sandwich** task.

single instruction that combines two existing tasks by an *AND* conjunction, the full task representation consists of a single *AND* node as the root of the task tree.

4.3.1.2 IKEA EKET Base Assembly

In the previous scenario, the robot's *Place(Object)* behavior was limited to a single parameter, which was a particular object which needed to be placed. By enabling the parsing of prepositions and adjectives, the behaviors can be parameterized with destination location (e.g., *on the green leg*) as well as specifics of the objects involved

Step	Dialogue and Actions
1.	**H:** *"Make recipe."*
2.	**R:** *"Ok, make recipe. What is the name of the recipe?"*
3.	**H:** ***Tea time.***
4.	**R:** *"Ok, recipe for* ***tea time****. Please start with the instructions."*
5.	**H: Sandwich** and **tea**
6.	**R:** *"Ok,* ***sandwich*** *and* ***tea****. What else?"* Builds controller for **sandwich** and **tea** (Figure 4.4).
7.	**H:** *"Store recipe."*
8.	**R:** *"Ok, recipe for* ***tea time*** *is complete and stored."* Executes the task.

TABLE 4.2: Human-robot dialogue and interaction during training of the **tea-time** task.

(e.g., *pink leg*). The task consists of building an IKEA EKET base that has two bars that need to be placed on two legs. One of two tops (purple and orange) can be selected to be put on top of the built base. For easy recognition by a vision system and to showcase the use of adjectives, the legs were painted with green and blue, and the bars were painted with yellow and pink. Table 4.3 shows the human instructions (without the robot's responses, which follow the same pattern as in the previous examples).

Step	Human Instructions and Robot Actions
1.	**H:** *"Place the green leg in front of you."*
2.	**H:** *"Place the pink bar and the yellow bar on the green leg."*
3.	**H:** *"Place the blue leg onto the base."*
4.	**H:** *"Place the orange top or the purple top on the base."*

TABLE 4.3: Human-provided instructions during training of the **EKET** task.

Figure 4.5 shows the hierarchical representation of the learned task. During task execution, the location information provided by the prepositions (e.g., *FRONT-OF,*

ON, ONTO) is mapped to specific positions of the source object with respect to the destination object, based on a pre-determined table. Our current vision system does not yet provide pose information for the objects and the offsets of the prepositions give rough placement positions. Therefore, at task execution the robot asks a human user for assistance to precisely position the objects with respect to each other before making the final assembly. In future work this will be addressed by integrating a vision-based system that provides pose information as well as by specifying locations in a finer grain of detail, for example that a particular side of a bar should fit on a particular side of the leg.

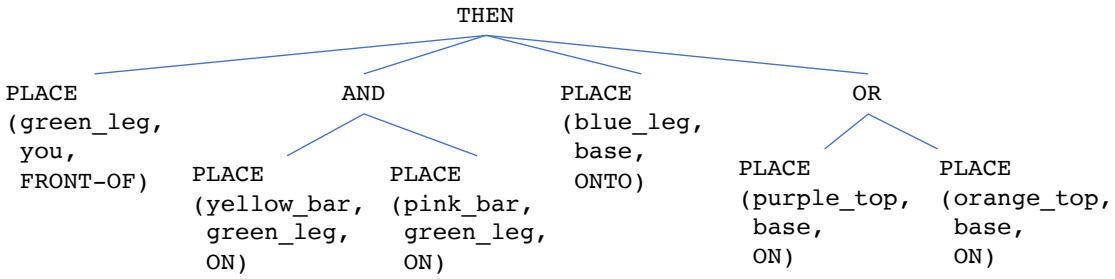


FIGURE 4.5: Representation of the learned IKEA EKET assembly task. The PLACE nodes contain the parameterizations for each object, i.e. the destination location and specifics of other objects involved.

4.3.2 General-Purpose Task Learning Experiments

This section provides additional results that demonstrate in more detail the full capabilities of the approach for parsing language instructions to controllers. These tasks have not been validated on a robotic system, but show the representational power of the approach.

4.3.2.1 Complex Task Execution Constraints

This section presents additional examples of complex task representations that can be constructed from a single verbal instruction. Figures 4.6-4.8 show the verbal instructions and the resulting task trees generated from our system for several instructions.

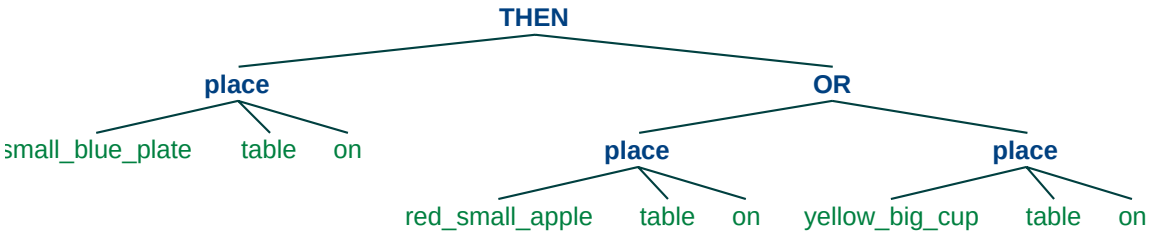


FIGURE 4.6: Instruction: "Place the blue small plate then the small red apple or the big yellow cup on the table."

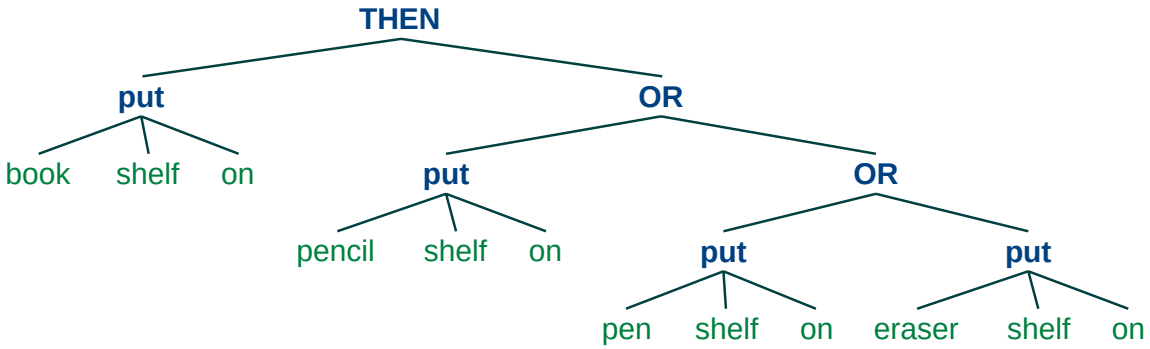


FIGURE 4.7: Instruction: "Put the books then the pencil or the pen or the eraser on the shelf."

All these examples include temporal sequencing constraints (shown by the *THEN* nodes), non-ordering constraints (shown by the *AND* nodes), as well as alternative paths of execution (shown by the *OR* nodes). In order to learn these constraints using demonstrations/instructions that solely rely on sequential commands, the system would need to be provided with multiple demonstrations that illustrate all the alternative ways of execution and sequencing/non-sequencing constraints. The proposed

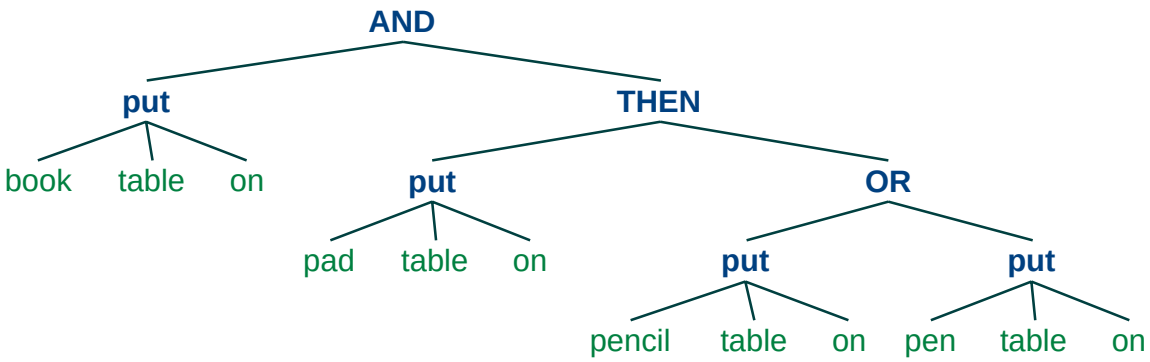


FIGURE 4.8: Instruction: "Put the book and the pad then the pencil or the pen on the table."

method of mapping verbal instructions into controllers that encode the constraints provided by the *THEN, AND,* and *OR* conjunctions enables complex execution constraints to be conveyed to a robot in a single instruction.

4.3.2.2 Use of Adjectives

The proposed system can handle all descriptive adjectives, and multiple adjectives can be used to refer to the same noun. Positive, comparative, and superlative are also successfully parsed. In contrast, the following adjectives are not currently handled: quantitative adjectives (*some, few, all*), demonstrative adjectives (*this, that, these*), possessive adjectives (*my, your, his*), distribute adjectives (*each, every, either*). Handling these types of adjectives is a topic for significant future work. Examples of sentences that use combinations of descriptive adjectives are shown in Figure 4.9.

Push the large tall chair around the small pretty table
`(push tall_large_chair pretty_small_table around)`

Move the tool above the table
`(move tool table above)`

Put the small yellow book under the brown round table
`(put yellow_small_book brown_table under)`

Move the sharp tool near the tiny red box
`(move sharp_tool red_tiny_box near)`

Move the big purple ball from the tiny red table
`(move purple_big_ball red_tiny_table from)`

Chase the big man to the right door
`(chase big_man right_door to)`

FIGURE 4.9: Sample sentences using prepositions.

4.3.2.3 Use of Prepositions

Given the nature of verbal instructions that we are interested in providing, the focus is on providing location information regarding placing or positioning of objects for a task. Our system can handle prepositions related to locations, such as those in the following list:

```
with, at, from, into, during, against, among, towards, upon, in,
on, by, over, through, of, throughout, to, for, about, after, under,
within, aboard, next to, in front of, along, across, behind, beyond,
but, up, out of, out, around, down, off, above, near, below, beside,
beyond, inside, onto, opposite, outside, underneath, unto, adjacent
to, ahead of, as of, other than, outside of, as far as
```

Figure 4.9 shows sentences that use such prepositions.

4.4 Conclusion & Summary

This chapter described a novel approach to transfer complex task knowledge from a human user to a robot, with the goal of exploiting the richness of natural language instructions in order to increase the complexity of task representations that a robot can learn. In particular, the focus was on learning tasks which *convey complex execution constraints* (such as alternative paths of execution, sequential or non-ordering constraints, as well as hierarchical representations), as well as on *enabling behavior parameterization* through the instruction. Specific algorithms have been developed for handling *conjunctions*, *adjectives*, and *prepositions* as well as for translating the parsed instructions into parameterized executable task representations. The method also enables learning increasingly complex tasks from multiple instructions. Experimental validation using a PR2 humanoid robot has been performed, demonstrating the feasibility of the proposed method to learn multiple task representations with complex constraints. Additionally, examples of parsed trees outside of the robot domain are provided in order to demonstrate the versatility of the method. This verbal instruction system allows the proposed generalized task structure to be utilized to teach robots to perform tasks through verbal instruction, as specified in one of the main contributions presented in Section 1.4.

Chapter 5

Human-Robot Collaboration and Dialogue for Fault Recovery on Hierarchical Tasks

Robotic systems typically follow a rigid approach to task execution in which they perform the necessary steps in a specific order, but fail when having to cope with issues that arise during execution. To address this issue, this chapter proposes an approach that handles such issues through dialogue and human-robot collaboration. The main contribution of the proposed approach is a hierarchical control architecture built upon the work presented in Section 3.1 which *1) autonomously detects and is cognizant of task execution failures*, *2) initiates a dialogue with a human helper to obtain assistance*, and *3) enables collaborative human-robot task execution through*

extended dialogue in order to *4) ensure robust execution of hierarchical tasks with complex constraints, such as sequential, non-ordering, and multiple paths of execution.* The architecture ensures that the constraints are adhered to throughout the entire task execution, including during failures. The recovery of the architecture from issues during execution is validated by a human-robot team on a building task.

The incorporation of a fault recovery system which is able to detect and inform users of failures and resolve them through dialogue allows for a more robust task allocation scheme. This type of resolution is necessary to ensure that collaborative tasks for human-robot teams are completed successfully. This contribution allows for the proposed generalized task structure to be utilized in complex, hierarchical tasks which are prone to failures as well as those which require collaboration between humans and robots.

5.1 Control Architecture with Fault Recovery

In this chapter we extend the control architecture presented in Section 3.1 to incorporate a dialogue-based management system of task faults capable of autonomously detecting issues and resolving them through human-robot collaboration. Section 5.1.1 describes the additions made to this architecture to incorporate the dialogue-based fault recovery. Section 5.1.2 provides details about the dialogue module used by the robot to initiate an extended dialogue with the human to resolve faults. Section 5.1.3

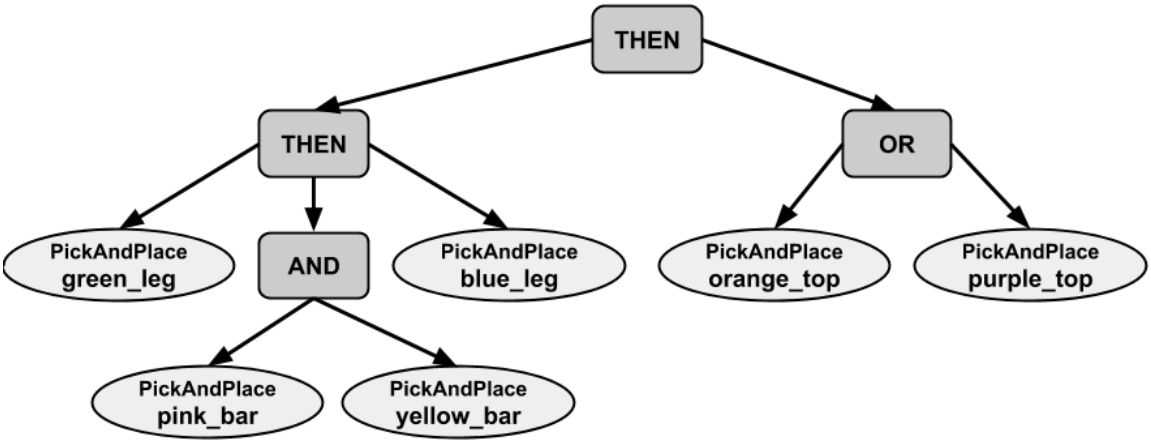


FIGURE 5.1: The task tree for the IKEA EKET building task. The dark gray rectangles are goal nodes and the light gray ovals are behavior nodes.

details how the robot uses on-board sensors to autonomously detect unexpected situations resulting in faults. These additions result in a robust control architecture which is capable of detecting and recovering from various faults during the execution of complex, hierarchical tasks through the use of human-robot collaboration and dialogue.

5.1.1 Interfacing with the Control Architecture

In order to allow the architecture to handle interruptions that come from the fault detection system, the update loop of the nodes from Section 3.1 was modified by adding a checking mechanism that allows the loop to continue as normal unless a failure is detected. In the case of a detected fault, a Robot Operating System (ROS) message is published to the corresponding node's *issue* topic. Once the node receives such a message, the node's *issue* callback function is triggered (Figure 5.2).

In this function, a ROS message is published on the *dialogue* topic to initialize the dialogue that corresponds to the specific failure that was detected. This initiates the dialogue between the robot and human and allows the human to provide assistance, as described in Section 5.1.2. After the dialogue is initialized, a while loop stops the current behavior in the architecture, as well as the physical motion of the robot, from finishing until a resolution has been reached through the dialogue between the robot and human. Since the node that the robot is working on is active at the time of the detected failure, no other nodes can be activated until that node is done or reset, allowing the entire architecture, and therefore task progress, to be paused from within a single node. This pause ensures that no task constraints are broken during the handling of the fault.

Once a resolution message is received from the dialogue system, changes are made to the node's state based on the type of resolution. If the resolution involves either the human, robot, or both to complete the task then the node's state is set to *done* and its *activation level* is set to zero. In the case that the resolution is *human_finish*, the human will perform the required work to complete the task. If the resolution is *robot_finish*, then the robot will continue on with the remaining work required to finish the task, after being briefly assisted by the human (i.e. the human hands the robot an object that is out of its reach). Once the human completes the action, the robot is able to finish the task without further help. This assistance varies based on the task at hand and the issue found. If the resolution is *collab_finish*, then the human must work together with the robot simultaneously to complete the task (i.e.,

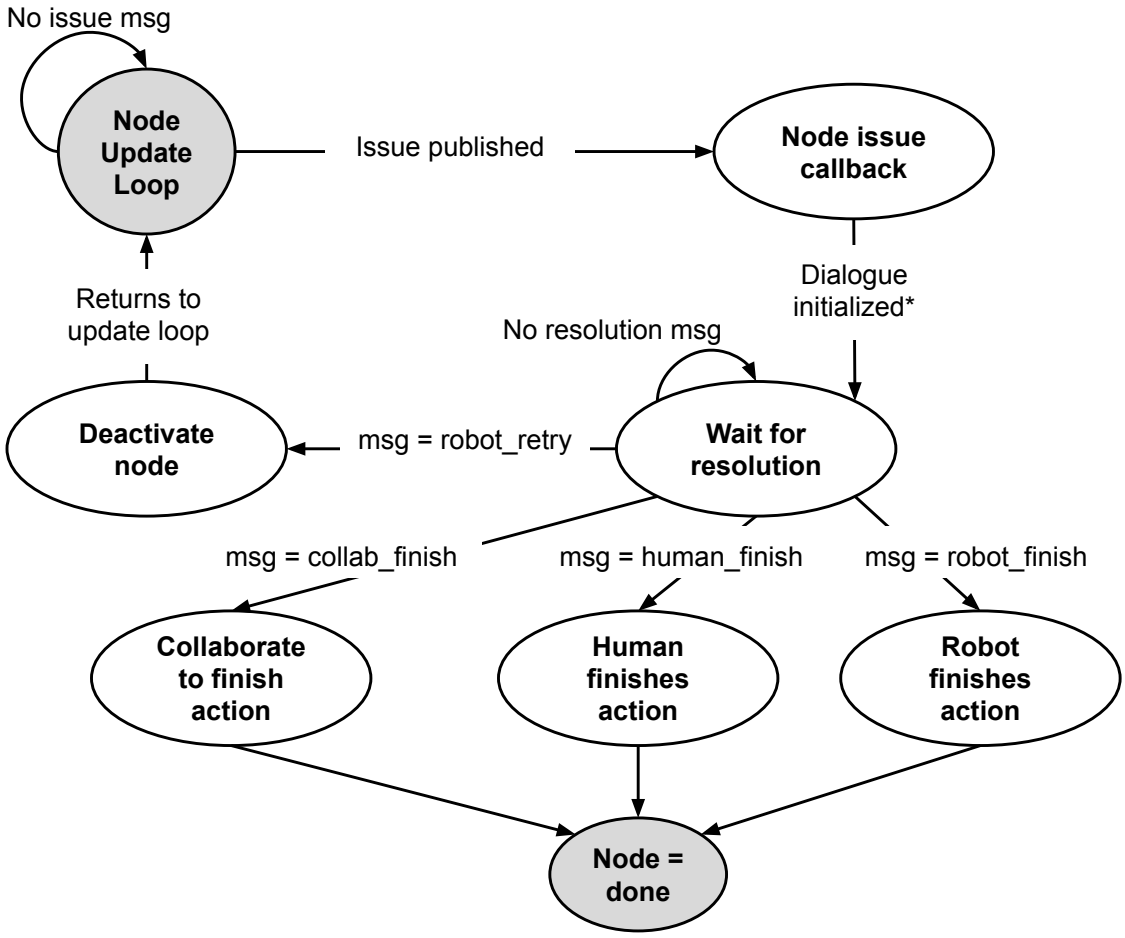


FIGURE 5.2: State machine diagram of architecture flow upon issue detection. The *Node Update Loop* state is the starting state in which the state machine stays until a issue message is published. *Node=done* is the final state which signals that the node's behavior has finished executing. The *Dialogue initialized** transition is where the dialogue flow (Figure 5.3) interfaces with this state machine. The resolution message can trigger different actions.

the human must hold and align an object as the robot connects another object). This type of resolution requires both agents to work together at the same time in order to fully complete the task.

Lastly, if the robot is required to retry the execution (*robot_retry*), the node gets deactivated. This deactivation sets the node's state back to what it was before the

node was activated, thereby ensuring that the task constraints encoded by the task tree are still upheld after the conflict is resolved. The node's state is set to *not done* and its *activation level* is reset to its original level upon activation. If a node is deactivated, it can be chosen for activation at a later time and the robot can attempt the execution of that behavior again.

The development of the pause and deactivate functionalities ensures that our control architecture is able to maintain the task constraints during the entire task execution. Additionally, the various resolution messages allow the architecture to utilize multiple ways to resolve a conflict. Furthermore, these different resolutions illustrate that the architecture is able to handle different levels of conflict. The resolutions which result in the node being set to done only require a temporary pause of the architecture until the work is completed, illustrating handling of a minor fault. On the other hand, the resolution forcing the robot to retry the task illustrates a major fault as it requires both pausing and deactivating the node, which in turn resets part of the task tree. Thus the addition of the dialogue-based management system increases the robustness of the architecture and allows for fault recovery during the execution of complex, hierarchical tasks.

5.1.2 Dialogue Module

When a ROS message is published to the *dialogue* topic, the dialogue is initialized, as shown in Figure 5.2. This initiates a communication between the robot and human.

The high-level flow-chart for the initiated dialogue is shown in Figure 5.3. This flow-chart illustrates the major interactions that occur between the human and robot that encompass the extended dialogue. There are two main components to this interaction that are specific to the failure that was detected:

- **Detected issue:** Name of the issue detected.
- **Action:** The action that needs to be performed.

Additionally, there are two internal checks in the interaction which affect the outcome of the dialogue: *1) Human collaboration required? and 2) Should robot complete task now?* The first one checks if human-robot collaboration is required to complete the task. This means that the human and robot must work together simultaneously to finish the task. An example of this is the *positioning* issue where the human must help to align an object while the robot connects another object. Because this type of issue requires human assistance, the robot automatically requests help from the human. This check will return *no* if the task can be completed by either the human or robot alone or with minimal, asynchronous assistance from the other. The second check determines whether the robot should complete the task at the current time. And example of where this check returns *yes* is the *unreachable* issue. In this case, the human has handed the object to the robot, so the robot should finish the task as it is now holding the object. Examples of where this check returns *no* are the *dropped* and *missed* issues. In these cases, the human has to replace the object to a location

graspable by the robot, so the robot must restart the task from the beginning once the objects are reset.

If the issue does not require human collaboration to complete the task then the robot will ask if it should complete the task. If the human replies with a *yes*, then the robot will briefly provide the human with instructions on how to reset the objects to enable the robot to complete the task on its own. Then, depending on the second check, the robot will either finish the task at the current time or inform the human that it will retry the task again later and the corresponding resolution message is published. If the human replies with a *no*, then the robot will ask if the human will complete the task. If the human again responds with *no*, the resolution message is published to enable the robot to retry and the human is notified. If the human responds with *yes*, the robot will thank the human and the resolution message for the human completing the task is published.

Simply following the high-level dialogue flow utilizing these main components and internal checks provides a simple way for new issues to be added into the system. Although specific details (such as the exact dialogue exchanges) will vary based on the issue, this flow outlines all of the necessary interactions that would occur between the human and the robot for any simple issue that could be added. This further emphasizes the generality of our the proposed dialogue-based management system for fault recovery.

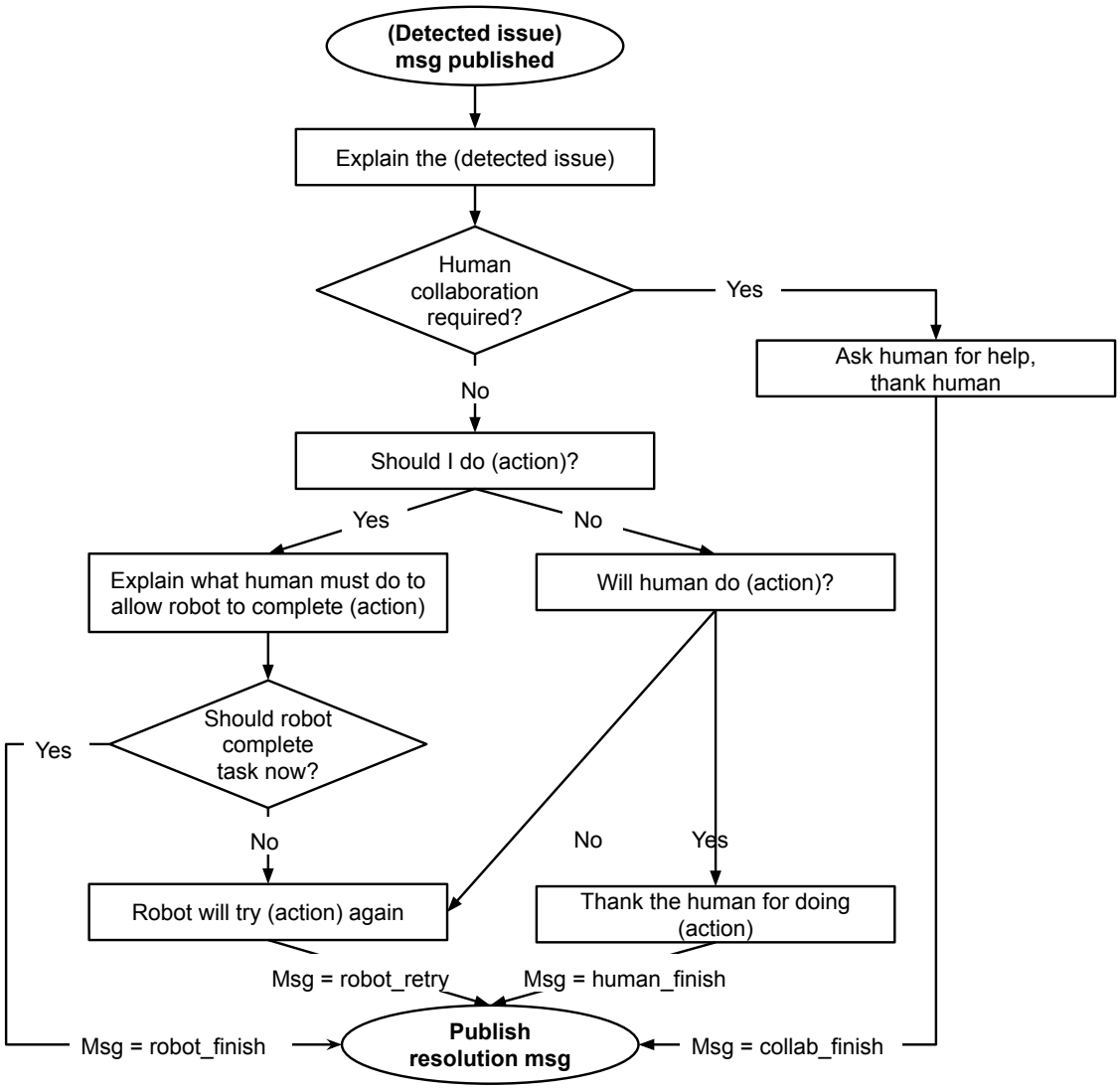


FIGURE 5.3: High-level flow-chart of the dialogue initiated between robot and human when an issue is detected. Details of the dialogue are filled in depending on which specific issue was detected.

To illustrate how the complete dialogue flow works with concrete examples, the faults detected for our assembly scenario (Section 5.2) are summarized below:

- **Missed:** The *missed* issue message is raised when the robot misses an object during pick-up. The robot explains it missed the object and asks to try again. If the human agrees, the robot will ask the human to place the object to its

original position on the table, and says it will try picking it again later. If the human disagrees, the robot asks if the human will place the object. If the human says *yes*, they will place it to the final location. Otherwise, the robot says it will try again later.

- **Dropped:** The *dropped* issue message is raised when the robot drops an object after picking it up. The dialogue flow is exactly the same as in the *missed* case, except the robot explains it dropped the object (instead of missed it).

- **Unreachable:** The *unreachable* issue message is raised if a robot is unable to reach an object. The robot will ask if the human can hand the object to the robot. If the human complies, the robot will grab the object and finish completing the task. If the human refuses, the robot will ask if the human will place the object. If the human says *yes*, the human will place the object to its final location. Otherwise, the robot says it will try again later.

- **Positioning:** The *positioning* issue message is raised if a robot needs assistance with precisely positioning an object as it is placed. The robot will ask the human for help placing the object and thank the human. The motion of the robot is then slowed down and the human can assist with the positioning of the object.

In the *dropped*, *missed*, and *unreachable* cases, the human can choose to help the robot or not and the architecture is able to adjust to both responses and handle the failures accordingly. Additionally, the architecture is able to handle cases which

require human-robot collaboration to complete the task, as seen through the *positioning* issue. Upon completion of the dialogue, a resolution message is published to the node's *issue* callback function and the corresponding node is either reset or set to done as discussed in Section 5.1.1.

In order to accomplish the dialogue, several components are needed. The robot utilizes a on-board speaker and a ROS driver called sound_play [97] to allow it to communicate with the human. In turn, the human speaks into a microphone and the verbal response is registered through a speech recognition engine called PocketSphinx [93]. Once the robot poses a question to the human, a ROS service request is sent to PocketSphinx to listen for the human's response. Once a *yes* or *no* response is recognized, the dialogue flow between the robot and the human continues accordingly.

5.1.3 Fault Detection System

In order for the architecture to detect issues, a fault-monitoring system has been added to each node in the task tree for the base control architecture discussed in Section 3.1. Once a node gets activated, the system begins monitoring for faults during the execution of the node's work. In this work, the monitoring happens during the execution of the *PickAndPlace* node behavior. This behavior performs the following steps in order: *1) move above the pick position, 2) move to the pick position, 3) close the gripper, 4) move back above the pick position, 5) move above the place position, 6) move to the place position, 7) open the gripper, and 8) move back above the place*

position. To ensure the arm is not colliding with objects as it moves between pick and place locations, the arm is moved above (positive z-offset) the pick and place position after opening/closing the gripper.

During this sequence of steps, the monitoring system checks for various fault cases using a combination of the robot's on-board sensors. If a fault is detected, this system publishes a ROS message to the node's *issue* topic, which in turn pauses the architecture and triggers the dialogue with the human. Additionally, for the *dropped* and *missed* cases, the robot's motion along this path is interrupted and the arm is moved to a neutral location to wait until a resolution is reached.

In order to extend this monitoring system to new issues, two main components are required:

- **Start step:** the step along the *PickAndPlace* sequence which starts the monitoring of the new issue.
- **Stop step:** the step along the *PickAndPlace* sequence which stops the monitoring of the new issue.

Based on these components, monitoring of new issues can be performed along any two points in the sequence. However, the sensors used to check if the issue has occurred will vary based on the specifics of the issue. Additionally, these sensors might require specific settings, such as locations of objects in a particular camera. Aside from the

issue-specific settings, only the starting step, stopping step, and issue topic must be defined in order to interface a new issue with the architecture's monitoring system.

For the *unreachable* fault, the starting and stopping steps are the first step in the sequence (i.e. move to *above pick location*). Before this motion occurs, the system checks if the object is within the robot's graspable range using a simple distance check from the robot to the object's initial location as detected from the Kinect on the PR2 robot's head. If the object is out of reach, the system registers an *unreachable* fault and publishes the issue to the node's *issue* topic, which triggers the dialogue between the robot and the human. If the human chooses to hand the robot the object, the robot will extend its arm towards the robot, grab the object, and then finish placing the object starting the motion from the *above place location* step.

For our implementation, a simple color blob detector, implemented with OpenCV [98], is used to find objects in an image. HSV-segmentation of pre-trained color histograms, combined with morphological open/close operations, isolates large regions of color in the image. These regions represent each object. The monitoring system uses these trained colors to identify whether or not the object is in the gripper by running the color blob detector on the RGB image from the PR2's right forearm camera. During the monitoring, the fault detection system searches the image for the color blob of the object's respective color. If the center of the blob is not within a predefined range of values in the image, it registers either the *missed* or *dropped* fault, depending on which part of the motion was being executed when the fault was detected.

The fault-monitoring system checks for a missed fault between the two *above pick location* steps (steps 1-4). If the color blob detection does not detect the object in the correct location in the forearm camera, it registers a missed fault and publishes the *missed* issue to the node's *issue* topic. The system then checks for a *dropped* issue between the second *above pick location* step and the first *above place location* step (steps 4-5). At any point between these steps in the execution, if the color-blob detection does not detect the object in the correct location in the forearm camera, it registers a dropped fault and publishes the *dropped* issue to the node's *issue* topic.

The *positioning* issue is only checked at the first *above place location* step (step 5). The system checks whether or not the carried object is in the list of predefined objects which require assistance for placement. If the object requires help, the system registers a positioning fault and publishes a *positioning* issue message. The robot then slows down the movement from the *above place location* to the *place location* (steps 5-6), which allows the human ample time to align the necessary object as the robot attempts to connect the new object. The robot then moves back to the *above place location* (step 8) and returns to its regular speed.

These issue resolutions illustrate that the proposed architecture is able to handle varying degrees of faults. The *dropped*, *missed*, and *unreachable* failures represent major issues as they require a complete interrupt of the robot motion and the architecture. Without assistance from the human, the robot would be unable to complete the task. On the other hand, the *positioning* issue is a lesser fault. The robot requires assistance

to align the objects perfectly, but neither the robot motion nor the architecture need to be interrupted in this case. Therefore, our dialogue-based management system for fault recovery of hierarchical tasks through the use of human-robot collaboration allows the distributed control architecture to recover from faults of varying degrees.

5.2 Experimental Validation

The proposed architecture has been validated with a robot-human team in a scenario specifically designed to illustrate the key proposed contribution: *a control architecture that 1) autonomously detects and is cognizant of task execution failures, 2) initiates a dialogue with a human helper to obtain assistance, and 3) enables collaborative human-robot task execution through extended dialogue in order to 4) ensure robust execution of hierarchical tasks with complex constraints, such as sequential, non-ordering, and multiple paths of execution.* Most of the proposed additions to the architecture are outlined by general methods, so a concrete scenario with example cases for each addition is utilized to validate the combined functionality of the architecture.

The task used to validate the architecture involves building a modified EKET base from IKEA, with all parts painted in different colors for disambiguation. The task structure is shown in Figure 5.1. Building tasks have inherent constraints due to the way parts fit together. These constraints are reflected by our architecture. The task is performed as follows: *1) place the green leg in front of the robot, 2) attach the pink*

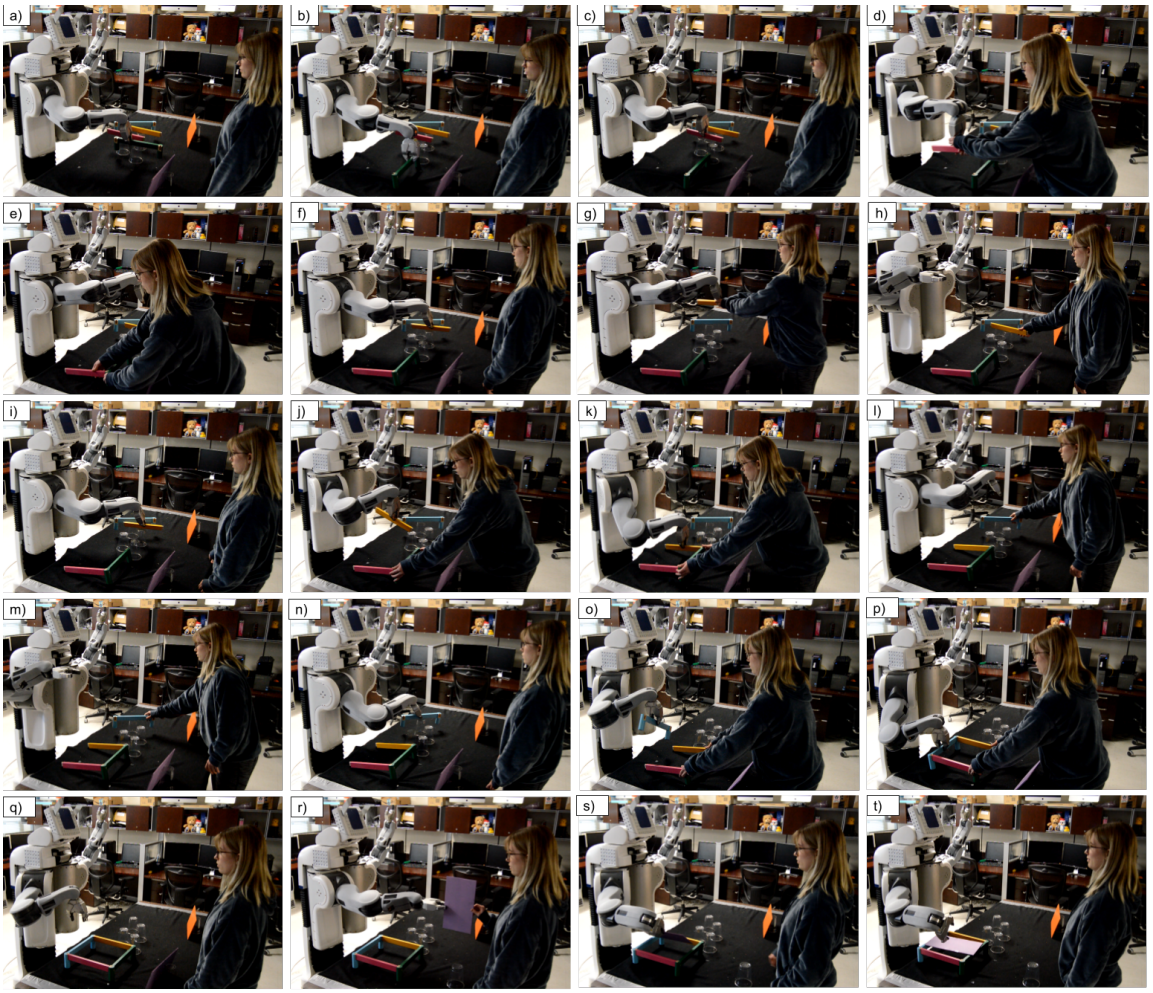


FIGURE 5.4: Execution of the task with issues and assistance provided by the human. In (d) the human steals the pink bar before it is placed, resulting in a *dropped* issue. The human then follows the dialogue to place it. In (g) The human steals the yellow bar after it is picked, resulting in a *dropped* issue. The human then follows the dialogue to allow the robot to pick and place it again. In (j) the robot encounters a *positioning* issue and asks for help from the human. In (l) the human steals the blue leg before it gets picked up, resulting in a *missed* issue. The human then follows the dialogue to allow the robot to try again. In (q) the robot encounters an *unreachable* error. The human then follows the dialogue to hand the robot to the object so it can be placed.

and yellow bars in either order, 3) attach the blue leg, and 4) place either the purple or orange top on top. The ordering of these steps reflect the sequential constraints of the building task. Step 2 reflects the non-ordering constraints of the task since the order of placing the pink and yellow bars does not matter. Step 4 reflects the alternate paths of execution in the task since either one of the tops can be placed. Due to the precise nature of building tasks, multiple steps in the task require assistance from the human to complete, making this a perfect human-robot collaboration task.

In order to accomplish this task, the parts are placed on a table in front of the PR2 robot. The *PickAndPlace* behavior nodes get their respective pick locations from an initial detection of objects using the Kinect's RGB camera by applying the same type of color blob detection utilized on the forearm cameras as discussed in Section 5.1.3. The respective place locations of the *PickAndPlace* behavior nodes are set as pre-specified locations in order to allow the objects to be attached together. End-effector trajectories to the pick/place locations are generated using the MoveIt library [99].

Utilizing this setup, the proposed architecture is validated by running an experiment in which the robot completes the building task. During the experiment, the robot determines the order of actions to take based on the activation spreading mechanism defined in our previously developed architecture [4]. As discussed in Section 3.1, this mechanism creates a dynamic ordering in which to complete the sub-tasks based on the environmental conditions and guarantees this ordering adheres to the constraints of the task. In order to validate the fault recovery of the proposed architecture,

a human simulates each of the possible fault cases by interrupting the task. This validates that the architecture can recover from various types of faults occurring within a single task.

5.2.1 Task Execution

The execution of the experiment is shown in Figure 5.4. In order to illustrate the recovery and collaboration capabilities of the proposed architecture, the human interfered by stealing several objects during the execution of the task, causing the robot to detect the various types of faults handled by the architecture.

The task execution begins with the robot performing the *PickAndPlace(green_leg)* behavior in pictures (a) and (b) in Figure 5.4. This sub-task completed correctly without any faults, illustrating that the control architecture is able to perform as usual under normal conditions. Next, the control architecture specifies that the robot must place the pink and yellow bars in any order. The pink bar is closer to the robot's gripper so the architecture tells the robot to grab it first using the activation mechanism in Section 3.1. Thus, in (c) the robot begins the *PickAndPlace(pink_bar)* behavior. However, the human steals the pink bar right before the robot places the object (d). After the human steals the object, the fault detection system detects a fault as described in Section 5.1.3. At this point, the vision system running on the forearm camera no longer detects the object in the robot's gripper. The fault detection system then uses the point during execution at which the object was lost to

determine which fault occurred. Since the robot was en route to the place location, the fault system triggers a *dropped* issue to be published, which then causes the robot to begin a dialogue with the human as described in Figure 5.3. The human follows the branch of the dialogue flow that leads to the human placing the pink bar as shown in (e).

After the fault is resolved and the object is placed, the architecture resumes and the robot continues the task by beginning the *PickAndPlace(yellow_bar)* behavior in (f). After the robot picks up the yellow bar, the human immediately steals it as shown in (g). Again, the fault detection system loses track of the object after the robot had successfully grasped the object so it triggers the *dropped* issue to be published, which causes the robot to begin a dialogue with the human. This illustrates that the dropped issue can get triggered in multiple parts of a sub-task's execution. This time however, the human follows the dialogue path that leads to the robot having to try again. In (g) the human places the yellow bar back on the table, as prompted by the dialogue, and the architecture resets the corresponding part of the task. Due to the task constraints, both the pink bar and yellow bar must be placed before moving on to the next part of the task, so the robot attempts the *PickAndPlace(yellow_bar)* behavior again in (i). During the placement of the yellow bar (j-k), the fault detection system determines that the object is one which was specified to require human assistance for placement. It then raises the *positioning* issue and asks the human for help in placing the object as described in the rightmost branch in Figure 5.3. Once the object has been placed with the human's assistance, the robot continues to the next part of the task.

In (l) the human steals the blue leg right before the robot picks it up during the *PickAndPlace(blue_leg)* behavior. At this point during the execution, the fault detection system expects the object to be detected in the gripper. However, the object is not detected, which means that the robot did not successfully grasp the object. Thus, the fault detection system triggers the *missed* issue to be published, which causes the robot to begin a dialogue with the human. The human follows the dialogue flow which leads to the robot trying again. In (m) the human places the blue leg back on the table, as prompted by the dialogue, and the architecture resets the corresponding part of the task. Due to the sequential task constraints, the blue leg must be placed before the next part of the task can happen, so the robot attempts the *PickAndPlace(blue_leg)* behavior again as shown in (n). In (o-p) the fault detection system once again triggers the *positioning* issue and asks the human for help since this object was also determined to be one which was specified to require assistance.

Based on the task constraints, the robot can then choose to place either the orange or the purple top. Since the purple top is closer to the robot's gripper, the architecture chooses to place the purple top. In (q) the robot begins the *PickAndPlace(purple_top)* behavior. The fault detection system discovers that the purple top is out of the robot's reachable space since the distance check described in Section 5.1.3 fails. It then raises an *unreachable* issue. This triggers the dialogue and the human follows the flow that results in a hand-off between the human and robot (Figure 5.3, left-most path) in (r). The robot finishes placing the purple top in (s). Finally, (t) shows the completed task with the fully assembled IKEA EKET base.

5.2.2 Discussion of Experiment

The execution of the experiment (Section 5.2.1) validates that our proposed architecture effectively utilizes the dialogue-based management system for fault recovery of hierarchical tasks. The detection of faults is done entirely with sensors on-board the robot through a combination of views from multiple cameras. The robot was able to complete the *PickAndPlace(green_leg)* behavior without fault. This shows that the architecture can complete tasks without assistance when no faults occur (Section 3.1). The execution of the *PickAndPlace(pink_bar)* and *PickAndPlace(yellow_bar)* behaviors both illustrated examples of a *dropped* failure. These objects were detected as dropped at different points along the behaviors' execution which demonstrates that the system is able to detect and resolve faults at various points, as long as they occur between the start and stop steps during which the issue is monitored. Furthermore, they illustrate that failures can be resolved through either the human assisting with the task or having the robot make another attempt to complete the task. The execution of the *PickAndPlace(blue_leg)* behavior demonstrated that the system is able to both detect and handle a *missed* failure. The execution steps for placing the blue leg and the yellow bar demonstrated that the dialogue-based management system is able to handle various major faults which require it to reset parts of the control architecture. Furthermore, these steps show that the task constraints are upheld after the architecture is reset, since the architecture completed the placements of these objects before moving on (as defined by the task tree constraints). Lastly, the

PickAndPlace(purple_top) behavior illustrates that the system is able to manage the *unreachable* fault and negotiate a hand-off between the human and the robot.

The various behaviors demonstrate that the verbal instruction system is able to assist with the task execution in multiple resolution cases by utilizing the extended dialogue between the human and the robot. By showing each of these failure cases in a single scenario, it shows that the proposed system is able to robustly handle faults that occur during the execution of complex, hierarchical tasks. The robot is able to autonomously detect faults occurring from execution failures, begin a dialogue with the human to resolve these faults, and resume the normal task execution upon fault recovery without breaking any constraints. This capability allows for a more robust and generalizable task structure for human-robot collaboration.

5.3 Conclusion & Summary

This chapter presents an extension to our previously developed distributed control architecture (Section 3.1), which incorporates a dialogue-based management system for fault recovery of hierarchical tasks through the use of human-robot collaboration. The contribution of this approach is a control architecture that *1) autonomously detects and is cognizant of task execution failures*, *2) initiates a dialogue with a human helper to obtain assistance*, and *3) enables collaborative human-robot task execution through extended dialogue* in order to *4) ensure robust execution of hierarchical tasks*

with complex constraints. The proposed method is able to autonomously detect faults occurring from execution failures, begin a dialogue with the human to resolve these faults, and resume normal task execution upon fault recovery. Furthermore, the architecture is able to adhere to all constraints defined by the task tree while the faults are being handled. The extended dialogue with the human allows for multiple avenues to resolve a detected fault, instead of a single request for help. The faults are detected autonomously with on-board sensors, through the robot's multiple cameras. The proposed approach is validated on a building task with a human-robot team. The proposed system is able to robustly detect and recover from faults that occur during the execution of a complex, hierarchical task through the use of human-robot collaboration and dialogue.

The incorporation of a fault recovery system which is able to detect and inform users of failures and resolve them through dialogue allows for a more robust task allocation mechanism, as specified in one of the main contributions presented in Section 1.4. This type of resolution is necessary to ensure that human-robot teams are able to successfully complete collaborative tasks. This contribution enables the proposed generalized task structure to be utilized in complex, hierarchical tasks which are prone to failures as well as those which require collaboration between humans and robots.

Chapter 6

Collaborative Human-Robot Hierarchical Task Execution

This chapter addresses the problem of human-robot collaborative task execution for hierarchical tasks. The main contributions are the ability for dynamic allocation of tasks in human-robot teams and opportunistic task execution given different environmental conditions. The proposed work is an extension of the multi-robot control architecture described in Section 3.1 to the human-robot domain. This effort is one of the major contributions towards developing a generalized task structure that allows human-robot teams to work together on joint tasks. The work provided in this chapter was joint work performed as part of [100].

To enable human-robot collaboration, the architecture has been modified to perform intent recognition on a human's motions in order to allow the robot to infer which

part of a task a human is performing. The architecture treats the human as a second robot and uses the human's intent in order to set the state of the second robot accordingly. This allows for the robot to identify which part of the task the human has already completed as well as what the human is currently working on, so that the robot does not repeat these parts of the task. This scheme allows the agents to work independently on the task until a collision occurs where they both attempt to grab the same object at the same time. This collision is then resolved through dialogue and the agents continue working on the task alongside each other. The proposed approach is validated on a tea-time task scenario with both overlapping and non-overlapping sub-tasks completed by a human and a Baxter robot.

6.1 Human-Robot Collaborative Architecture

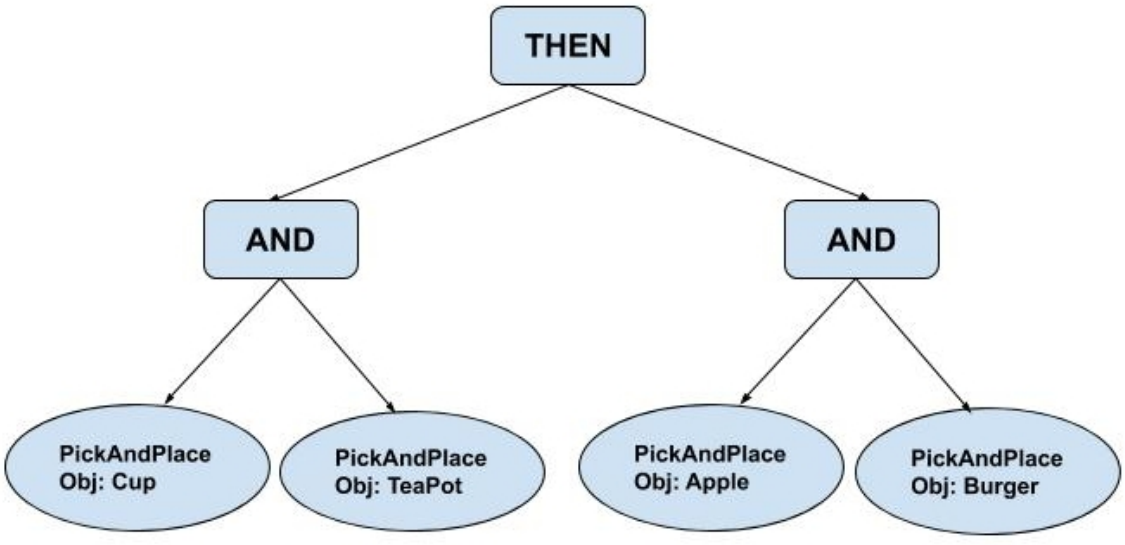


FIGURE 6.1: Hierarchical task representation used for the collaborative human-robot experiment.

6.1.1 Human-In-The-Loop Hierarchical Architecture

In order to extend the previously developed architecture described in Section 3.1 from the multi-robot domain to the human-robot domain, several modifications must be made. Instead of two robots each maintaining their own task representations, a robot working alongside a human must maintain both its own task representation and an updated, simulated version of the human's task representation. The human's representation is a copy of the robot's representation as in the multi-robot case, except this representation is updated based on the human's actions. Section 6.1.2 described how this update occurs. The human completes the task with the same constraints as the robot. Message passing between peer nodes of the human's and robot's task representations occur as in the multi-robot scenario in order to ensure the robot can monitor the human's progress on the task throughout the task execution.

If the task that the human is attempting to work on can be inferred (as described in Section 6.1.2), the corresponding node's *activation potential* in the human's simulated task tree will be increased making the node *active*. As a result, the robot can use the human's task tree to identify what part of the task the human is working on. Two cases can arise for each step during the task execution:

1. **Non-overlapping tasks**: This case occurs if the human and the robot decide to work on picking and placing two different objects at the same time. For example, looking at Figure 6.1, if the human and the robot decide to work on

the cup and the teapot respectively, the robot will infer that its task is safe to continue by checking the status of the peer node of the teapot on the human's controller. This process is the same as the process described for the multi-robot scenario with non-overlapping tasks (Figure 3.2).

2. **Overlapping tasks**: This case occurs if the human and the robot decide to work on picking and placing the same object at the same time. For example, looking at Figure 6.1, if the human and the robot both decide to work on the cup, the node status will indicate to the robot that the human is also working on this task. This is similar to the process described for the multi-robot scenario with overlapping tasks (Figure 3.3), with one major change. In this case, instead of the agent who started first completing the task, the robot will initiate a dialogue with the human in order to negotiate the conflict. A *dialogue* topic and *issue* topic to each corresponding node are added to the architecture to allow initiation of the dialogue when needed.

6.1.2 Human Intention Recognition

In order to infer the human's intention, we developed a system which tracks the human's hand as it moves around. During execution of the task, the robot continuously updates the hand position of the human as shown in Fig. 6.2. By finding the largest skin contour in the image frame, we are able to detect the position of the human hand because the only skin in the robot's view is the hand.

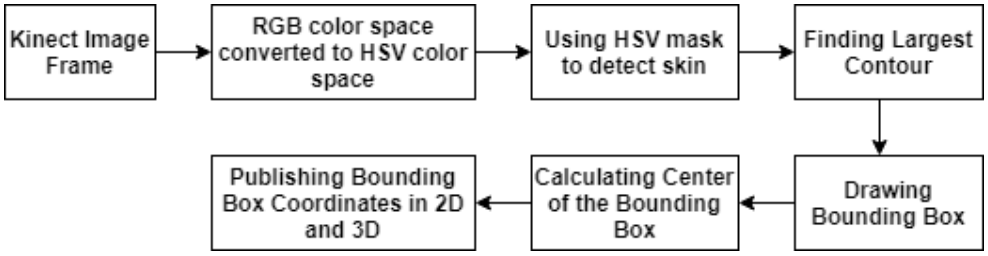


FIGURE 6.2: A step-by-step description of the continuous hand detection system from the Kinect image frame to infer the human intention

From the motion of the hand, we calculate similarity score (*SimScore*), chance score (*Chance*), started value (*Started*), and done value (*Done*) for each object.

- **Similarity Score:** The similarity score (*SimScore*) for each object is calculated for the updated hand position ($h_{x,y,z}$) in the frame. The initial normalized vector between the initial hand position ($h_{X,Y,Z}$) and an object's position ($obj(i)_{x,y,z}$) are calculated for each object $i \in 1, ..., n$. For each new hand position, the cosine similarity between the initial normalized vector and the updated normalized vector are calculated and stored in the *SimScore* list as shown in equation 6.1.

$$SimScore_i = Cosine\_Similarity(\hat{V_{X_i,Y_i,Z_i}}, \hat{V_{x_i,y_i,z_i}}) \tag{6.1}$$

where $\hat{V_{X_i,Y_i,Z_i}}$ and $\hat{V_{x_i,y_i,z_i}}$ are the initial normalized vector and updated normalized vector for object $i \in 1, ..., n$.

- **Chance:** The *Chance* value for the object that has the highest *SimScore* is incremented for every new hand position. If multiple objects have the same maximum score, the *Chance* value will be incremented for all of them. In this

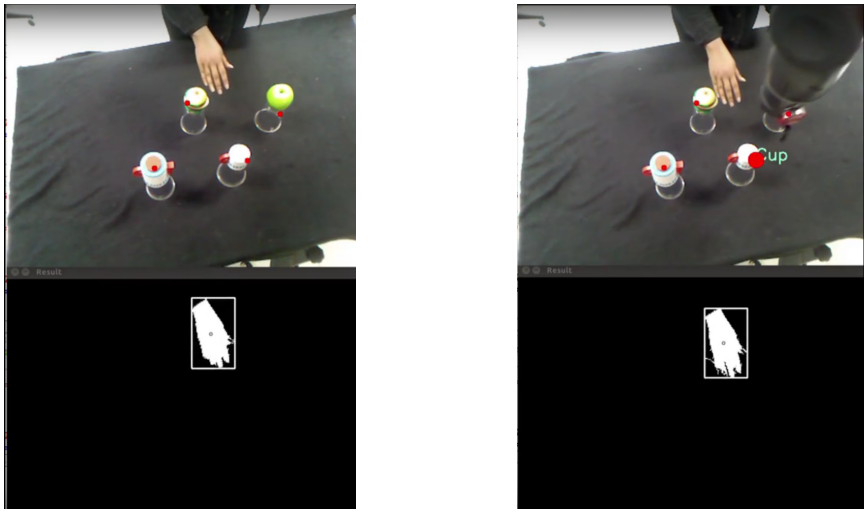
situation, the *Chance* value of the object which had the highest similarity score in the previous iteration will instead be incremented twice.

- **Started:** A Boolean variable which is initially 0 for each object; it will be set to 1 if it is inferred that the human is going for the object by checking the maximum *Chance* value.

- **Done:** A Boolean variable that will be initially 0 for each object; it will be set to 1 if the task for the object is completed by the human.

The likelihood that the person is intending to pick up each object based on the updated hand position for each frame is published as an object status message to each object's dedicated status topic using ROS [94]. The object status message contains the *Chance*, *Started*, and *Done* information for each object. The messages allow the human's simulated task tree to be updated based on the content of the object status message for each object. The human's simulated task tree will activate an object's node when the *Started* value for that object is set to 1.

Fig. 6.3 shows an example of the human hand going for the cup during the task. The system hasn't detected the intention yet in Fig. 6.3a. However, in Fig. 6.3b the human's intention can now be inferred and is being shown with a large red circle on the object. After the itention can be inferred, the *Started* value is set to 1.



(A) (B)

FIGURE 6.3: Human intention system with the contour of the hand detection. (a) The system hasn't detected the intention yet. (b) The system is detecting the intention with a large red circle on the object.

6.1.3 Collision Detection and Handling

In most human-robot collaborative tasks, collisions can occur where both the human and the robot attempt to grab the same object at the same time. This scenario is briefly discussed in Section 6.1.1 in regards to overlapping tasks. Collisions must be handled for smooth collaboration between the human and the robot. Since the message passing between each of the agent's task trees occurs continuously, the architecture is able to catch collisions. If both agent's attempt the same task at the same time, the status of the node for a specific object in both agent's trees will be *active*, which will trigger a collision.

If a collision is detected, a ROS message will be published to the corresponding node's *issue* topic which will enable the callback function to publish a ROS message to the

dialogue topic. This initiates the dialogue between the robot and the human which is used to negotiate the conflict. The robot will ask, "It appears that you are going to grab the (Object Name). Should I grab the (Object Name)?" If the human replies "Yes" then the robot will answer "Alright I will place the (Object Name)." The robot will then continue on its path to pick and place the object, while the human will instead go for the next available object in the task tree. If the human replies "No," then the robot will answer "Okay, then please place the (Object Name). Thank you." It will then let the human finish the pick and place task and instead go for the next object according to the task tree.

6.2 Experiment Design

To demonstrate the capabilities of this modification of the architecture, a collaborative task between a human and a robot was designed. The task was performed in a lab environment with a human and a Baxter humanoid robot standing on opposite sides of a table containing the objects as shown in Fig. 6.4. The 3D location of each object is provided by a vision system [101]. A Kinect v1 camera next to the Baxter was used to observe human's intent, and a Kinect v2 camera on top of the Baxter's head was used for the robot end of the architecture.

A joint tea-making task was designed based on the task tree which encodes the constraints of both THEN and AND nodes (Fig. 6.1). The scenario was performed in

such a way that it contained both overlapping and non-overlapping sub-tasks between the human and the robot.

The experiment ran as follows: First, the robot and the human both attempted to pick and place the cup, which resulted in a collision. This triggered the robot to began a dialogue, during which the human told the robot to finish the current task. While the robot was performing the task, the human moved to the next object, which was picking and placing the teapot. Next, another collision was detected as the human and the robot were both going for the apple. Again, this started a dialogue between the robot and the human. The human decided to perform the current task themselves and informed the robot. Thus, the robot stopped going for the apple, and moved to the next task to pick and place the burger while the human completed picking and placing the apple.

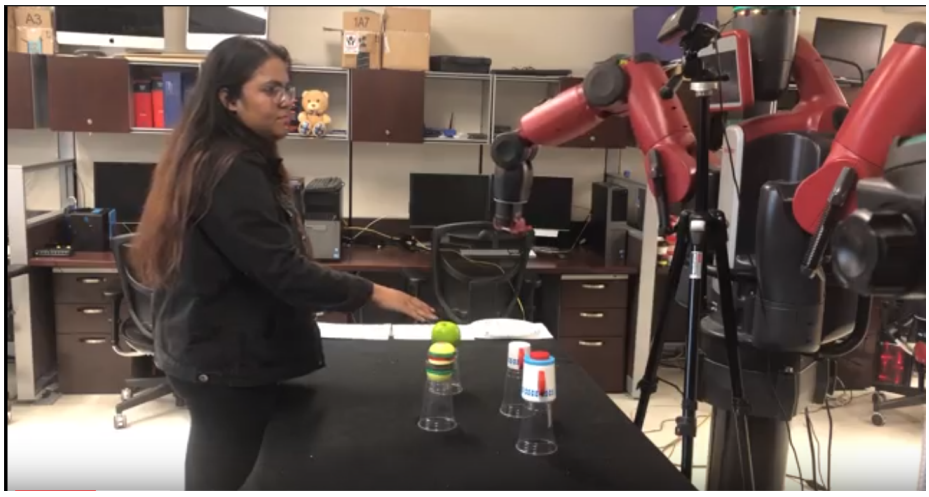
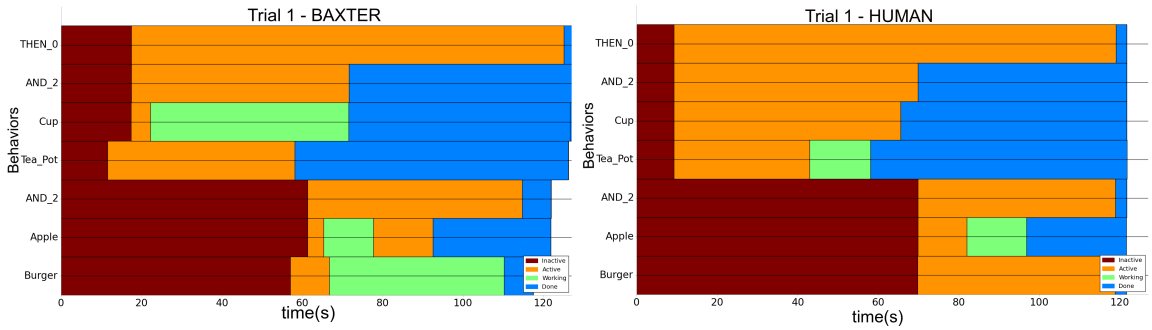


FIGURE 6.4: A sample view of the experimental setup used to perform a human-robot collaborative task.



(A) Test Scenario: Baxter (B) Test Scenario: Human

FIGURE 6.5: The timing diagrams of the tea-time task scenario on the human and the Baxter. These show the times at which the state of a node in a given task tree changed. Each row corresponds to a behavior node named as its corresponding object. The horizontal axis is increasing time. Brown → *inactive*, Orange → *active*, Green → *working*, and Blue → *done*.

6.2.1 Results

The timing diagrams (Fig. 6.5) illustrate the state for each node during the experiment using the task structures of the human and robot shown in Fig. 6.1. There are four state types in the diagram: *inactive*, *active*, *working*, and *done*. Each state is shown with different color bars in the diagram for each node.

When the task starts, both the cup and teapot are eligible for both agents to grasp (due to the task tree constraints), thus becoming *active*. At first, both agents choose to go for the cup which caused a collision and began a dialogue. As in the task design, the human let the robot finish the task for this collision resulting in the cup status of the robot being changed to *working* (Fig. 6.5a). While the robot was finishing the task, the human moved on to pick and place the teapot, which changed the teapot node status for the human to *working* in Fig. 6.5b, due to the human's action. After

placing the cup and the teapot, the status of both objects became *done* in both agents.

After the teapot and cup were completed, the apple and burger became eligible for grasping by both agents (due to the task tree constraints), and so their status became *active*. The second collision occurred on the apple task. After the Baxter began working on the apple task, the human started the same task, which triggered a collision and began a dialogue. The human told the robot to stop. The robot stopped working on the apple task (changing its state back to *active*) and moved on to the burger, changing its state to *working* (Fig. 6.5a). Fig. 6.5b shows the human's apple node status changed to *working* (after the robot stopped working), as the human chose to finish the apple task. Once the apple was placed, the status was changed to *done* for both agents. Likewise, after the burger was finished by the robot, the status was set to *done* for both agents.

Based on the experiment, we see that our architecture is able to dynamically allocate tasks in a human-robot team. The system allows a human and robot to use dialogue to negotiate how to resolve collisions that arise during the task execution in order to complete the joint task. This experiment validates that our system is able to dynamically allocate tasks between an human and a robot working together on a joint task which contains complex, hierarchical constraints.

6.3 Conclusion & Summary

This chapter addresses the problem of human-robot collaborative task execution for hierarchical tasks. The main contributions are the ability for dynamic allocation of tasks in human-robot teams and opportunistic task execution given different environmental conditions. The proposed work is an extension of the multi-robot control architecture described in Section 3.1 to the human-robot domain. This effort is one of the major contributions towards developing a generalized task structure that allows human-robot teams to work together on joint tasks, as specified in one of the main contributions presented in Section 1.4.

In the modified architecture, the robot maintains its own state and the state of its collaborative human partner. A human intent system, designed as an extension to our previous control architecture, continuously publishes a message containing the human's intent status information for each object. This allows for agents to operate independently when all agents are working on non-overlapping tasks; however, when the agents' goals overlap, a collision occurs, and dialogue is used to resolve the collision. This allows one agent to finish the task and the other to move on to another available task based on the task tree constraints.

The extension to the human-robot domain was validated through a tea-time task scenario with both overlapping and non-overlapping sub-tasks completed by a team consisting of a human and a Baxter robot. Based on the experiment, we see that our architecture is able to dynamically allocate tasks in a human-robot team. The system

allows a human and robot to use dialogue to negotiate how to resolve collisions that arise during the task execution in order to complete the joint task. This experiment validates that our system is able to dynamically allocate tasks between an human and a robot working together on a joint task which contains complex, hierarchical constraints.

Chapter 7

Dynamic Hierarchical Task Allocation of Manipulation Tasks for Heterogeneous Robot Teams

One of the major components necessary in order to create a generalized task structure which is able to learn and transfer skills between agents is the development of a mechanism for incorporating the varying skills between agents in a heterogeneous robot team. This variable heterogeneity must be utilized to encourage a robust and reliable task allocation scheme by allocating robots to tasks which best fit their specific skills. Therefore, one of the major contributions necessary to develop a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for multi-robot teams is the development of a task allocation mechanism which

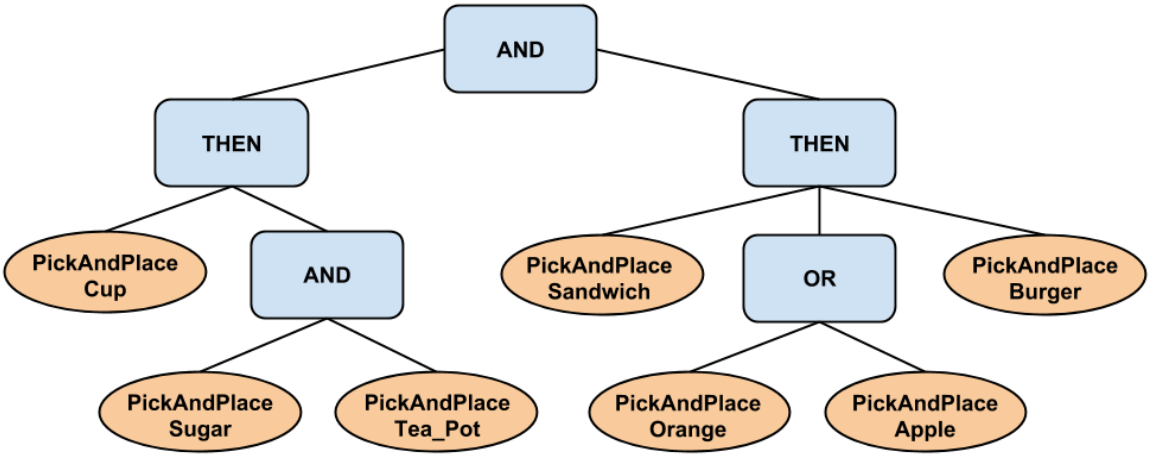


FIGURE 7.1: Representation of the joint task used in the experiments for the team of heterogeneous robots. The blue nodes represent the *goal nodes* and the orange nodes represent the *behavior nodes*.

factors in the varying capabilities of a team of heterogeneous robots. This chapter presents a mechanism which can be used to handle variable heterogeneity.

The architecture described in Section 3.1 assumes that all the robots in the team are capable of performing all the sub-tasks/behaviors and relies on a distance-based metric to compute the *activation potential* of each behavior node. This chapter extends this work to remove this assumption. This extension of the architecture introduced a generalized and extensible approach for taking into account the fact that a robot's degree of ability to perform a task: *1) covers a discrete and continuous spectrum* and *2) varies continuously during task execution based on different environmental conditions*. This modulation of the architecture is able to handle cases where robots have different capabilities. This degree of ability can be utilized to compute the *activation potential* to reflect the dynamic grasp capabilities of the different robots.

FIGURE 7.2: The setup for the multi-robot task. The PR2 (left) and the Baxter (right) are collaborating to complete a food serving task. The fruit must be placed in the bowl, the tea-set should be placed next to the bowl, and the sandwich and burger should be placed on the plate.

7.1 Task Allocation for Heterogeneous Teams with Dynamic Capabilities

7.1.1 Task Allocation using Activation Potential

To handle the variable heterogeneity between robots, several factors are incorporated into a single metric representing a robot's perceived level of capability for executing a specific behavior. In addition, the metric is continuously updated this during the task execution, enabling the team to take into account the most recent environmental conditions for task allocation. For the team of humanoid robots used for this work, the main type of behavior node used is a manipulation (pick and place) behavior;

therefore, the cues considered relevant for the metric are specific for manipulation tasks. The components taken into account are the distance between the arm and the objects and a grasp score that represents a robot's perceived effectiveness of grasping an object. Different environmental conditions are reflected in the grasp score, provided through a novel sensory pipeline, which is able to generate grasps for objects with unknown initial locations. To represent a strong heterogeneity between the robots used in our task, different constraints were placed on the grippers for each robot. For the PR2, the enforced constraints only allow the robot to get grasps in which the gripper is sideways, For the Baxter, a similar method is used to enforce the gripper to grasp the objects top down. These constraints enforce the maximum distinction in grasping functionality between the robots to illustrate the extent of heterogeneity for which the proposed method allows. These constraints force several of the objects to become nearly un-graspable by the PR2, namely the apple, the orange, and the sandwich. The grasp scores for these objects returned by our novel perception-manipulation pipeline (Section 7.1.2) are close to 0 due to the fact that the objects are too wide for the PR2's gripper to fit around them when the gripper is constrained in this manner. The constraints on the Baxter do not inhibit its grasp capabilities for any of the objects, but result in different grasp scores for the robot in different environments. These metrics are combined using a weighted linear combination, as shown in Equation 7.1:

$$activation\_potential = w_d \cdot distance\_score + w_g \cdot grasp\_score \tag{7.1}$$

w_d and w_g represent weights assigned to each metric, *distance_score* encapsulates how far the end effector is from the object to be grasped, and *grasp_score* encodes how good is the grasp returned from the grasp pipeline. Details on the *grasp_score* are provided in Section 7.1.2. The *distance_score* is computed in Equation 7.2, where $\vec{x}_{obj}$ and $\vec{x}_{arm}$ represent the 3D positions of the object and arm, respectively.

$$distance\_score = \frac{1}{||\vec{x}_{obj} - \vec{x}_{arm}||} \tag{7.2}$$

The values of weights used can be selected to assign a higher or lower significance to each metric; in this work $w_d = 1$ and $w_g = 0.001$, to ensure that the grasp score is the same order of magnitude as the distance metric.

This metric is incorporated into our distributed control architecture through the activation potential (Section 3.1). The activation potential for each individual *PickAndPlace* node is computed and updated at each step using Equation 7.1, which takes into account the different metrics described above. This value is then used by each robot to determine what behavior node to activate. In order to ensure proper coordination with the other robot teammates, such that no two robots decide to execute the same behavior, the process described in Section 3.1 is followed.

7.1.2 Object Detection, Recognition and Grasping Pipeline

The metrics used in Equation 7.1 are continuously computed from sensory data, through the pipeline shown in Figure 7.3. This pipeline is capable of generating grasps for objects with unknown initial locations. This allows for two contributions to the previously developed architecture described in Section 3.1: 1) the metric is able to accurately reflect the varying capabilities of the robots in different environmental conditions and 2) the architecture is able to automatically grasp objects. In the previous architecture, the grasps of the objects were pre-determined based on specific orientations. Utilizing this pipeline, grasps can be automatically generated for objects with arbitrary positions and orientations. This enables us to extend the architecture to allow for dynamic task allocation with different environmental conditions.

The perception-manipulation pipeline consists of multiple modules. The first module performs object detection using a vision system developed as part of the grant that funded this work [102]. This system uses input from a head-mounted Kinect. A sliding window approach with quick rejection of distant areas from the robot is used; two sets of features are extracted: 1) a Histogram of Oriented Gradients (HOG) [103]; 2) a flattened 5-by-5 2D color histogram for the HSV (hue and saturation channels only) and CIELUV (u and v channels only) color spaces separately. The two color histograms contribute a total number of 50 features, whereas the HOG produces 180 features. In order to prevent overfitting during the training and to reduce the computational load of the classifier during the test time, the HOG features pass

through a two-layered feature reduction stage to drop their number to 60. The feature reduction is done by performing Principal Component Analysis (PCA), followed by a Linear Discriminant Analysis (LDA). The system uses a one-versus-rest strategy multi-class Support Vector Machine (SVM). The overlapping detections are merged by choosing the strongest one via the non-maximum suppression strategy. In order to keep the object detection system real-time, Median Flow [104] tracking is used between any two detection rounds.

The object detection module returns objects and their locations in the 2D camera view; this is combined with depth information to obtain locations of the objects in the robot's 3D coordinate frame. This is used to compute the end effector distance to each of the objects in the scene, and as input to a module that computes possible grasps for the detected objects, based on the GPD library [105]. Given a point cloud, the GPD library is designed to return a set of grasps consisting of a 6-DOF position, orientation, and a grasp quality score. According to [105], the grasps returned by GPD are robust and reliable in cluttered environments (grasps were shown to have a 93% success rate).

However, GPD makes one of two assumptions: 1) any graspable object in the scene is acceptable; or 2) only a single object is in the point cloud. These assumptions did not hold for our use, so we made several modifications to utilize GPD. To force GPD to return grasps on a single object, each object in the task tree utilizes its own instance of the GPD library, which observes a subset of the point cloud from the Kinect

centered around the location of the object (from the object detection module). As the object moves around, its respective location in the point cloud changes. The GPD work space associated with that object has to be continually updated as well, which required some modifications to GPD's interface with the Robot Operating System (ROS) [94]. We modified GPD further to extract, for each object, the grasp with the highest rating score from the set of possible grasps.

These modifications to GPD resulted in a single rating score per object. We take the continuous-valued score returned by GPD and add two additional discrete filters to the score in order to obtain the final grasp score. The first filter checks whether or not the grasp is withing the robot's reachable space. If it is, the original score is kept. Otherwise, the grasp score is set to 0. The grasp score returned by the first filter is then fed into the second filter. The second filter checks whether the orientation of the grasp is within a provided range of orientations which represent the possible configurations of the robot's gripper. For the purposes of this experiment, this range is hard-coded to enforce different grasp patterns on the robots as specified in Section 7.1.1 but could be automatically computed using the computed grasp position and an inverse kinematics solver for the robot. If the grasp is within the set of feasible configurations for the robot, the score passed from the first filter is kept, otherwise the score is set to 0. The result of this second filter becomes the final grasp score for the given object. This grasp score therefore provides a measure of the robot's perceived capability for grasping an object in varying environmental conditions with unknown orientations.

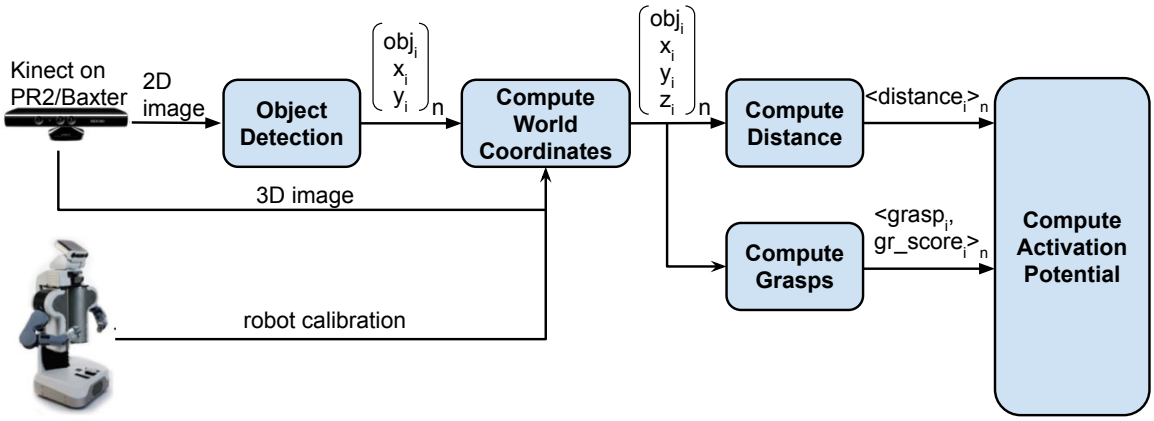


FIGURE 7.3: Perception-manipulation pipeline.

The grasps generated by GPD along with their newly computed grasp scores are returned by the compute grasps module. The distance and the grasp scores are then used to compute the activation potential for each of the objects in the scene as explained in Equation 7.1. This value is used by the update loop to determine which node should be activated by each robot, as described in Section 3.1.

7.2 Experimental Validation

FIGURE 7.4: Placement of objects for the different scenarios. Left to right: placements for Scenario 1, placements for Scenario 2, placements for Scenario 3.

The proposed architecture has been validated with a team of two humanoid robots in scenarios specifically designed to illustrate the key proposed contribution: *ability*

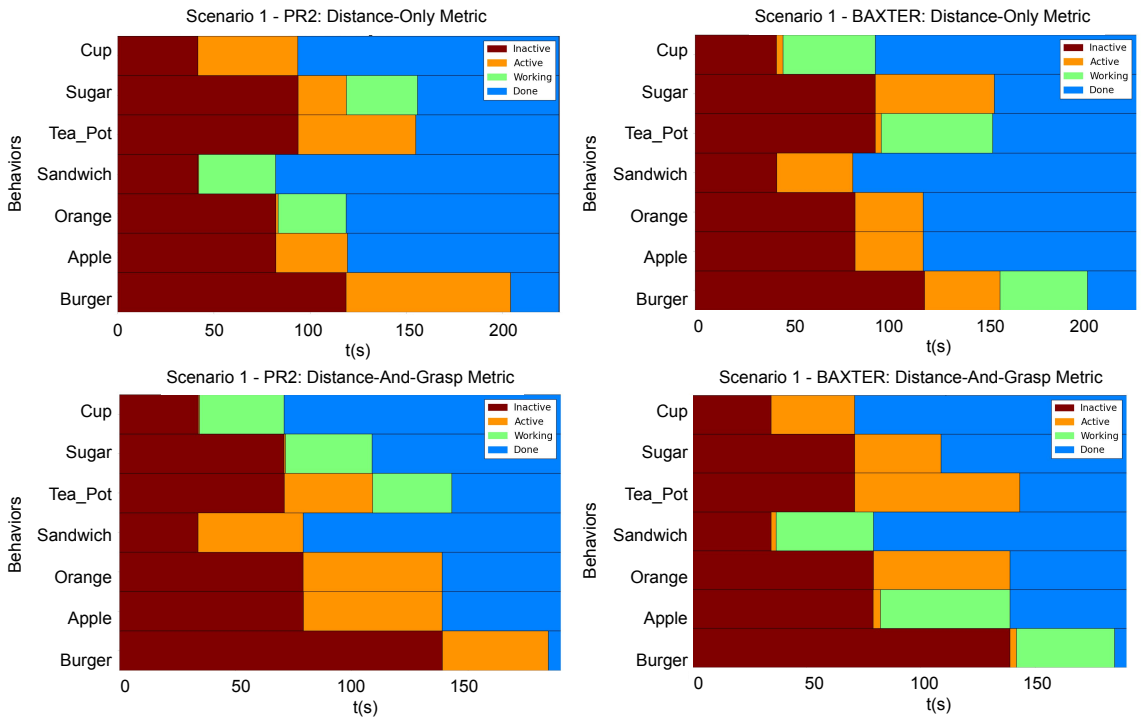


FIGURE 7.5: The timing diagrams for **Scenario 1**. These diagrams represent the times at which the state of a node in a given task tree changed. **Top row:** Provides the timings for the PR2 and the Baxter using the distance-only metric. **Bottom row:** Provides the timings for the PR2 and the Baxter with the distance-and-grasp metric which utilizes the heterogeneity component. **Within each graph:** Each row corresponds to a behavior node named according to its corresponding object. The horizontal axis is increasing time. Brown → *inactive*, Orange → *active*, Green → *working*, and Blue → *done*.

to handle dynamically changing robot capabilities in the context of multi-robot teams performing tasks with complex sequencing and temporal constraints.

The objects used in the experiments include a wooden tea-set (consisting of a cup, a sugar container, and a teapot) in addition to several fake food objects (namely an apple, a burger, an orange, and a sandwich). The joint task structure is shown in Figure 7.1 and consists of two main sub-tasks that can be executed in parallel. The first is a tea-setting task which is shown in the left sub-tree of the task structure. This sub task consists of first placing the cup, then placing the sugar and the teapot

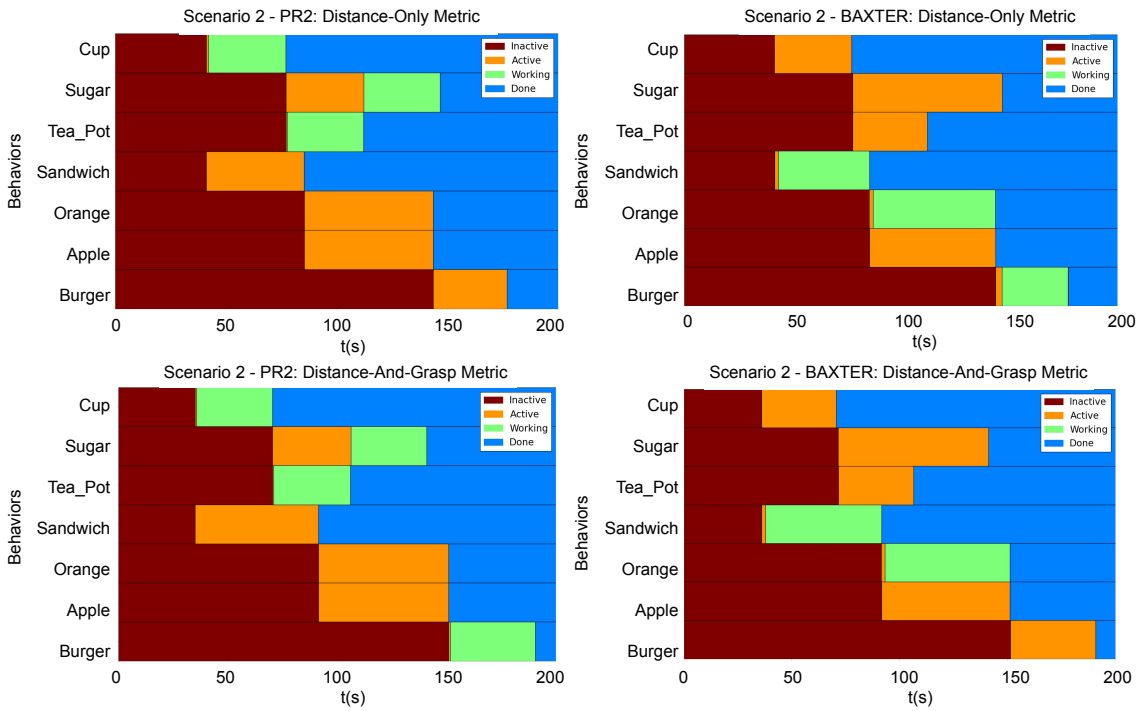


FIGURE 7.6: The timing diagrams for **Scenario 2**. These diagrams represent the times at which the state of a node in a given task tree changed. **Top row:** Provides the timings for the PR2 and the Baxter using the distance-only metric. **Bottom row:** Provides the timings for the PR2 and the Baxter with the distance-and-grasp metric which utilizes the heterogeneity component. **Within each graph:** Each row corresponds to a behavior node named according to its corresponding object. The horizontal axis is increasing time. Brown → *inactive*, Orange → *active*, Green → *working*, and Blue → *done*.

(in any order) to their goal locations. The second sub-task is the food-setting task which is shown in the right sub-tree of the task structure. This sub-task consists of first placing the sandwich, followed by placing either the orange or the apple, and then finally placing the burger to their respective goal locations.

The *PickAndPlace* nodes take as input the desired grasp location of an object provided by the perception-manipulation pipeline (Section 7.1.2) and place the object at a pre-specified location. End-effector trajectories to the grasp location are generated using MoveIt [99]. The right arm on each robot was used. The complete motion must be

completed in order for the *PickAndPlace* node to be marked as *done*. Until the place command finishes, the robot waits before it activates another node, since only one node per robot can be doing work at any given time.

The initial setup for the experiments is shown in Figure 7.2. For this setup, the Baxter and PR2 robots were placed on either side of a table with the objects in between them. Three different scenarios were performed, in which the objects were placed in different locations to show that the architecture can dynamically determine different task allocations based on the specifics of the environment. Figure 7.4 shows the placement of the objects for the different scenarios. For all of the scenarios, the goal locations were as follows: the apple and the orange are placed into the bowl on the right side of the PR2; the cup, sugar, and teapot are placed next to the bowl; and the burger and the sandwich are placed onto the plate on the left side of the PR2.

In order to assess the impact of the heterogeneity metric as defined in Section 7.1.1, each scenario consisted of two separate trials. The first trial used a metric (named *distance-only*) that took into consideration only the distance from the robot's gripper to the objects for the activation potential (first term in Equation 7.1). The second trial used the full heterogeneity metric (named *distance-and-grasp*) shown in Equation 7.1 which incorporates both the distance and the grasp score into the activation potential.

PR2	Scenario 1	Scenario 2	Scenario 3
Cup	**25**	**33**	**14**
Sugar	**57**	**17**	**23**
Tea_Pot	**13**	**41**	**50**
Sandwich	0	0	0
Orange	0	0	0
Apple	0	0	0
Burger	15	**91**	5

TABLE 7.1: Table of grasp scores for the PR2 for each scenario (rounded to nearest integer). Scores in bold are the objects which the PR2 grabbed during each scenario.

7.2.1 Results and Discussion

The timing diagrams for the different scenarios are shown in Figures 7.5-7.7. In each figure, the results of the two trials within a single scenario are shown. The top row illustrates the results of the first trial using the distance-only metric on the PR2 and Baxter. The bottom row illustrates the results of the second trial using the distance-and-grasp metric. Each of the individual timing diagrams illustrate the change of state of each node in the task tree for a given robot. The different color bars in the figure represent the times during which a particular *PickAndPlace* behavior node is in one of the following states: inactive, active, running, or done. The intervals corresponding to the running state identify when a given node is being executed and are thus indicative of the order in which various sub-tasks have been performed. Additionally, the grasp scores for the different scenarios for each robot are given in Tables 7.1-7.2. These scores differ across trials and robots due to the different environmental conditions in each scenario as well as different grasp capabilities of each robot. The results of each scenario are discussed below.

Baxter	Scenario 1	Scenario 2	Scenario 3
Cup	10	8	2
Sugar	14	2	2
Tea_Pot	2	17	13
Sandwich	**44**	**26**	**27**
Orange	12	**14**	**8**
Apple	**13**	9	7
Burger	**16**	13	**11**

TABLE 7.2: Table of grasp scores for the Baxter for each scenario (rounded to nearest integer). Scores in bold are the objects which the Baxter grabbed during each scenario.

In Scenario 1 there were several differences in the allocation of objects between the trials for the distance-only and the distance-and-grasp metrics. For the distance-only trial, first the PR2 picked up the sandwich while the Baxter grabbed the cup, then the PR2 grabbed the orange while the Baxter grabbed the teapot, and lastly the PR2 picked up the sugar and the Baxter grabbed the burger. Since the metric used in this trial only utilizes distance to the objects, this allocation grasps the closest objects first, while adhering to the constraints defined in the task structure. However, because the PR2 cannot accurately grasp the sandwich or the orange due to the constraints of the gripper, these objects get knocked over during the execution of this task. During these experiments, the robots do not have the capability to detect that the object was dropped (as they did in Chapter 5), so they assume that the place behavior was successful and the task will continue on. In the trial utilizing the grasp score, the cup, sugar, and tea were all allocated to the PR2, and the other objects were allocated to the Baxter. This illustrates that by utilizing the grasp score in the metric for the activation potential, the architecture is able to allocate objects which are graspable by the robot, while still adhering to the various types of constraints provided in the

task structure.

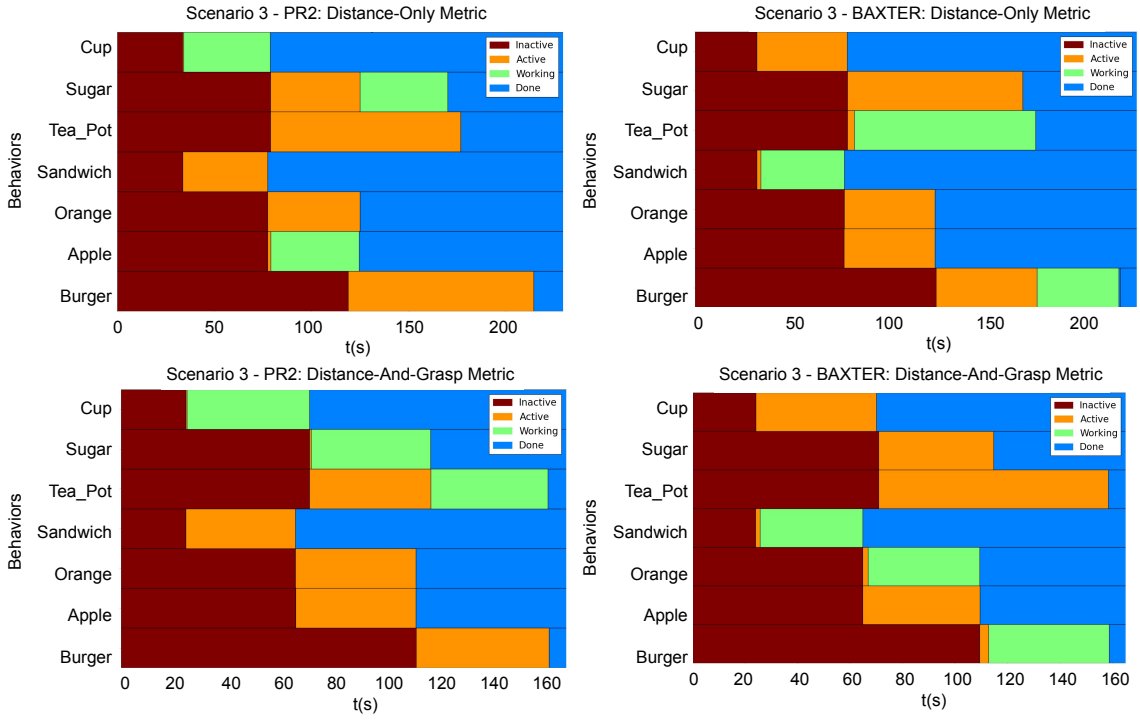


FIGURE 7.7: The timing diagrams for **Scenario 3**. These diagrams represent the times at which the state of a node in a given task tree changed. **Top row:** Provides the timings for the PR2 and the Baxter using the distance-only metric. **Bottom row:** Provides the timings for the PR2 and the Baxter with the distance-and-grasp metric which utilizes the heterogeneity component. **Within each graph:** Each row corresponds to a behavior node named according to its corresponding object. The horizontal axis is increasing time. Brown → *inactive*, Orange → *active*, Green → *working*, and Blue → *done*.

Scenario 2 illustrates the continuous-valued element of the proposed heterogeneity metric. At the time when the allocation of the burger was determined, the Baxter's gripper was at the goal location for the orange and the PR2's gripper was at the goal location for the sugar. Due to the fact that the pick location of the burger is slightly closer to the orange's goal location than to the sugar's goal location, in the distance-only trial the Baxter picked up the burger since only the distance from gripper to object was used. However, in the distance-and-grasp trial, the burger's

grasp score for the PR2 is higher than that of the Baxter (91 vs 13). Thus, in this trial the burger is allocated to the PR2 instead of the Baxter. This illustrates that the proposed distance-and-grasp metric is able to properly allocate objects based on a skill which can be performed to various degrees (continuous-valued score) rather than a simple binary (yes/no) skill.

Scenario 3 illustrates a combination of the findings from the previous two scenarios. Utilizing the proposed distance-and-grasp metric which accounts for the variable heterogeneity: 1) allows the objects to be allocated such that the robots can grasp all of the objects allocated to them and 2) is able to allocate objects with a higher chance at being grasped according to a continuous-valued metric. Using the distance-only metric, the apple is allocated to the PR2. However, the apple cannot be reliably grasped by the PR2 due to the gripper constraints. Thus, in the trial with the distance-and-grasp metric, the apple is instead allocated to the Baxter. This is similar to the finding in Scenario 1. Using the distance-and-grasp metric, at the time which the burger is allocated, the PR2 is at the sugar goal location and the Baxter is at the orange goal location. The PR2 has a higher grasp score on the teapot than the burger (50 vs 5); while the Baxter's scores are very similar (11 vs 13). Thus, the PR2 grabs the teapot while the Baxter grabs the burger. This illustrates that, similar to Scenario 2, the distance-and-grasp metric which accounts for the heterogeneity is able to allocate the objects to the robots which have a higher chance of grasping them reliably.

These scenarios illustrate that the inclusion of the heterogeneity component in the scoring metric of the architecture results in allocations of the objects to the robots which are best suited to grasp them. The proposed architecture is able to handle *variable heterogeneity* during the task allocation which takes into account the most recent environmental conditions as the task progresses.

7.3 Conclusion & Summary

One of the major components needed to create a generalized task structure which is able to learn and transfer skills between agents is a mechanism for incorporating the varying skills between agents in a multi-robot team of heterogeneous robots. This variable heterogeneity must be utilized to encourage a robust and reliable task allocation scheme by allocating robots to tasks which best fit their specific skills. Therefore, the work presented in this chapter is one of the major contributions, as described in Section 1.4, which is necessary towards developing a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for multi-robot teams.

This chapter presents a real-time distributed control architecture for collaborative task execution by heterogeneous robot teams. Three main contributions of the approach are the *ability to handle variable robot heterogeneity*, *ability to handle automatic grasping of objects with unknown initial locations*, and *collaborative execution*

of tasks with hierarchical representations and multiple types of constraints. This is achieved through the use of a continuous-valued metric that encodes a robot's ability to perform a particular task component; the metric is updated continuously during task execution, allowing for dynamic task allocation that takes into account the most recent environmental conditions.

Additionally, the architecture provides a novel perception-manipulation pipeline which is able to automatically generate grasps on objects with arbitrary positions and orientations. This pipeline is utilized by the updated metric which allows it to accurately reflect the varying capabilities of the robots in different environmental conditions. Experimental validation is performed with a team of two humanoid robots performing household manipulation tasks. The outcomes of the experiments support the proposed contributions: different environmental conditions result in different and continuously changing values for the robot's task execution ability, resulting in dynamic task allocation among the heterogeneous robot team performing complex hierarchical tasks. Therefore, this work is a major contribution towards developing a generalized task structure which is able to transfer learned skills between heterogeneous robots while accounting for their varying capabilities.

Chapter 8

Interdependence Constraint for Collaborative Multi-Robot Task Allocation Using a Distributed Control Architecture

For robot teams to complete tasks in real world situations, they must be able to understand task constraints and operate cooperatively and efficiently. Multi-robot task allocation attempts to allocate tasks to the robots in order for the robot team to complete its overall goal, while still following any constraints that were imposed. To allow for explicit cooperation between multiple robots, an interdependence constraint must be used which requires several parts of a task to be completed together. One

type of task which requires explicit cooperation is building tasks. These tasks require one agent to hold a part in place while another agent connects another piece. Our proposed work is focused on these types of manipulation tasks which require a holding behavior while another task component is completed.

In this chapter, an interdependence task constraint was implemented, tested, and added to the distributed multi robot task allocation architecture described in Section 3.1 in order to increase the generalizability of the proposed task structure. This constraint was also integrated into the verbal instruction system developed in Chapter 4 to further emphasize the generalizability in the learning capabilities of the task structure.

8.1 Integration of WHILE into Architecture

8.1.1 Addition of interdependence Constraint

In order to extend the existing architecture (Section 3.1) to add an interdependence constraint, the following was implemented:

- **WHILE constraint:** The WHILE behavior is a new *goal node* which enforces an interdependence constraint on its children, meaning the completion of one sub task is dependent on the completion of the other sub task.

- **HOLD behavior:** A new *behavior node*, HOLD, was implemented to allow one robot to hold an object, while the other robot completes another portion of the task requiring explicit coordination between the agents. The task that must be completed will be referred to as the *action task*. HOLD is intended to be a child of the WHILE node only.

To implement the WHILE constraint and HOLD behavior, the following was added to the architecture:

- **SET flag:** A flag was created in order to signal to the architecture that the *action task* was completed, and the item that is being held can now be placed.
- **Update and Spread Activation Algorithms:** Each previous *goal node* used *Update Activation* and *Spread Activation* algorithms to propagate activation potential and activation levels throughout the tree. The implementation of the WHILE constraint required its own version of these algorithms as well.

The integration of the WHILE constraint and HOLD behavior into the control architecture is illustrated in a sample task tree representation shown in Figure 8.1. In this figure, the left child of the WHILE node is the HOLD behavior and the right child of the WHILE node is the *action task* that must be completed for the WHILE constraint to be completed.

The *Update Activation* algorithm, Algorithm 6, first checks to see if the peer robot has completed the action task. If this task is completed, the flag is set. If the HOLD

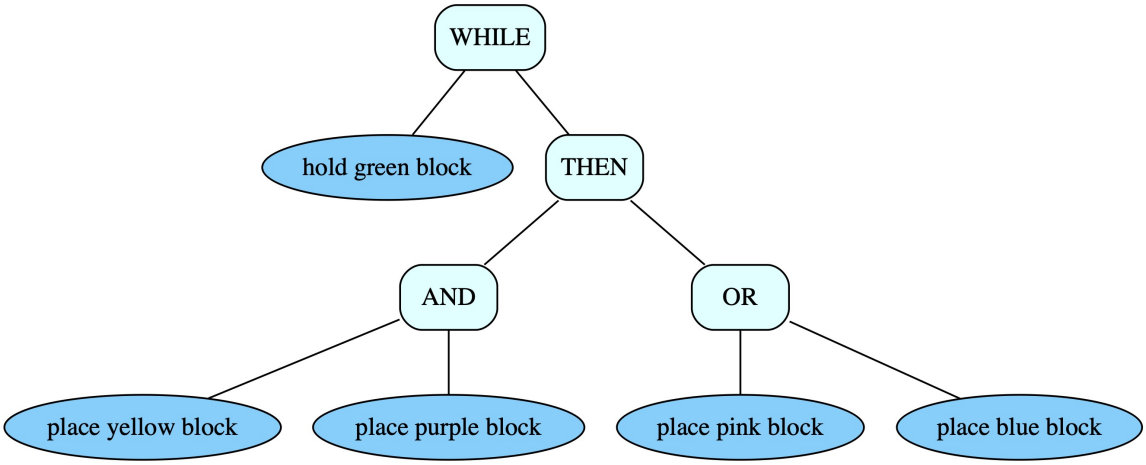


FIGURE 8.1: Task tree illustrating WHILE functioning as a root node, with THEN, AND, and OR constraints nested underneath WHILE. The left child of the WHILE node (hold green block) is the HOLD behavior and the right child of the WHILE node (THEN) is the *action task* that must be completed. In this case, the *action task* is a compound node and consists of the THEN node along with its entire sub-tree.

Algorithm 6 Update Activation

```
1: if action.peer_done == TRUE then
2:     setflag
3: end if
4: if hold.active == FALSE then
5:     activation_potential = hold.activation_potential
6: end if
7: if hold.active == TRUE then
8:     activation_potential = action.activation_potential
9: end if
```

child is activated, then the *activation potential* of the WHILE constraint is set to the action child's *activation potential*. This is because an item is currently being held, so the next task to be completed will be the *action task*. If the HOLD child isn't active, then the *activation potential* of the WHILE constraint is set to the *activation potential* of the HOLD child. This is because the first child of the WHILE behavior that is activated is the HOLD child, as seen in the *Spread Activation* algorithm, Algorithm 7.

Algorithm 7 Spread Activation

1: $msg \leftarrow \{activationlevel = 1.0\}$
2: $SendToChild(hold, msg)$
3: **for** $child \in children$ **do**
4: **if** *previous child active* **then**
5: $SendToChild(action, msg)$
6: **end if**
7: **end for**

The *Spread Activation* algorithm, Algorithm 7, first spreads its activation to the HOLD child. Once the HOLD child is active, then the following children of the WHILE constraint receive the *activation level*. This is to ensure that the architecture first instructs one agent to hold the object, then activates the *action task* on the second agent to enable the explicit coordinate on the task. The *action task* can be either a singular *behavior node* or a compound node consisting of both *goal nodes* and *behavior nodes*. Once the action task is completed, the state flag is set, and the object that was being held by the first agent can now be placed.

8.2 Integration of WHILE into Verbal Instructions

In addition to integrating the interdependence constraint into the control architecture, the WHILE capability was also added into the verbal instruction system outlined in Chapter 4. This allows the WHILE constraint to be utilized in the generation of task trees through verbal instruction which can be directly executed by a team of robots to complete tasks which require explicit cooperation between the agents.

8.2.1 Command Parsing

For the purposes of this work, the interdependence constraint focuses on tasks manipulation tasks which require holding an object while another task component is completed. Therefore, in order to formulate a verbal instruction which conveys a task using the interdependence constraint, the sentence must contain at least two verbs and one conjunction. The first verb is the HOLD behavior and the second is the verb associated with the other task component which needs to be completed. For example, the sentence "place the pink bar on the green leg while holding the yellow bar" has two verbs and one conjunction. This type of sentence structure is necessary to allow the interdependence constraint to represent meaningful actions. However, the work presented in Chapter 4 was not able to correctly parse these types of sentences. Therefore, the parsing of the verbal instruction system defined in Chapter 4 was modified to allow for these types of sentences. In order to solve this problem, the system was modified to include a preposition argument check within the Answer Constraint Engine (ACE) tool [70]). This check allows ACE to select the minimal recursion semantics (MRS) list [106] with the correct arguments attached to each preposition. The modified parsing method is described below.

Algorithm 8, *searchForCorrectPrepArgs* takes in a dictionary of strings (*MRS*) and determines if each preposition within the MRS has the desired arguments. If the passed *MRS* has a preposition with the desired arguments or has no preposition at all, it will return *True*. However, if the passed *MRS* has a preposition with incorrect

Algorithm 8 *searchForCorrectPrepArgs*(MRS)

```
1: args = ∅
2: prepCount = gatherArgs(args, MRS)
3: if prepCount == 0 then
4:    return True
5: else
6:    return checkArgs(prepCount, args)
7: end if
```

arguments, the algorithm will return *False*. The argument *args* (line 2) is a list of strings, starting off as an empty set, that will contain the arguments for each preposition. The variable *prepCount* is an integer tracking the number of prepositions used in the *MRS*. The *gatherArgs* function is presented in Algorithm 9 and the *checkArgs* function is presented in Algorithm 10.

The *gatherArgs* function, Algorithm 9, finds the two relevant arguments attached to a preposition and appends them to a list. This is done for each preposition that is found in the *MRS*. The final argument list is eventually copied into the argument list, *args*, upon the return of the call to this function from the *searchForCorrectPrepArgs* function described in Algorithm 8. Additionally, the algorithm returns the total number of preposition in the *MRS* as an integer.

The *checkArgs* function, Algorithm 10, checks the two arguments passed in to determine if they are the desired arguments for each of the prepositions found in Algorithm 9. The first argument can either be a noun or a conjunction. The second argument must be a noun. If these conditions hold for all propositions found, the algorithm passes and returns *True*.

Algorithm 9 $gatherArgs(args, MRS)$

```
1: prepCount = 0
2: for line in MRS do
3:    if Prepositionfound then
4:       prepCount+ = 1
5:       for i from 0 to 2 do
6:          args.append(findPrepArgs(line, MRS)[i]
7:       end for
8:    end if
9: end for
10: return prepCount
```

Algorithm 10 $checkArgs(prepCount, args)$

```
1: for i from 0 to prepCount do
2:    // get first argument preposition i
3:    ARG1 = argL_{2*i}
4:    // get second argument preposition i
5:    ARG2 = argL_{2*i+1}
6:    if (ARG1 != noun or ARG1 != conj) or (ARG2 != noun) then
7:       return False
8:    end if
9: end for
10: return True
```

The *findPrepArgs* function, Algorithm 11, takes a list of strings as input and returns a list consisting of two strings. It must first find the labels in the *MRS* corresponding to the arguments, then convert them to strings referring to the words in the original command. These two words correspond to the arguments of a preposition present in the *MRS*. The list *lbls* (line 9) is a list of labels for each prepositional argument. In line 8, *nextWord* is a string that appears after "*ARG1*" or "*ARG2*" in the passed in line.

The *searchForLabel* function, Algorithm 12, searches a dictionary, *MRS*, for a key. If

Algorithm 11 *findPrepArgs*(*line*, *MRS*)

```
counter = 0; argCounter = 0
for word in line do
    counter+ = 1
    if argCounter == 2 then
        break
    else if word == "ARG1" or word == "ARG2" then
        argCounter+ = 1
        nextWord = line[counter]
        lbls.append(nextWord)
    end if
    for i from 0 to argCounter do
        args.append(searchForLabel(lbls[i], MRS))
    end for
end for
return args
```

Algorithm 12 *searchForLabel*(*lbl*, *MRS*)

```
for line in MRS do
    if len(line) > 10 and line[0] == "[" then
        if line[5] == lbl then
            return line[1]
        end if
    end if
end for
return False
```

the label is found, the corresponding string will be returned. The *lbl* passed in is the key that is being searched for within the *MRS*. The *line[5]* refers to a string keeping track of the key in that line, and *line[1]* is a string corresponding that key.

This modified parsing implementation utilizing Algorithms 8-12 is able to accept a string command with up to two verbs and one conjunction as an input, select the correct MRS from a list, and return that MRS with the correct preposition arguments. With this modification, the verbal instruction system is able to correctly parse the

types of sentences necessary to formulate a verbal instruction which conveys a task using the interdependence constraint. For example, it can now parse sentence such as "place the pink bar on the green leg while holding the yellow bar" and ensure that the correct object information is linked to the correct verbs within the parse tree.

8.2.2 Command Converting

In addition to modifying the parsing implementation as described in Section 8.2.1 to incorporate the WHILE constraint, the system used to convert from parsed sentences into task trees as described in Chapter 4 also had to be modified to include this constraint. The previous version of the system was only able to convert sentences with one verb. However, as discussed in Section 8.2.1, commands incorporating the WHILE constraint must contain at least two verbs and one conjunction. Therefore, the conversion system was modified to incorporate sentences with multiple verbs using the following steps. First, calculate the number of verbs in the sentence. If there are two verbs present, split the sentence into two sentences, process each separately, and stitch them together by placing the conjunction in the correct location at the end. These steps are outlined from Algorithm 13 which is discussed in more detail below. With these modifications, the architecture can accept an input sentence that contains up to two verbs and one conjunction, and convert it into a parenthesized command, which allows the verbal instruction system to work with commands containing the WHILE constraint.

Algorithm 13 *multi_verb_parsing*(*relLines*)

```
1: verbL = returnVerbs(relsLines)
2: verbCount = len(verbList)
3: if verbCount >= 2 then
4:    relsLinesL = splitSentenceIntoTwo(relsLines)
5:    conj = returnConj(relsLinesL[2])
6:    commandL = MRS_Crawling(verbCount, verbL, relsLinesL)
7:    finalCommand = addConjToEnd(commandL, conj)
8: else
9:    relsLinesL = [relsLines]
10:   commandL = MRS_Crawling(verbCount, verbL, relsLinesL)
11:   finalCommand = commandL[0]
12: end if
13: return finalCommand
```

The *multi_verb_parsing* function, Algorithm 13, takes a list of parsed strings from an *MRS*, known as the *relsLines*, and processes it into a final, parenthesized command. This algorithm is able to process basic sentence structures with up to two verbs separated by a conjunction. If there is one verb present, the sentence will be converted into a parenthesized command by Algorithm 14, *MRS_Crawling*. If there are two verbs separated by a conjunction, the sentence will be split at the conjunction using Algorithm 15, and each section will be processed separately by Algorithm 14. Algorithm 14 is a modified version of Algorithm 1 described in Chapter 4.

In Algorithm 13, *verbL* is a list of the verbs present in the sentence, *verbCount* is an integer tracking the number of verbs in the sentence, *relsLinesL* is a list of *relsLines* for each sentence structure that has been separated based on verbs, *conj* is a string tracking the conjunction separating the two sentence structures, and *finalCommand* is a string that contains the final parenthesized command. The *returnVerbs* function

Algorithm 14 *MRS_Crawling*(*verbCount*, *verbL*, *relsLinesL*)

```
1: for i from 0 to count do
2:    rels = simplify( relsLinesL[i] )
3:    ConnectAdjectives(rels)
4:    HandlePrepositions(rels)
5:    // add to list of parenthesized commands
6:    commandL.append(BuildCommand(rels, verbL[i])
7: end for
8: return commandL
```

is described in Algorithm 16, the *returnConj* function is described in Algorithm 17, and the *addConjToEnd* function is described in Algorithm 18.

Algorithm 14, *MRS_Crawling*, is a modified version of Algorithm 1 described in Chapter 4. In the original version, only a sentence containing a single verb would be parsed correctly into a parenthesized string. This modified version is able to correctly parse sentences with up to 2 verbs, such as sentences containing a WHILE constraint, into parenthesized strings. The algorithm parses a passed *relsLinesList* into a parenthesized string command for each verb found in the passed sentence (up to 2 verbs).

The *splitSentenceIntoTwo* function, Algorithm 15, takes a passed list of strings, splits them into two separate lists of strings, appends each new list into a larger list, and returns that final list. The sentence is split once a conjunction is found in the sentence. The conjunction is added to the end of the final list. In the algorithm, *conjFound* is a boolean value referring to if the conjunction was found or not, *conj* is a string corresponding to the conjunction in the sentence, *sent1* is a list of strings before the conjunction is present in the sentence, *sent2* is a list of strings after the conjunction

Algorithm 15 *splitSentenceIntoTwo*(*sentence*)

```
1: conjFound = False
2: for  word in sentence do
3:    if word is conjunction then
4:       conj = word
5:       conjFound = True
6:    else if conjFound == False then
7:       sent1.append(word)
8:    else if conjFound == True then
9:       sent2.append(word)
10:   end if
11: end for
12: sentL = [sent1, sent2, conj]
13: return  sentL
```

is present in the sentence, and *sentL* is a list with elements consisting of *sent1, sent2,* and *conj*.

The *returnVerbs* function, Algorithm 16, searches a passed list of strings and returns any verbs within that passed sentence. The *returnConj* function, Algorithm 17, takes in a passed string and attempts to match it to a conjunction. It then returns a command string corresponding to that conjunction. The *addConjToEnd* function, Algorithm 18, takes in a list of parenthesized commands and a conjunction, and stitches them together to make one, final parenthesized command string. In the algorithm, *commandString* is a string corresponding to the final parenthesized command, *conj* is a command string corresponding to the conjunction in a sentence, and *pCommand[i]* is a list of strings, at element i, that corresponds to a single parenthesized command for one part of the sentence.

With the modified conversion method, the architecture can accept an input sentence

Algorithm 16 *returnVerbs*(*sentence*)

```
1: for word in sentence do
2:     if word == verb then
3:         verbL.append(word)
4:     end if
5: end for
6: return verbL
```

Algorithm 17 *returnConj*(*conj*)

```
1: if conj == 'and' then
2:     return 'AND'
3: else if conj == 'or' then
4:     return 'OR'
5: else if conj == 'then' then
6:     return 'THEN'
7: else if conj == 'while' then
8:     return 'WHILE'
9: end if
```

Algorithm 18 *addConjToEnd*(*pCommandL*, *conj*)

```
1: commandString = '(' + conj + ' '
2: for i from 0 to len(pCommandL) do
3:     commandString + = pCommandL[i] + ' '
4: end for
5: commandString + = ')'
6: return commandString
```

that contains up to two verbs and one conjunction, and convert it into a parenthesized command. Using the last step of the verbal instruction system outlined in Chapter 4, this parenthesized command can then be converted into a task tree which can be directly executed by a team of robots. Therefore, these modifications allow for the WHILE constraint to be incorporated into verbal instructions which can be converted to task trees and executed by a team of robots to complete tasks which require explicit

cooperation between the agents.

8.3 Experimental Setup

To illustrate the incorporation of the interdependence constraint (WHILE) into the generalized task structure, several experiments were completed. These experiments are broken down into two sets. The first set of experiments is used to verify that the WHILE constraint was correctly incorporated as a new type of constraint into the previously developed control architecture (Section 3.1). These experiments demonstrate that the WHILE constraint can be utilized correctly by a team of agents to complete tasks which require explicit cooperation. This verification is performed in a simulated environment as described in Section 8.3.1. The second set of experiments is used to verify the that WHILE constraint can be correctly utilized in the verbal instruction system (Chapter 4) to generate a task tree from a verbal instruction, which can then be directly executed by the robot. This verification is described in Section 8.3.2.

8.3.1 Architecture Integration Experiments

8.3.1.1 Validation Plan

In order to validate the addition of the interdependence constraint (WHILE) to the previously developed control architecture (Section 3.1), different types of task trees

were used to evaluate different situations for the WHILE node. The task trees used for validation determined if WHILE could function as a root node, if WHILE could be nested under other constraints, and if other constraints could be nested under WHILE. These combinations of nodes illustrate that the WHILE constraint can be used correctly in conjunction with the existing capabilities of the control architecture.

Figure 8.1 shows a task tree which combines various elements that were tested to ensure the WHILE worked properly within the architecture. The tree shows WHILE acting as a root node, as well as demonstrating that the *action task* of the WHILE node functions properly as a compound node. Additionally, this tree contains all four constraints types that the architecture can handle, demonstrating that the WHILE constraint does not disrupt the behavior of the other *goal nodes*.

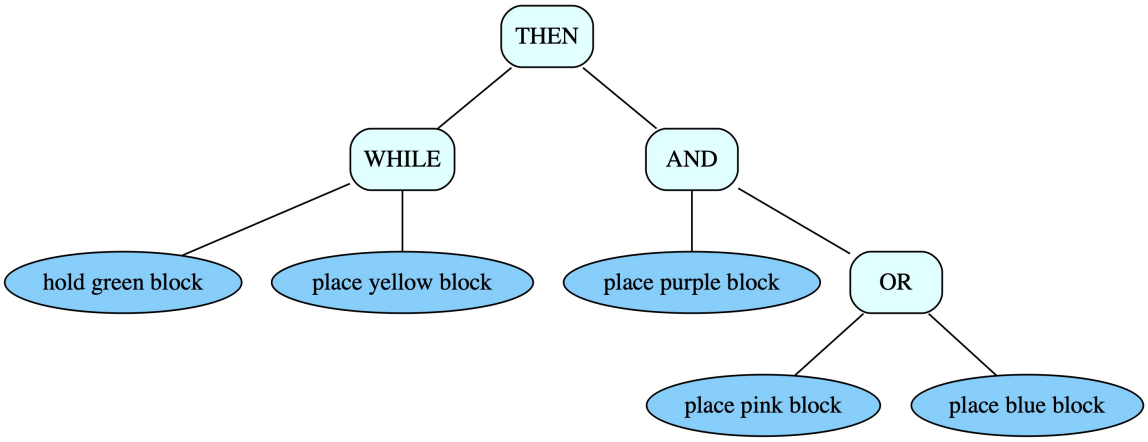


FIGURE 8.2: Task tree illustrating all four constraints (THEN, AND, OR, WHILE), as well as the nesting capability of the WHILE, since it is nested underneath another THEN constraint. The left child of the WHILE node (hold green block) is the HOLD behavior and the right child of the WHILE node (place yellow block) is the *action task* that must be completed. In this case the *action task* is a singular **behvaior node**.

Figure 8.2 also encompasses all four constraints of the architecture in the same task tree. In this task, the WHILE constraint is shown nested underneath another constraint, and the *action task* of WHILE is shown to be a singular *behavior node*.

The following section will discuss the results of validating the architecture with the trees shown in Figures 8.1 and 8.2. By validating the architecture using these task trees, we can see that the WHILE behavior is able to work properly with any type of task tree that may be created. The trees demonstrate that the WHILE constraint will work properly in conjunction with the other constraints, both nested underneath other constraints as well as a root node, and with both singular *behavior nodes* and compound nodes (combinations of *behavior* and *goal* nodes) as the WHILE's *action task*.

8.3.1.2 Experimental Validation

The modified architecture incorporating the interdependence constraint as specified in Section 8.1 was tested in simulation using the task trees provided in Figures 8.1 and 8.2. Presently, the architecture has yet to be validated through robot experiments in the real world.

Figure 8.3 shows the simulation that was used to validate the additions to the architecture. The circles represent the two robots which are collaborating to complete a joint task using the modified architecture. For this experiment, the blue circle represents the robot that will correspond to the PR2 in future real-world experiments

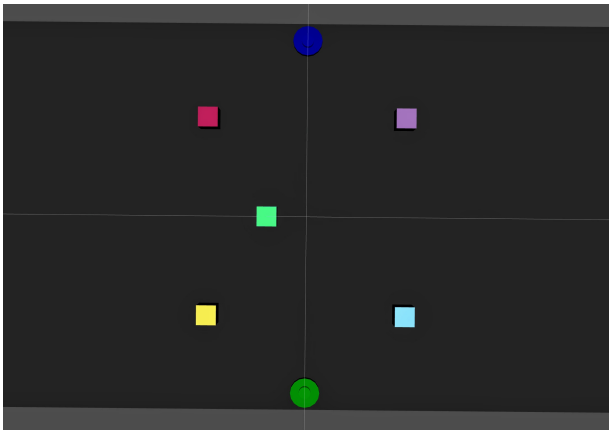


FIGURE 8.3: The simulation used to validate the architecture. Blocks are objects grabbed by the robots and circles are the robots themselves.

and the green circle represents the robot that will correspond to the Baxter in future real-world experiments. The objects are represented by blocks of various colors. Picking and placing an object is simulated by the "robots" moving over to a block, and then moving the block to a different location. HOLD is simulated by a robot moving over to a block, and "holding" the object, or hovering over the object, until the *action task* is completed. The block that was held is then placed by the robot. For simplicity, all of the blocks are placed in the same location for this simulation. For the simulation, the task completed by the robots corresponds to a particular task tree (either the tree shown in Figure 8.1 or the tree shown in Figure 8.2).

Figure 8.4 shows a heat map that corresponds to the results from validating the architecture with the task tree shown in Figure 8.1 by simulating the robots completing this task a total of 30 times. The objects which need to be grabbed during the task are shown on the left axis. The bottom axis represents the order in which they were grabbed by the robots, with the *1* being the object grabbed first and *4* being the

last object grabbed. The frequency in which the objects were grabbed in a particular order is shown on the right axis. The task tree in Figure 8.1 was tested in the simulation for a total of 30 trials. The heat map shows that the green block, which is the object that must be held by the WHILE node, was grabbed first by the robots in the simulation for every trial. This behavior is correct, since the HOLD child is the child that is activated first. Underneath the WHILE node is a THEN node, with AND and OR as its children. As shown in the heat map, the children of the AND node are grabbed second and third, alternating between the yellow and purple block. This behavior is expected, since AND is a non-ordering constraint. Finally, one of the children of the OR node is grabbed by the robot. In this case, the blue block was chosen 10 times, and the pink block was chosen 20 times. Given the OR constraint, a particular block was chosen depending on which one was closest to the robots at the time when the OR node was activated which differed across trials.

Figure 8.5 shows the heat map depicting the validation results from the task tree depicted in Figure 8.2 by simulating the robots completing this task a total of 30 times. The root of the tree is a THEN node, with WHILE and AND nodes as children. Because THEN is a sequential ordering constraint, WHILE is always activated first. The green object is again grabbed by the robots first, because it is the object that must be held. The pink block is always grabbed second, because placing the pink block is the *action task* of the WHILE node, and the WHILE behavior must be completed before moving on to AND, since they are both children of a THEN constraint. The children of the AND behavior can be completed in any order, shown in the heat map

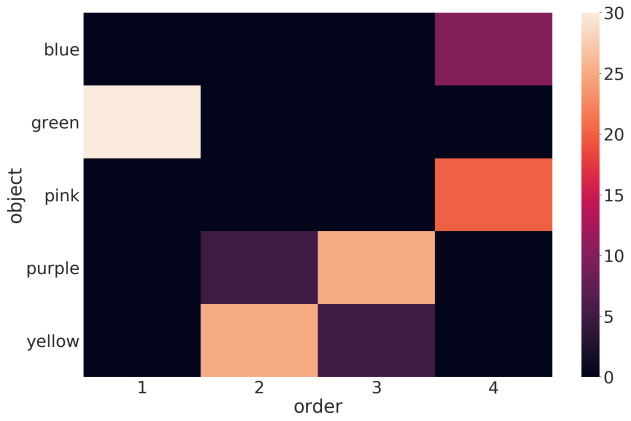


FIGURE 8.4: Heat map depicting the results from simulating the robots completing the task corresponding in the task tree depicted in Figure 8.1 a total of 30 times. The objects are shown on the left axis, the order in which they were grabbed by the robots are on the bottom axis, and the frequency in which they were grabbed in that order is shown on the right axis. We see that the green block, which is the object that must be held, was grabbed first each time, which is the correct behavior.

by the variation of objects that are grabbed third and fourth. The behavior of the OR node is shown as well, since the pink block was grabbed for 20 out of the 30 trials and the blue block was grabbed for 10 out of the 30 trials. Given the OR constraint, a particular block was chosen depending on which one was closest to the robots at the time when the OR node was activated which differed across trials.

8.3.2 Verbal Instruction Integration Experiments

To illustrate the incorporation of the interdependence constraint (WHILE) into the verbal system defined in Chapter 4, several experiments were completed.

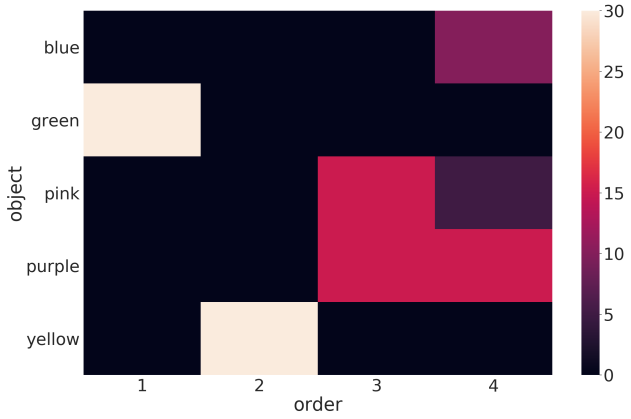


FIGURE 8.5: Heat map depicting the results from simulating the robots completing the task corresponding in the task tree depicted in Figure 8.2 a total of 30 times. The objects are shown on the left axis, the order in which they were grabbed by the robots are on the bottom axis, and the frequency in which they were grabbed in that order is shown on the right axis. Because THEN is a sequential ordering constraint, WHILE is always activated first, which is the correct behavior.

The experiments consist of verbal commands consisting of various combinations of verbs, prepositions, and conjunctions. These experiments show that the verbal instruction system is able to convert many different commands into the format of the task trees required by the control architecture as described in Section 3.1. It is also able to recognize and correctly parse multiple conjunctions (AND, OR, THEN, WHILE), two verbs (PLACE, HOLD), and multiple prepositions. The experiments are provided in Table 8.1.

These experiments verify the that WHILE constraint can be correctly utilized by the modified verbal instruction system described in Section 8.2 to generate the parenthesize command corresponding to a given verbal instruction. These parenthesized commands can then be converted into a task tree which can then be directly executed by the robot using the last step of the verbal instruction system outlined in

Num Verbs	**Num Preps**	**Num Conjs**	**Verbal and Parenthesized Commands**
1	0	0	**Input**: *"hold the pink bar"* **Output**: *(HOLD pink_bar)*
1	1	0	**Input**: *"place the green leg in front of you"* **Output**: *(PLACE green_leg robot in+front+of)*
1	1	1	**Input**: *"place the pink bar and the yellow bar on the green leg"* **Output**: *(AND (PLACE pink_bar green_leg on_l) (PLACE yellow_bar green_leg on_r))*
2	0	1	**Input**: *"place the yellow bar then hold the magenta bar"* **Output**: *(THEN (PLACE yellow_bar) (HOLD magenta_bar))*
2	1	1	**Input**: *"place the pink bar on the green leg while holding the yellow bar"* **Output**: *(WHILE (PLACE pink_bar green_leg on_l) (HOLD yellow_bar))*
2	1	1	**Input**: *"place the pink bar on the green leg or hold the blue leg"* **Output**: *(OR (PLACE pink_bar green_leg on_l) (HOLD blue_leg))*
2	2	1	**Input**: *"place the pink bar on the blue leg while placing the yellow bar on the green leg"* **Output**: *(WHILE (PLACE pink_bar blue_leg on_l) (PLACE yellow_bar green_leg on_r))*

TABLE 8.1: Verbal instruction experiments with varying numbers of verbs (Num Verbs), prepositions (Num Preps), and conjunctions (Num Conjs). The input to the system is a verbal instruction and the output is the parenthesized command. The experiments illustrate the system can correctly parse multiple conjunctions (AND, OR, THEN, WHILE), two verbs (PLACE, HOLD), and multiple prepositions

Chapter 4. Therefore, these experiments show that verbal instructions containing the WHILE constraint can be used to teach robots to complete tasks which require explicit cooperation between the agents.

8.4 Conclusion & Summary

The WHILE constraint presented in this chapter adds an interdependence task constraint to the previously developed control architecture (Section 3.1). In order to implement the WHILE constraint, a new *behavior node* was added to the architecture; HOLD. Previously, the robots picked up and immediately placed objects. With HOLD implemented, an object was picked up and held by one of the robots until a task was completed by the other robot, encoding the interdependence behavior of WHILE. The task trees used to validate the architecture were tested in simulation, showing a combination of situations that behaved correctly. While robot trials were not used to test the grasping and hold ability of the robots, the simulation validation is able to show that the fundamental behavior of the WHILE constraint is compatible and functional with the rest of the constraints in the architecture. Further experiments were completed to illustrate the integration of the WHILE constraint into the verbal instruction system presented in Chapter 4. These experiments modified the verbal instruction system to recognize and correctly parse multiple conjunctions (AND, OR, THEN, WHILE), two verbs (PLACE, HOLD), and multiple prepositions. Together, these experiments illustrate the integration of the interdependence task constraint into the proposed generalized task structure. Therefore, the generalized task structure is able to handle tasks which require explicit coordination between agents, which is one of the main contributions specified in Section 1.4.

Chapter 9

Generalized Task Structure Learning

This chapter focuses on generalized task structure learning for tasks with complex, hierarchical constraints. For the purposes of this work we assume the task structure follows the hierarchical architecture defined in our previous work 3.1. The methods in this chapter describe a learning framework (based on a genetic algorithm) which is able to learn the structure of a complex, hierarchical task through the use of human demonstrations. This learning framework can be used to teach a robot how to perform a task through human demonstration which further extends the capabilities of the proposed generalized task structure presented in this work.

9.1 Problem Representation

Learning from human demonstration consists of several major components. In order to learn a particular task, a human may provide a robot with several demonstrations of the task. Given these demonstrations, the first step is segmenting out the individual tasks from the demonstration. In our case, the individual tasks are the pick and place movements of a particular object. The second step is using these segmented demonstrations to learn how to perform the task. In our case, this entails learning the sequence in which the objects were placed. The last step is transferring these learned tasks to the robot to ensure the robot can completely complete the learned tasks.

For the purposes of the work presented in this chapter, we are focusing on the second step as the learning is the most important component for developing a generalized task structure as proposed in this work. In future work, the first step can be done in several ways, such as parsing video sequences or kinesthetic teaching, depending on the capabilities of the robot. However, this step is outside the scope of this work. The third step focuses on transferring the tasks to the robot. The success of this transfer depends on the capabilities of the robot and the scope of the task. This step is already addressed by several other chapters in this work, namely Chapter 5 to ensure correct execution through fault monitoring and Chapter 7 to incorporate the varying capabilities of the robot into the task allocation to ensure the robots are performing tasks which they are best suited for. Therefore, this chapter only

performs validation of the learning method on generated task trees rather than robot demonstrations. Experiments combining each of the main steps to ensure the learning performs correctly from start to finish will be explored in the future.

As mentioned above, the learning task in this work entails learning the ordering in which objects can be placed. A single demonstration in our case is represented by a particular ordering in which objects are placed. For example, in the tea task (Figure 4.4) one demonstration might be placing the *cup* first, then placing the *tea*, and finally placing the *sugar*. However, due to the constraints of a task, there may be multiple ways to perform a given task. Therefore, the learning scheme must be able to encompass the set of possible orderings within a single task structure. Because of this reason, we assume that the possible set of orderings are represented by a hierarchical task representation with a set of constraints (THEN, AND, OR) as discussed in Section 3.1. We also make a few assumptions regarding the demonstrations provided. The demonstrations provided must represent a fully completed task. In order to learn the ordering (*THEN*) constraint, we also incorporate a set of bad demonstrations which represent incorrect ways to perform the task. These bad demonstrations can be either completed tasks or partial tasks. Therefore, this chapter aims are solving the following problem: *Given a set of demonstrations, generate a hierarchical representation which accurately represents the constraints inherent in the demonstrations.*

9.2 Generalized Task Learning Framework

The goal of the generalized task learning framework is to learn a hierarchical representation which accurately represents the constraints inherent in a task. The learning is performed using a set of demonstrations. These demonstrations represent various orderings in which the objects can be placed for a particular task. For example, in the tea task (Figure 4.4) one demonstration might be placing the *cup* first, then placing the *tea*, and finally placing the *sugar*. To simplify the terminology used in this framework, each object is mapped to a unique number. Therefore, instead of a demonstration consisting of (*cup, tea, sugar*), a demonstration is represented by a numerical sequence where each number corresponds to a particular object such as (*1,2,3*) where *cup=1*, *tea =2*, and *sugar=3*. This numerical representation for a demonstration is used throughout the remainder of this chapter.

The learning framework proposed in this chapter is built around a Genetic Algorithm (GA). The method uses a novel compression-like encoding scheme to represent the chromosomes for the GA. The encoding scheme is discussed in Section 9.2.1. The provided demonstrations are used in the fitness function of the GA to determine how well the generated chromosomes fit the constraints inherent in the task, as discussed in Section 9.2.2. The modified GA algorithm is presented in Section 9.2.3.

9.2.1 Compression-based Encoding Scheme

The purpose of the GA is to generate chromosomes which represent a hierarchical representation for a given task. This work assumes that the representations are binary trees. The goal is to generate simple, human-interpretable trees in which each object for a given task appears only once. Given these two restrictions, a generated task tree can have at most (*n-1*) constraints (internal nodes) where n is the number of objects in the task. Because of this, for large tasks, the chromosomes can get very large. Therefore, we have designed a compression-based encoding scheme to maintain a consistent size of chromosomes for simplicity in the GA.

The basis of the compression-based encoding is a dictionary which stores the compressed chromosomes. Initially, the dictionary only contains the set of objects in a given task stored by their numerical representation used to convert the demonstration from object names to numbers as discussed in the beginning of Section 9.2. Therefore, for a task with n objects, the dictionary would contain the numbers *0-n* which represent the numerical representation of the objects. For example, given the tea task (Figure 4.4), the dictionary would initially consist of {*1: cup; 2: tea; 3: sugar* }.

The GA generates chromosomes of the form (*number_left, constraint, number_right*) where *number_left* and *number_right* are two different numbers in the dictionary and *constraint* is one of the constraints handled by the hierarchical representation (*THEN, AND, OR*). Therefore, in the initial population of the GA, the chromosomes are simple as they are built entirely from objects with a single constraint between them.

Looking back at the tea example, some examples of initial chromosomes would be *1T2, 2A3, 1O3*. The chromosome *1T2* would represent placing the *cup THEN tea*; the chromosome *2A3* would represent placing the *tea THEN sugar*; the chromosome *1O3* would represent placing the *cup OR sugar*.

For later generations of the GA, the chromosomes get more complex. At each generation, new rules may be added to the dictionary based on their fitness (Section 9.2.2). Therefore, for later generations, the *number_left* and *number_right* will no longer be numbers representing objects, but instead will be numbers in the dictionary representing other chromosomes. Looking back at the tea task, assuming the chromosomes *1T2* and *2A3* were stored in the dictionary as rules *4* and *5* respectively, the next generation could include a complex rule such as *1T5*. This rule is complex since *5* already represents another chromosome saved in the dictionary.

In this manner, the dictionary maintains the compressed chromosomes to allow complex chromosomes to be generated. Using this compression-based encoding, the chromosomes generated by the GA will always be of the form *(number_left, constraint, number_right)*. After the GA finishes, the dictionary can be used to decode the compression in order to get the complete chromosome as represented solely by the base numbers (*0-n*, corresponding to the objects) along with the constraints generated between them. Looking back at the tea task, assuming the ending chromosome was *1T5*, this tree can be decoded as (1T(2A3)). Applying the reverse mapping from numbers

to objects, this tree becomes *(Cup THEN (Tea And Sugar))*. The tree from the complete chromosome represents the entire hierarchical task structure in the form of the parenthesized notation mentioned in Chapter 4. Therefore, the ending chromosome can be directly executed by a robot to perform the learned task.

9.2.2 Fitness Function

In order to score the generated hierarchies, we must define a fitness function which takes into account the desired structure of our learned hierarchical task representation. This fitness function allows us to quantitatively evaluate how well a generated hierarchy represents the constraints inherent in the demonstrations. By using a quantitative measure, we are able to compare different hierarchies and determine which one best represents our desired output. In our context, a good fit will be a simple task tree which is interpretable by a human yet accurately represents all of the sequences provided as demonstrations. A bad fit would be things such as a tree which is overly complex and incomprehensible to a human or a tree whose constraints do not fit the majority of sequences provided. In other words, the aim is to generate trees which are compact yet are able to represent the complete set of constraints inherent to a task. The correctness of the trees is considered by taking into account how well the tree reflects the provided demonstrations. Further details on these measures are provided below.

Our system must be able to learn tasks which contain three types of constraints: *THEN, AND, OR*. There are several measures we must take to ensure that each of these types of constraints can be learned. The *THEN* and *AND* constraint have similar behaviors in terms of the sequence of demonstrations. A sequence which contains *(A,B)* can be represented as either *A THEN B* or *A AND B*, unless there is some notion of ordering illustrated within the demonstrations. By looking solely at good demonstrations, it is very difficult to enforce this ordering constraint. Therefore, we also include a set of bad demonstrations, demonstrations which provide incorrect orderings of the task, to incorporate some form of ordering constraint to allow the GA to learn the *THEN* constraint easier. If a good demonstration has *(A,B)* and a bad demonstration has *(B,A)*, then we know the constraint in the task must be a *THEN* instead of an *AND* since the *AND* constraint holds for this bad demonstration when it should result in a failure, just like the *THEN* does.

The fitness function used in the GA is defined in Equation 9.1.

$$score = count_{good} * multiplier - count_{bad} * multiplier \tag{9.1}$$

$$multiplier = (multiplier_{left} + multiplier_{right}) * w_{constraint} \tag{9.2}$$

$$where\ w_{constraint} = \begin{cases} 4 & THEN \\ 1 & AND \\ 1 & OR \end{cases} \tag{9.3}$$

The fitness function uses a multiplier style score which weights the rules based on the combinations of constraints within the rule. Since higher-level *THEN* constraints, those which appear closer to the root of the tree, are the hardest to illustrate through demonstrations alone, this scoring metric was designed to prioritize trees which have higher-level *THEN* constraints. The multiplier grows with the depth of the tree. The multiplier component has a maximum value of $multiplier^{(n-1)}$ where n is the number of objects in the task. This comes from the fact that the trees represented are binary trees and thus the max depth of the tree is at most *n-1*. At each depth in the tree the multiplier is calculated as in Equation 9.2. Each type of constraint has a different multiplier value associated with it. The $multiplier_{left}$ is the multiplier for the left sub-tree at a given level and $multiplier_{right}$ is for the right sub-tree. Therefore, the multiplier component factors in the number of each type of constraint that exists in the overall tree, which has at most *n-1* constraints. The *THEN* constraint has a higher multiplier than the *AND* and *OR* nodes. Therefore, trees with *n-1* which have *THEN* constraints closest to the root will have the highest score. This allows the GA to learn the higher-level *THEN* constraints which are difficult to illustrate through demonstration alone. In order to ensure the GA is learning the true constraints,

and not just a tree with all *THEN* constraints, the human demonstrations are also factored into the fitness function.

In addition to the multiplier components in the fitness function, the function takes into account the number of demonstrations which accurately reflect the constraints represented in a particular chromosome. The $count_{good}$ represents the number of good demonstrations which fit the chromosome's constraints. The $count_{bad}$ represents the number of bad demonstrations that fit the chromosome's constraints. Again, the bad demonstrations are examples of orderings which break the real constraints of a task and therefore, if a chromosome fits these demonstrations, then they are not learning the correct constraints. Therefore, the fitness function (Equation 9.1 uses the weighted difference between these two values to ensure that the chromosome fits the good demonstrations but not the bad ones. Determining whether or not a chromosome fits a particular rule is discussed in Section 9.2.2.1 below.

If the score is above a certain threshold (in our case we used the number of good demonstrations), the chromosome gets saved as a new rule in the dictionary. This rule is then able to be used to generate new chromosomes in future iterations. Otherwise, the chromosome is saved into another dictionary which stored the bad functions. In order to save time, the scores for a particular chromosome are saved inside of the corresponding entry in the dictionary. This way, the scores do not have to be recomputed each time this chromosome gets generated, but instead the scoring becomes a

look-up process using the dictionary. The same look-up process happens for chromosomes which were found to be bad and were stored in the bad function dictionary.

9.2.2.1 Evaluation Method

A chromosome fits a certain demonstration if the ordering and constraints between the objects are all reflected in the demonstration. Assist functions were designed for each of the possible constraints (*THEN, AND, OR*) and were used to determine if a particular chromosome fits a demonstration.

The assist function for *THEN* takes a chromosome of the form *(number_left, THEN, number_right)* where *number_left* and *number_right* are two different numbers in the dictionary of rules. The assist function checks whether *number_left* and *number_right* occur in the correct order in a given demonstration. Additionally, it checks whether or not a certain object appears more than once in a given chromosome. To keep track of duplicates, the dictionary entry for a given rule also contains a list of objects which appear in the rule. Therefore, the assist function can search the rule entry for the *number_left* and *number_right* entries to search for duplicate objects. If *number_left* occurs before *number_right* in the demonstration and no objects appear more than once in the chromosome, then the assist function for *THEN*returns a pass, meaning that the chromosome fits the constraints of the demonstration. Otherwise, it returns a fail, meaning it does not fit the demonstration's constraints.

The assist function for *AND* behaves in a similar manner to that of the assist function for *THEN*. It takes a chromosome of the form *(number_left, AND, number_right)* where *number_left* and *number_right* are two different numbers in the dictionary of rules. The assist function checks if both *number_left* and *number_right* appear in the demonstration. In this case, it does not matter which one appears first in the demonstration, since the *AND* is a non-ordering constraint. Additionally, it checks for duplicate objects in the same way as the *THEN* assist function. If both *number_left* and *number_right* appear in the demonstration, in any order, and no objects appear more than once in the chromosome, then the assist function for *AND* returns a pass, meaning that the chromosome fits the constraints of the demonstration. Otherwise, it returns a fail, meaning it does not fit the demonstration's constraints.

The assist function for *OR* also behaves in a similar manner. It takes a chromosome of the form *(number_left, OR, number_right)* where *number_left* and *number_right* are two different numbers in the dictionary of rules. The assist function checks if either *number_left* or *number_right* appear in the demonstration, but not both. Additionally, it checks for duplicate objects in the same way as the *THEN* assist function. If either *number_left* or *number_right* but not both appear in the demonstration, and no objects appear more than once in the chromosome, then the assist function for *OR* returns a pass, meaning that the chromosome fits the constraints of the demonstration. Otherwise, it returns a fail, meaning it does not fit the demonstration's constraints.

These assist functions can be stacked in order to recursively evaluate a complex

chromosome with multiple constraints. In this case, the *number_left* and *number_right* would be complex chromosomes stored in the dictionary which are encoded using the scheme described in Section 9.2.1. Therefore, to evaluate if the complex chromosome fits the constraints, both chromosomes encoded by *number_left* and *number_right* need to be evaluated for correctness. In this way, the complex chromosomes can be recursively evaluated using the appropriate assist functions at each step to determine whether or not all constraint that exist in the complex chromosome hold. If all constraints within the chromosome hold, meaning all recursive assist function calls return pass, then the entire complex chromosome fits a given demonstration. If at any point, a constraint is not held, the entire chromosome fails and is determined to not fit a given demonstration.

Another benefit of these assist functions is that they help to limit the complexity of the rules which are generated. If there are duplicates of the objects, the rule fails. This implies that no objects can be repeated in the sequence. Therefore, this formulation only allows trees to be generated which have at most a depth of *n-1*. This stops the trees generated by the GA from becoming too complex for a given task, which upholds the aim of generating simple, human-interpretable task trees.

9.2.3 Modified Genetic Algorithm

The framework uses a modified version of the standard GA method. The modified algorithm consists of two major changes to a standard GA: 1) each generation, the

chromosomes which have a fitness above a certain threshold are added to the dictionary which contains the compressed chromosomes (Sections 9.2.1 and 9.2.2) and 2) the best resulting chromosome at the end of the GA must be decoded using the dictionary to get the complete hierarchical plan ((Sections 9.2.1). The modified algorithm is presented in Algorithm 19.

Algorithm 19 Modified genetic algorithm for learning a hierarchical task representation

```
1: Generate initial population of k individuals.
2: for each individual in the population do
3:    Evaluate the fitness
4:    if fitness > THRESH then
5:       Add individual to dictionary
6:    else
7:       Add individual to bad dictionary
8:    end if
9: end for
10: while i < MAX_ITERS do
11:    Select top 70% of previous population as offspring
12:    Generate other 30% of offspring through the following:
13:    Generate 15% new individuals with any numbers in the dictionary as number_left and number_right
14:    Generate 15% new individuals with simple numbers corresponding only to the objects as number_left and number_right
15:    Cross over offspring to form new offspring with probability p_c
16:    Mutate new offspring with probability p_m
17:    Place new offspring in a new population
18:    for each individual in the population do
19:       Evaluate the fitness
20:       if fitness > THRESH then
21:          Add individual to dictionary
22:       else
23:          Add individual to bad dictionary
24:       end if
25:    end for
26: end while
27: Find the individual with highest fitness in dictionary
28: Decode the individual to get the complete hierarchical plan
```

The main implementation for the GA was done using the DEAP framework [107]. Although DEAP was able to provide the standard GA methods, several major components of the GA had to be manually adapted to fit the problem specified in this work for learning hierarchical task structures. The individuals were generated using the form *(number_left, constraint, number_right)* as described in Section 9.2.1. Evaluation of the fitness was done as described in Section 9.2.2. The selection in line 11 was performed through the roulette selection method specified by DEAP. The crossover in line 15 was performed with the one point crossover method specified by DEAP.

The mutation in line 16 was performed using a manually defined method as described in Algorithm 20. The algorithm mutates one of the components of an individual: *number_left*, *number_right*, or *constraint*. When a mutation occurs, the algorithm will choose to mutate *number_left* and *number_right* 25% of the time each. The other 50% of the time, the algorithm will choose to mutate *constraint*. For mutation of *number_left*, the algorithm will generate a number of a chromosome that exists in the dictionary. If this number is the same as the *number_right* in the given individual, meaning a duplicate object would be found, or the number is the original *number_left* in the given individual (no mutation actually occurred), the algorithm generates a new number. The opposite logic occurs for the mutation of *number_right*. For a mutation of the *constraint*, a *THEN* constraint will be generated 66% of the time, and *AND* and *OR* constraints will be generated 33% of the time.

Algorithm 20 Mutate(individual)

```
mutated = individual
Generate random position pos for mutation from 1 to 4
if pos == 0 then
    generate random number mut from 0 to length of dictionary
    while mut == individual.number_left or mut == individual.number_right do
        generate a new random number mut from 0 to length of dictionary
    end while
    mutated.number_left == mut
else if pos == 2 then
    generate random number mut from 0 to length of dictionary
    while mut == individual.number_left or mut == individual.number_right do
        generate a new random number mut from 0 to length of dictionary
    end while
    mutated.number_right == mut
else if pos == 1 or pos == 3 then
    generate constraint type const out of {THEN, AND, OR} with probabilities
    (66%, 33% ,33%) respectively
    mutated.constraint == const
end ifRETURN mutated
```

9.3 Experimental Validation

The GA method (Algorithm 19) is validated on three different experiments. In order to evaluate whether or not the GA produced valid representations of a set of demonstrations, each experiment is designed from a human-generated task tree to act as ground truth. Each of these task trees contain one of the constraints (*THEN*, *AND*, *OR*) as a root node of the tree and various combinations of constraints underneath the root. These task trees are used to generate both a set of good demonstrations which represent valid orderings for each tree and a set of bad demonstrations which represent invalid orderings, or orderings which break the constraints of the tree. These bad demonstrations can be either full sequences (containing a full ordering of objects)

or partial sequences (containing a partial ordering of a few objects). These demonstrations are passed as input to the GA in order to learn a hierarchical representation which reflects the inherent task constraints. The human-generated task trees are not given to the GA as input. Instead, these trees are used to verify whether or not the task trees generated by the GA reflect the same constraints as those provided by the human. This verification is done by hand. The experiments used for validation along with discussion of each are described below. Together, these experiments illustrate that the GA is able to learn hierarchical tasks with complex constraints through the use of human demonstration, as specified in one of the main contributions given in Section 1.4.

For our experiments, the following values were used for the variables in Algorithm 19: k was set to 1000, *THRESH* was set to the number of good demonstrations provided, *MAX_ITERS* was set to 20, p_c was set to 80%, and p_m was set to 40%.

9.3.1 Experiment for THEN constraint as root

The first experiment tests whether or not the GA is able to learn a task tree with a *THEN* constraint as the root node and combinations of other constraints underneath. The human-generated task tree, input demonstrations, and task tree generated by the GA are provided in Figure 9.1. We see that the demonstrations accurately reflect the constraints given in the human-generated tree. For instance, the first four objects are always ordered within the first part of the sequence and the other three objects are

ordered in the last part of the sequence, which represents the high-level *THEN* node at the root of the tree. In the bad demonstrations, we see that several constraints are broken, such as the *THEN* constraint at the root. This is evident through the swapping of the first and second group of objects mentioned above. Another example of a broken constraint is the *OR* constraint since both *2* and *3* appear in the same demonstration.

If the GA is working correctly, the constraints in the task tree generated should be similar to those in the human-generated tree. In Figure 9.1 (c) we see that this is indeed the case. The trees generated by the GA reflect the same task constraints that were given in the ground truth tree provided by a human. In this experiment, we see that multiple representations can reflect the same constraints since the trees are not unique. Since *THEN* is an ordering constraint, the left child of a *THEN* must be completed first. Thus in all three trees we see that *0, (2 OR 3), 1, and 4* all happen in the same ordering as that shown in the human's task tree. We also see that the *5* and *6* can happen in either order since they are related by an *AND* constraint, as illustrated in the ground truth. The fitness of each sub-tree is given for each tree an the final fitness is given in bold font. We see that different structure in the third tree results in a slightly lower fitness even though it produces a correct output since our fitness function encourages higher-level *THEN* nodes. From this experiment we see that the GA is able to correctly learn a task tree with the *THEN* constraint as a root and combinations of other constraints underneath.

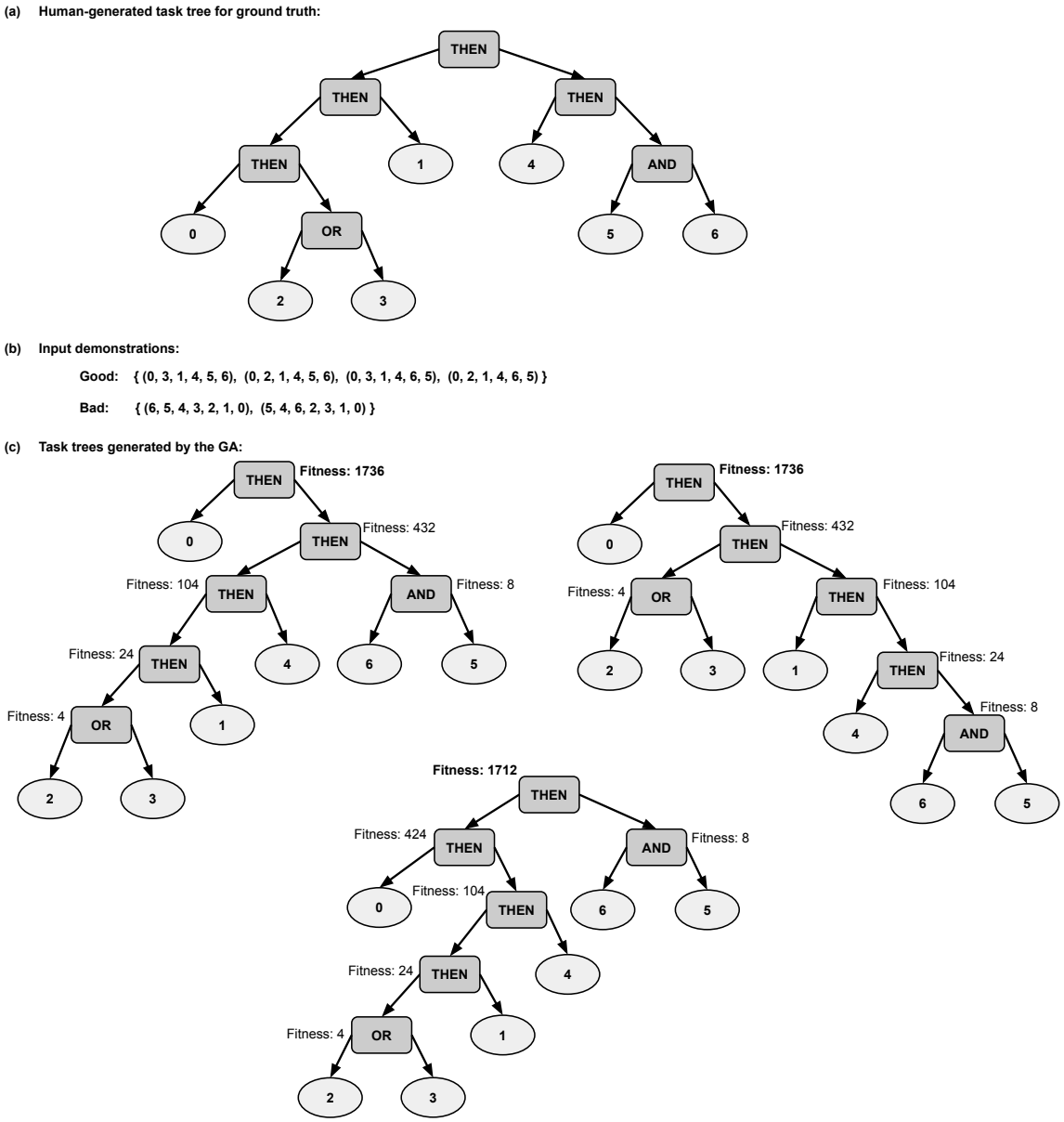


FIGURE 9.1: Experiment for the task tree with the *THEN* constraint at the root and combinations of other constraints below. (a) The human-generated task tree used for ground truth for the experiment. (b) The demonstrations (both good demonstrations and bad demonstration) used as input to teach the GA. (c) Three sample task trees generated by the GA with the fitness of each sub-tree provided next. The total fitness for the tree is given at the root in bold font. We see that the trees in (c) resemble the tree in (a) so the GA is able to learn correct trees in this case.

9.3.2 Experiment for AND constraint as root

The second experiment tests whether or not the GA is able to learn a task tree with a *AND* constraint as the root node and combinations of other constraints underneath. The human-generated task tree, input demonstrations, and task tree generated by the GA are provided in Figure 9.2. We see that the demonstrations accurately reflect the constraints given in the human-generated tree. For instance, the first four objects are always grouped together and the last three objects are always grouped together in the orderings (due to the THEN constraints for these sets of objects), but these groups can occur either at the beginning or the end of the task, which reflects the *AND* node at the root of the tree. As mentioned above, the bad demonstrations can be full orderings, such as those in the first experiment, or partial orderings of a subset of objects, such as those in this experiment. We see that the partial orderings represent infractions of the constraints in the left and right sub-trees of the root node.

If the GA is working correctly, the constraints in the task tree generated should be similar to those in the human-generated tree. In Figure 9.2 (c) we see that this is indeed the case. The trees generated by the GA reflect the same task constraints that were given in the ground truth tree provided by a human. We see that given the root *AND* constraint, the left and right subtrees in the ground truth appear on either side of the *AND* in the trees generated by the GA. This illustrates that the GA is able to correctly learn the non-ordering constraint of the *AND*. We also see as in the *THEN* experiment above, since the trees are not unique, the *THEN* constraints

appear in a different structure, but still reflect the same constraints as the ground truth. Lastly, we see that the *OR* constraint can have either the *2* or the *3* appear first, which illustrates that the ordering of the *OR*'s children doesn't matter, since we only care about at most one of the children appearing in the overall task. We see that different structure in the first tree results in a slightly lower fitness even though it produces a correct output since our fitness function encourages deeper branches off of *THEN* nodes due to the multiplier component. From this experiment we see that the GA is able to correctly learn a task tree with the *AND* constraint as a root and combinations of other constraints underneath.

9.3.3 Experiment for OR constraint as root

The third experiment tests whether or not the GA is able to learn a task tree with a *OR* constraint as the root node and combinations of other constraints underneath. The human-generated task tree, input demonstrations, and task tree generated by the GA are provided in Figure 9.3. We see that the demonstrations accurately reflect the constraints given in the human-generated tree. For instance, either the first two objects appear in the demonstration or the second two objects appear, but not both, which reflects the *OR* node at the root of the tree. In the bad demonstrations, we see that this *OR* constraint at the root is broken since all of the objects are represented in the task.

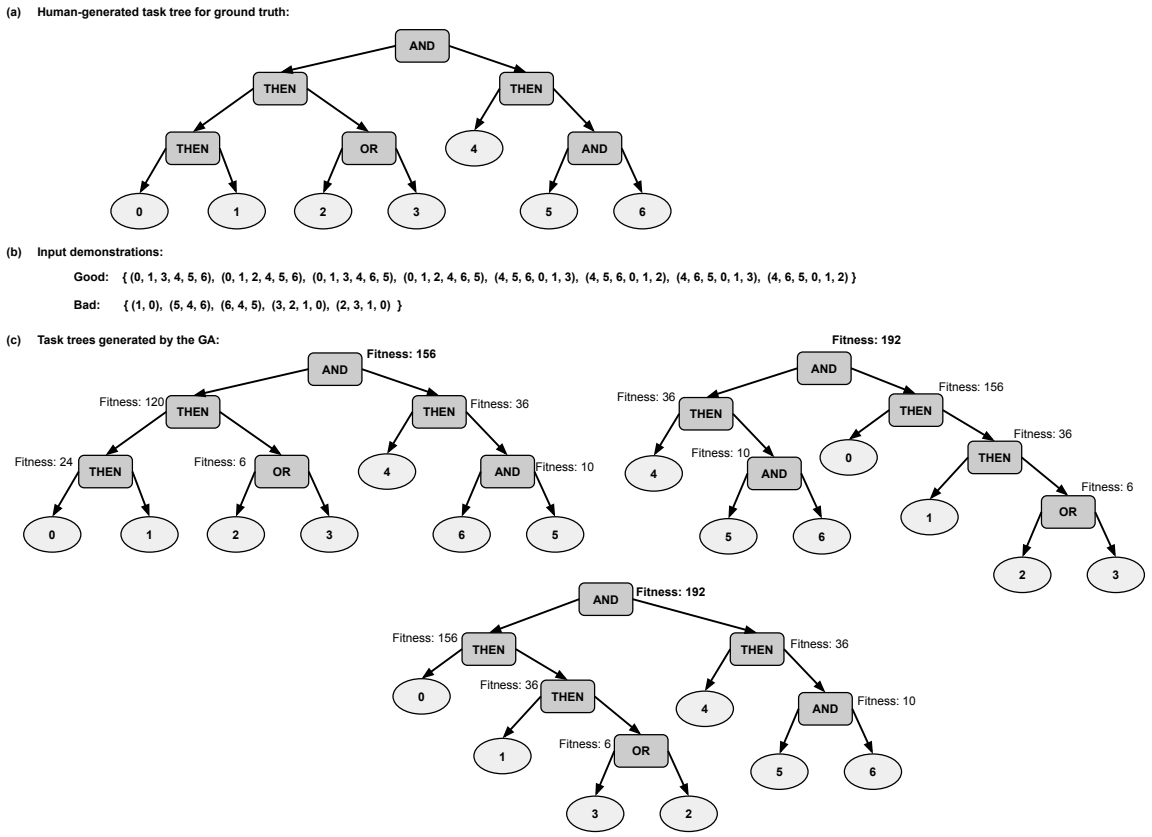


FIGURE 9.2: Experiment for the task tree with the *AND* constraint at the root and combinations of other constraints below. (a) The human-generated task tree used for ground truth for the experiment. (b) The demonstrations (both good demonstrations and bad demonstration) used as input to teach the GA. (c) Three sample task trees generated by the GA with the fitness of each sub-tree provided next. The total fitness for the tree is given at the root in bold font. We see that the trees in (c) resemble the tree in (a) so the GA is able to learn correct trees in this case.

If the GA is working correctly, the constraints in the task tree generated should be similar to those in the human-generated tree. In Figure 9.3 (c) we see that this is indeed the case. The trees generated by the GA reflect the same task constraints that were given in the ground truth tree provided by a human. We see that the task trees generated by the GA contain the same sub-trees of the *OR* as in the ground truth. The ordering of these sub-trees in the generated tasks can be in either branch, since

the *OR* constraint doesn't care about the ordering of the child, but only that at most one of the children appears in the task. We also see in the *AND* sub-tree, the *0* and *1* can appear in either order, since the *AND* is a non-ordering constraint. We see that each of the three trees have the same structure so they all have an equivalent fitness. From this experiment we see that the GA is able to correctly learn a task tree with the *OR* constraint as a root and combinations of other constraints underneath.

9.4 Conclusion & Summary

This chapter focuses on generalized task structure learning for tasks with complex, hierarchical constraints. For the purposes of this work we assume the task structure follows the hierarchical architecture defined in our previous work (see Section 3.1). The methods in this chapter describe a learning framework (based on a genetic algorithm) which is able to learn the structure of a complex, hierarchical task through the use of human demonstrations. This learning framework can be used to teach a robot how to perform a task through human demonstration which further extends the capabilities of the proposed generalized task structure presented in this work.

Learning from human demonstration consists of several major components. In order to learn a particular task, a human may provide a robot with several demonstrations of the task. Given these demonstrations, the first step is segmenting out the individual tasks from the demonstration. In our case, the individual tasks are the pick and

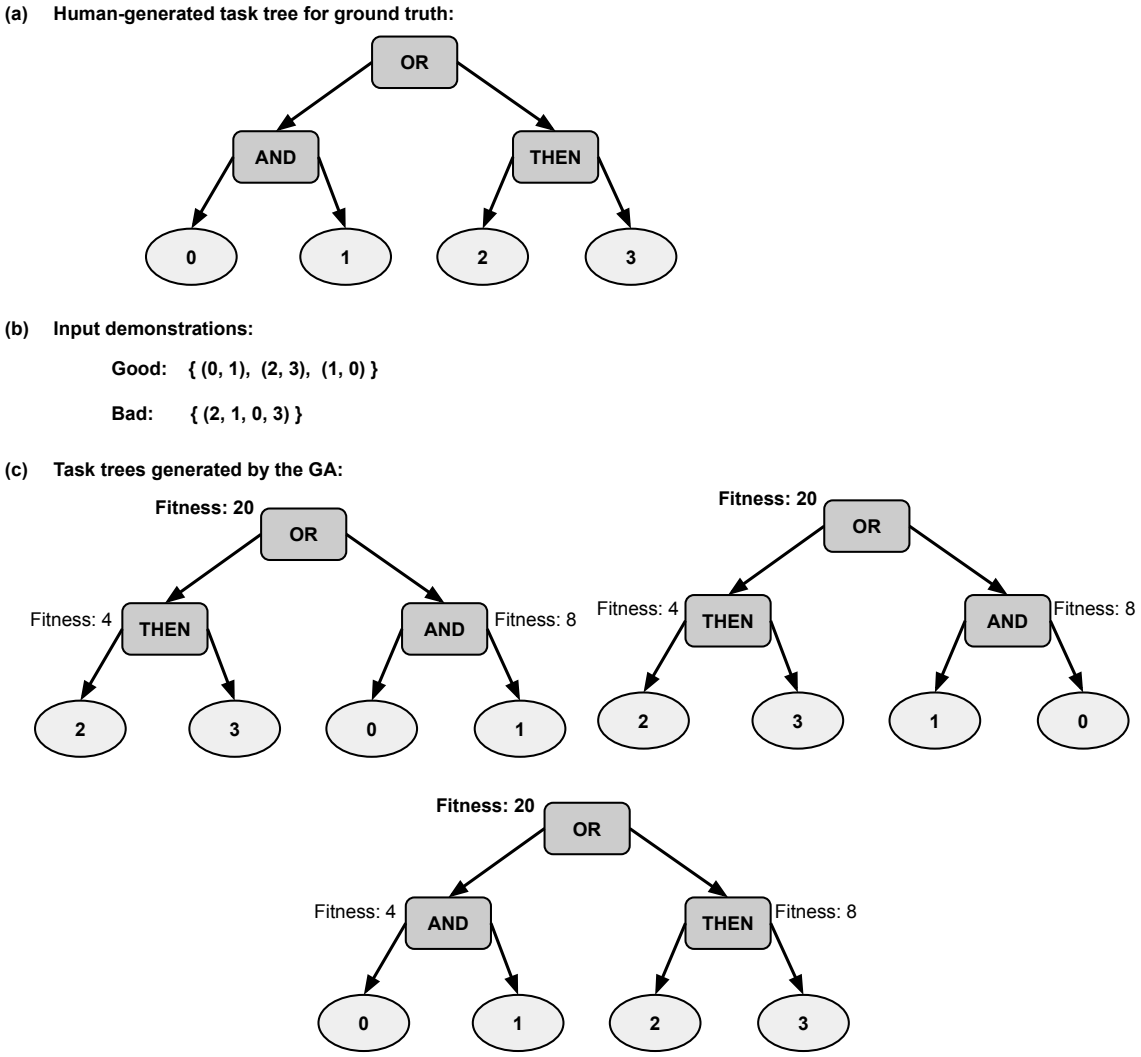


FIGURE 9.3: Experiment for the task tree with the *OR* constraint at the root and combinations of other constraints below. (a) The human-generated task tree used for ground truth for the experiment. (b) The demonstrations (both good demonstrations and bad demonstration) used as input to teach the GA. (c) Three sample task trees generated by the GA with the fitness of each sub-tree provided next. The total fitness for the tree is given at the root in bold font. We see that the trees in (c) resemble the tree in (a) so the GA is able to learn correct trees in this case.

place movements of a particular object. The second step is using these segmented demonstrations to learn how to perform the task. In our case, this entails learning the sequence in which the objects were placed. The last step is transferring these learned tasks to the robot to ensure the robot can completely complete the learned tasks. For the purposes of the work presented in this chapter, we are focusing on the second step as the learning is the most important component for developing a generalized task structure as proposed in this work.

The learning task in this work entails learning the ordering in which objects can be placed. A single demonstration in our case is represented by a particular ordering in which objects are placed. However, due to the constraints of a task, there may be multiple ways to perform a given task. Therefore, the learning scheme must be able to encompass the set of possible orderings within a single task structure. Because of this reason, we assume that the possible set of orderings are represented by a hierarchical task representation with a set of constraints (THEN, AND, OR) as discussed in Section 3.1. Therefore, this chapter aims are solving the following problem: *Given a set of demonstrations, generate a hierarchical representation which accurately represents the constraints inherent in the demonstrations.*

The learning framework proposed in this chapter is built around a Genetic Algorithm (GA). The method uses a novel compression-like encoding scheme to represent the chromosomes for the GA. The encoding scheme is discussed in Section 9.2.1. The provided demonstrations are used in the fitness function of the GA to determine how

well the generated chromosomes fit the constraints inherent in the task, as discussed in Section 9.2.2. The modified GA algorithm is presented in Section 9.2.3.

The GA method (Algorithm 19) is validated on three different experiments. In order to evaluate whether or not the GA produced valid representations of a set of demonstrations, each experiment is designed from a human-generated task tree to act as ground truth. Each of these task trees contain one of the constraints (*THEN*, *AND*, *OR*) as a root node of the tree and various combinations of constraints underneath the root. These task trees are used to generate both a set of good demonstrations which represent valid orderings for each tree and a set of bad demonstrations which represent invalid orderings, or orderings which break the constraints of the tree. These demonstrations are passed as input to the GA in order to learn a hierarchical representation which reflects the inherent task constraints. The human-generated task trees are used to verify whether or not the task trees generated by the GA reflect the same constraints as those provided by the human. Together, the three experiments illustrate that the GA is able to learn hierarchical tasks with complex constraints through the use of human demonstration, as specified in one of the main contributions given in Section 1.4.

Chapter 10

Conclusion & Future Work

10.1 Conclusion

The goal of the proposed work is to develop a generalized task structure which enables collaborative task allocation for complex, hierarchical tasks for both multi-robot and human-robot teams. The basis of this work is a previously developed collaborative multi-robot control architecture described in Chapter 3. This architecture focuses on the problem of task allocation under hierarchical constraints imposed on a joint task. Real-world tasks are not only a series of sequential steps, but typically exhibit a combination of multiple types of constraints, where some parts of the task are sequential, some have no ordering constraints, and others allow for alternative paths of execution. Therefore, to enable multi-robot and human-robot teams to complete joint tasks in the real world, the design of a generalized hierarchical control architecture

which is able to encompass all of these types of constraints is necessary. One primary example which illustrates this concept is a building task. In order to correctly build a piece of furniture, certain parts have to be connected first whereas others can be attached at various points in the process.

In order to realize the development of a generalized task structure for task allocation for both multi-robot and human-robot teams, several major extensions to our previously developed control architecture are proposed in Chapters 4-9. Each of the extensions proposed in these chapters correspond to a major contribution towards the development of such a generalized task structure. The proposed extensions of this work fall into two categories: multi-robot capabilities and human-robot capabilities. These contributions are described below along with the major conclusions drawn from their corresponding chapters as well as their connection to the development of a generalized task structure.

Human-robot capabilities:

- A novel approach to robot task learning from verbal instruction.
  - Chapter 4 describes a novel approach to transfer complex task knowledge from a human user to a robot, with the goal of exploiting the richness of natural language instructions in order to increase the complexity of task representations that a robot can learn. In particular, the focus was on learning tasks which *convey complex execution constraints* (such as alternative paths of execution, sequential or non-ordering constraints, as well

as hierarchical representations), as well as on *enabling behavior parameterization* through the instruction.

  - This verbal instruction system allows the proposed generalized task structure to be utilized to teach robots to perform tasks through verbal instruction.

- A fault recovery system able to detect and inform users of failures and resolve them through dialogue.

  - Chapter 5 described development of a hierarchical control architecture that *1) autonomously detects and is cognizant of task execution failures*, *2) initiates a dialogue with a human helper to obtain assistance*, and *3) enables collaborative human-robot task execution through extended dialogue* in order to *4) ensure robust execution of hierarchical tasks with complex constraints, such as sequential, non-ordering, and multiple paths of execution.*

  - The incorporation of a fault recovery system which is able to detect and inform users of failures and resolve them through dialogue allows for a more robust task allocation mechanism. Additionally, it enables the proposed generalized task structure to be utilized in complex, hierarchical tasks which are prone to failures as well as those which require collaboration between humans and robots.

- An extension of our previously developed control architecture to facilitate collaboration by human-robot teams.

  - Chapter 6 describes the extension of the multi-robot control architecture to the human-robot domain through the development of a human intent recognition system which enables to robot to identify which part of the task the human has already completed as well as what the human is currently working on so that the agents can work independently on the joint task with minimal interruption.
  - This contribution allows the generalized task structure to be utilized for human-robot teams performing collaborative tasks as well as for tasks in which the agents can work independently.

Multi-robot capabilities:

- An extension of our previously developed control architecture for incorporating the varying capabilities of a team of heterogeneous robots.
  - Chapter 7 describes the development of an architecture that enables *collaborative execution of tasks with hierarchical representations and multiple types of execution constraints* by teams of robots with *variable heterogeneity* through the utilization of a continuous-valued metric representing the agents' ability to perform a given task.
  - The variable heterogeneity capabilities can be utilized by the generalized task structure to encourage a robust and reliable task allocation scheme by allocating robots to tasks which best fit their specific skills.

- An extension of our previously developed control architecture for an interdependence constraint which requires explicit coordination between agents.
  - Chapter 8 describes the addition of a *WHILE* constraint and a *HOLD* behavior to the control architecture in order to enable explicit coordination between agents on a particular part of a given task through the interdependence constraint. This additional constraint was also incorporated into the verbal instruction system described in Chapter 4 to allow a human to train a robot for tasks which utilize this constraint through verbal instruction.
  - This addition allows the generalized task structure to represent tasks which require explicit coordination between agents on various parts of a given task.
- A novel method based on a genetic algorithm which is able to take sequences of demonstrations and learn a hierarchical task representation.
  - Chapter 9 describes a learning method based on a genetic algorithm which which is able to learn the structure of a complex, hierarchical task through the use of human demonstrations. This structure can then be used directly by the robot to execute the trained task.
  - This contribution defines a scheme which allows the generalized task structure to learn tasks from human demonstration.

10.2 Future Work

10.2.1 Multi-Robot Task Allocation

One future area for exploration in the multi-robot domain is using both arms of each robot. This would lead to increased team efficiency as well as allow for completion of more complex tasks. To accomplish this, specific modules for obstacle avoidance will need to be developed in order to avoid collisions during the task execution.

Several extensions can be made to further account for the varying skills in a team of heterogeneous robots. One extension is to include additional features in the computation of the robot's performance metric, such as a feature that provides insight into the efficiency of the trajectories computed to reach the objects and the destination. Another extension would be to add to the repertoire of types of tasks that the generalized task structure can handle. The easiest addition would be to develop a new *behavior node* which utilizes the navigation skills of certain robots. This would emphasize the varying capabilities between stationary and non-stationary robots and would allow for a wider range of tasks to be utilized by the proposed generalized task structure.

10.2.2 Human-Robot Task Allocation

An immediate extension of the proposed work to allow for both human-robot and multi-robot teams is to incorporate both the proposed fault recovery system (Chapter 5) and verbal instruction system (Chapter 4) into teams consisting of both humans and multiple robots. This extension would allow the dialogue systems developed in both chapters to be utilized for communication not only between human and robot teammates, but also between multiple robot teammates. This would allow for robots to ask for assistance both from the human as well as the other robot teammates during execution failures. This would also allow for humans to use verbal instructions to teach teams of robots to learn new tasks simultaneously. Both of these extensions would help to increase the human-robot collaboration capabilities of the proposed generalized task structure. They would also allow for a wider range of applications which require more than two agents to complete.

10.2.3 Generalized Task Learning

In order to further the proposed research there are two main outlets which can be pursued for generalized task learning. The first one is to further extend the proposed generalized task learning method to allow it to learn the primitives used to generate the sequences through visual representation. As it stands, the proposed work assumes that the set of primitives used to generate the demonstration sequences are predefined. However, this requires the encoding of these primitives to physical actions on the robot

to be defined by hand. This extension would avoid this by allowing the system to first learn these primitives through raw visual data. This extension would begin to combine a type of generalized policy learning method with the proposed generalized task learning method to create a complete end-to-end system capable of learning complex task representations and transfer them to teams of heterogeneous robots through only the use of raw visual data.

The second is to develop a system which enables joint learning from multiple robots on-line. In this extension there would be two added benefits: i) allowing the system to learn on-line through rehearsing the task multiple times in order to learn how to best complete it and ii) allowing the generated task representation to fuse the on-line observations of multiple robots at a time to create a single task representation that works for both robots. In this case, the tasks would be split between two robots and each robot would perform certain tasks which they find they are more suited for. Therefore some parts of the task may only be seen by a single robot, which means that the robots would need to communicate and collaborate in order to generate a complete task representation which encompasses the correct hierarchy of sub-tasks.

Bibliography

[1] Luke Fraser, Banafsheh Rekabdar, Monica Nicolescu, Mircea Nicolescu, David Feil-Seifer, and George Bebis. A compact task representation for hierarchical robot control. In *International Conference on Humanoid Robots*, pages 697–704, Cancun, Mexico, November 2016. IEEE. ISBN 978-1-5090-4717-8. doi: 10.1109/HUMANOIDS.2016.7803350.

[2] Luke Fraser, Banafsheh Rekabdar, Monica Nicolescu, Mircea Nicolescu, and David Feil-Seifer. A hierarchical control architecture for robust and adaptive collaborative robot task execution. In *Robotics: Science & Systems: Workshop on Planning for Human-Robot Interaction: Shared Autonomy and Collaborative Robotics*, Cambridge, MA, June 2016.

[3] Brian P. Gerkey and Maja J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004. doi: 10.1177/0278364904045564. URL `https://doi.org/10.1177/0278364904045564`.

[4] Janelle Blankenburg, Santosh Balajee Banisetty, Seyed Pourya Hoseini Alinodehi, Luke Fraser, David Feil-Seifer, Monica N. Nicolescu, and Mircea Nicolescu. A distributed control architecture for collaborative multi-robot task allocation. In *17th IEEE-RAS International Conference on Humanoid Robotics, Humanoids 2017, Birmingham, United Kingdom, November 15-17, 2017*, pages 585–592, 2017. doi: 10.1109/HUMANOIDS.2017.8246931. URL `https://doi.org/10.1109/HUMANOIDS.2017.8246931`.

[5] 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.

[6] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 156–163. IEEE, 2015.

[7] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing-solving sparse reward tasks from scratch. *arXiv preprint arXiv:1802.10567*, 2018.

[8] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274, 2013.

[9] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[10] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[11] William H Montgomery and Sergey Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.

[12] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *arXiv preprint arXiv:1807.04742*, 2018.

[13] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J Johnson, and Sergey Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *arXiv preprint arXiv:1808.09105*, 2018.

[14] Maja J Matarić. Reinforcement learning in the multi-robot domain. In *Robot colonies*, pages 73–83. Springer, 1997.

[15] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[16] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[17] Chongjie Zhang and Victor Lesser. Coordinating multi-agent reinforcement learning with limited communication. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1101–1108. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[18] Daniel Garant, Bruno Castro da Silva, Victor Lesser, and Chongjie Zhang. Accelerating multi-agent reinforcement learning with dynamic co-learning. Technical report, Technical report, 2015.

[19] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[20] Yan Wu and Yiannis Demiris. Hierarchical learning approach for one-shot action imitation in humanoid robots. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, pages 453–458. IEEE, 2010.

[21] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[22] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25, 2009.

[23] David Freelan, Drew Wicke, Keith Sullivan, and Sean Luke. Towards rapid multi-robot learning from demonstration at the robocup competition. In *Robot Soccer World Cup*, pages 369–382. Springer, 2014.

[24] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*, 35(14):1760–1778, 2016.

[25] Felix Duvallet and Anthony Stentz. Imitation learning for task allocation. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3568–3573. IEEE, 2010.

[26] Bradford A Towle and Monica Nicolescu. An auction behavior-based robotic architecture for service robotics. *Intelligent Service Robotics*, 7(3):157–174, 2014.

[27] Nathan Michael, Michael M Zavlanos, Vijay Kumar, and George J Pappas. Distributed multi-robot task assignment and formation control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 128–133. IEEE, 2008.

[28] M Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

[29] Anahita Mohseni-Kabir, Sonia Chernova, and Charles Rich. Collaborative learning of hierarchical task networks from demonstration and instruction. In *RSS Workshop on Human-Robot Collaboration for Industrial Manufacturing, Berkeley, CA*, 2014.

[30] Bradley Hayes and Brian Scassellati. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5469–5476. IEEE, 2016.

[31] Monica N Nicolescu and Maja J Mataric. Experience-based representation construction: learning from human and robot teachers. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 740–745. IEEE, 2001.

[32] Monica N Nicolescu and Maja J Mataric. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 241–248. ACM, 2003.

[33] Katie Browne and Monica Nicolescu. Learning to generalize from demonstrations. *Cybernetics and Information Technologies*, 12(3):27–38, 2012.

[34] Hajime Asama, Akihiro Matsumoto, and Yoshiki Ishida. Design of an autonomous and distributed robot system: Actress. In *IROS*, volume 89, pages 283–290, 1989.

[35] Lynne E Parker. Alliance: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, volume 2, pages 776–783. IEEE, 1994.

[36] Brian P Gerkey and Maja J Matarić. Murdoch: Publish/subscribe task allocation for heterogeneous agents. In *Proceedings of the International Conference on Autonomous Agents*, pages 203–204. ACM, 2000.

[37] Ronald C. Arkin. *An Behavior-based Robotics*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262011654.

[38] Maja J Matarić. Behavior-based control: Main properties and implications. In *Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pages 46–54, 1992.

[39] Lynne E Parker. L-alliance: Task-oriented multi-robot learning in behavior-based systems. *Advanced Robotics*, 11(4):305–322, 1996.

[40] Barry Brian Werger and Maja J. Matarić. Broadcast of local eligibility: Behavior-based control for strongly cooperative robot teams. In *Proceedings*

of the Fourth International Conference on Autonomous Agents, AGENTS '00, pages 21–22, New York, NY, USA, 2000. ACM. ISBN 1-58113-230-1. doi: 10.1145/336595.336621. URL `http://doi.acm.org/10.1145/336595.336621`.

[41] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, July 2006. ISSN 0018-9219. doi: 10.1109/JPROC.2006.876939.

[42] Lynne E. Parker. Multiple mobile robot systems. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, pages 921–941. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[43] Zhongya Wang, Min Li, Jie Li, Jinge Cao, and Hanqing Wang. A task allocation algorithm based on market mechanism for multiple robot systems. In *Real-time Computing and Robotics (RCAR), IEEE International Conference on*, pages 150–155. IEEE, 2016.

[44] G. P. Das, T. M. McGinnity, and S. A. Coleman. Simultaneous allocations of multiple tightly-coupled multi-robot tasks to coalitions of heterogeneous robots. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pages 1198–1204, Dec 2014. doi: 10.1109/ROBIO.2014.7090496.

[45] Sahar Trigui, Anis Koubaa, Omar Cheikhrouhou, Habib Youssef, Hachemi Bennaceur, Mohamed-Foued Sriti, and Yasir Javed. A distributed market-based algorithm for the multi-robot assignment problem. *Procedia Computer Science*, 32:1108–1114, 2014.

[46] José Guerrero and Gabriel Oliver. Multi-robot coalition formation in real-time scenarios. *Robotics and Autonomous Systems*, 60(10):1295 – 1307, 2012. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2012.06.004. URL `http://www.sciencedirect.com/science/article/pii/S0921889012000942`.

[47] Y. Zhang and L. E. Parker. Iq-asymtre: Synthesizing coalition formation and execution for tightly-coupled multirobot tasks. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5595–5602, Oct 2010. doi: 10.1109/IROS.2010.5651186.

[48] Y. Zhang, L. E. Parker, and S. Kambhampati. Coalition coordination for tightly coupled multirobot tasks with sensor constraints. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1090–1097, May 2014. doi: 10.1109/ICRA.2014.6906990.

[49] L. Chaimowicz, T. Sugar, V. Kumar, and M. F. M. Campos. An architecture for tightly coupled multi-robot cooperation. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 3, pages 2992–2997 vol.3, 2001. doi: 10.1109/ROBOT.2001.933076.

[50] T. Huntsberger, P. Pirjanian, A. Trebi-Ollennu, H. Das Nayar, H. Aghazarian, A. J. Ganino, M. Garrett, S. S. Joshi, and P. S. Schenker. Campout: a control architecture for tightly coupled coordination of multirobot systems for planetary surface exploration. *IEEE Transactions on Systems, Man, and Cybernetics -*

Part A: Systems and Humans, 33(5):550–559, Sept 2003. ISSN 1083-4427. doi: 10.1109/TSMCA.2003.817398.

[51] Terry L. Huntsberger, Ashitey Trebi-Ollennu, Hrand Aghazarian, Paul S. Schenker, Paolo Pirjanian, and Hari Das Nayar. Distributed control of multi-robot systems engaged in tightly coupled tasks. *Autonomous Robots*, 17(1): 79–92, Jul 2004. ISSN 1573-7527. doi: 10.1023/B:AURO.0000032939.08597.62. URL `https://doi.org/10.1023/B:AURO.0000032939.08597.62`.

[52] Mary Koes, Illah Nourbakhsh, and Katia Sycara. Heterogeneous multirobot coordination with spatial and temporal constraints. In *Proceedings of the National Conference on Artificial Intelligence*, AAAI'05, pages 1292–1297. AAAI Press, 2005. ISBN 1-57735-236-x.

[53] V. A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, and P. F. Palamara. Petri net plans - A framework for collaboration and coordination in multi-robot systems. *Autonomous Agents and Multi-Agent Systems*, 23(3):344–383, 2011. doi: 10.1007/s10458-010-9146-1.

[54] Saeed Saeedvand, Hadi S Aghdasi, and Jacky Baltes. Robust multi-objective multi-humanoid robots task allocation based on novel hybrid metaheuristic algorithm. *Applied Intelligence*, 49(12):4097–4127, 2019.

[55] Vieri Giuliano Santucci, Emilio Cartoni, Bruno Castro da Silva, and Gianluca Baldassarre. Autonomous open-ended learning of interdependent tasks. *arXiv preprint arXiv:1905.02690*, 2019.

[56] Matthew Johnson, Jeffrey M Bradshaw, Paul J Feltovich, Catholijn M Jonker, M Birna Van Riemsdijk, and Maarten Sierhuis. Coactive design: Designing support for interdependence in joint activity. *Journal of Human-Robot Interaction*, 3(1):43–69, 2014.

[57] Stanislao Lauria, Guido Bugmann, Theocharis Kyriacou, and Ewan Klein. Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3):171 – 181, 2002. ISSN 0921-8890. doi: https://doi.org/10.1016/S0921-8890(02)00166-5. URL `http://www.sciencedirect.com/science/article/pii/S0921889002001665`. Advances in Robot Skill Learning.

[58] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. *Learning to Parse Natural Language Commands to a Robot Control System*, pages 403–415. Springer International Publishing, Heidelberg, 2013. ISBN 978-3-319-00065-7. doi: 10.1007/978-3-319-00065-7_28.

[59] Felix Duvalet. *Natural Language Direction Following for Robots in Unstructured Unknown Environments*. PhD thesis, Carnegie Mellon University, 2012.

[60] T. C. Lueth, T. Laengle, G. Herzog, E. Stopp, and U. Rembold. Kantra-human-machine interaction for intelligent robots using natural language. In *Proceedings of 1994 3rd IEEE International Workshop on Robot and Human Communication*, pages 106–111, Jul 1994. doi: 10.1109/ROMAN.1994.365947.

[61] M. Ralph and M. A. Moussa. Toward a natural language interface for transferring grasping skills to robots. *IEEE Transactions on Robotics*, 24(2):468–475, April 2008. ISSN 1552-3098. doi: 10.1109/TRO.2008.915445.

[62] Yonatan Bisk, Deniz Yuret, and Daniel Marcu. Natural language communication with robots. In *Proceedings North American Chapter of the Association for Computational Linguistics*, 2016.

[63] Guang-Hong Wang, Ping Jiang, and Zu-Ren Feng. Extraction of robot primitive control rules from natural language instructions. *International Journal of Automation and Computing*, 3(3):282–290, Jul 2006. ISSN 1751-8520. doi: 10.1007/s11633-006-0282-7. URL `https://doi.org/10.1007/s11633-006-0282-7`.

[64] Jesse Thomason, Shiqi Zhang, Raymond Mooney, and Peter Stone. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1923–1929. AAAI Press, 2015. ISBN 978-1-57735-738-4. URL `http://dl.acm.org/citation.cfm?id=2832415.2832516`.

[65] Ana Ramirez Chang. *User-Extensible Natural Language Spoken Interfaces for Environment and Device Control*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2008. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-162.html`.

[66] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Lawson L. S. Wong, and Stefanie Tellex. Accurately and efficiently interpreting human-robot instructions of varying granularities. *CoRR*, abs/1704.06616, 2017. URL `http://arxiv.org/abs/1704.06616`.

[67] Matthias Scheutz, Evan Krause, Brad Oosterveld, Tyler Frasca, and Robert Platt. Spoken instruction-based one-shot object and action learning in a cognitive robotic architecture. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 1378–1386, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems. URL `http://dl.acm.org/citation.cfm?id=3091282.3091315`.

[68] Dipendra K. Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016. doi: 10.1177/0278364915602060. URL `https://doi.org/10.1177/0278364915602060`.

[69] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashish G. Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1507–1514, San Francisco, CA, August 2011.

[70] Woodley Packard. Answer constraint engine, 2015. URL `http://sweaglesw.org/linguistics/ace/`.

[71] Cynthia Breazeal, Guy Hoffman, and Andrea Lockerd. Teaching and working with robots as a collaboration. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1030–1037. IEEE Computer Society, 2004.

[72] Paul E Rybski, Kevin Yoon, Jeremy Stolarz, and Manuela M Veloso. Interactive robot task training through dialog and demonstration. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 49–56. ACM, 2007.

[73] Anahita Mohseni-Kabir, Charles Rich, Sonia Chernova, Candace L Sidner, and Daniel Miller. Interactive hierarchical task learning from a single demonstration. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, pages 205–212. ACM, 2015.

[74] Alessandro Roncone, Olivier Mangin, and Brian Scassellati. Transparent role assignment and task allocation in human robot collaboration. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1014–1021. IEEE, 2017.

[75] Terrence Fong, Charles Thorpe, and Charles Baur. Robot, asker of questions. *Robotics and Autonomous Systems*, 42(3):235 – 243, 2003. ISSN

0921-8890. doi: https://doi.org/10.1016/S0921-8890(02)00378-0. URL http://www.sciencedirect.com/science/article/pii/S0921889002003780. Socially Interactive Robots.

[76] T. Fong, C. Thorpe, and C. Baur. Multi-robot remote driving with collaborative control. *IEEE Transactions on Industrial Electronics*, 50(4):699–704, Aug 2003. ISSN 0278-0046. doi: 10.1109/TIE.2003.814768.

[77] Ross A. Knepper, Stefanie Tellex, Adrian Li, Nicholas Roy, and Daniela Rus. Recovering from failure by asking for help. *Autonomous Robots*, 39(3):347–362, Oct 2015. ISSN 1573-7527. doi: 10.1007/s10514-015-9460-1. URL https://doi.org/10.1007/s10514-015-9460-1.

[78] Ronald C. Arkin, Masahiro Fujita, Tsuyoshi Takagi, and Rika Hasegawa. An ethological and emotional basis for human–robot interaction. *Robotics and Autonomous Systems*, 42(3-4):191–201, Mar 2003. doi: 10.1016/s0921-8890(02)00375-5. URL https://doi.org/10.1016/s0921-8890(02)00375-5.

[79] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva. *International Journal of Robotics Research*, 19(11):972–999, 2000. doi: 10.1177/02783640022067922. URL https://doi.org/10.1177/02783640022067922.

[80] Kerstin Severinson-Eklundh, Anders Green, and Helge Hüttenrauch. Social and collaborative aspects of interaction with a service robot. *Robotics and*

Autonomous Systems, 42(3-4):223–234, mar 2003. doi: 10.1016/s0921-8890(02)00377-9. URL https://doi.org/10.1016/s0921-8890(02)00377-9.

[81] M. Hans, B. Graf, and R.D. Schraft. Robotic home assistant care-o-bot: past-present-future. In *Proceedings of Robot and Human Interactive Communication*. IEEE. doi: 10.1109/roman.2002.1045652. URL https://doi.org/10.1109/roman.2002.1045652.

[82] R.O. Ambrose, H. Aldridge, R.S. Askew, R.R. Burridge, W. Bluethmann, M. Diftler, C. Lovchik, D. Magruder, and F. Rehnmark. Robonaut: NASA's space humanoid. *IEEE Intelligent Systems*, 15(4):57–63, jul 2000. doi: 10.1109/5254.867913. URL https://doi.org/10.1109/5254.867913.

[83] J. Khurshid and Hong Bing-rong. Military robots - a glimpse from today and tomorrow. In *ICARCV Control, Automation, Robotics and Vision Conference*, 2004. doi: 10.1109/icarcv.2004.1468925. URL https://doi.org/10.1109/icarcv.2004.1468925.

[84] J. Casper and R.R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 33(3):367–385, jun 2003. doi: 10.1109/tsmcb.2003.811794. URL https://doi.org/10.1109/tsmcb.2003.811794.

[85] Muhammad Awais and Dominik Henrich. Human-robot collaboration by intention recognition using probabilistic state machines. In *19th International*

Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010). IEEE, jun 2010. doi: 10.1109/raad.2010.5524605. URL https://doi.org/10.1109/raad.2010.5524605.

[86] Zhichao Wang, Bin Wang, Hong Liu, and Zhaodan Kong. Recurrent convolutional networks based intention recognition for human-robot collaboration tasks. In *International Conference on Systems, Man, and Cybernetics (SMC)*, 2017. doi: 10.1109/smc.2017.8122856. URL https://doi.org/10.1109/smc.2017.8122856.

[87] Przemyslaw A. Lasota and Julie A. Shah. Analyzing the effects of human-aware motion planning on close-proximity human–robot collaboration. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 57(1): 21–33, jan 2015. doi: 10.1177/0018720814565188. URL https://doi.org/10.1177/0018720814565188.

[88] Ren C. Luo and Charly Huang. Human-aware motion planning based on search and sampling approach. In *Workshop on Advanced Robotics and its Social Impacts (ARSO)*, 2016. doi: 10.1109/arso.2016.7736286. URL https://doi.org/10.1109/arso.2016.7736286.

[89] Stefan Escaida Navarro, Maximiliano Marufo, Yitao Ding, Stephan Puls, Dirk Goger, Bjorn Hein, and Heinz Worn. Methods for safe human-robot-interaction using capacitive tactile proximity sensors. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, November

2013. doi: 10.1109/iros.2013.6696495. URL https://doi.org/10.1109/iros.2013.6696495.

[90] David Feil-Seifer and Maja Matarić. People-aware navigation for goal-oriented behavior involving a human partner. In *Proceedings of the International Conference on Development and Learning (ICDL)*, Frankfurt am Main, Germany, August 2011. doi: 10.1109/DEVLRN.2011.6037331.

[91] K. P. Hawkins, Nam Vo, S. Bansal, and A. F. Bobick. Probabilistic human action prediction and wait-sensitive planning for responsive human-robot collaboration. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 499–506, Oct 2013. doi: 10.1109/HUMANOIDS.2013.7030020.

[92] K. P. Hawkins, S. Bansal, N. N. Vo, and A. F. Bobick. Anticipating human actions for collaboration in the presence of task and sensor uncertainty. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2215–2222, May 2014. doi: 10.1109/ICRA.2014.6907165.

[93] David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W Black, Mosur Ravishankar, and Alexander I Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I. IEEE, 2006. doi: 10.1109/ICASSP.2006.1659988.

[94] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 2009.

[95] D. Flickinger. English resource grammar, 2013. URL `http://www.delph-in.net/erg/`.

[96] Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A Sag. Minimal recursion semantics: An introduction. *Research on language and computation*, 3 (2-3):281–332, 2005.

[97] Blaise Gassend. sound_play ros package. URL `http://wiki.ros.org/sound_play`.

[98] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[99] Ioan A Sucan and Sachin Chitta. Moveit! *Online at http://moveit.ros.org*, 2013.

[100] Bashira Akter Anima, Janelle Blankenburg, Mariya Zagainova, Muhammed Tawfiq Chowdhury, David Feil-Seifer, Monica Nicolescu, Mircea Nicolescu, et al. Collaborative human-robot hierarchical task execution with an activation spreading architecture. In *International Conference on Social Robotics*, pages 301–310. Springer, 2019.

[101] S. Pourya Hoseini A., M. Nicolescu, and M. Nicolescu. Handling ambiguous object recognition situations in a robotic environment via dynamic information fusion. In *Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)*, June 2018. doi: 10.1109/COGSIMA.2018.8423982.

[102] Seyed (Pourya) Hoseini, Janelle Blankenburg, Mircea Nicolescu, Monica Nicolescu, and David Feil-Seifer. A dual-camera robotic vision system based on the concept of active perception. In *Proceedings of the International Symposium on Visual Computing (ISVC)*, Lake Tahoe, NV, October 2019.

[103] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[104] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *Pattern recognition (ICPR), 2010 20th international conference on*, pages 2756–2759. IEEE, 2010.

[105] M. Gualtieri, A. ten Pas, K. Saenko, and R. Platt. High precision grasp pose detection in dense clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 598–605, Oct 2016. doi: 10.1109/IROS.2016.7759114.

[106] Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3

(2):281–332, Jul 2005. ISSN 1572-8706. doi: 10.1007/s11168-006-6327-9. URL https://doi.org/10.1007/s11168-006-6327-9.

[107] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.