LAAS-CNRS

Doctoral Thesis

Decision Making in Human-Robot Interaction

Author:
Michelangelo Fiore

Supervisor:
Dr. Rachid Alami

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy in Robotics

in the

August 2016

LAAS-CNRS

Abstract

Institut National des Sciences Appliquees de Toulouse

Doctor of Philosophy in Robotics

Decision Making in Human-Robot Interaction

by Michelangelo Fiore

There has been an increasing interest, in the last years, in robots that are able to cooperate with humans not only as simple tools, but as full agents, able to execute collaborative activities in a natural and efficient way. In this work, we have developed an architecture for Human-Robot Interaction able to execute joint activities with humans. We have applied this architecture to three different problems, that we called the robot observer, the robot coworker, and the robot teacher. After quickly giving an overview on the main aspects of human-robot cooperation and on the architecture of our system, we detail these problems.

In the observer problem the robot monitors the environment, analyzing perceptual data through geometrical reasoning to produce symbolic information. Using a rule-based framework our system is able to model and maintain humans' mental beliefs, allowing the robot to understand what humans currently know about the state of the world. We also show how the system is able to infer humans' actions and intentions by linking physical observations, obtained by reasoning on humans' motions and their relationships with the environment, with planning and humans' mental beliefs, through a framework based on Markov Decision Processes and Bayesian Networks. We show, in a user study, that this model approaches the capacity of humans to infer intentions. We also discuss on the possible reactions that the robot can execute after inferring a human's intention. We identify two possible proactive behaviors: correcting the human's belief, by giving information to help him to correctly accomplish his goal, and physically helping him to accomplish the goal.

In the coworker problem the robot has to execute a cooperative task with a human. In this part we introduce the Human-Aware Task Planner, used in different experiments, and detail our plan management component. The robot is able to cooperate with humans in three different modalities: robot leader, human leader, and equal partners. We introduce the problem of task monitoring, where the robot observes human activities to understand if they are still following the shared plan. After that, we describe how our robot is able to execute actions in a safe and robust way, taking humans into account. We present a framework used to achieve joint actions, by continuously estimating the robot's partner activities and reacting accordingly. This framework uses hierarchical Mixed Observability Markov Decision Processes, which allow us to estimate variables, such as the human's commitment to the task, and to react accordingly, splitting the decision process in different levels. We present an example of Collaborative Planner, for the handover problem, and then a set of laboratory experiments for a robot coworker scenario. Additionally, we introduce a novel multi-agent probabilistic planner, based on Markov Decision Processes, and discuss how we could use it to enhance our plan management component.

In the robot teacher problem we explain how we can adapt the plan explanation and monitoring of the system to the knowledge of users on the task to perform. Using this idea, the robot will explain in less details tasks that the user has already performed several times, going more in-depth on new tasks. We show, in a user study, that this adaptive behavior is perceived by users better than a system without this capacity.

Finally, we present a case study for a human-aware robot guide. This robot is able to guide users with adaptive and proactive behaviors, changing the speed to adapt to their needs, proposing a new pace to better suit the task's objectives, and directly engaging users to propose help. This system was integrated with other components to deploy a robot in the Schiphol Airport of Amsterdam, to guide

groups of passengers to their flight gates. We performed user studies both in a laboratory and in the airport, demonstrating the robot's capacities and showing that it is appreciated by users.

Version française

Acknowledgements

TODO...

Contents

List of Figures

List of Tables

Abbreviations

MDP	**M**arkov **D**ecision **P**rocess
POMDP	**P**artially **O**bservable **M**arkov **D**ecision **P**rocess
MOMDP	**M**ixed **O**bservability **M**arkov **D**ecision **P**rocess
MAMDP	**M**ulti **A**gent **M**arkov **D**ecision **P**rocess
BN	**B**ayesan **N**etwork
DBN	**D**ynamic **B**ayesan **N**etwork
IG	**I**ntention **G**raph
HATP	**H**uman-**A**ware **T**ask **P**lanner
TPNU	**T**emporal **P**lan **N**etwork with **U**ncertainty
I-POMDP	**I**nteractive **P**artially **O**bservable **M**arkov **D**ecision **P**rocess

Part I

Overview

In this part we give a general overview on the thesis. Chapter 1 introduces the problem of Human-Robot Interaction, motivating our work and how we have contributed to this important subject. Chapter 2 presents some architectures for Human-Robot Interaction and gives a bird's eye view on the system that we have developed.

Chapter 1

Introduction

In the current days robots are starting to be introduced in our lives more and more, and we can expect that, in the next decade, they will complete the transition from mechanic tools, used mostly in industries, to true partners and companions. There is an increasing interest in studying how robots should behave in environments inhabited by humans, and, in some works, robots have been deployed in crowded and dynamic environments, like airports and museums.

Human-Robot cooperation poses a multitude of problems. Imagine a mobile robot working in a warehouse, carrying and sorting crates in different locations. Already, we are presented with quite a complex problem, where the robot needs to have a good representation of the world (i.e. position of the crates, obstacles, layout of the warehouse), to create plans to reach the goal (which crates to move, where to bring them, which paths to follow), and to have sufficient motion and manipulation skills to achieve them. If humans are present in the environment, they should be represented and considered by the robot in its plans and actions. Modeling humans as simple moving obstacles might not be enough if we consider issues of trust, legibility, and acceptability. The robot should respect a number of social rules in the presence of humans, like maintaining a socially acceptable distance, whenever possible, from them, and approaching them from a visible position.

The problem becomes even more complex when robots and humans need to cooperate to solve a goal, for example by sorting together the crates, or even by sharing the load of heavy objects. To understand how to approach this problem we can observe how humans cooperate with themselves. Research in psychology and philosophy (Pacherie, 2012) characterizes the execution of cooperative actions as 'joint actions'. Sebanz et al. (2006) have proposed that the execution of a joint action depends on three different abilities: sharing representations, predicting actions, and integrating predicted effects of own and other's actions. These abilities can be achieved by the combination of different mechanisms:

- Joint Attention. The ability to direct a partner's attention, in order to create a shared representation of objects and events. Humans use a large number of social cues, like gaze direction or pointing

gestures, to indicate what is currently under observation. This mechanism helps filling important gaps in the knowledge of a partner, and points to the importance of understanding what others know and perceive.

- Action Observation. Observing other partners' actions is crucial in understanding what are their goals. Studies have shown that observing a person performing an action produces a motor resonance, which increases with the observer's level of expertise in the action. Understanding what others are doing allows to predict the outcomes of their activities, and even their next movements.

- Task Sharing. Humans are able to predict, in some circumstances, what others will do even without direct observation. A notable example is a well trained sport team, which is able to act like a single entity, coordinating seamlessly. This ability suggests that humans possess a shared representation of tasks, which include actions that should be performed by each partner of the team.

- Action Coordination. Predicting actions is not enough. Humans also need to choose a complementary action and adjust its parameters to partners, like the exact moment and place where it should be performed.

It seems that robots need to have an equivalent of these mechanism, in order to cooperate in a natural and acceptable way with humans. Is this enough to create robot partners? Unfortunately, we just scratched the surface of the problem. While these areas are already very complex, and not completely understood, humans possess other skills that should be translated to robots. For example, when a robot's behavior shows a degree of intelligence, humans usually try to have a conversation with it, which can lead to frustation or disbelief in the actual capacities of the robot. Issues such as dialogue, representation, and refinement of knowledge are very complex and will not be a direct focus of this work.

The goal of this work is, instead, to provide a framework to allow a robot to work in social environments and execute joint actions with humans in a natural way. We have built our system using psychology as an inspiration, without trying to replicate accurately human mechanisms, an area of work studied in cognitive systems.

Contributions

The main contributions of this work are the following:

- Building a supervision system for Human-Robot Interaction (HRI). The aim of this system is being able to execute cooperative human-robot actions in a natural and efficient way. It has been built by integrating new components, developed in this work, with existing ones.

- Developing a novel algorithm to infer human intentions and actions. This algorithm, based on Markov Decision Processes (MDP), and Bayesian Networks (BN), allows the robot to better understand humans by reasoning not only on their actions, but also on their knowledge and awareness of the environment. The algorithm is efficient and scalable, and has been tested in a user study to prove that it is able to approach the capacity of humans to infer intentions.

- Developing a framework to execute joint actions between the robot and a human. This framework is based on Mixed Observability Markov Decision Processes (MOMDP) and has been applied to two different examples: the handover and the robot guide.

- Developing a multi-agent probabilistic planning algorithm, based on MDPs. This planner is able, starting from a list of single-agent MDP models, to build a multi-agent model, taking into account issues such as cooperation and conflicts in the agents' actions. The planner was developed to work on tightly coupled problems, where agents interactions are frequent. The complexity associated to MDPs is partially mitigated by introducing features such as parameters, abstract states, and hierarchical models.

- Evaluating the different components in user studies. We have performed two user studies to evaluate the capacity of the system to infer human intentions, and how its adaptive plan explanation and management skills are perceived by humans. In addition, parts of the system, joined with components of other partners in an European project, where evaluated with real users in a robot guide scenario, ran at the airport of Schiphol in Amsterdam.

1.1 Organization of the Thesis

During the development of our work we identified three different aspects of our robot: the observer, the coworker, and the teacher. This thesis is organized in different parts, reflecting these aspects.

Part II introduces the problem of the robot observer, where a robot is constantly monitoring the activities of humans in the environment, trying to infer their goals and to help them achieve it. Chapter 3 shows how the robot is able to use geometrical reasoning to understand what humans know about the current environment. Chapter 4 introduces our intention and action recognition algorithm, showing how the robot can use it to infer what actions humans execute, and how they connect to his possible intentions. Chapter 5 shows our experiments in this problem, where we compared our algorithm with humans prediction in a user study.

Part III introduces the robot coworker problem, where a robot has to complete a cooperative scenario with a human helper. Chapter 6 analyzes how our system is able to produce shared plans, which include both the robot and humans, and how it is able to manage them, synchronizing with humans. Chapter 7

explains how the robot is able to execute tasks and, in particular, introduces the Collaborative Planners, a framework that we use to represent and execute joint actions. Chapter 8 introduces a human-robot cooperative scenario, showing how our system is able to handle several possible situations. Chapter 9 introduces the Multi-Agent Markov Decision Process, a probabilistic model that we have recently developed to extend the capacity of the robot coworker.

Part IV discusses about how a robot can teach a task to users, adapting its explanations to their expertise in the domain. Chapter 10 explains how we are able to model and update a human's expertise in a task and how the robot can adapt its explanations to this model. We also evaluate different strategies in collaborative planning, favoring teaching or efficiency. Chapter 11 shows a user study that evaluates our algorithm to teach a cooperative task.

Part V presents a complex case study, where we cooperated in the development of a human-aware robot guide, which was then deployed in the Schiphol Airport of Amsterdam. Chapter 12 shows how we designed this system, adapting the capacities explained in the previous parts. Chapter 13 shows our experiments, performed both in a laboratory environment and in the airport, with real passengers.

Finally, Part VI concludes this thesis. Chapter 14 reviews our achievements and discusses possibile future works. We also present in this part appendix A, where we introduce in a formal way the models used in the our work.

1.2 Published Works

- Milliez, Grégoire, et al. "Simulating human-robot interactions for dialogue strategy learning." Simulation, Modeling, and Programming for Autonomous Robots. Springer International Publishing, 2014. 62-73.
- Triebel, Rudolph, et al. "SPENCER: A socially aware service robot for passenger guidance and help in busy airports.", 2015.
- Fiore, Michelangelo, et al. "An Adaptive and Proactive Human-Aware Robot Guide." Social Robotics. Springer International Publishing, 2015. 194-203.
- Fiore Michelangelo, et al. "On planning and task achievement modalities for human-robot collaboration." Experimental Robotics. Springer International Publishing, 2016.
- Milliez, Grégoire, et al. "Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring." The Eleventh ACM/IEEE International Conference on Human Robot Interation. IEEE Press, 2016.

- Devin, Sandra et al. “Some essential skills and their combination in an architecture for a cognitive and interactive” robot arXiv preprint arXiv:1603.00583, 2016

- Caccavale, Riccardo, et al. “Attentional Supervision of Human-Robot Collaborative Plans.” The IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), 2016.

Chapter 2

System Overview

In this chapter we provide an overview on our system. Section 2.1 provides an overview of some relevant architectures, which consider humans at multiple levels. Section 2.2 presents the main characteristics of our system, while section 2.3 shows a high-level overview of its architecture, introducing the main layers.

2.1 Introduction

While different authors have studied HRI, most works focus on specific aspects of the problem, like how robots are perceived by children (Kozima et al., 2007), or how they can navigate in human environments (Sisbot et al., 2007b). Few robotic architectures, at the moment, take humans into account at all levels, from planning to execution. We will briefly review some examples, and will detail a part of their specific aspects in the future chapters:

- Trafton et al. (2013) present ACT-R/E, a cognitive architecture, based on the ACT-R architecture, used for human robot interaction tasks. The architecture aims at simulating how humans think, perceive and act in the world. ACT-R/E has being tested in different scenarios, such as theory of mind and hide and seek, to show its capacity of modeling human behaviors and thoughts. Being a cognitive architecture, ACT-R/E is focused on modeling human cognition, and not on efficiently solving tasks. For this reason, the system could encounter issues in handling complex human-robot cooperative tasks.

- In Fong et al. (2006) the authors present HRI/OS, an agent-based system that allows humans and robots to work in teams. In this system, a central unit, called Task Manager, coordinates the achievement of a goal, by decomposing the problem in smaller tasks and assigning these to human and robot agents. Agents are assumed to work in parallel and independently on their own tasks. When an agent encounters a difficulty, it can send a request for help, which other agents can answer to provide the needed assistency. HRI/OS implements a spatial reasoning component, which allows

communication between agents to be more natural and effective. This system is conceived for team based work on loosely coupled problems. It does not provide a sophisticated cognitive model, or the skills required to execute in a natural way tightly coupled problems, where the agents need to solve a task by frequently interacting and performing actions together.

- The HAMMER architecture (Demiris* and Johnson†, 2003) is a system, inspired by research in biology, that can be handle HRI scenarios. The main idea of this architecture is using the robot schemas for both prediction and execution of motions. Johnson and Demiris (2005) have extended the system to include visual perspective taking, increasing the cognitive capacities of the robot. This architecture has been tested in experimental settings involving imitation of another robot's movements and with a human demonstrator. The system is mainly focused on prediction tasks and does not present a way to model the mental state of other humans, which is very important to correctly understand others' actions and goals.

- In Breazeal et al. (2009) the authors present a cognitive architecture for HRI. The system uses a simulation approach, similar to HAMMER, where the robot schemes can be used for both prediction and for execution. This architecture is able to infer human goals and actions, as well as modeling human beliefs on the state of the world. While supporting human-robot cooperation, the architecture does not directly present models for aspects such as sharing plans, monitoring their execution, and executing joint actions.

- In Clodic et al. (2009) the authors present the LAAS architecture for human robot interaction, tested in domestic environments to perform tasks such as serving a drink to a person. The decisional layer of this architecture is composed by three modules: the Task Agenda, which manages the high-level goals and tasks of the robot; SHARY, a supervision system able to execute tasks, monitor the execution of actions performed by humans, and perform simple exchanges of information between the robot and humans; and the Human-Aware Task Planner (HATP), able to build plans that include humans and robots. Our work is an evolution of this architecture which includes new aspects, like spatial reasoning and modeling of joint actions.

2.2 Characteristics

In this work, our goal was developing a system allowing a robot to interact with a human in a way that is pleasing for the human, but also efficient and safe. To achieve this objective, we have developed an architecture composed by different layers, and, at each layer, we have asked ourselves the question: how can we consider the human at this level? While working on this problem we realized that our robot can show three different faces, which we called the *robot observer*, the *robot coworker*, and the *robot teacher*.

In the robot observer case, the robot is costantly monitoring the environment, reasoning in order to understand the current situation. Including humans in this kind of scenario means reasoning on their activities and on their effects on the environment. In particular, we are interested in having an estimation of a human's *intention*. In literature about psychology (Bruner, 1981) and philosophy (Bratman, 1984), an intention is the wish and will to achieve a goal. Intentions emerge from contextual causes (motivations) and are present until the goal is achieved or abandoned, pushing agents to undertake actions leading to that goal. While a human can have many goals, short or long-termed, we are interested in inferring what is the current goal that he is trying to achieve through his activites, and see if the robot can provide help. The robot observer can help an elderly person to remember to take its medicine, or indicate to forgetful person the locations of his keys, if he is looking for them.

In the robot coworker case, the robot has to perform a task together with a human. In this situation, we stress the point that the goal of this robot is not to complete a task, but to complete it *together* with a human. It is important that the robot is able to produce and manage collaborative plans, which include not only its actions, but also the ones of the human. We will call *shared plan* a collaborative plan, which includes the robot and humans, that is known (and expected to be followed) by all the participants. This is an area of artificial intelligence called multi-agent planning. In our system an *agent* is a robot or a human which is able to act and influence the environment.

While we have developed our system as a generic architecture, able to work in different tasks and problems, we are particularly interested in tightly coupled tasks, where the human and the robot need to interact, often by performing *joint actions*. We consider as a *joint action* a cooperative task, performed by the human and the robot together. In this situations creating a classical plan could not be enough, since the robot needs to costantly adapt to the human's actions. For example, in the case of handover, where the robot is giving an object to the human, the robot can not simply extend its arm and release the object at some point, but needs to find a good position where the exchange can be performed, eventually changing this position depending on the human's actions; understand when the object can be safely released; and also if the human is actually going to take the object or is, momentarely or for a longer period of time, doing something else.

Finally, in the robot teacher case, the robot is trying to explain a complex task to a human, in an efficient and interesting way. In order to do so, it is important the robot understands what the human already knows about the task. Problems are often composed by repetitive or similar operations, and explaining every single action can be boring for a human, particularly if he already has some expertise in the domain. For example, if the robot is teaching a human to cook a dish, which requires julienning three eggplants, it is not necessary that it explains in details how to cut each eggplant. The robot can teach the human how to julienne the first eggplant, and then just ask him to repeat the same operation on the

other two eggplants. Of course, the robot should consider that the human might not have understood well its explanation, and my need further advice.

It is important to understand that these three aspects of the robot are not actually separated. The robot coworker needs the reasoning skills of the observer, in order to produce efficient plans for the current situation and to cooperate with the human. Similarly, the robot observer needs the coworker's abilities if it wants to help the human to achieve his goal. The robot teacher needs the coworker's (and in turn, the observer's) capacity to create shared plans. While we can imagine simpler versions of these scenarios, where the three aspects are actually separated (e.g. a robot teacher with predefined strategies, that does not need to plan), we believe that their integration provides a richer framework for HRI.

2.3 Architecture

To respect these characteristics, we designed a supervision system composed by the following layers, as shown in figure 2.1:

- Situation Assessment. This layer uses geometrical reasoning, starting from perceptual data, to produce symbolic information, like the reachability of objects, which actions have been performed by humans, and the spatial relationships (distance, orientation, and their variations) between humans and the robot. These symbolic information are stored in a Database, which is able to represent the knowledge of other humans, as viewed by the robot. Using this feature, the robot can represent, for example, the fact that a human does not know, or that he has wrong information, about the location of an object.

- Goal Management. This layer manages the different goals of the robot. Goals can be directly received from exernal inputs, like a human or a terminal, or generated by the robot starting from information present in the Database. For example, after deducing that the human is looking for his glasses in the Situation Assessment layer, the Goal Management layer can create a goal to fetch them. In this work we developed very simple rule mechanisms to manage goals. Since this layer is not a crucial point in our system we will not present it in the thesis.

- Plan Management. This layer is in charge of managing plans to achieve the current goal. The system receives as input a multi-agent plan, which includes the actions of the robot and of other humans. When managing a plan, the robot will execute its own actions and synchronize with the human, by monitoring its activities and comparing them to what is expected from the plan. Plans can be managed in three different modalities: robot leader, human leader, and equal partners. This layer also supports human-aware explanation of plans, by adapting the explanation process to the expertise of users in the tasks.

- Task Execution. This layer handles the execution of the robot's actions, including joint actions shared with other agents. Human safety and robustness are achieved by stopping and resuming operations when unexpected or dangerous situations arise, like a human moving into the operative area of the robot.

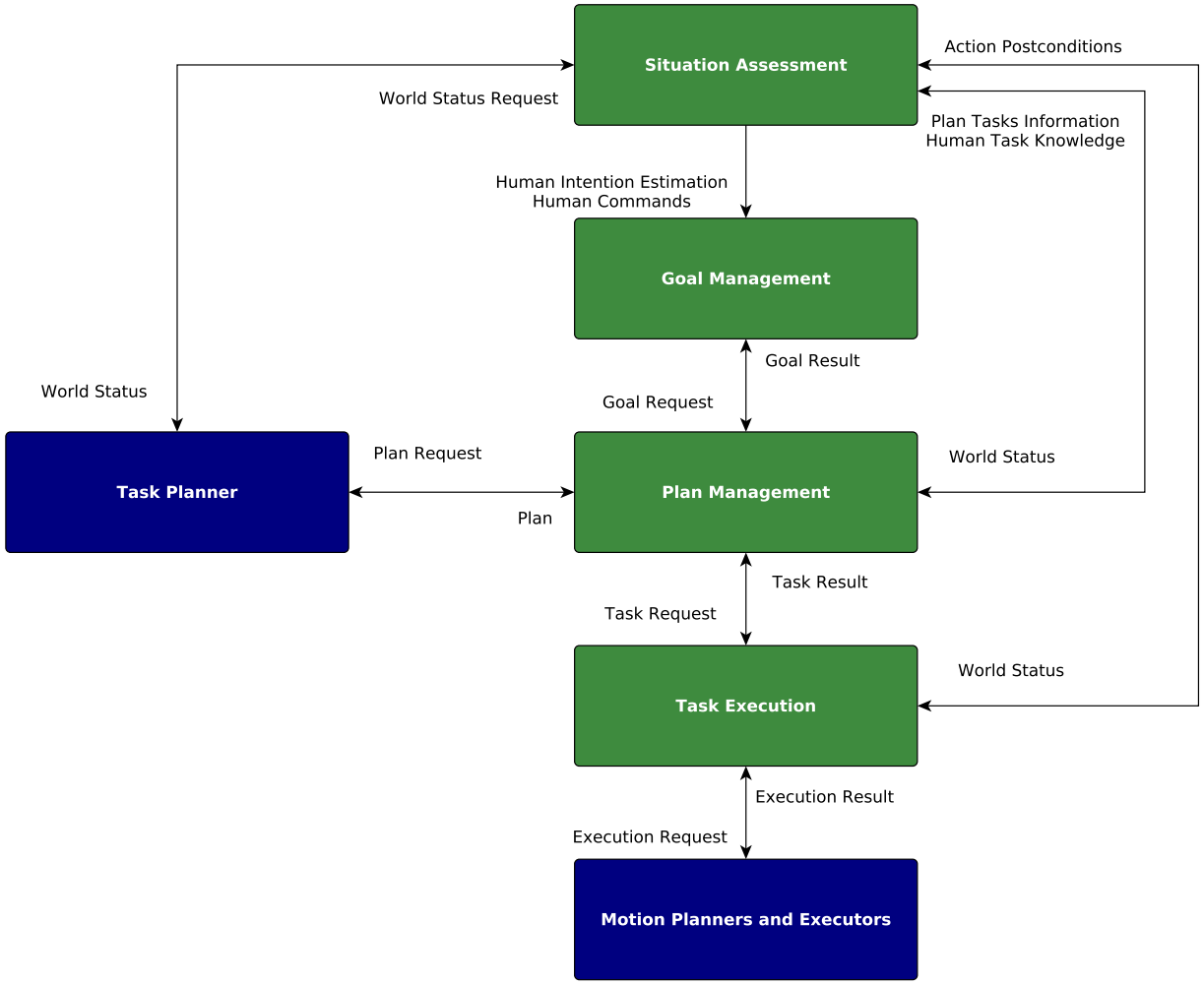


FIGURE 2.1: This picture shows the different layers of the system. Green rectangles represent parts of the supervision system, while blue rectangles external modules.

The system can also easily interact with different modules, which can be changed depending on the current needs:

- Task Planner. The Plan Management layer provides an interface for different planners. New planners can be introduced by creating a bridge that respects the interface provided, producing plans in the format requested by the Plan Management layer and reading information about the world state from the Situation Assessment layer.

- Motion Planners and Executors. The Task Execution layer is interfaced with a set of motion planners and an executor to accomplish the robot's movements and actions. New modules can be introduced by respecting the interface used in the layer.

To achieve these goals we use the well-known ROS framework[1], which naturally supports different robots and modules. The system has been implemented and tested in simulation, using the GAZEBO simulator[2], and on two different robots, the PR2 by Willow Garage[3], and the SPENCER robot[4], developed in an European research project.

The system has been used in two european projects: SPENCER, whose tasks was building a human-aware robot to guide passengers in an airport scenario; and SAPHARI[5], where it was used to execute a cooperative scenario, where a robot and a human had to clean a table together.

[1] http://www.ros.org/
[2] http://gazebosim.org/
[3] https://www.willowgarage.com/pages/pr2/overview
[4] http://www.spencer.eu/
[5] http://www.saphari.eu/

Part II

The Robot Observer

In any application that is not entirely composed by repetitive, precomputed actions, robots need reasoning skills, which severely depend on the quality of the representation of the current environment. This representation can be more or less complex, depending on the application.

Imagine, for example, a robot whose task is cleaning the floor of a room. In the simplest case, this robot would only be provided with an elementary set of sensors. Without the capacity to understand which areas actually need cleaning, this robot could only move through the room, randomly or with some strategy, achieving the task in a longer amount of time than what would actually be needed.

Now, imagine a household robot that needs to actively help a family that lives in an apartment, by fetching objects, providing information, and helping to accomplish various tasks. Let us imagine that one of the members of the family, Greg, is moving in the living room, searching around, while exclaming 'Where are my glasses?'. In the ideal situation, our robot would try to help Greg, by giving him information such as 'They are on the table to your right', or even by fetching them for him.

Clearly, in this scenario, the robot needs deeper reasoning skills. It needs to understand that the user is looking for its glasses, to link them to their actual physical location, to compute the spatial relationship between the table and the glasses, and to provide information in a natural way. In fact, if the robot would tell Greg that the glasses are in the position $(3.2, 5.0, 1.3)$, Greg would likely be very perplexed. A more natural way would be to inform Greg that his glasses are on a table, whose location is pointed taking into account Greg's position.

In this case, having sophisticated sensors is, of course, important but not sufficient. The robot needs also to *reason* on the sensor's data in order to produce meaningful information. For example: laser points and camera images need to be integrated to recognize objects and humans; spatial relationships (e.g. the glasses are on the table) have to be properly modeled; actions performed by humans, and their effects on the environment, need to be recognized; and so on.

The process of reasoning on data to produce symbolic information is called *situation assessment*. Endsley explained in Endsley (1995) that this process is deeply linked to the quality of decions of the robot.

While in many applications robots can benefit from a situation assessment component, being able to perform complex reasoning on data is particularly important in HRI. If the robot is able to take better decisions (i.e. efficient, safe, socially acceptable, natural) than it will be perceived in a more positive manner by humans.

Situation assessment is the fundamental skill for the *robot observer*. In this part, we will show the situation assessment mechanisms of our system. Chapter 3 shows how we can use geometrical reasoning in order to understand what the robot and other humans know and perceive about the current environment. Chapter 4 shows how our robot is able to infer the current intention of a human and the

actions it performs. Finally, chapter 5 details a user study that we developed to validate our intention recognition algorithm.

Chapter 3

Belief Management

In this chapter we show how our system is able to manage agents beliefs. Section 3.1 provides an introduction to the subject, with a discussion on some relevant works. Section 3.2 shows an overview of our approach, discussing its key aspects and modules. Sections 3.3 shows the simplified mechanism that we use to detect objects and humans. In section 3.4 we explain the geometrical reasoning capacities of our system, introducing the main symbolic facts that are computed. Finally, in section 3.5 we show how we build human belief models starting from these facts.

3.1 Introduction

In HRI, it is important to represent humans not as simple obstacles, but as acting entities, with different beliefs on the state of the world and with the capacity to affect the environment.

A simple example to prove this point is related to proxemics. Normally, when we approach an object, like a table, in an empty environment, we just choose the quickest path. If, intead, the environment is populated by other people, we will follow a set of social rules, like keeping a certain distance from them. If the robot does not follow these rules, it might freighten people, for example by approaching quickly from behind the person, or be considered as 'rude'.

Theory of Mind (Premack and Woodruff, 1978) is a skill used to reason about humans' beliefs and thoughts, and how they affect actions. An ability linked to this concept is perspective taking, which is widely studied in developmental literature. Flavell in Flavell (1977) describes two levels of perspective taking: perceptual perspective taking, the capacity to understand that other people see the world differently (Tversky et al., 1999); and conceptual perspective taking, the capacity to attribute thoughts and feelings to other people (Baron-Cohen S, 1985).

Through perceptual perspective taking we are able to compute information such as 'I can see the glasses, but you can not', 'you can reach the bottle, but I can not'. Conceptual perspective taking is

equally important, and allows us, for example, to understand that a friend, when choosing the flavors for its ice cream, would choose its favourites, which might differs from ours.

An important study linked to conceptual perspective taking is the *divergent belief task*. Formulated in Wimmer and Perner (1983), this kind of task requires the ability to recognize that others can have beliefs about the world that differ from the observable reality. A typical test used to study divergent belief is the Sally and Anne test (Figure 3.1). In this test, Sally and Anne are two puppets. A short scene is shown to participants, where Sally takes a ball and hides it in her basket, before leaving the room. While she is away, Anne takes the ball from Sally's basket and puts it in her box. After Sally returns, participants are asked a question: "Where will Sally look for her object?". Young children, or those affected by autism, might answer (wrongly) that Sally will look for her object in Anne's box, not taking into account that she does not know that it was moved.

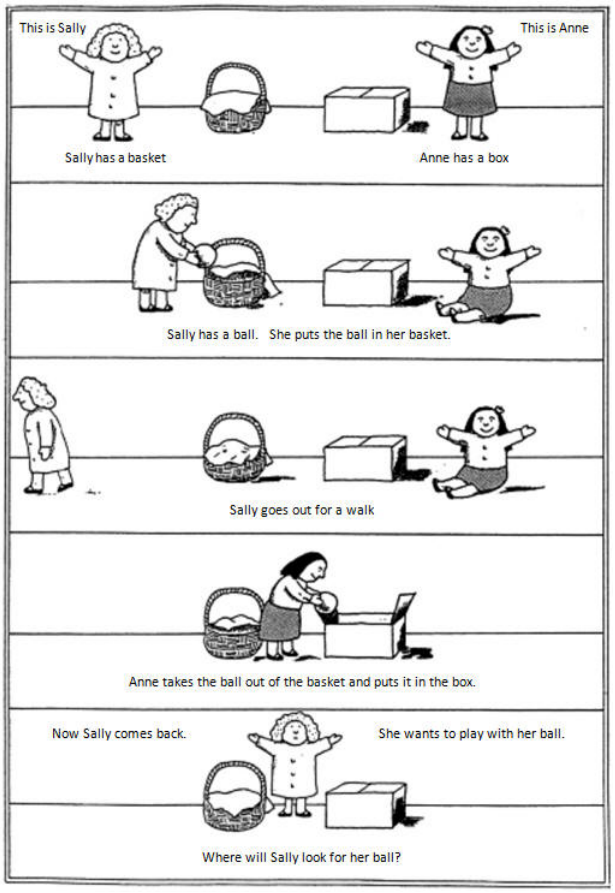


FIGURE 3.1: The Sally and Anne test. Original artwork by Axel Scheffler.

Studies on individuals that do not possess the required mechanisms to perform perspective taking (Frick et al., 2014) have shown that they encounter great difficulties in social relationships, confirming the importance of this ability.

These studies make us think that perspective taking could be very important in robotics. For example, let us imagine a scenario where Greg and a robot are performing some household repairs, which require the use of different tools. The robot might create a plan to achieve this goal where Greg needs to use

a screwdriver, which is actually behind a box, hidden from his sight. Without perspective taking, the robot will procede in its task while Greg is desperately looking for the screwdriver. By using perspective taking, instead, the robot would infer that Greg does not know where the screwdriver is, and can not see it. In this situation, the robot might inform Greg of the location of the screwdriver, give it to him, or create a plan where Greg does not need to use it, thus being a more likable helper.

Previous works in robotics have actually demonstrated that enhancing the robot's perspective taking abilities improves its reasoning capabilities, leading to more appropriate and efficient task planning and interaction strategies (Breazeal et al., 2006, Ros et al., 2010, Trafton et al., 2005). Let us review some implementations of perspective taking.

The HAMMER system, previously presented in chapter 1, has been extended in Johnson and Demiris (2005) to introduce perceptual perspective taking, using the concept of forward and inverse visual models. Forward vision models analyze sensory data to produce information, like the geometrical coordinates of objects. Inverse visual models, instead, receive as input desired properties and states (for example, the presence of objects of a certain shape and color in a particular location), and try to produce an appropriate visual image that respects these inputs.

Perspective taking is performed by introducing two forward vision models. The first one produces the location and relationships between objects and end effectors (like grippers), while the second computes the gaze direction of other agents. Using the results of these models as input of an inverse visual model, the system can reconstruct the scene as seen by another agent. The representation of knowledge by agents is not explicitly handled in the system.

The architecture of Breazeal et al. (2009), also presented in chapter 1, is another cognitive system which uses simulation mechanisms, but the work includes both perceptual and conceptual perceptive taking. In this system the robot's schemas are used to build its mental model and execute its tasks, but also to build humans mental models and infer their actions and goals.

The robot is able to build a belief on the world state by analyzing perceptual data, producing symbolic information, such as the location or color of an object. This belief is constantly maintained, by adding new information and erasing those that are no longer valid.

To build a belief model of a human, the robot manipulates perceptual data, by filtering objects that can not be seen by the human (because they are occluded, for example), and transforming the data to simulate a first-person experience on the human. By using the same mechanisms to manage the belief for itself and for humans, the robot can infer divergent belief situations. This method has been tested on variants of the Sally and Anne test with good results.

The belief management algorithm of this systems does not take into account information that can not be directly perceived, but must be inferred. For example, in a domestic scenario, a human might not

be aware that a mug containing liquid is hot and should not be touched. These aspects could be inferred by taking into account the results of actions, like pouring hot liquid in the mug.

Another system able to model agent beliefs is Scheutz (2013). This work is oriented toward problems where a distributed team of agents needs to communicate over a remote connection to solve a task. Each agent is assumed to be able to create his own mental model using his perception capacities, even though the precise rules for these mechanisms are not explained. When an agent receives information, he updates his mental belief using a set of rules, introducing the new data and checking for incongruencies with his previous information. Using the same mechanism, each agent forms a belief model of other agents that have received the same information.

We created a rule-based approach for belief management, based on geometric reasoning and on inference, which we will introduce in the following sections.

3.2 Belief Management Overview

3.2.1 Process overview

Managing beliefs means being able to build and maintain a model of the belief of each agent. Our system is able to accomplish this task with different steps.

Of course, to reason on the current situation the system needs to be able to detect humans and objects in the environment. In some case, to be generic, we will call an agent or an object an *entity*. Section 3.3 will show how we detect and track entities.

We can assign *attributes* to entities, parts of entities (e.g. the arm of a human), and areas to represent different information, like properties, the state of entities or relationships between entities. Examples of attributes are: an agent can be in a specific area; a box can be opened and can contain objects; a bottle can contain liquids; a mug can be hot; a room can be a silent area, where the robot should avoid making any noise. We divide attributes in two classes, which influence the capacity of agents to perceive them:

- Fully observable. These attributes can be observed by any present agent looking at the linked entity or area (e.g. the box is open, Greg is in the kitchen).
- Partially observable. These attributes can be observed by present agents only in specific situations, represented as rules linked to the object and the attributes, e.g. an agent can see that a box contains items only when it is open, an agent can detect that the mug is hot only when he touches it).

The locations of entities are integrated with other data produced by sensors through geometrical reasoning in order to produce symbolic information, such as spatial relationships between objects and

humans. We represent these information as facts, where a fact is a tuple *subject predicate value*. An example of fact is $CUP\ isOn\ TABLE$, which defines the location of an object. The instance of an attribute is represented as a fact, and so, each fact will correspond to exactly one attribute. More details about this aspect will be shown in section 3.4.

We call *world state* the collection of all facts describing the current situation. A *belief model* is a collection of facts representing the knowledge of an agent on the world state. Each agent can have a different belief model, since the environment can change without an agent being able to perceive it. To represent the lack of knowledge of an agent, the value of a property can be *unknown*. For example, the robot could take a tool from the table, while Greg is in another room, and place it in a box. Greg has not seen this event, and so his belief model will contain the (wrong) information that the tool is still on the table. Section 3.5 will show in, details, our approach to update the belief model of an agent.

In general, beliefs models are updated when the system infers that an *action* has been executed, bringing a change to the environment. We define an action as a tuple $(name, preconditions, target, postconditions)$. The *name* of an action is a unique string that identifies it. The *preconditions* are a set of facts that must be true in order to realize the action. In our system, an action is executed on a *target*, which can be a physical object, like a cup, but also an area of the environment, like a room. The *postconditions* are the set of facts, and their values, affected by the action's execution.

Since we are interested on reasoning and not perceptual aspects, we use inference, as explained in chapter 4, in order to understand when a human has performed an action. Through the predefined *postconditions* of actions we can also infer changes in the state of objects, e.g. the human opens a box, so the box is now open.

3.2.2 Architecture

These aspects are represented in the Situation Assessment layer, and in particular in the following modules, shown in figure 3.2.

- Sensor Data. Data produced by different possible sensors (e.g. lasers, camera, etc.).
- Entity Detection. Different components detect and track humans and objects in the environment.
- Geometrical Reasoning. Symbolic facts are produced starting from perceptual data.
- Belief Management. The system maintains a mental model of each agent.
- Database. The Database stores symbolic facts produced by the system.

- Intention and Action Recognition. The system infers the current intentions of a human, and which actions are executed, by integrating geometrical reasoning, the human's belief model, planning, and bayesian inference.

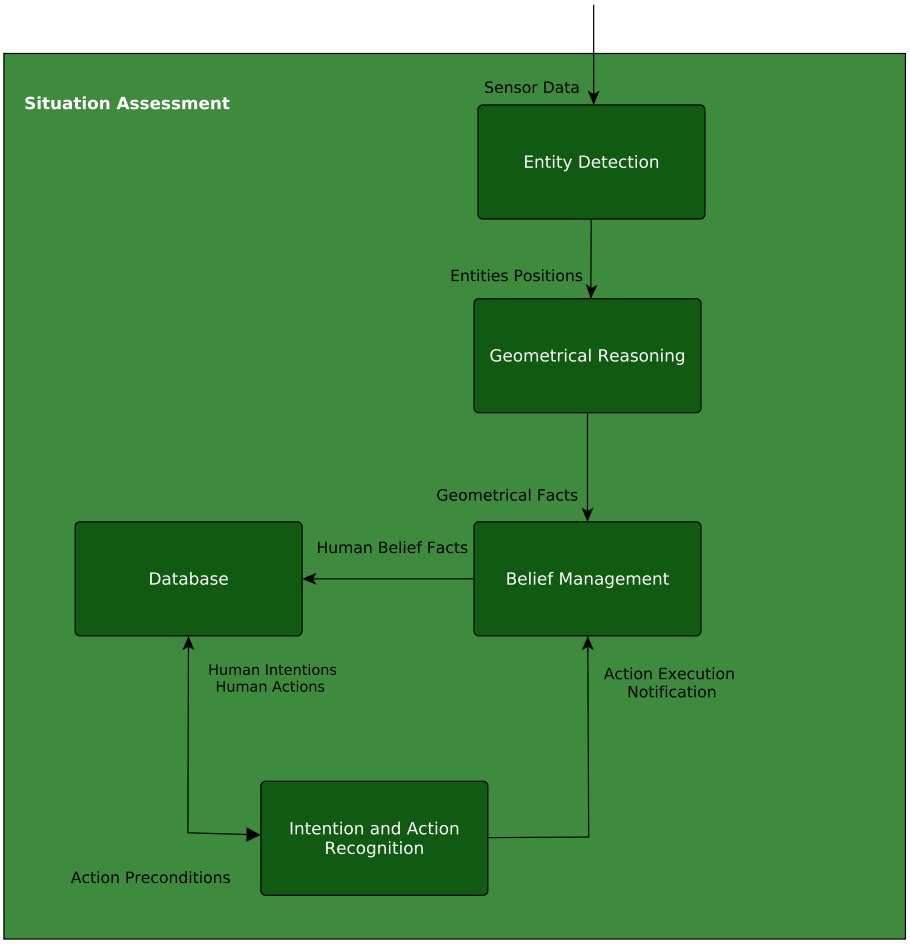


FIGURE 3.2: Overview of the different modules composing the Situation Assessment layer.

Symbolic facts are constantly produced, starting from the sensors data and from the position of entities. The Belief Management module collects these information to maintain the belief model of each agent. The belief model of humans and of the robot is used by the Intention and Action Recognition module to infer the most likely human intentions, and which actions they perform. All these information are introduced in the Database, and can be read by other components. For example, the Goal Management layer can choose a goal based on human commands or an estimation of humans' intentions. In a similar way, the Task Execution layer will read the Database in order to obtain the state of the world, to check action preconditions.

The geometrical reasoning and belief management capacities of this layer were presented in Milliez et al. (2014), where they were used to pass the Sally and Anne test on a robotic platform.

Dialogue can be a very important source of information. Agents often communicate, while executing a task together, or even when working independently, to clarify ambiguities and obtain missing information.

While we will not present a specific dialogue component in this work, in Ferreira et al. (2015) our belief management component was integrated with a situated dialogue system in a simulator. This model was compared with a basic system (without belief awareness) in a study with 60 interactions, in a simulated environment. We successfully showed that the dialogue management system significantly improves its efficiency, reducing the number of dialogue turns in the interaction, and its accuracy, with a higher success rate when a divergent belief situation appears.

In the following sections we will show how our system is able to manage agents' belief models. Actions will be treated, at this point, as input received by the Belief Management module. The intention and action recognition capacities of our system will be explained in chapter 4.

3.3 Entity Detection

In our system, we chose to simplify perception issues, focusing on reasoning aspects. We associate a unique tag to every object that is interesting in a particular scenario. When the robot observes a tag using a camera, it detects the corresponding object using a tag-matching algorithm. Regarding humans, we use a motion capture software to identify and track agents moving in the environment. Using different tags, we can track the head, shoulders, and right arm of a human. Our situation assessment component has also been tested using a laser and RGB based detector, detailed in Linder et al. (2016), in the SPENCER european project. We also experimented using a depth camera, mounted on the ceiling, and a color-recognition algorithm to identify humans.

3.4 Geometrical Reasoning

Using its perception abilities the robot can build a representation of the environment, starting with entities' positions. With geometrical reasoning we can compute spatial relationships between entities, e.g. the glasses are on the shelf, the human is moving toward the library, the glasses are reachable by the human, the bottle is visible for the human. These reasonings provide a base for the perspective taking abilities of the robot. The geometrical reasoning capacities of our system were introduced in Sisbot et al. (2011). This work was updated to include the production of new symbolic facts. We will show a list of some of the most important facts that our system is able to produce through geometrical reasoning.

- *isOn*. Used when an object is on top of another, for example $CUP\ isOn\ TABLE$.
- *isNextTo*. Used when two objects are on the same surface and close to each other, for example $CUP\ isNextTo\ BOX$.

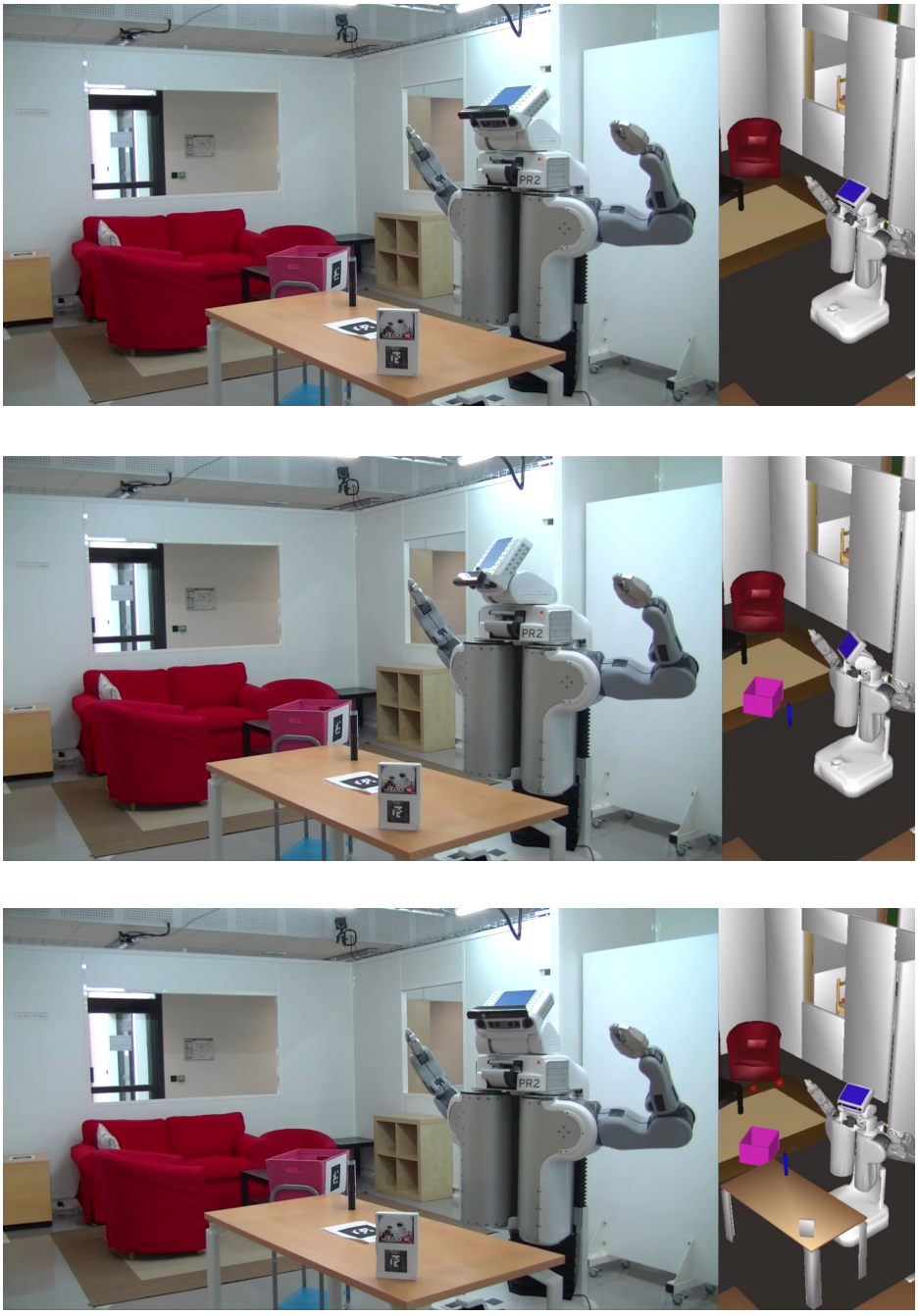


FIGURE 3.3: The robot explores the environment, recognizing objects through tags, and building a representation of the environment.

- *isReachable*. Used when an object is reachable by an agent, computed through inverse kinematics. An example of such predicate is $CUP\ isReachableBy\ ROBOT$

- *isVisible*. Used when an entity is visible by an agent. This is calculated by computing the field of view of an agent, and checking if there is a sufficient portion of the object not hidden by occlusions in this field. An example of this predicate is $CUP\ isVisibleBy\ ROBOT$.

- *isMoving*. Used when an agent is currently moving. This is computed by checking if the agent's

displacements in a pre-determined time unit is bigger than a pre-determined threshold. We use an hysteresis filter to avoid continuous oscillations in the value of the fact. An example of $isMoving$ is $ROBOT\ isMoving\ TRUE$.

- *isMovingToward*. Used to indicate that an agent is moving toward a particular entity. This is computed by checking if the distance between the agent and the entity is decreasing. An example of this predicate is $ROBOT\ isMovingToward\ GREG$.

- *isOrientedToward*. Used to indicate that an agent is oriented toward a particular entity, for example $ROBOT\ isOrientedToward\ GREG$.

- *pose*. Used to indicate that an agent is in a particular pose, for example $GREG\ pose\ HANDOVER$.

- *isAt*. Used to indicate that an agent is in a particular location, for example $GREG\ isAt\ LIVING\_ROOM$.

3.5 Belief Management

We have created a rule based framework in order to build the beliefs of each agent and update them when needed. Human belief models are updated using the perspective taking skills of the robot. When the robot detects the execution of an action in the world, with the mechanisms shown in chapter 4, it updates the belief model for itself and for every human that can perceive the action, adding its *postconditions* to their models. When an action is not perceived by a human (e.g. the user was in another room), his belief model will not be updated, as he is not aware of the changes that occurred.

However, when he comes back and looks at the environment, we assign him a new belief state following a set of rules, which we will now explain. We call p a fact, H the agent, HB his belief model, and RB the robot's belief model. We also create the following predicates: $obs(p)$ means that p is observable for H, $valid(p, x)$ means that p does not contradict the current perception data of agent x, $value(p, m)$ is the value p in belief model m, and $vis(p, x)$ means that agent x has visibility on the linked entities of p (e.g. if p is $MUG\ isOn\ TABLE$ the linked entities are MUG and $TABLE$) . The rules for the *valid* predicate will be different in each attribute. For example the fact $MUG\ isOn\ TABLE$ will not be valid for agent Max if he can see that there is no mug on the table. For each fact $p \in HB \cup RB$:

- if $p \in RB, \quad p \notin HB, \quad obs(p), \quad vis(p, H) \rightarrow value(p, HB) = value(p, RB)$.

- if $p \notin RB, \quad p \in HB, \quad obs(p), \quad vis(p, H) \rightarrow remove \quad \mathrm{p} \quad from \quad HB$.

- if $p \in RB, \quad p \in HB$ then:

- if $value(p, HB) \neq value(p, RB)$,
  $obs(p), \quad vis(p, H) \rightarrow$
  $value(p, HB) = value(p, RB).$
- if $value(p, HB) \neq value(p, RB)$,
  $!obs(p), \quad !valid(p, H) \rightarrow$
  $value(p, HB) = unknown.$

The idea of this set of rules is updating an agent's mental belief model for a fact only if it is observable, or if it is not observable and perception data contradicts the current value of the fact (e.g. the mug was moved from the table to the kitchen while the agent was in another room. While the agent can not see where is the mug, he can see it is no longer on the table).

Chapter 4

Inferring Human Actions and Intentions

In this chapter we show how our system is able to infer human actions and intentions. Section 4.1 introduces the problem of intention recognition, discussing some related studies. Section 4.2 shows our approach at solving this problem, while section 4.3 discusses the proactive behaviors that the robot can execute after inferring the human's intention. Finally, section 4.4 shows an example of a run of our algorithm.

4.1 Introduction

A crucial skill to interact with humans is recognizing others' actions and goals. This process is is directly linked to modeling humans' beliefs, since, as explained by Byom and Mutlu (2013) "as humans, we generally believe that others act in ways that are consistent with their beliefs and goals".

The recognition of human activities is an important topic in computer science research, which can be studied at different levels. Anticipating human actions and movements allows the robot to adapt its behavior and proactively help humans, as studied in Koppula and Saxena (2013).

Sequences of actions can be linked to plans, a well-known topic called plan recognition. Several approaches have been studied in this domain using, for example, classical planning (Ramırez and Geffner, 2009), probabilistic (Bui, 2003) or logic techniques (Singla and Mooney, 2011).

As previously said, we call an intention the wish and will to achieve a goal. Intention recognition is intrinsically linked to plan recognition, since, if an agent is acting by following a plan we can assume that it has the intention of achieving a goal. In the rest of this chapter, we will consider plan recognition as a way to recognize intentions.

Other approaches that can be used to estimate the intention of a human are Interactive Partially Observed Markov Decision Processes (I-POMDP) and Inverse Learning. I-POMDP (Gmytrasiewicz and Doshi, 2004) offer a rich framework that extends Partially Observed Markov Decision Processes

(POMDP) in a multi-agent setting. Inference in these models can be extremely complex, but there have been attempts at solving this issue, like in Doshi and Gmytrasiewicz (2009), Hoang and Low (2013).

Inverse Reinforcement Learning (Ng et al., 2000) formulates the problem of computing an unknown reward function of an agent after observing his behavior. This strategy has been applied, with Bayesian Networks (BN), in Nagai et al. (2015), in order to learn the mental model of another agent, and choose appropriate actions for a relationship building task. A linked approach is inverted planning, which has been applied in a bayesian framework in Baker et al. (2009) for human action understanding.

Contextual information can be used to further disambiguate complex situations. For example, if it is currently raining (context), we could think that it is more likely that Greg will look for his umbrella (intention) if he has to go out. Liu et al. (2014) show a system using BNs to understand users' intentions with an emphasis on contextual information. This BN is constructed using object affordance nodes (e.g. a cup can be washed or used for drinking), context nodes (e.g. it's a hot day, the cup was recently used), and intention nodes (e.g. drinking from a cup or washing it). The causal links between contexts and intentions are learnt through a user study, which uses an online questionnaire where participants need to rate the strenght of the connection betwen an intention and a context. The work does not study how to adapt this BN to complex plans, composed by sequences of actions.

It is very important to consider humans' beliefs when estimating their intentions. In a dynamic environment, agents can execute actions, modifying the state of the world without other agents being able to perceive the changes. Let us imagine a scenario. Bob comes back home from work and would like to relax while reading. He lays down on a sofa with a book, and reaches to a nearby table to grab his glasses. He does not know that his wife, during the day, moved the glasses to another room. If we would ignore Bob's beliefs on the world (i.e. he does not know that the glasses are not on the table) we could infer that, for example, Bob would like a drink while he is sitting on the sofa, or the tv remote controller. If, instead, we would know that Bob thinks his glasses are on the table (and we would use other contextual information perhaps, like Bob's habitudes) we would be able to correctly infer Bob's current intention, that is, taking his glasses, and warn him that they are not there, perhaps even fetching them for him.

In robotics, an interesting framework that considers this issue is the Bayesian Theory of Mind (Baker and Tenenbaum, 2014), used to represent the inference process of an observer looking at another agent's behaviors. The acting agent is modeled as a POMDP, whose richness is able to represent his possible beliefs about the world. The observer's process is modeled as a DBN, built starting from the agent's POMDP but considering his reward function (that represents his desires) as hidden. The system has been tested against some alternative models and compared, in user studies, with human capacities, to understand how well it models theory of mind. Since the models used are quite complex, scalability in

the model could be an issue. Also, the study is focused on a single-agent scenario, and does not consider collaborative problems.

Let us examine the two simulation-based systems that we already presented in the previous section, HAMMER (Demiris, 2007), and the architecture of Breazeal et al. (2009), and see how these cognitive architectures are able to infer actions and intentions.

The HAMMER system is organized with couples of inverse and forward models. Inverse models receive as input the goal and state of the system, producing the motor commands which are needed to achieve the goal. Forward models, instead, receive as input motor commands and compute the predicted future state. When these two models are linked, the forward model receives as input the motor commands produced by an inverse model. This link can form a loop, with the output of a forward model returning to its inverse model, which can adjust a range of parameters if the predicted future state does not match exactly the desired state. These models can be organized in parallel schemas and used to recognize actions performed by a demonstrator.

In this case, the demonstrator's current state, as perceived by the robot, is fed in the inverse models, which in turn send their output to the forward models. The state predicted by the forward models are compared with the demonstrator's state at the next time step. This comparison produces a score, which can be used to infer the most likely action performed.

Forward and inverse models can be organized in hierarchical schemas, to infer tasks and plans. The complexity of these schemas could be quite significant, particularly when trying to recognize a goal which can be achieved in many different ways, depending on the context.

Breazeal et al. (2009) uses a similar ideas, where all the possible robot movements are represented as a graph of connected poses, with arcs showing possible transitions between the poses. This graph is used both to represent the robot's movements and to map observed trajectories. Tasks are represented as schemas, which can be organized in sequential and hierarchical structures to model complex goals.

When trying to infer an agent's intention, the robot looks for a schema whose motor action matches the observed activities of the agent. After that, the schema is traversed in reverse in order to try to determine the real intention. The system is not able to deal with ambiguities, and this algorithm stops if it comes to a point where there is more than a possible explanation for the current behavior.

An example of non simulation-based system in this topic is Talamadupula et al. (2014). This architecture is used to coordinate human-robot teams, based on intention recognition and belief modeling. Creating and maintaining beliefs is handled using the strategy explained in Scheutz (2013), presented in the previous chapter. Prediction of other humans' intentions is based on the plan recognition algorithm of Ramırez and Geffner (2009). While this algorithm uses efficient replanning to increase its efficiency,

in complex domains, where the users have many different strategies to achieve a goal, the system would need to execute frequent replans to infer the actual strategy chosen by the user, which can be expensive.

Our goal was developing an algorithm able, through inference, to recognize human actions and connect them to possible intentions. To avoid errors in interpretation, we decided to use humans' belief models, and not the robot knowledge, to perform our reasonings. We also decided to use contextual information to further disambiguate our estimation. Another goal was to make this algorithm fast and scalable.

This algorithm was introduced in Devin et al. (2016) and we will show, in the next sections, how it was designed.

4.2 Intention and Action Inference

In order to infer actions and intentions, we will provide the following information to the robot: a list of known contexts, a list of known intentions, a list of known actions, a set of observations of human actions, and belief model of humans and of the robot itself.

We introduce a simplification in our model: at each time step, a human can execute only one action and has only one intention, decreasing the ambiguities in the inference process.

We propose, as central model used for intention estimation, a framework based on BNs. We call our implementation of BN an Intention Graph (IG). An IG is linked to a specific human, and composed by the following layers of nodes:

- Context Nodes: these nodes represent contextual information, modeled as boolean variables (e.g. HotDay, ColdDay).
- Intention Nodes: these boolean nodes represent the set of possible intentions. Each intention can depend on several contexts.
- Action Nodes. This is the set of human actions whose preconditions are satisfied in the human's mental belief when the IG is created. Each of these nodes is dependent on all the intention nodes.
- Observation Nodes. We associate to each action a different set of observation nodes, that depend on the associated action node. For example, the distance of a human from the *target* of an action can be an Observation Node.

In a typical usage, the robot will create, for each monitored human, an IG, formed by the Context and Intention Nodes, which we consider statically known by the robot, and a variable list of Action and Observation nodes, which depends on the human's belief model. The robot will create action nodes for each known action whose *preconditions* are *satisfied in the human's belief model*, and their related

Observation Nodes. We stress, in particular, that actions are created based on the human's belief model, since the human might be trying to execute actions that are actually impossible in the current situation (e.g. the human tries to take an object from a basket without looking inside it. He does not know that the object is not there anymore). These IGs will be updated every time that an agent performs an action, creating and removing Action and Observation nodes, depending on the state of the world after the action was performed.

When monitoring a human, we set Context Nodes and Observation Nodes as *evidence*, considering them observable by the robot. These information will allow us to have a good estimation of the most likely actions and intentions of the human, as explained in subsection 4.2.4.

An example of IG, taken from an experiment, can be seen in figure 4.1. In the following paragraphs, we will explain the role of these layers of nodes, and how the conditional dependencies between them are computed. After that, we will show, in section 4.4, an example showing how an IG is created and updated following a sequence of human actions, and how the robot is able to use it to infer the most likely human intention.

4.2.1 From Contexts to Intentions

We introduce a set of contexts in our domain. We consider as context any information that can be used to characterize and motivate an intention (Abowd et al., 1999). We model a context as a fact, which can assume different values and influences the probability of a user having a particular intention. For example, we imagine that a human is more likely to be cooking at dinner time, or to drink a hot mug of tea on a cold day.

Contextual nodes can directly influence one or more Intention Nodes. In this work, we chose to learn these conditional dependencies from humans, as explained in subsection 5.2.2.

4.2.2 From Intentions to Actions

To understand how actions are linked to intentions the robot needs to answer the following question: what actions would a human take, in this situation, given his belief of the world, in order to achieve its intention? Our idea is based on the principle of rationality (Dennett, 1989), which states that agents tend to choose the most efficient actions, taking into account their beliefs about the world, in order to achieve their desires.

In Blakemore and Decety (2001), the authors explain that "the attribution of intentions to actions might rely on simulating the observed action and mapping it into representations of our own intention". We represent this idea by providing the robot with a set of planning models. Each one of these planning models is related to an intention, and represents all the known plans to achieve its linked goal. In this

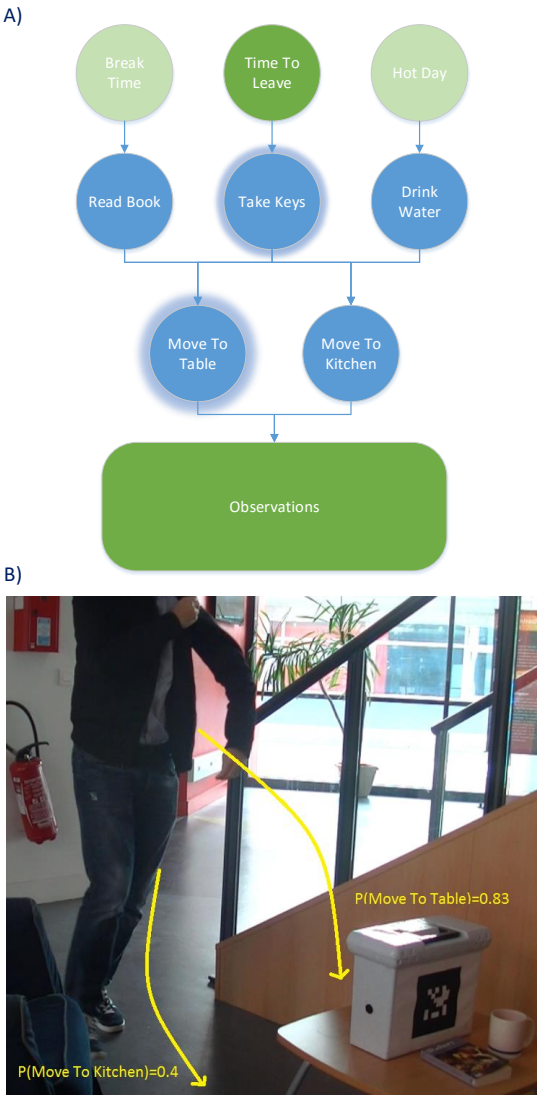


FIGURE 4.1: A scene from our experiment. The yellow arrows show possible actions and their associated probabilities. The diagram represents the current IG. Green circles represent evidence nodes and blue circles other nodes. For the Context Nodes (top of the graph), we represented nodes with a false value as greyed out, and nodes with a true value as green. The most likely nodes in the graph are represented with a glowing effect. The observation nodes were compressed in a single block to simplify the diagram

way, we can estimate how much the current human actions are compatible with the plans related to an intention.

In our implementation, for each intention known by the robot, we will create an associated Markov Decision Process (MDP), to represent all the possible plans linked to this intention. After solving the set of MDPs we will use the calculated action value function $Q(s, a)$, to create conditional dependencies between Intention and Action Nodes in the IG. We define $P(a|I_i = 1)$, the probability that action a will be performed if intention I_i is true, as as:

$$P(a|I_i = 1) = \frac{Q_i(s, a)}{\sum_b (Q_i(s, b))} \tag{4.1}$$

where we normalize the value function $Q_i(s, a)$ for intention i and action a in the human's belief state s, over the value function $Q_i(s, b)$ calculated on all the monitored actions b.

The key idea in this problem is to use the human's belief state as input for the MDPs' value functions. In this way we are using perspective taking at a planning level, since the human action will be consistent with his intention in his own belief state but may be not optimal, irrelevant, or even dangerous in the real world (e.g. in case of wrong belief).

Our idea is similar to Karami et al. (2010). In this work, the robot planning model is a POMDP, where the human intention is a hidden variable. The transition function for the human intention is computed starting from the action values obtained from a set of human MDPs, that simulate human policies related to different intentions. In this work, we used a BN for the inference process, instead of a POMDP. This allows us to include in a simple way more information in the inference process, such as context, and to separate the mechanisms used for inference and for the robot's actions. Also, we improve the recognition process by including the belief state of the human.

4.2.3 From Actions to Observations

Intentions will be inferred from human actions, so the robot needs to monitor their execution. For each Action Node we can define a different set of Observation Nodes, which depend on the specific actions. Typical examples are: the distance of the human's body from the action's *target*, its variation, the distance of the human's hand from the action's *target*, and its variation. The conditional dependencies of the Observation Nodes are precomputed.

4.2.4 Intention and Action Inference

We assume, in this work, that at each moment a human can only execute a single action, and the robot will react only to his most likely intention. The most likely action and intention are inferred from the BN in the following way. We call $P(n)$ the inferred probability of a node n, $B(n)$ the set of brothers of n (that is, nodes on the same layer), and δ_1, δ_2 two thresholds. The robot selects the most likely action and intention following these rules:

- $P(n_i) > \delta_1$
- $\forall b \in B(n_i) : P(n_i) > P(b) + \delta_2$, where n_i is the node associated to the interested intention or action.

To infer that an action has been performed, the robot uses an estimation of the action's probability and geometrical reasoning. For example, the robot infers that a human has taken a bottle if his hand is closer to it than a threshold σ and the action's probability in the IG respect the previous rules.

For another example, the robot infers that a human has mixed some ingredients in a bowl if his hand approaches the bowl and then leaves after some seconds, always taking into account the probability of the action in the IG.

When the robot infers that an action has been performed, it updates the world state with its *postconditions*, triggering an update on the beliefs of all present agents. The current human intention is recorded in the Database, and will be used by the Goal Management layer.

4.2.5 Action Inference with Unknown Intentions

In some situations, we are interested in inferring human actions without using information from intentions. When the robot is cooperating with a human, we assume that the human will follow a set of actions to achieve the cooperative goal following a shared plan. The MDPs used in our implementation are able to plan only for a single-agent, and can not be used to represent the shared plan. This could be a problem, because when agents act alone, they might execute different actions than when acting together. For example, if Greg and the robot are reordering a room together, Greg might give the robot a book, because in the shared plan the robot will put it in its place. If Greg is acting alone, he will need to place the book back by himself.

In this situation, we will create an IG using only actions and observations. In this way, we can still infer which actions are executed, but we have to rely only on geometrical reasoning. Since we are using limited perception algorithms, we could easily encounter ambiguous situation, where, for example, the human moves his hand toward two close objects. In this situation, without using higher level information, the system could make mistakes, and infer that the wrong action has been performed.

We will discuss, in chapter 9 about a possible multi-agent MDP implementation to solve this issue.

4.3 Proactive Behaviors

Information about the most likely intention and action will be introduced in the Situation Assessment layer's Database, to be read by the Goal Management layer. Based on this information, the robot can execute two different proactive behaviors: correcting the belief state of the human, and proactively helping him to achieve his task.

4.3.1 Correcting Belief State

Having a wrong or incomplete belief on the world state can lead agents to execute non optimal, useless, or even dangerous actions. The robot needs to detect these situations in order to warn the human. For

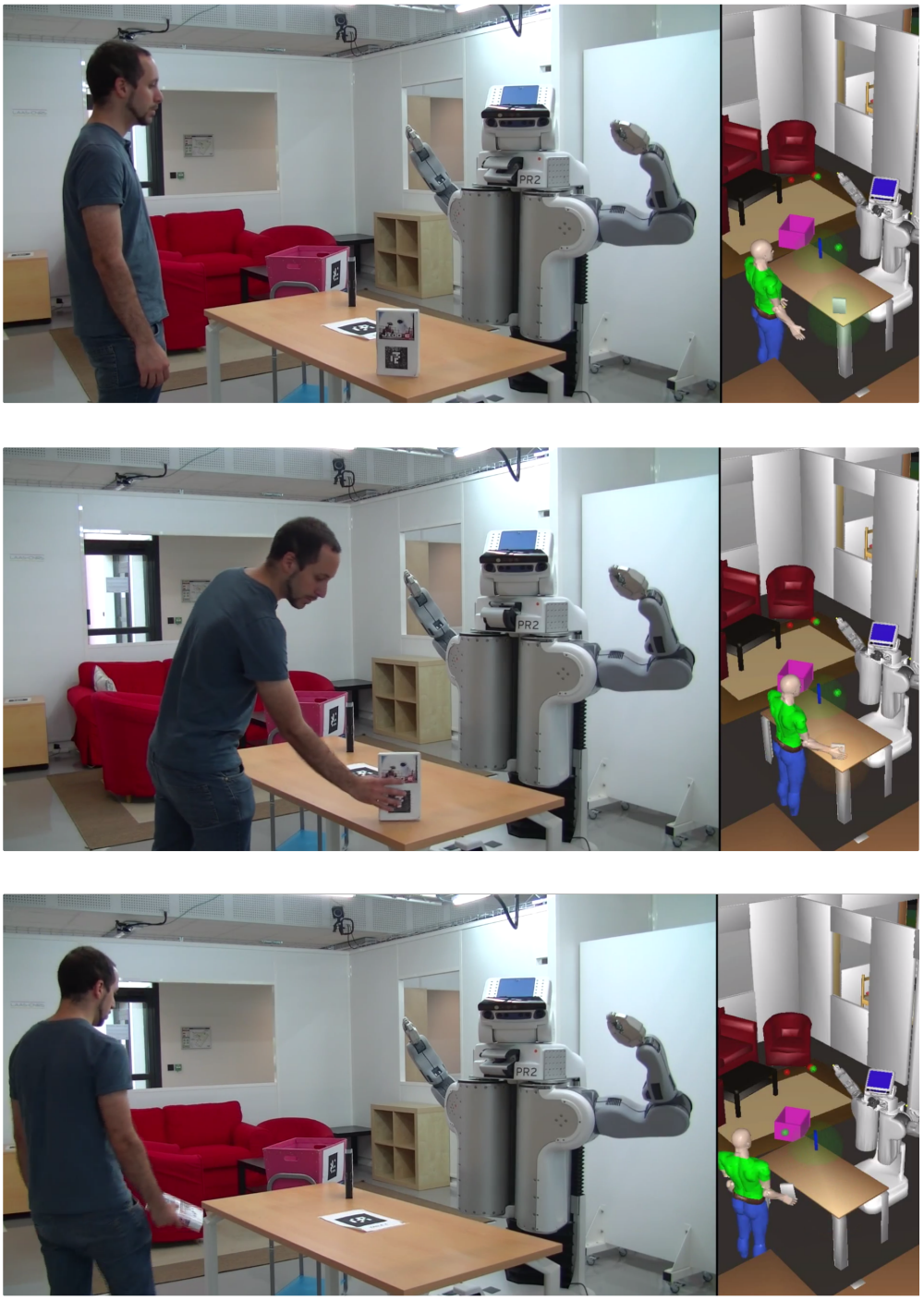


FIGURE 4.2: The human picks an object and the robot updates accordingly the world state. The spheres surrounding objects show when the human's hand is considered to be near an object.

example Greg could search for his glasses in a wrong location if he does not know that they were moved, or could touch a hot object if he does not know its temperature.

The robot assumes that it always holds a correct belief state. Our solution uses the expected action

rewards introduced in 4.2.2. The idea is comparing the expected reward for performing an action according to the human and the robot's belief states. To formalize: we compare the action values $Q_m(s_h, a_h)$ and $Q_m(s_r, a_h)$, where m is the most likely intention, s_h and s_r are the robot's and human's belief state, and a_h is the most likely action. If these values are not equal the human expects a different outcome from its action than what should actually happen.

We propose a simple solution, where the robot warns the human of the detected divergent belief for that action. For example, Bob wants to drink tea from a closed, opaque bottle, which the robot knows is empty (perhaps because another agent drank the last glass), while Bob does not. When he approaches the bottle, the robot detects that his most likely intention is to drink tea.

The system calculates the expected rewards from taking the bottle in the two belief models and obtains different values. The system checks the facts related to the attributes associated to the target of the action (i.e. the bottle) in the two mental models, and extracts the differences. Using this information, the robot corrects the divergent belief, informing the human that the bottle is now empty.

If there is no difference in the belief models, it means that the divergent belief is actually related to a future action that the human is expected to execute. For example, let us imagine a scenario where Greg is trying to prepare a dish, requiring pasta and tomatoes. Let us imagine that Greg does not know that there are no more tomatoes, and moves to take the pasta. Using Greg's belief model, there will be a certain reward to take the pasta, since it is a necessary step to prepare the dish. In the robot's belief model, the reward will be zero (assuming that he correctly inferred the intention of Greg to prepare pasta with tomato sauce), because there are no more tomatoes. While there are is no divergent belief on attributes related to the pasta, it would be useful if the robot would inform Greg that there are no more tomatoes.

A solution is extracting an horizon of future actions from the MDP related to the *prepare pasta with tomato sauce* intention. Our system can do this by simulating an update on the MDP, considering as starting state the current human belief model, choosing actions from the agent's policy computed when solving the model, and taking as next states the most likely effects of the actions. We can repeat the reasoning on the targets of each action in this horizon, in order to find a divergent belief.

If we can still not find a divergent belief, the robot can only inform a human that there might be a problem to achieve its goal, but it is not able to identify it.

In a real scenarios, the human might have several divergent beliefs at the same time. While the robot could inform the human as soon as it detects a divergent belief, it would risk overloading him with unneeded information. For example, Greg might not know that his wife drank a mug of coffee or moved the remote control to the table, but maybe he will not need these information in the near future. We chose, with this approach, to give information about a divergent belief only when it risks impacting the actions of a human.

4.3.2 Performing a part of the plan

There are situations in which the robot should help the human achieve its goal by physically acting. The Goal Management layer will consider new inferred intentions as possible goals for the robot, and will communicate with the Plan Management layer to achieve them, as explained in chapters 6 and 7.

4.4 Intention Graph Example

4.4.1 Scenario

We will now show an example of use of an IG. We start by defining a scenario with a human, Greg, and two possible intentions: drinking water and reordering a table, by moving all the objects on top of it to the kitchen. We set three different locations: a *table*, a *shelf*, and a *kitchen*. We consider two objects: a *glass* and a *bottle*. Both these objects can contain water, but the *bottle* is opaque and Greg can not observe if there is water or not in it. We simplify the situation by imagining that the bottle is already open. Of course, this scenario is not realistic and is chosen just to illustrate the IG. In a real situation Greg would notice that the bottle is empty when taking it, because it would be too light, or by looking through its hole.

At the start of the scenario, Greg is at the *shelf*, the *bottle* and *glass* are on the *table*, and the they are both empty. We set a situation of divergent belief, where Greg does not know that the bottle is empty, but the robot has this information.

We introduce two predicates: *isAt*, that represented the location of an entity, and can assume the values *TABLE*, *SHELF*, and *KITCHEN*; and *capacity*, which indicates if an object contains water, and can assume the values *0* and *1*.

The belief models of Greg and of the robot are shown in table 4.1, while the set up for this scenario is shown in figure 4.3.

Robot	Human
GLASS isAt TABLE	GLASS isAt TABLE
BOTTLE isAt TABLE	BOTTLE isAt TABLE
GLASS capacity 0	GLASS capacity 0
BOTTLE capacity 0	BOTTLE capacity 1

TABLE 4.1: Belief models for Greg and for the robot in the IG example scenario. There is a divergent belief situation, where Greg does not know that the bottle is currently empty, represented by the fact *BOTTLE capacity VALUE*

We introduce two contexts in the scenario: *HotDay*, representing the fact that the day is particularly warm, and *AlreadyDrank*, representing the fact that the human has recently drank water.

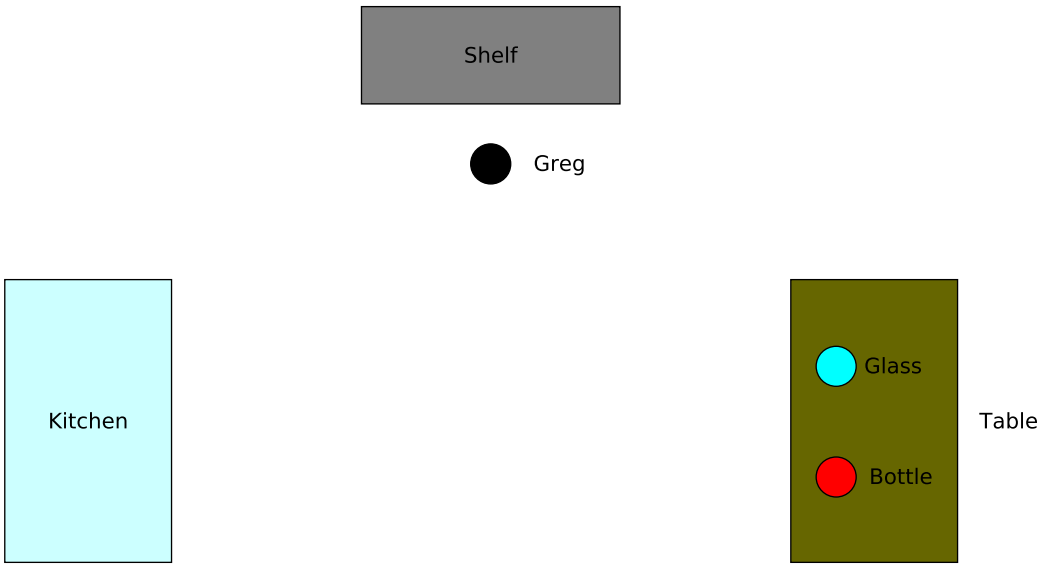


FIGURE 4.3: The image shows the set-up for the IG example scenario. The locations are represented as different rectangles. Greg is represented as a black circle. The glass and bottle are represented, respectively, by a blue circle and a red circle.

We introduce this possible set of actions in the scenario: taking the bottle, taking the glass, filling the glass using the bottle, drinking from the glass, moving to the kitchen, moving to the table, placing the objects in different locations. We simplify this set to only include the actions relevant to this example, and so we do not include the possibility for Greg to refill the bottle with water. Also, we consider that Greg can only hold a single object at a moment.

4.4.2 Building the Intention Graph

We build a starting IG with the following nodes:

- Context Nodes: *HotDay*, *AlreadyDrank*.
- Intention Nodes: *DrinkWater*, *ReorderTable*.
- Action Nodes: *MoveTable*, *MoveKitchen*. These two actions are introduced in the IG because they are the only ones whose *preconditions* are currently satisfied, since Greg can move in any location where he is not at the moment.
- Observation Nodes: *BodyDistanceTable*, *TowardTable*, *BodyDistanceKitchen*, *TowardKitchen*. The *BodyDistance* nodes represent the distance between Greg and the target of an action, and can assume the values *close*, *medium*, *far*, *out of range*. The *Toward* nodes are *true* if the distance between *Greg* and the object is decreasing, and false otherwise.

In this scenario, we precompute the probability table of the causal links between Context Nodes and Intention Nodes, and between Action Nodes and Observaton Nodes. We set a causal link between

HotDay and *DrinkWater*, and one between *AlreadyDrank* and *ReorderTable*. In both cases, we use the probability table of table 4.2.

Context	Intention	
	0	1
0	0.6	0.4
1	0.4	0.6

TABLE 4.2: Conditional probabilities of the Intention Nodes in the IG example scenario.

We consider as slightly more likely that Greg wants to drink water in a Hot Day, and that he wants to reorder the table if he drank recently.

We set a causal link between each action and its observation nodes. For example, *MoveTable* will have a causal link with *BodyDistanceTable* and *TowardTable*. In both actions, we use the probability table of tables 4.3 and 4.4.

Action	Toward	
	0	1
0	0.8	0.2
1	0.2	0.8

TABLE 4.3: Conditional probabilities of the Toward Nodes in the IG example scenario.

Action	Distance			
	Close	Medium	Far	Out of Range
0	0.16	0.2	0.25	0.39
1	0.39	0.25	0.2	0.16

TABLE 4.4: Conditional probabilities of the Distance Nodes in the IG example scenario.

If Greg is executing an action, it more likely that he is moving toward the action's *target*. For example, the *MoveTable* action is more likely the closer Greg is to the table and if the distance to it is decreasing.

4.4.3 Building the Human MDPs

To set the conditional probabilities between Intention and Action Nodes, we created two different MDPs, each related to one of the intentions. We will now show the state space S, action set A and reward function R of the two MDPs. We do not include the transition function of the model as it is extensive and does not help understanding this example.

- DrinkWater:
  - S: $\{agent\_isAt, glass\_isAt, bottle\_isAt, bottle\_capacity, glass\_capacity\}$.

  - A: $\{agent\_move\_table, agent\_move\_kitchen, agent\_move\_shelf, agent\_take\_bottle, agent\_take\_glass, agent\_fill\_glass\_bottle, agent\_drink\_glass, agent\_place\_bottle\_table, agent\_place\_bottle\_kitchen, agent\_place\_bottle\_shelf, agent\_place\_glass\_table, agent\_place\_glass\_kitchen, agent\_place\_glass\_shelf\}$.
  - $R(s, a) = 1000$ if
    $glass\_isAt == agent\_isAt$ AND $glass\_capacity == 1$ AND $a == agent\_drink\_glass$.

- ReorderTable:
  - S: $\{agent\_isAt, glass\_isAt, bottle\_isAt\}$.
  - A: $\{agent\_move\_table, agent\_move\_kitchen, agent\_move\_shelf, agent\_take\_bottle, agent\_take\_glass, agent\_place\_glass\_kitchen, agent\_place\_bottle\_kitchen\}$.
  - $R(s, a) = 1000$ if
    $(glass\_isAt == kitchen$ AND $bottle\_isAt == human$ AND $a == human\_place\_bottle\_kitchen)$
    OR
    $(glass\_isAt == human$ AND $bottle\_isAt == kitchen$ AND $a == human\_place\_glass\_kitchen)$.

After solving these MDPs we used their action value functions to compute the conditional probabilities for the Action Nodes, as shown in section 4.2.2.

4.4.4 Simulated Run

We will now show a simulated run of this scenario in three steps. This run was created by introducing a stream of observations in the system, including the actions that Greg will perform. We consider that Greg's intention in this example is *Drink Water*. We will conduct three different tests for each step:

- Test 1. We create the IG using the human's mental belief state to compute the causal links between Action and Intention Nodes. We do not use context to help disambiguate the intentions.
- Test 2. We add contextual information to Test 1, setting *HotDay* to *true* and *RecentlyDrank* to *false*.
- Test 3. We do not use contextual information and use the robot's mental belief state to compute the causal links between Action and Intention Nodes. This means that the robot will evaluate Greg's action by considering that he knows that the bottle is currently empty (even though he does not know).

We will now describe and discuss these stages:

- Greg is approaching the *table*, as shown in figure 4.4. The observations are updated correspondingly, and the probabilities of the actions are computed. The action *MoveTable* has the highest probability.
  - Test 1. At this stage, the system is not able to infer the correct intention. Greg could be going to the *table* to take the objects and bring them to the *kitchen* or to drink water. The probability of the two intentions is both 0.50.
  - Test 2. By using context, the system is already able to infer the correct intention. The probability of *DrinkWater* becomes 0.69, and that of *ReorderTable* 0.3. The robot could already fire a proactive behavior to inform Greg that the *bottle* is, in fact, empty.
  - Test 3. Without context, the system thinks that Greg is trying to reorder the table, since it is not possible to drink water because the *bottle* is empty. The system would infer a wrong intention.
- Greg has arrived to the *table*, as shown in figure 4.5. The *postconditions* of the action are added to the mental models of the robot and of Greg, changing Greg's location. The system creates a new IG, setting as actions those whose *preconditions* are now satisfied: *MoveKitchen*, *MoveShelf*, *TakeGlass*, and *TakeBottle*. At this point, Greg's hand moves toward the *bottle*, and the observations nodes are set appropriately.
  - Test 1. The system is still not able to disambiguate between the two intentions. Greg could be taking the *bottle* to fill the *glass* or to bring it to the *kitchen*.
  - Test 2. In this case, the context is still providing the missing information to understand that Greg wants to drink water.
  - Test 3. In this case, since it is not possible to drink water in the robot's mental belief, the system is still inferring the wrong intention.
- Greg has taken the *bottle*, and is now bringing it to the *glass*, as shown in figure 4.6. A new IG is created, containing the actions that are now executable: *MoveKitchen*, *MoveShelf*, *PlaceBottleTable*, and *FillGlass*. In test 3, the IG created would not contain the action to fill the glass with water, since it is not executable in the robot's mental belief model, where the *bottle* is empty.
  - Test 1. At this point, the system has enough information to understand that Greg's intention is *DrinkWater*, which assumes a probability of 0.90. The system can now fire a proactive behavior, with the robot informing Greg of the divergent belief. Even though the actions *PlaceBottleTable* and *FillGlass* could be ambiguous from a geometric point of view (since, in both cases, the arm of the human will approach the table), the system infers that the action performed was *FillGlass* since it is more *useful* in the current situation and with the considered intentions.

- Test 2. Context confirms this intention, bringing the probability to 0.95.
- Test 3. In this case, Greg's movement would not be understood, since in the robot's mental belief model it is not possible to fill the glass with water. Greg's action would actually be inferred as *PlaceBottleTable*, which has no use for any intention.

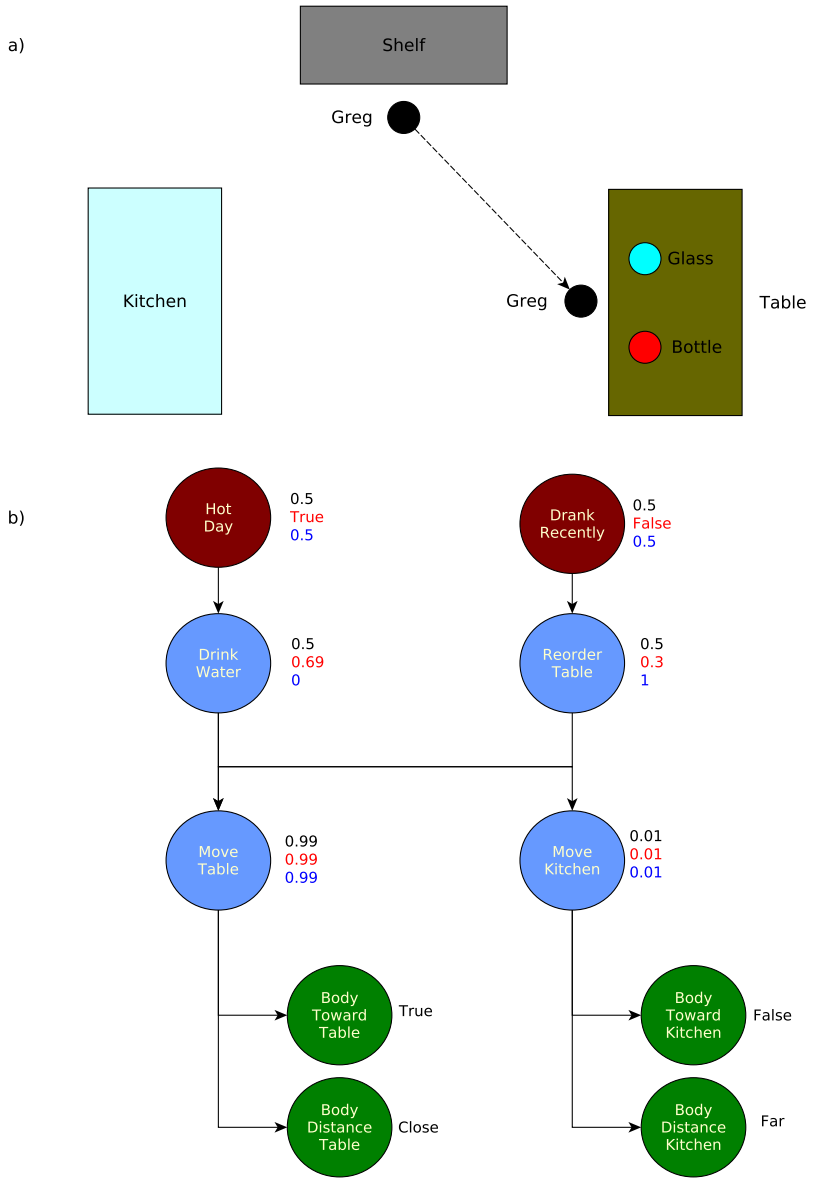


FIGURE 4.4: a) Greg, represented as a black circle, is approaching the table. The two black circles correspond to his starting and ending location, with the dotted arrow showing the direction of his movement. b) The corresponding IG graph. Nodes are represented as circles and causal links as arrows. Intention and Action Nodes are represented as blue. Observation Nodes are represented as green to show that we consider them as evidence, fixing their values. Context Nodes are represented as red to mean that in the first and third test they are treated as standard nodes, and in the second as evidence. For each node we show the probability that its value is true or its current value, if the node is treated as evidence. We show three different values for each node: the black one shows the value if we compute the probability by using the human's mental belief (test 1), the red if we use the human's mental belief and treat Context Nodes as evidence (test 2), and the blue if we do not use context and use the robot's mental belief for the computation (test 3). Test 1 shows that at this point the system is not able to disambiguate between the two intention, which have a value of 0.50. Test 2 shows that context would help the robot to infer correctly the intention. Test 3 shows that by not using the human's mental belief the robot would discard the drink water intention, considering it not achievable.

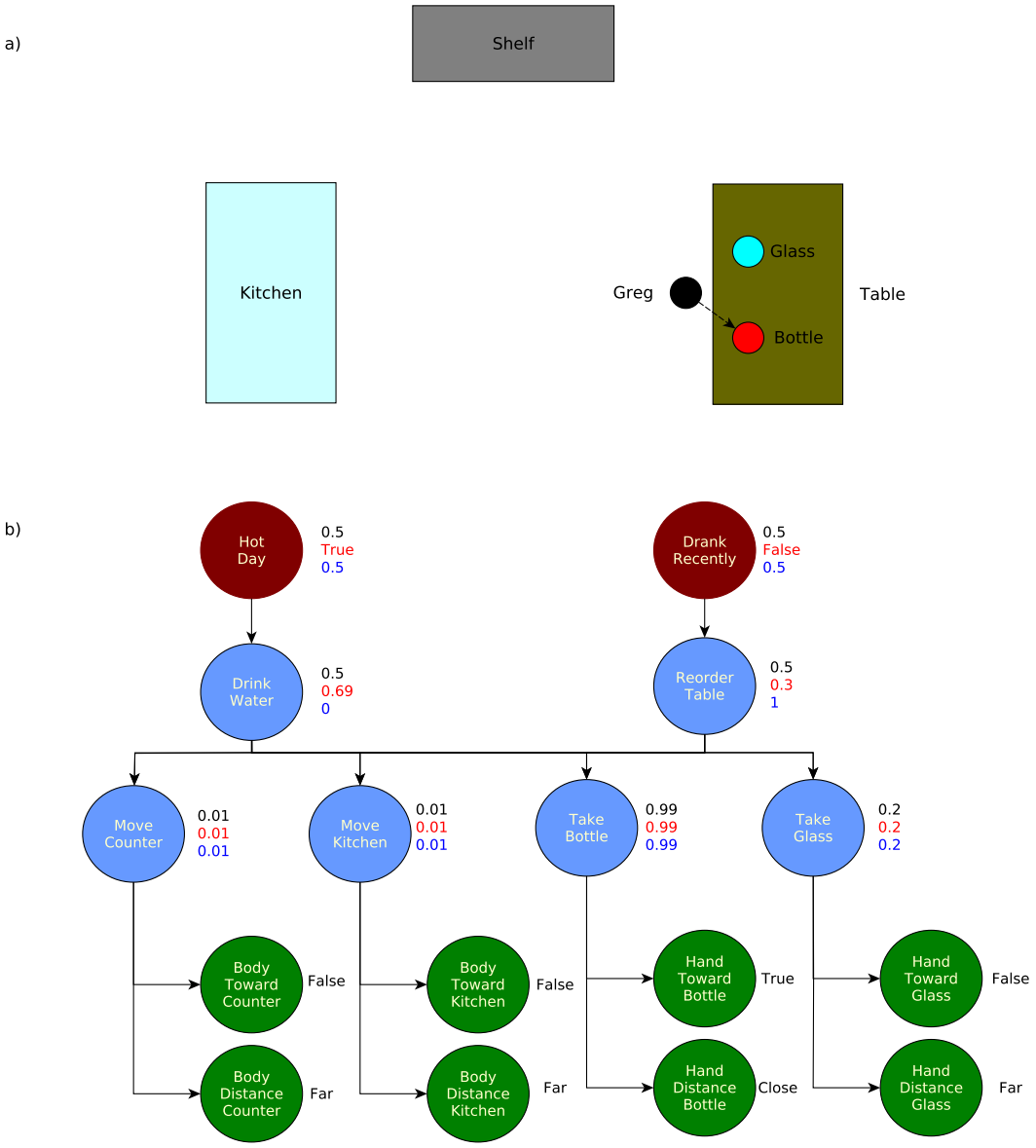


FIGURE 4.5: a) Greg's hand is approaching the bottle. b) The corresponding IG graph. As before, test 1 is shown in black, test 2 in red, and test 3 in blue. The results of the test are very similar to the previous time step, shown in figure 4.4.

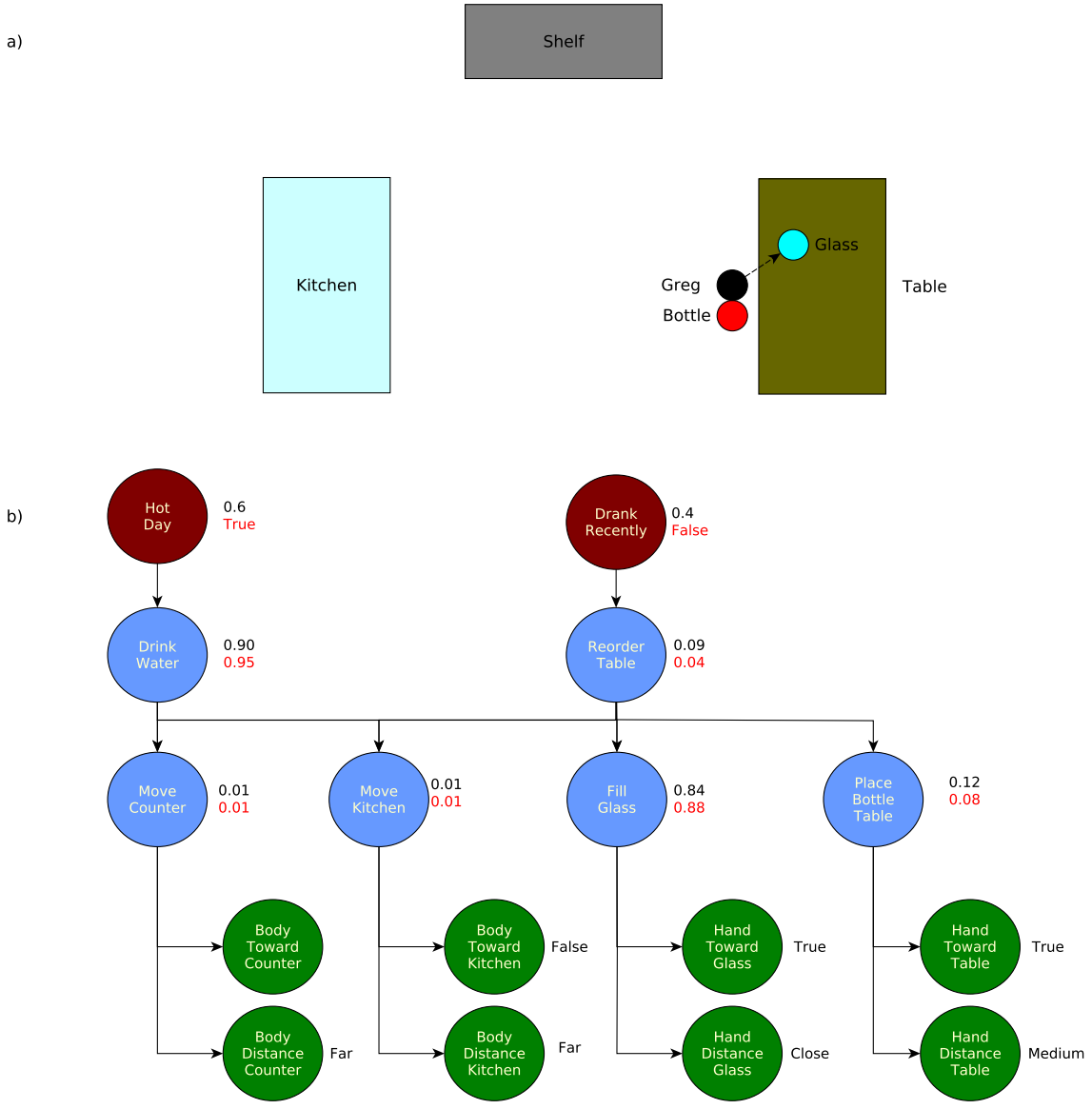


FIGURE 4.6: a) Greg has taken the bottle, shown by placing the red circle and black circle close. His hand, with the bottle, is now approaching the glass b) The corresponding IG graph. As before, test 1 is shown in black and test 2 in red. Test 3 is not shown since the fill glass action would not be executable in the robot's mental belief model, and so the corresponding IG would be different. In test 1 the system has now sufficient information to infer the correct intention, which has a value of 0.90. Test 2 confirms this choice.

4.5 Discussion

This component is able to estimate the likelihood of a human's intention by combining BNs, MDPs, geometrical reasoning, and the capacity to model human's beliefs. Contextual information can further help to disambiguate the inference process.

Another advantage of our approach is its good scalability. Computing the probabilities in a BN can be done using efficient and well know algorithms, which scale well with the size of the network, meaning that the IG is able to accomodate the addition of new actions, observations and contexts.

Adding a new intention means creating and solving another MDP. Since this process is done offline this would not impact the run of the system. When computing the conditional probabilities of the action nodes, the system uses the action value function of the MDPs, which is stored in memory and can be directly accessed.

Chapter 5

Results and Experiments

This chapter presents an experiment used to evaluate the intention recognition capacity of our system. Section 5.1 introduces our case study, where we decided to compare the prediction of our system with those of humans. Section 5.2 explains how the experiment was actually conducted. We show and discuss our results in section 5.3.

5.1 Case Study

5.1.1 Study Description

Evaluating the capacity of the system to estimate human intentions is not easy, since intentions are not directly observable. A possible solution, as shown in Baker and Tenenbaum (2014), is comparing the estimation of human intentions, performed by other humans, with the predictions of our system. In order to perform this comparison we created a user study where we showed participants several videos, asking them to estimate the likelihood of a set of intentions for each video, and collected their results. The same tests were simulated on the system, streaming as input a sequence of observations corresponding to the actions shown in the video (e.g. if the video shows a human approaching the table, we will stream to the system a trajectory of coordinates that leads to the table position). All of the test videos ended in a situation of ambiguity. For example, in one test we showed a human approaching a table with different objects, and stopped the video before users could see which object the human wanted to take. Some videos include more information than others, like comments by humans or situations that help to disambiguate the intention.

We have performed an equivalence test, comparing users' intentions predictions with those of the system, following the two one-sided tests (TOST) approach. We choose as a threshold for equivalence the standard deviation σ of the users' answers. The idea behind this choice is that, if the system's answers

are closer than a standard deviation to the average human answers, its predictions are comparable to an average human answer from our user group.

We defined our hypothesis as follow:

- H_0: $\mu_{hi} - \mu_{si} \leq -\sigma_{hi} \quad \text{OR} \quad \mu_{hi} - \mu_{si} \geq \sigma_{hi}$
- H_A: $-\sigma_{hi} < \mu_{hi} - \mu_{si} < \sigma_{hi}$

where μ_{hi} and μ_{si} are the human average and the system's answer for test i, σ_{hi} is the variance of the human answers for test i.

We have performed tests to evaluate: a) prediction in absence of clues, b) prediction in the presence of contextual clues, c) prediction in the presence of belief state clues.

We have built a household environment with a fixed set of furniture: a *Kitchen Shelf*, a *Table*, a *Sofa*, and a *Chair*. In this environment, we have created two scenarios, composed by several tests, with two agents, *Max* and *Bob*, performing different actions. Each scenario contained a set of objects, and a constrained set of intentions. For the tests related to belief states, we start by showing the users a specific sequence of events, allowing them to build a mental model of the agents. A corresponding simulated sequence will be streamed to the system for this test. We will describe in details the two scenarios and the relative tests.

5.1.2 Cookie Scenario

- Objects: a *Cookie Box*, a *Mug*, and a *Bottle of Water* were placed on the *Table*, close to each other. A pack of *Cookies* was placed on the *Kitchen Shelf*. The *Cookie Box* could contain, or not, *Cookies*.
- Intentions: *Eating a Cookie*, *Drinking Water*, *Reading the Book*.
- Tests:
  - *No Clues*: *Max* approaches the *Table*.
  - *Contextual Clues*: *Max* approaches the *Table* commenting on the warmth of the day.
  - *Divergent Belief No Cookies*: *Max* approaches the *Table*.
  - *Divergent Belief Cookies*: *Bob* approaches the *Table*.
- *Divergent Belief Event*: *Max* and *Bob* are chatting on the *Sofa*. Max eats the last *Cookie* from the *Cookie Box* before closing it and leaving. While *Max* is away, *Bob* takes *Cookies* from the *Kitchen Shelf*, fills the *Cookie Box* with them, and closes it, before leaving.

The *Divergent Belief Event* was shown to the users (and its simulation streamed to the system) between the *Contextual Clues* and the *Divergent Belief No Cookiies* events.

We have deliberately included an intention, *Reading the Book*, without placing a book in the visible environment, introducing a confusing element in the scenario. This scenario can be seen in figure 5.1.

FIGURE 5.1: The cookie intention scenario

5.1.3 Keys Scenario

- Objects: a *Box* was placed on the *Table*, that partially occluded the sight of people approaching. A *Book* and a *Mug* where placed behind the *Box*, so that they could be seen from the sofa but not from approaching people.
- Intentions: *Taking the Mug*, *Taking the Keys*, *Reading the Book*.
- Tests and Events:
  - *No Clues*: *Max* approaches the *Table*.
  - *Contextual Clues*: *Max* approaches the *Table* in a hurry, while putting on a coat.
  - *Divergent Belief Max*: *Max* approaches the *Table* in a hurry, while putting on a coat.
- *Divergent Belief Event*: *Max* is sitting on the *Table*, drinking from the *Mug*, while having the *Keys* in his hands. His phone rings, so he drops the *Keys* and the *Mug* on the *Table*, behind the *Box*, and leaves the room. While *Max* is away, *Bob* comes and sits on the *Sofa*, reading a *Book*. When he sees the *Keys*, he takes them, places the *Book* on the *Table*, and leaves.

The *Divergent Belief Event* was shown to the users (and its simulation streamed to the system) between *Contextual Clues* and the *Divegent Belief Max* events. This scenario can be seen in figure 5.2.

FIGURE 5.2: The keys intention scenario

5.2 Experiment

5.2.1 User Study

We built an online user study, where we presented videos related to the tests and events of the two scenarios to users, who had to evaluate the likelihood of each intention of the scenario on a five-level Likert scale. The user study was conducted in three languages, with users living in two different countries[1]. We collected answers from 78 adults, performed an average, and converted them to percentile scores, in order to compare them with the system's predictions.

Looking at users' answers (figure 5.3), we can see that, in the absence of clues, people rated similarly the two intentions related to visible objects. Contextual clues had the highest influence on users' ratings. This is particularly visible in the *Contextual Clues* test of the *Keys Scenario*, where users chose as the most likely intention *Take Keys*, even if no keys were visible in the video. Divergent beliefs also influenced users decisions, but not as strongly as context. The strongest responses, over all, where given by the *Divergent Belief Max* test on *Keys Scenario*, which uses both divergent belief and contextual information.

5.2.2 System Implementation

In this subsection, we show how we implemented this experiment on our system. We will start by showing how we created a simulation of the scenarios. Then we will explain how we computed conditional dependencies between intentions and contexts, and finally we will discuss about the actual implementation of the IGs in the two scenarios.

[1] A version of this user study was provided at http://goo.gl/forms/YiuFHnF63c

5.2.2.1 Setting up the Simulation

We built a simulation to represent these two scenarios. For each scenario, we set the positions of the objects and computed a starting world state.

At the start of the scenario, the system does not have information about what each human knows, and so it will assume that they have the same belief model as itself. The system will be able to form a different belief model of each human only with the *Divergent Belief* event. This is a simplification. In the real world, we believe that humans have complex mechanisms to infer others' mental beliefs, based on many aspects. For example, if we would go to the home of a friend, we would probably infer that he has knowledge even about attributes that he can not immediately perceive (e.g. he will know that that he can find a glass in a cabinet in the kitchen even if he can not see it at the moment). Instead, if other people come at our home, we would infer that they do not know where objects are located until they see them. Our system is not able to replicate this idea, leading to our simplification.

For each test of our simulation, we streamed coordinates related to the body and hand positions of agents, following what was shown in the videos of the user study. For example, if in a test we showed Max moving from the hallway to the table, we streamed a trajectory of coordinates representing this path.

The inference process was performed by building different IGs for the scenarios. Each test had a different graph, related to its main agent.

5.2.2.2 Contextual Information

We considered three different Context Nodes for our simulation: *Hot Day*, true when the day is particularly warm; *Break Time*, true when the agents are taking a pause; *Time to Leave*, true when it is late in the day, and the humans usually leave work and return home.

As previously said, we have chosen to follow Liu et al. (2014) in order to learn the link between Context and Intention Nodes. We submitted a questionnaire to 15 users with 16 questions, representing every possible combination of the intentions and contexts used in our scenarios. In each question, the users had to rate how they perceived that a context influenced a particular intention. The rating was based on a five-level Likert scale, representing the range (*very negatively*, *very positively*). For example, one of the questions asked how a very warm day would influence the probability that the user would drink water.

The conditional dependency between intention I_i and context C_j was computed in the following way:

$$P(I_i|C_j = 1) = \frac{\sum_{k=1}^{5} ncj_k \times v_k}{n\_users} \tag{5.1}$$

where ncj_k is the number of persons who rated the influence of context C_j on intention I_i with the value k, v_k is the probability value that we associated to k (where 1=0.1, 2=0.3, 3=0.5, 4=0.7, 5=0.9) and $n\_users = 15$ is the number of users that participated in the test.

We set the values of Context Nodes manually in the tests, depending on the information shown in the videos. While we simulated this aspect, we will give some ideas on how the values of these Context Nodes could be inferred in a real scenario.

- *Hot Day*. The temperature of the day could be measured by a sensor or obtained from a meteo application. Also, in one of the tests Max is commenting that the day is very warm. A speech recognition software could capture this information.
- *Break Time* and *Time to Leave*. These information could be either hardcoded (e.g. in the company, workers have an hour of break from 13 to 14 and leave at 17:30) or learnt by observing the agents' activities.

5.2.2.3 Cookie Scenario

The starting world state for the cookie scenario is shown in table 5.1.

MUG isAt TABLE
BOTTLE isAt TABLE
COOKIEBOX isAt TABLE
COOKIEBOX capacity 1
COOKIES_SUPPLY isAt KITCHEN

TABLE 5.1: The starting world state for the cookie scenario.

With our perception capacities, we would not be able to detect if the cookie box is full or empty, and so we consider it as full (using the attribute *capacity*, which can have a value of 1 or 0) at the start of a test, and update its value using the *postconditions* of inferred human actions. We consider the box as empty when we infer that a human has taken a cookie from inside, and as full when we infer that a human has put a cookie in it.

In the *Cookie Scenario* the graph for the tests is constructed from the following nodes:

- Context Nodes: *Hot Day*, *Break Time*, *Time to Leave*
- Intention Nodes: *Fill Cookie Box*, *Eat Cookie*, *Drink Water*, *Read Book*.
- Action Nodes: *Move to Table*, *Move to Kitchen*.
- Observation Nodes: distance of the agent's body and hand to each action's associated *target*.

We introduce the *Fill Cookie Box* intention, not present in the human test, to allow the system to detect when Bob fills the *Cookie Box* during the *Divergent Belief Event*.

As previously said, we set Context Nodes to plausible values, that could be extracted by watching the videos. For the *Contextual Clues* test, we set the value of *Hot Day* to true (since Max is commenting about the temperature), and *Break Time* and *Time to Leave* to false (since no data in the video points to one of these contexts being true. Max and Bob seem to have taken a break from work before the other events are shown, in the Divergent Belief Event).

Divergent Belief Event, *Divergent Belief No Cookie*, and *Divergent Belief Cookie* were streamed sequentially to the system, which updated the agents' mental models and created new IGs accordingly. After the *Divergent Belief Event*, the robot inferred that Max has a divergent belief on the *Cookie Box*, because he does not know that Bob filled it. The mental states of the two humans, at this point, is shown in table 5.2.

Bob	Max
MUG isAt TABLE	MUG isAt TABLE
BOTTLE isAt TABLE	BOTTLE isAt TABLE
COOKIEBOX isAt TABLE	COOKIEBOX isAt TABLE
COOKIEBOX capacity 1	COOKIEBOX capacity 0
COOKIES_SUPPLY isAt KITCHEN	COOKIEBOX_SUPPLY isAt KITCHEN

TABLE 5.2: The table shows the belief models of Max and Bob after the Divergent Belief Event.

During the *Divergent Belief Event* several IGs need to be created with different Action and Observation Nodes, to follow the sequence of actions by the two agents. For example, when *Max* leaves the room, *Bob* has the possibility to execute the actions *Take Mug*, *Take Water Bottle*, *Open Cookie Box*, *Move to Kitchen Shelf* or *Leave Room*. Intention and Context nodes remains the same in all the IGs of the scenario.

5.2.2.4 Keys Scenario

The starting world state for the keys scenario is shown in table 5.3.

BOX isAt TABLE
BOOK isAt TABLE
MUG isAt TABLE

TABLE 5.3: The starting world state for the keys scenario.

The IG for this scenario is similar to the one for the *Cookie Scenario*, with the following differences.

- Context Nodes: *Hot Day*, *Break Time* and *Time to Leave*.
- Intention Nodes: *Drink Water*, *Take Keys*, *Read Book*.

Action Nodes and Observation Nodes are the same as the previous scenario, and follow the same ideas during the *Divergent Belief Event*. An example of IG used in the tests can be seen in figure 4.1, presented in chapter 4. For the *Contextual Clues* and *Divergent Belief* test, we set the *Time to Leave* context value to *true* (since Max is putting on a coat and seems in a hurry), and other Context Node values to *false*. Using the component described in the chapter 4 and these IGs the system was able to obtain predictions from the user actions.

5.3 Discussion

We performed TOST tests for each intention in the scenarios, comparing the humans' answers with the system's, for a total of 21 tests. We calculated p-values and performed our tests using a significance value $\alpha = 0.05$.

Analyzing the results of our equivalence tests, shown in figure 5.3, produces some interesting information.

- **The behavior of our system is often close to human capacities**. 16 tests out of 21 passed our requirements with very low p-value scores.
- **Context and Divergent Belief are necessary**. A system without these skills would only have been able to model properly the *No Clues* cases.
- **There are still some missing aspects in our system**.
  - When our system does not find a strategy to achieve a goal, the probability of the related intention is inferred as zero. Humans, instead, perform a different kind of reasoning. This is particularly evident in the *Contextual Clues* test of the *Keys Scenario*, where our system produced very different results than the users' answers. In this case, contextual information made users think that Max wanted to take the keys, even if no keys where actually present. Somehow, humans are able to infer the mental belief of Max, deducing that maybe he thinks that there are keys on the table. Our system is not able to replicate this reasoning.
  - In some situations it seems that humans perform complex temporal reasonings. In the *Divergent Belief Cookie* of the *Cookie Scenario*, the users' average answer for the *Eat Cookie* intention was quite high. We believe that users thought that, since Bob, in the previous videos, filled the box, probably he wants to eat a cookie. Contextual information about the warmth of the day where less strong in this case.

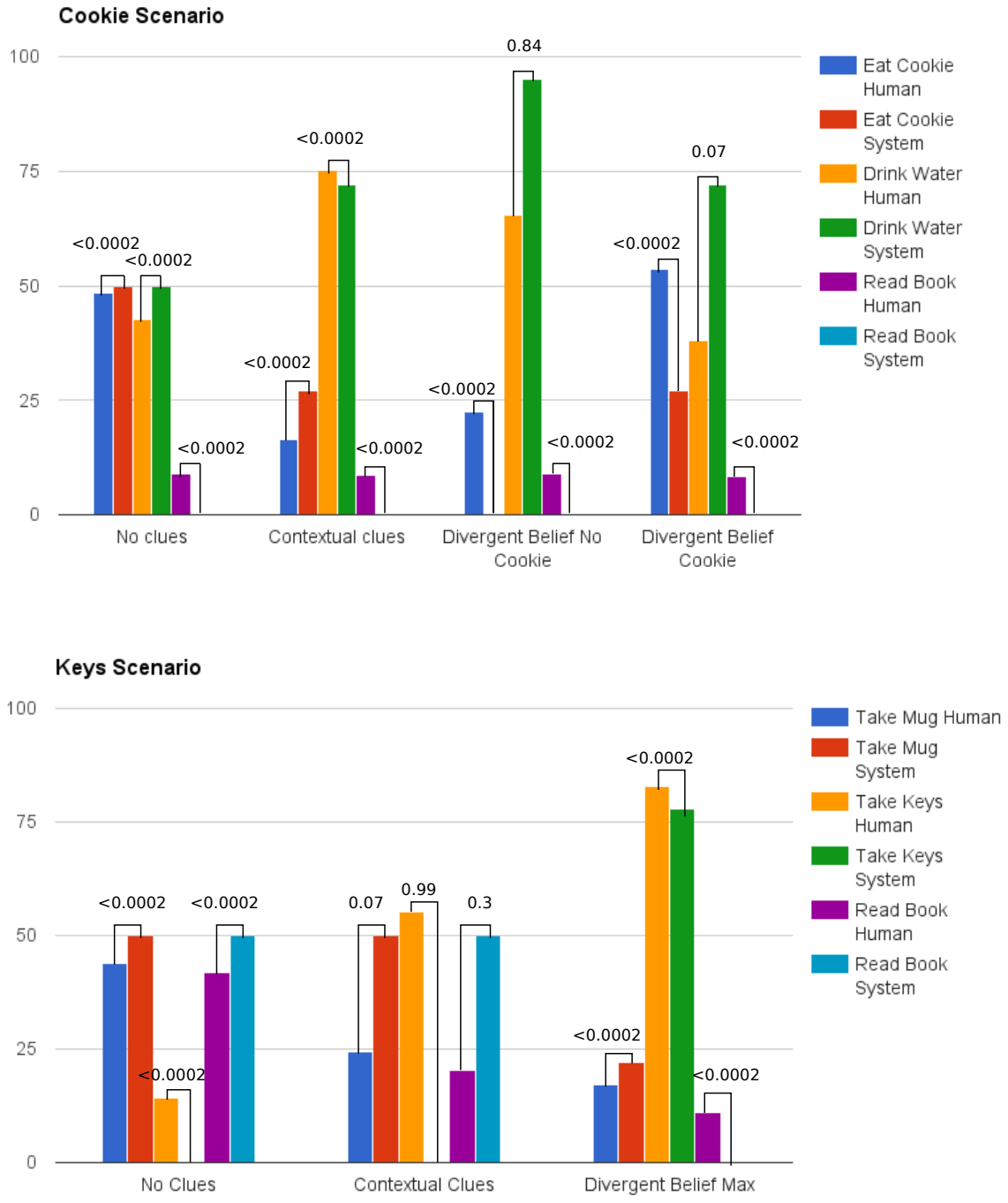


FIGURE 5.3: Experiment results. Results from the two scenarios are represented as graphs. Intentions, as estimated by the humans and the system, are represented by different colors, as shown in the legend of the graphs, with estimations of the same intention by the system or the human placed in adjacent positions. Each column represents the likelihood of an intention, expressed as a percentile score. P-values from the equivalence tests are shown, linking the estimation of an intention by the humans and by the system.

Part III

The Robot Coworker

Robots can be used to perform a large number of different operations. Some of these will be simple enough that the robot can just achieve its task by performing a prefixed sequence of elementary actions. In other cases, the robot might have to achieve complex goals, which require the ability to create plans and to adapt them to the current state of the world. When cooperating with other agents, the robot has to build a shared plan, which includes the actions that every agent need to perform, in order to coordinate and ensure the corrent achievement of the goal. We can imagine the following process:

- The system receives a new goal. This can be directly introduced by a human, or chosen after some kind of reasoning by the robot.
- One of the agents (the robot or the humans) proposes a plan to achieve the goal, and presents it to the other agents.
- The agents negotiate the plan. In some situations, one of the agents might not be able (or might not want) to perform a specific action, or sequence of actions. The agent can refuse the plan, proposing a correction or a completely different plan.
- The agents execute the plan. Each agent performs its part of the plan. In addition, agents may check the state of others to monitor the correct execution of their part of the plan or to cordinate with them.
- An agent might fail its part of the plan. If this happens the agents need to create a new plan to account for this failure.
- The process continues until the goal is achieved or it becomes unachievable (for example, a needed resource is no longer available).

When humans cooperate this process can be very quick. For simple tasks humans are able to coordinate without explicitly forming a plan, in particular if they are used to cooperating together. Other times, when there are unexpected problems during the execution of a plan, humans are able to quickly readapt their plan, without completely restarting this process. In order to cooperate in a natural way with humans, robots need to reproduce these mechanisms.

In this part we will introduce the mechanisms that we developed for the *robot coworker* problem. In this problem, the robot has to achieve a goal together with a human, cooperating to solve the task. We are particularly interested in tightly coupled problems, where the agent and the robot need to interact often in order to achieve a goal. The main aspects of our work are the management of plans, shown in chapter 6, and their execution, particularly the execution of joint actions, shown in chapter 7. Chapter 8 shows our experiments in the robot coworker problem. We also show, in chapter 9 a recent extension to this work, involving multi-agent probabilistic planning.

Chapter 6

Plan Management

In this chapter, we introduce the Plan Management capacity of our system. Section 6.1 introduces the subject, with a review of two systems able to manage cooperative plans. Section 6.2 shows the main aspects of this component. Our system is able to use different plan management modalities, as explained in section 6.3. Section 6.4 introduces HATP, a human-aware multi-agent planner interfaced with our system. Section 6.5 introduces our algorithm to manage plans. Finally, section 6.6 explains how we compare human activities with the current cooperative plan.

6.1 Introduction

When agents cooperate, they agree on a common plan to solve the task, implicitly or explicitly. We call this sort of plan a *shared plan*. Participants in a shared plan do not limit themselves to simply execute their parts of the plan, but, in fact, need to constantly monitor other participants, to synchronize their actions and adapt their plans to them. Let us an imagine a situation where the robot is executing a shared plan with Greg. If the robot execute its own actions without observing Greg we might encounter several situation where the two will put in danger, perhaps even preclude, the achievement of the goal. For example, Greg might be late in executing a crucial action, and the robot should wait for him. An even more dangerous example happens if Greg has starts following another plan, for personal choice or for other circumstances, and does not inform the robot about this change. If the robot does not notice that Greg has changed his strategy, and does not adapt its own plan, the two might not be able to achieve their goal.

We call *plan management* the process where the robot executes its own actions while coordinating with others and monitoring their activities.

Two examples of systems able to execute shared plans are Chaski (Shah et al., 2011) and Pike (Karpas et al., 2015, Levine and Williams, 2014).

Chaski is an executive system that enables the robot to anticipate and adapt to other agents actions. Chaski is based on human teamwork strategies, including ideas such as least moment commitment, frequent communications on the task status, and considering the consequences of the robot's choices on other agents. Chaski is able to execute plans in two different modalities: Equal Partners and Leader and Assistant. The system receives as input a shared plan, which includes the activities that need to be performed, the capacities of each agent, and the deadlines for these activities. Chaski produces a compact representation of all the possible schedulings of activities, based on this plan, which is used to take decisions on the fly during execution and to adapt to human choices. Results show that Chaski is able to reduce human's idle time in an equal partners scenario. A possible problem of this approach is that, if an agent completely deviates from the chosen plan, the system needs to create a new plan, which needs to be encoded again.

Pike is another executive, able to simultaneously recognize human plans and adapt to them. Pike receives as input a plan, represented as a Temporal Plan Network with Uncertainty (TPNU). Pike represents this plan by considering controllable choices (i.e. actions) for the robot and uncontrollable choices for the human. The idea is considering that these choices are not independent. In this way the system can infer what actions the human would rationally take to achieve his goal and what actions the robot should take to help him. The system receives a stream of human choices, which allows it to determine the robot's actions. Pike has been tested in simulation and with a real robot with good results, managing, on average, to take decisions with a low latency.

If the human does not follow the TPNU Pike will return a failure. The authors discuss integrating the system with a generative planner in the future to overcome this limitation.

In the next sections, we will present our approach to the problem.

6.2 Overview

6.2.1 Process Overview

Our Plan Management module receives, as input, a goal, which can be generated by the system or introduced by a human, using a tablet interface. After receiving a goal, the system will request a multi-agent plan to the task planner, including the actions of the robot and of other humans. We do not deal, in this module, with issues of task scheduling, which we leave to the task planner. While, in this chapter, we will concentrate on cooperative plans, our system is completely able to manage plans with only one agent, robot or human.

One of the goals of our system is flexibility; we consider important the possibility to interface with external components. Different planners can interface with our plan management algorithm, if they respect our interface.

For each scenario, we define a planning *domain*, which includes every entity and task that can be used in plans for that scenario. We consider that plans can be decomposed in a set of sub-parts, that we call *tasks*. In general tasks can be further decomposed in simpler sub-tasks, until reaching the most basic form of task of a domain, which we call *action*. Both actions and tasks follow the same representations, $(name, preconditions, target, postconditions)$, that we introduced in chapter 3. In chapter 4 we introduced our intention and action recognition module. As we said, this this module possesses a list of known actions. Every action in a planning domain that can be performed by humans need to be present in this list. In this way, the system will be able to monitor the execution of actions by humans, by using the mechanisms of the robot observes. This process will be explained in more details in section 6.6.

In some situations, to be more generic, we will use the word *task* to refer to both tasks and actions, since actions are actually tasks that can not be decomposed in the current planning model.

Each planner used by our system needs to represent its plans as a set of streams, one for each agent. A stream is a sequence of nodes, where each node corresponds to task assigned to be executed by the agent. Nodes can be connected by causal links, even among different streams, to ensure synchronization. A causal link $l = (t_1, t_2)$ indicates that task t_1 should be execute before task t_2. This ensures that all the preconditions to execute t_2 are fullfilled. Moreover, if t_1 and t_2 are executed by different agents, and if there is a shared resource connected to the task, the causal link indicates that the resource will be released by the agent only after t_1 is completed. An example of this data structure is shown in figure 6.1.

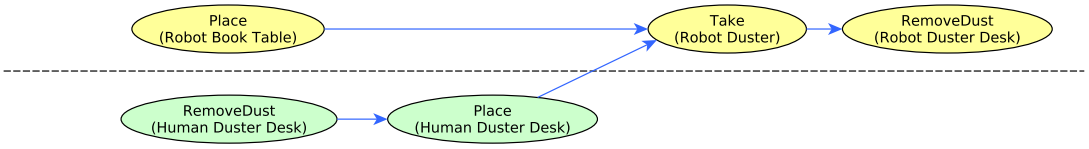


FIGURE 6.1: A plan stream structure. The upper stream represents the robot, and the lower one a human. Each node represents a task to be performed by the agent. For each task, the stream shows its name and, in parenthesis, the name of the agent and the parameters of the task. Causal links are represented as arrows between the nodes. Notice the causal link between the Place(Human Duster Desk) node in the human stream and the Take(Robot Duster) node in the robot stream. This link indicates that the robot should wait that the human places the duster before executing the action to take it, ensuring synchronization.

The system will manage each stream separately. By using the causal links, the robot is able to synchronize with humans. Joint actions (e.g. cooperative actions executed by two agents) will be treated by this module as actions of the robot, and be executed with a specific framework, explained in chapter 7.

Plans can be managed in three different modalities. The robot can be the leader, choosing a plan and guiding the human in its execution. The robot can also be an assistant, and follow the orders of the human. Finally, the robot and the human can be equal partners. This mechanisms will be explained in section 6.3.

6.2.2 Architecture

A number of modules implement these ideas, as shown in figure 6.2:

- Task Planner. Creates a shared plan for the involved agents. We consider the Task Planner as an external module. Our system has been integrated with the HATP planner, which we will introduce in section 6.4. We have also recently introduced a probabilistic multi-agent planner based on MDPs, which we will discuss in chapter 9.
- Plan Manager. Manages the current plan, interacting with the Task Execution layer to execute the robot's tasks and with the Situation Assessment layer to monitor humans' tasks.

After receiving a goal from the Goal Management layer, the Plan Manager module sends a request to the Task Planner to look for a suitable plan. After receiving a plan, this module will interact with the Task Execution and Situation Assessment layers to execute it.

Parts of this chapter were presented in Fiore et al. (2014), Lallement et al. (2014).

6.3 Plan Management Modalities

When acting together, agents sometimes do not have the same decision power, with one of them assuming the role of a leader. We represent this idea, in our system, by proposing three different modalities: *robot leader*, *human leader*, and *equal partners*. The robot is able to switch from one modality to another during the execution of a plan. For example, if the current modality is *robot leader* and rhe Robot receives a command from a user, it will switch to the *human leader* modality, after interrupting its current action.

6.3.1 Robot leader

In this modality the robot, after computing the plan, will present it to the user and start executing it. The robot will track the status of humans, informing them of which actions they should execute. This modality can be helpful when interacting with naive users or in tasks where the robot has a better knowledge of the domain or of the environment than the other agents.

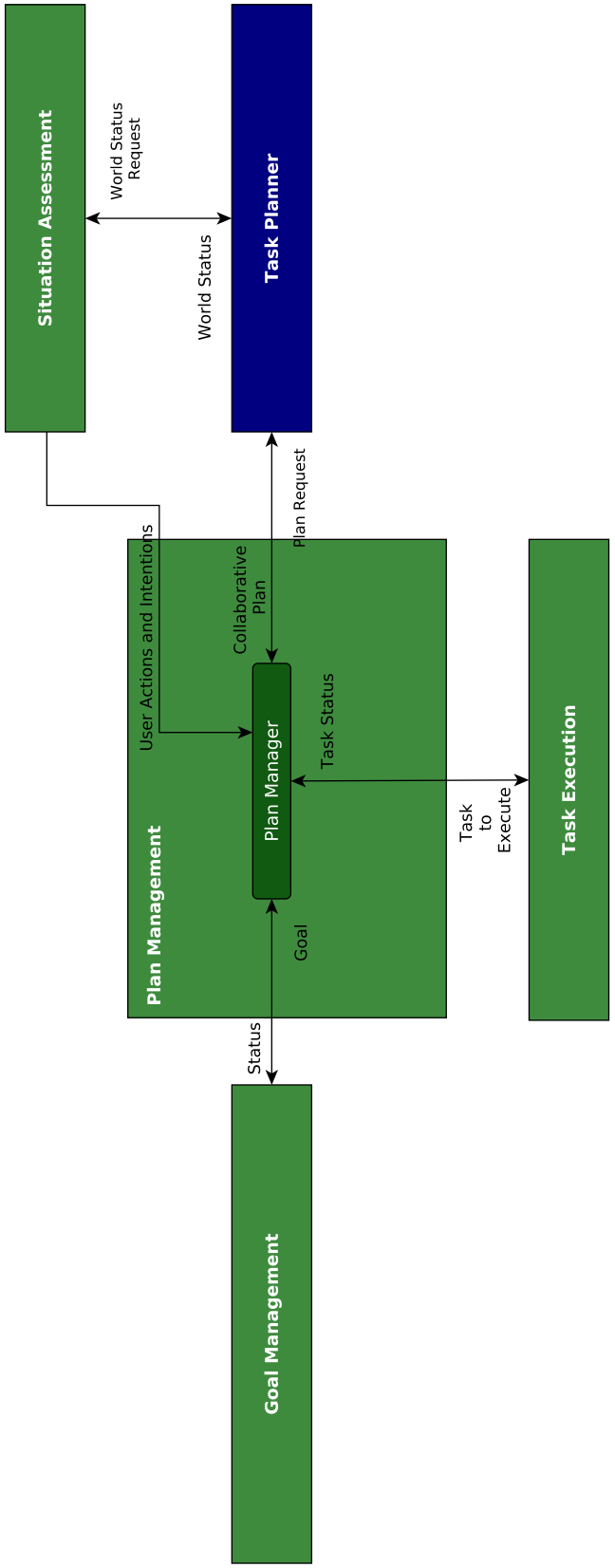


FIGURE 6.2: The architecture of the Plan Management layer. Light green rectangles represent modules, while dark green rectangles layers. The Task Planner is shown as a blue rectangle to indicate that it is a module external to the system. Arrows represent messages exchanged between components, with the label detailing the message.

We have developed two different algorithms to present the computed plan to the user. In the first, simpler case, the robot will verbalize sequentially all the actions, stating which agent should perform them.

For example, the plan presented in figure 6.1 would be explained in the following way: "I will place the book on the table. You will remove the dust from the desk and then place the duster on the desk. After that, I will take the duster and then I will remove the dust from the desk".

To generate this kind of verbalization we create a textual representation for each object and task, and then verbalize it with a text-to-speech application. Using simple rules, we follow the plan streams and verbalize each node. To create a more interesting and varied explanation, we use some connective words between nodes. We added some randomness in the algorithm. For example, the robot will randomly select words such as "At this point", "After that", or "Then" to introduce a task. Since this algorithm is not a focus of our work we will not present it in details.

We have also developed a more advanced algorithm to adapt this process to the expertise of a user in the tasks to execute. We will present this algorithm in chapter 10.

6.3.2 Human Leader

The human can also create plans, interacting with the robot by using a tablet application, shown in figure 6.3. Using the tablet, the human can choose a goal for the robot, but also pause or stop its actions. The human can choose, as goal, the execution of a simple action, but also of a more complex task, involving many actions.

In this modality the robot will simply observe the surroundings and wait for user inputs. User commands have a priority over the other two plan management modalities. If the robot receives a command from the application while it is in another modality, it will abandon its current plan, stopping its actions at a safe point, and then execute the user's command.

We feel that this interaction modality is important for two different reasons. First, some users will simply prefer to be in charge of the execution process, for a matter of personal preference or because they feel they have a deeper knowledge on how to realize the current task than the robot. We can picture, for example, industrial or medical scenarios, where the human is the leader and asks the robot to perform different tasks to help him, when needed.

A second use of this modality is in situations where the robot does not have a clear estimation of the users' intentions and goals. For example, in a domestic environment, a user could decide to order a robot to bring him a drink, a need that the robot can not always anticipate.

FIGURE 6.3: The human is giving a goal to the robot by using a tablet application.

6.3.3 Equal Partners

In the last presented operation modality the robot will try to help the human to complete a task. At the start of the scenario, the robot will observe the environment. After the user takes an action the robot will calculate a plan and try to help as it can, by performing actions related to that task and by giving helpful information to the user, for example to fill gaps in their knowledge. In this modality, the robot will not explain or negotiate the current plan and will not warn humans if their actions differ from the plan computed by the robot.

We feel that, particularly in non-critical tasks, where defining an accurate plan between the partners is not fundamental, this modality is a very natural way of interaction between different partners.

6.4 Human-Aware Task Planner

In this section we will briefly introduce HATP (Lallement et al., 2014), a planning framework that we used in many experiments with our system.

Computing a plan in complex environments can be very hard and time consuming. A useful approach to reduce the search space in planning is introducing the knowledge of an expert in the system, in order to guide the planner toward desirable states. An implementation of this idea is Hierarchic Task Network (HTN), where the domain expert specifies a hierarchical library of operations, called methods, when the operation is a node in the hierarchy, and actions, when the operation is a leaf. HATP extends HTN for human-robot interaction problems. Among the capacities of HATP we can find:

- Multi-Agent. HATP is able to include different agents in its domain, specifying which actions each one can execute. HATP is able to plan for different agents at the same time, humans and robots. The planner can also compute "joint actions", that involves more agents at the same time.

- Social Rules. The domain expert can introduce a set of rules, which represent desirable behaviors. Social rules help producing human-aware plans, that avoid behaviors that can be considered rude for humans. Using social rules, HATP can also balance in different ways the amount of effort of each agent in the plan. For example, we might choose that the human should have a minimum effort in the plan, or that the effort should be balanced between the human and the robot.
- Cost Driven. The domain expert can specify a cost for actions. Plan pruning allows to explore more efficiently the search space, discarding paths that are not promising.

Plans are represented as an HTN tree decomposition and as a set of streams, one per agent, which shows which actions each agent needs to perform. Causal links are introduced between streams to ensure synchronization.

6.5 Plan Management

Our plan management algorithm receives as input a plan composed by a set of streams, one for each agent. Each stream will be handled in parallel, using different threads of execution. We will now explain this algorithm, which is shown in figure 6.4.

- Each thread executes the part of the plan of an agent, composed by n different tasks.
- For each task, for every causal link (t_i, t_n), where t_n is the current task, and t_i is another task, the plan manager waits until t_i is completed, or there is an error. This process is handled in the *waitCausalLinks* procedure.
- When all the causal links have been satisfied, the execute\monitor procedure is called, depending if the thread is managing the robot or a human. The *executeTask* operation interacts with the Task Execution layer to complete the task, while the MonitorTask operation with the Situation Assessment layer.
- If these procedures succed, the Plan Manager switches to the next task, otherwise it returns a failure.
- The process is continued until there is a failure or the plan for the current agent is completed.

Failures in the algorithm lead to a replan request. If a new plan is found, the algorithm starts again, otherwhise the goal is considered as failed. Depending on the modality, the robot will also give different verbal information to the human.

If the current management modality is *robot leader*, the robot will warn the human the he executed a wrong action. If the replan is successfull, the robot will explain the new plan to the human.

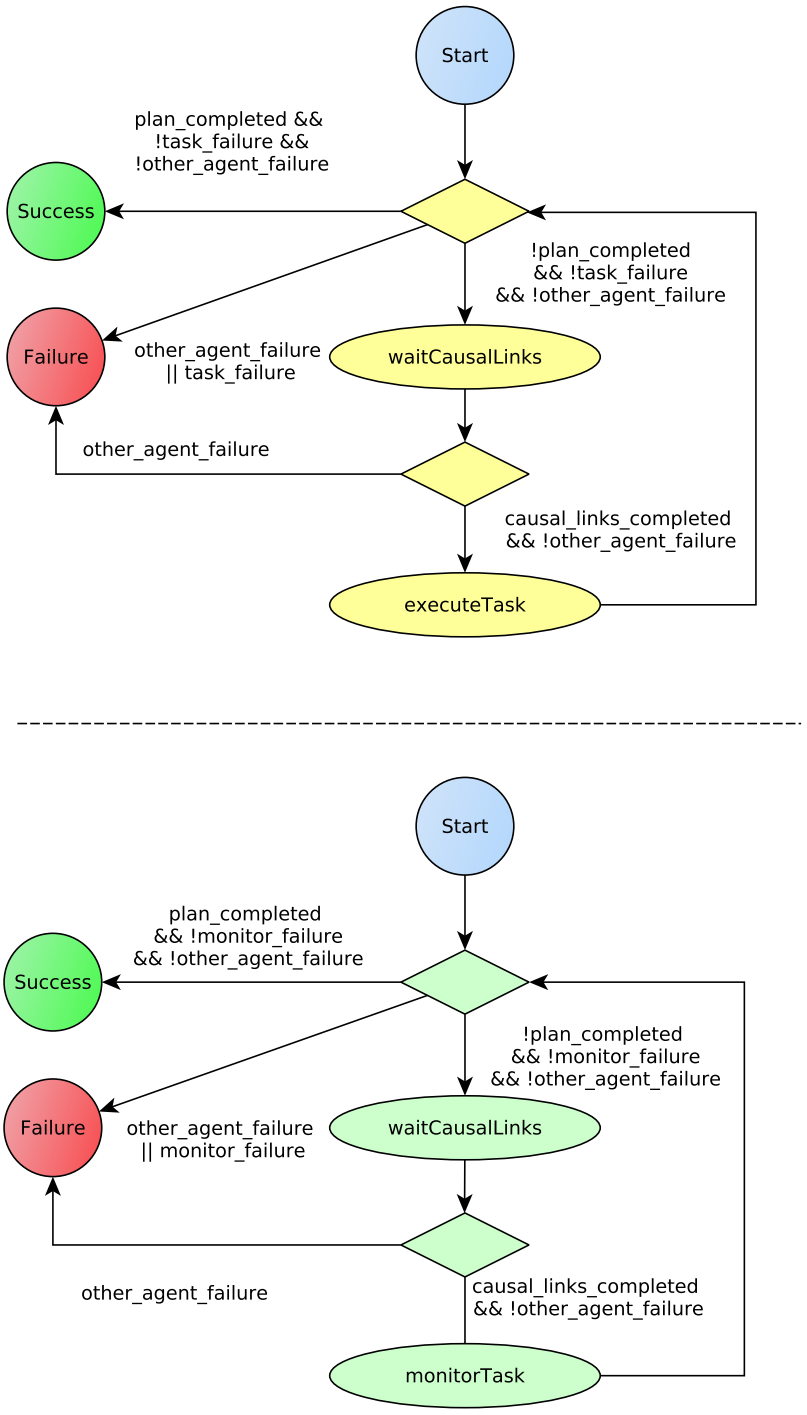


FIGURE 6.4: The plan management algorithm. The algorithm is composed by different threads, one for each agent. In this instance, the upper lane represents the robot's management thread, while the bottom a human's management thread. Elliptic nodes represent operations. Diamond nodes, representing divergences in the algorithm, where added to the graph only when they could simplify the understanding of the algorithm. Arrows imply a transition between nodes, with the label of the arrow representing the condition of the transition, when present. The absence of a label implies that a transition is always applied. The blue circle node, called "start", represents the start of the algorithm. The green and red circle nodes, instead, represent the success or failure of the algorithm.

When the human is the leader, the robot is following orders. The current order might be to execute a task which still requires coordination, and the robot might expect the human to follow a particular strategy. If the human deviates from this strategy the robot will replan, but not issue any warning.

A similar approach is used in the *equal partners* modality. In this case, the robot will adapt to the deviations in the humans' actions, but the replanning process should be almost hidden from the human, in order to be more natural. The robot will replan and start managing the new plan without informing the human.

6.6 Task Monitoring

During the execution of a plan, the robot will monitor other human partners. In general, having a shared plan, the robot knows what is the next task that the human should perform, and can monitor if it is accomplished. Task monitoring poses a number of different issues:

- Understanding when the next expected action has been performed. In some situations the robot will monitor the execution of a specific action. In this event, it needs to understand when the action has been completed.
- Understanding when the next expected task has been performed. In some situations, the robot wants to give a human cooperator the freedom to perform a task (that is, a complex operation that can be composed by a sequence of actions) as he sees fit. This is a more complex problem than monitoring a specific action, since the robot needs to reason on the results of sequences of actions.
- Evaluating the human engagement in the current task. The robot needs to understand if the human is trying to accomplish its current task, if he momentarily interrupted it, or if he abandoned it.

At the moment, we have implemented and integrated in our system only monitoring of actions. We will show how we include this idea, and then illustrate in the chapter 9 how we could extend our system to monitor tasks and evaluate the human engagement in the current task.

6.6.1 Monitoring Actions

At all time, as previously explained in chapter 4, the robot constantly observes the environment, monitoring which actions are executed. As previously said, each action that can be executed by a human needs to have the same representation in the Situation Assessment and Plan Management layers. When the Plan Manager needs to monitor an action a of a human stream, its *preconditions* will be satisfied (since, if we are monitoring a, all of its causal links have been already satisfied). This means that a will be present in the IG for the human, which will be monitoring its execution (chapter 4 gives more details about this mechanism).

As we said in section 4.2.5, the single-agent MDP mechanism that we introduced is not sufficient to monitor a joint goal, and so the IG for the human, in the coworker scenario, only includes Actions and Observation nodes.

The Plan Manager will wait for Situation Assessement to infer that an action has been performed by the human. If the action is different from a, it will return an error, prompting a replan. Otherwise, the plan management will continue with the next action in the stream, if there are any.

For example, let us imagine a scenario where Greg is near a table, with a *bottle* and a *book* on top. Let us say that, at the moment, Greg can execute three actions: *take book*, *take bottle*, and *move kitchen*. The current IG for Greg will include these actions, as well as observations to infer their execution. In this example, Greg's plan is represented by a stream of three sequential actions: *move to table, take bottle, move kitchen*. Greg has just completed the action *move to table*, and should now execute the action *take bottle*, if he follows this stream. If the system infers that Greg executes *take book* or *move kitchen*, the plan manager will replan, since Greg executed a different action that the one that was expected. If Greg executes *take book*, the system will start monitoring the next action in the stream, which is *move kitchen*.

6.6.2 Monitoring and Unseen Actions

Often, in cooperative tasks, agents will operate in different locations, and so they can not observe each other actions all the time. Perhaps one of the agents is cooking in the kitchen, while the robot is preparing the table for dinner. While we do not deal, in this work, with these issues, there are several studies on plan recognition in partially observable environments, like Geib and Goldman (2005).

Chapter 7

Task Execution

This chapter shows how our system is able to execute tasks, and particularly joint actions. Section 7.1 introduces the subject. Section 7.2 describes our approach and the components that implement it. Section 7.3 shows the main aspects of the Task Executor module, while section 7.4 introduces the framework we use to execute human-robot joint actions, the Collaborative Planners. Finally, section 7.5 shows an example of a Collaborative Planner for human-robot handovers.

7.1 Introduction

Acting in a human-crowded environment is a difficult problem. Even when acting independently, the robot needs: to ensure human safety, by taking others into account when planning and stopping if its actions could bring harm to them; to perform legible movements, so that its actions can be understood by humans (studied, for example, in Dragan et al. (2013)); and to be robust, trying to complete its tasks even in front of unexpected conditions.

These issues show us that humans should not be treated as simple obstacles by the robot, but need specialized reasoning and execution algorithms.

When performing a cooperative action with the human the robot needs to continuously monitor its partner, checking if he is involved in the task, stopping to wait for him, adapting its movements, and eventually abandoning the task, if the partner leaves. For example, if the robot is giving an object to a human, it will have to choose a position for its arm where the human can easily reach the object, change this position if the human is moving, and abandon the task if the human leaves the area.

Bussy et al. (2012) studied how to execute a transportation scenario jointly with a human partner, but the work is based more on haptic and control issues than actual reasoning, an area of the problem not deeply investigated.

7.2 Overview

7.2.1 Process Overview

We have built a module to execute tasks, represented, as in the previous chapter, as $(name, preconditions, target, postconditions)$, in a human-aware way. The focus of this module is not the planning or execution of the robot's motions, which are handled in external components, but the execution of a *task*, from start to finish. This problem presents several issues and steps:

- The robot needs to monitor if the task is actually achievable. Even though, if the task is part of a plan, our plan management algorithm, presented in section 6.5, ensures that its preconditions are satisfied when it is executed, the task could still not be achievable. This problem may arise for different reasons. First, the robot is planning by using its knowledge of the state of the world, which may be faulty. For example, the robot might believe, when planning, that an object is on a table, and produce a plan where it will take it. If the location of the object is different, because it was moved without the robot knowing, the task might not be completed. Second, external events can happen that make a task unachievable, and the robot might not be able to detect or link these events to the task. For example, if a gush of wind throws the object to the floor, the robot might not immediatly understand that this will prevent it, in the future, to take the object.

- The system needs to interact with motion planners and executors to produce the range of movements needed by the robot to achieve its task.

- The robot needs to constantly monitor its environment in order to ensure that the task is executed in a safe way. In particular, humans may move, while the robot is acting, in positions where they could be endanger by the robot's motions. To ensure human's safety, the robot needs to detect these situations and stop or pause its motions.

- After a task is executed, the system needs to infer its effects, in order to update the knowledge on the state of the world. In general, a part of the postconditions of the actions could be observed by the robot. For example, if the robot has placed an object on a table, the robot could observe, and not only infer, its location. Since our perception can be faulty, if we do not integrate observation with inference, we might encounter situations where the robot is not able to observe the effects of its actions. For example, the robot has placed an object on the table, but the light conditions or the orientation of the object do not allow its recognition by the perception algorithms. To avoid this problem, the robot will update the position of the object using inference, in its mental model, and then refine it through observation, if it is able to see it.

We are particularly interested in the execution of human-robot joint actions. We have defined, in section 2.2, a joint action as a cooperative action between the two agents. As previously said, this kind of action requires additional mechanisms to allow the robot to cooperate with the human in a natural and efficient way. We developed a specific framework, which we called Collaborative Planners, to deal with this issue, which we present in section 7.4.

7.2.2 Architecture

These ideas are represented in the following modules, as shown in figure 7.1.

- Task Executor. This module executes the robot's task in a robust, human-aware, and flexible way.
- Collaborative Planners. This set of planners are used to execute human-robot joint actions, allowing the robot to adapt its actions to the collaborators.
- Motion Planners and Executors. These planners, treated as external modules, are in charge of choosing trajectories for the robot, taking into account the environment and the present agents. We will not discuss this component, as it is outside the boundaries of this work. More details can be found in Mainprice et al. (2011), Pandey and Alami (2010), Sisbot et al. (2008).

We developed this layer with flexibility in mind. We can easily expand the system, by adding new actions, or switching motion planners and executors, without having an impact on the rest of the architecture.

Parts of this chapter where presented in Fiore et al. (2014).

7.3 Task Executor

The Task Executor handles task requests from the Plan Management layer. Following the ideas presented in section 7.2, the execution of a task is handled in several steps:

- Check Preconditions. The system can check if the preconditions of the task are valid by sending a query to the Situation Assessment layer. If the preconditions are not valid, the module returns an error.
- Execute Task. The task is executed, by communicating with the Motion Planners and Executors and, if the current task is a joint action, with the Collaborative Planners.
- Set Postconditions. The postconditions of the task are set in the Situation Assessment layer.

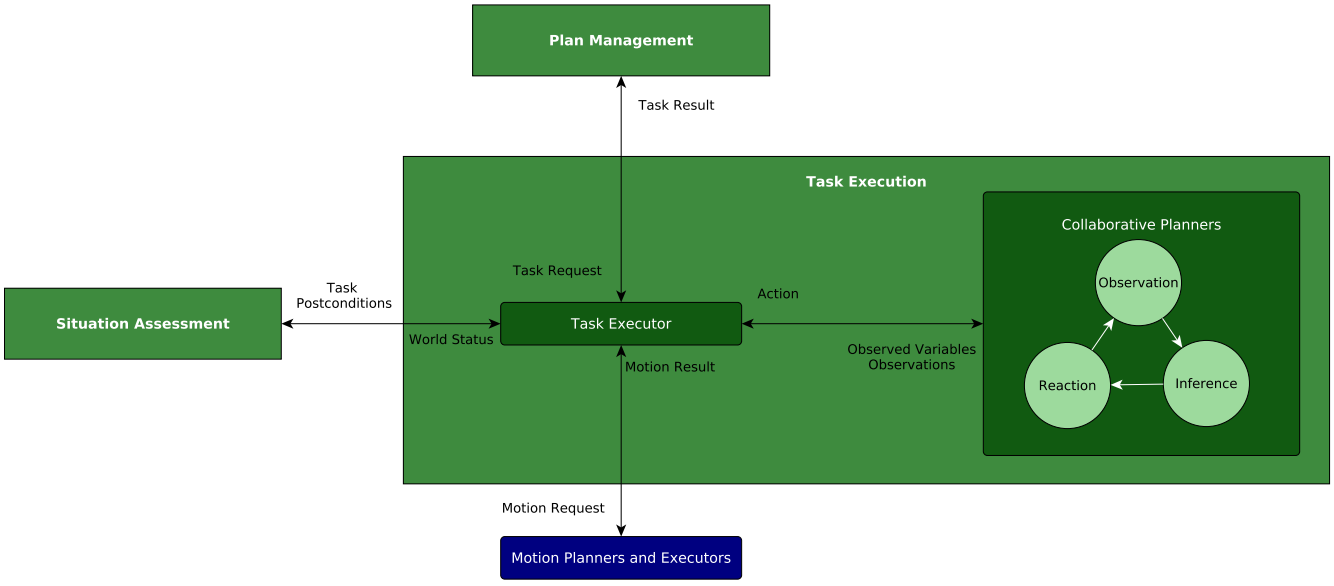


FIGURE 7.1: The architecture of the Task Execution layer. Dark green rectangles represent modules, while light green rectangles layers. Blue rectangles represent external modules. Arrows represent messages exchanged between components, with the label detailing the message. The white ellipses, and their arrows, represent the process of the Collaborative Planners, explained in Section 7.4. While in this image we present the Collaborative Planners as a single unit, the system actually has a set of planners, and will choose, during the execution of a joint action, which one to use.

The execution of a task can fail, and in that case the world status is updated consequently. For example, if the robot is trying to pick an object, and the motion planner does not manage to compute a trajectory to reach it, the object will be considered as *not reachable* by the robot in the current world status. When the Plan Management replans, the system could look for a plan where the robot changes position and tries again the pick, or even asks another agent to give him the object.

Tasks can be paused and resumed. We set a number of safety rules, where the robot will pause executing a motion if a human, or one of his body parts, is too close to the robot's grippers or arms.

The robot is also able to show some social clues during an action, for example by moving its head toward the tasks's *target*.

7.4 Collaborative Planners

When executing joint actions with humans, the robot needs to continuously observe its partner and react appropriately. The robot's action should be based on different aspects. First of all, the robot should act differently depending on the current status and level of advancement of the task. Second, the robot should consider the status of the world, in order to take appropriate actions. Then, as we said, the robot should observe its partner's behavior, which can give different information. Of course, the robot should coordinate with the human's movements. For example, if the two agents are performing an handover,

and the human extends his hand, the robot should extend its arm to bring the object in a reachable position.

Observing the human's activities can give us more subtle information, that represent how much he is engaged in the task. For example, in the case of the handover, if the human is oriented toward the robot and extending his hand, we can assume that he is currently engaged and cooperating. If, instead, the human is looking in another direction, or moving away, we can infer that he is currently doing something else, and perhaps has even abandoned the task.

To reason on all these variables and produce an appropriate reaction, we introduced a special framework to manage cooperative actions, called the Collaborative Planners, based on hierarchical Mixed Observability Markov Decision Processes (MOMDP). A MOMDP can be modeled as a tuple $(S_h, S_o, A, T, R, \Omega, O, \gamma)$, where S_h is the hidden state, S_o is the observed state, A is the set of actions, T is the transition function, R is the reward function, Ω is the observation function, O is the set of observations, and γ is a discount factor. More details about Markov Models are provided in appendix A.

Using n MMODP we can model observed variables, like the task and world status, and hidden variables, like the engagement level of the human, which we can infer from observations. Using a hierarchy of models, as explained in Pineau et al. (2001), we can represent complex scenarios and tasks with smaller, simpler models, that can be solved more easily, and expand them by adding more actions and complex behaviors as we see fit.

For each joint actions, we will introduce a new Collaborative Planner. When the system is executing a joint action between the robot and a human, the Task Executor will contact the related Collaborative Planner. The Task Executor will send requests containing the observations and observed variables, used to update the MMODPs, and the planner will return a sub-action to execute. To maintain this model generic the Collaborative Planners will output high-level actions, which the Task Executor will adapt to the current situation. This process will continue until the joint action is achieved, the planner chooses to abandon the task, or there is a failure.

We provide a *basic framework* that we use to implement Collaborative Planners, that can be adapted to the specific task. While it is not obligatory to implement a Collaborative Planner following these ideas, we believe that they offer a good basis for cooperative tasks:

- $S_h = \{human\_commitment\}$
  - $values(human\_commitment) = \{engaged, not\_engaged, not\_interested\}$.

  The human commitment to the task can be modeled as a hidden variable. A user can be engaged in the task, meaning that he is actively participating; not engaged, meaning that he is currently not participating, but we expect that he will resume the task (imagine, for example, a user receiving

a phone call while executing a joint task); or not interested, meaning that he has abandoned the task.

Modeling the human commitment is very important, since the robot should choose its action based on this variable, and of course on other task related variables. For example, if the user is not currently engaged in the task, the robot might choose to wait for him, or ask him if he has a problem. If he has abandoned the task, the robot might look for another plan, or ask the help of another human. If he is committed, the robot could proceed to accomplish the task, as long as the world state allows it.

- $S_o : \{task\_advancement\}$.
  - $values(task\_advancement) = \{not\_started, started, completed\}$.

  The status of advancement of the task can provide useful information to help the robot choose its actions. A simple way to represent it is considering just an initial state, where the agents have decided to execute a cooperative task, but have not started executing it yet; a middle state, where the agents are working to complete the task; and a final state, where the task is completed. Different tasks could further refine this variable.

- $A : \{continue, wait, abandon, engage\}$.

  When building an action set, the developer should consider how much control the MOMDP should have over the robot's actions. In our implementations, we chose to plan generic high level actions, that will be adapted by the Task Executor. This strategy allows us to simplify the state space of the MOMDP. In a basic Collaborative Planner, the robot is able to: continue its task, meaning that it will take the next logical action depending on the current state, maintained by the Task Executor; wait for the user, if the robot detects that he is momentarily not engaged in the task; abandon, if the user is not interested anymore in executing the task; engage, to communicate with the user, for example by giving him a warning.

- T, R and O are, of course, completely dependent on the joint task.

- Ω is dependent on the current application. Some typical observations are related to spatial relationships between the user and the robot (e.g. the user is more likely to be interested in the task if he is close to the robot and oriented toward him), or particular poses of the user (e.g. the user is ready to receive an object if he has extended its arm toward the robot).

Our idea is similar to the work of Ferrari and Mouaddib (2015). The authors propose a framework based on hierarchical POMDPs for cooperative tasks, with a three layer structure. This framework was applied to an escort application, where the robot had to guide a user to his location in a human-aware way,

adapting to his behaviors. In this application, the commitment of the user is estimated by reasoning on how much the human is focused on the robot, and on their distance. We implemented a similar application, for a robot guide, using our Collaborative Planners, which we will show in chapter **??**. The main difference between our framework and the one developed by the authors is that they chose to give a much stronger decision power to the POMDP, while in our system planning is actually split among several units, and the Task Execution module has a big role in adapting the actions of the Collaborative Planners. We also do note use a precise hierarchical framework, leaving to the single Collaborative Planners the choice on how many layers of task to use.

We will show, in the next section, an example of collaborative planner which develops this basic framework.

7.5 Collaborative Planner for Handover

The handover collaborative planner has the goal to handle both human-robot and robot-human handovers in a human-aware way. We will show parts of this model now, avoiding the observation and transition function, as they are extensive and do not provide particular help in understanding this example.

- S_h : $\{human\_commitment\}$. The hidden variable, and its values, follow the basic framework presented in the previous section.
- S_o : $\{task\_advancement\, timer, human\_distance\}$.
  - $value(task\_advancement) = \{not\_started, started, touching, completed\}$.
  - $values(timer) = \{not\_expired, expired\}$.
  - $values(human\_distance) = \{close, far, out\_of\_range\}$.

  The task advancement is represented in a similar way as the basic framework. We add the *touching* value to this variable, representing the fact that the gripper of the robot has detected some pressure (e.g. the human has placed an object in it, if the handover is human-robot, or he has grabbed the object kept in the gripper, if the handover is robot-human). We add a timer variable, which is started and controller by the Task Executor when the robot is waiting for the user. When the timer expires in the Task Executor, this variable will be set accordingly, and the robot can perform specific actions (e.g. ask the user if he still wants to perform the task). We also consider the human distance from the robot, classyfing it as *close* to the robot, far from the robot, and out_of_range if the human is not visible.
- A : $\{continue, wait, abandon, engage\}$.

The actions follow the basic framework. The *continue* action is chosen when the user is *engaged*. The Task Executor will react differently to this action, dependingly on the current *task_advancement*. If $task\_advancement \neq touching$ then the robot will extend (or keep extended) the arm toward the user. If $task\_advancement = touching$ the robot will release the object, or grab it from the human's hand, depending on the kind of handover.

The *wait* action is chosen when the user is *not_engaged*. In this case the robot will retract its hand to an intermediate position. The Task Executor will start a timer.

The *abandon* action is chosen when the user is *not_interested* or the task is *completed*.

Finally, the *engage* action is chosen when the *timer* is *expired*. In this case the robot will ask the human if he still wants to perform the handover.

- $O : \{human\_arm, human\_orientation, human\_distance\_variation\}$.
  - $values(human\_arm) = \{approaching, retracting, still, unknown\}$.
  - $values(human\_orientation) = \{toward\_robot, other, unknown\}$.
  - $values(human\_distance\_variation) = \{decreasing, increasing, still, unknown\}$.

  We present several human observations, used to estimate his engagement. In general, we consider that the human is more likely to be *engaged* when he is approaching the robot, he is turned toward it, and his arm is extending. If the human is turned away, or even leaving, it is more likely that he is currently *not_engaged* or even *not_interested*. Finally, if the human is out_of_range, his observations will be *unknown*, since the robot does not know his location.

- R: the robot is rewarded for completing the task, performing the handover, and for executing desirable behaviors, like waiting for a user if we detect that he is not engaged, and abandoning the task if he is no longer interested in the joint action.

An example of handover using this Collaborative Planner is shown in figure 7.2.

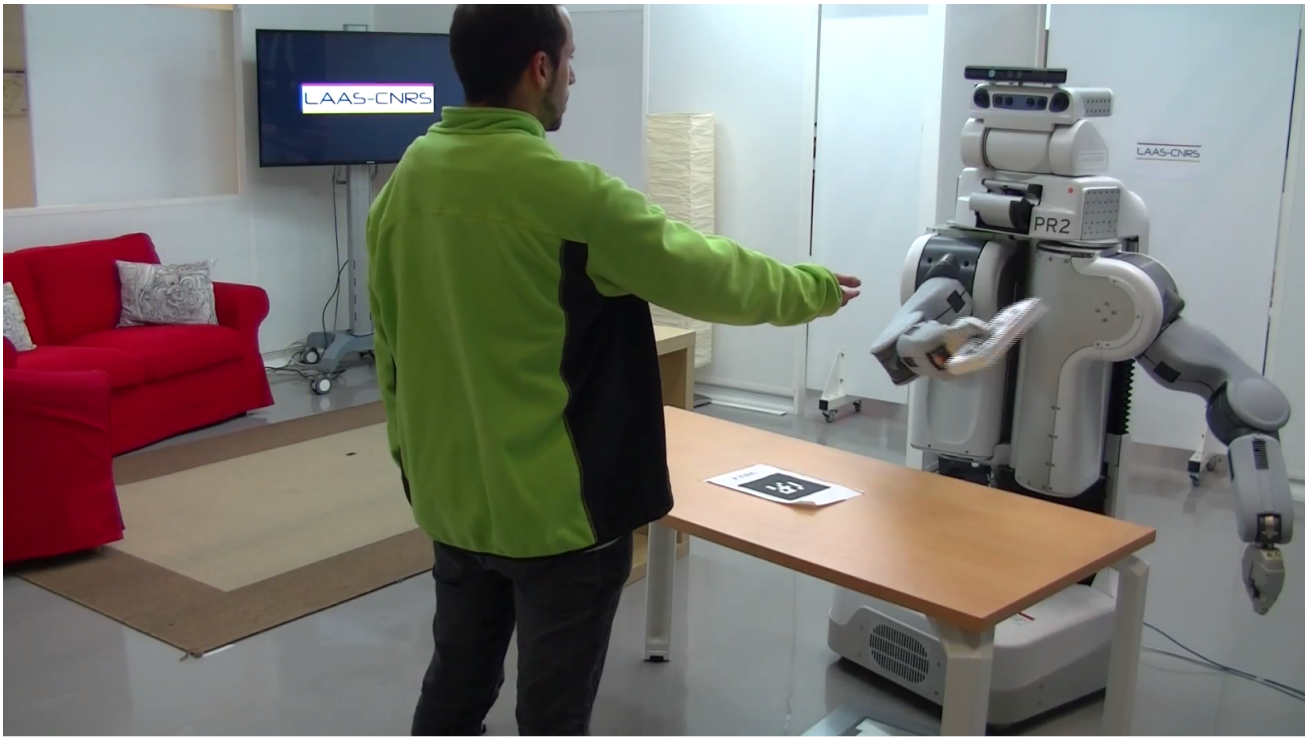


FIGURE 7.2: A robot-human handover

Chapter 8

Experiments and Results

This chapter presents an experiment used to evaluate the robot coworker problem. Section 8.1 introduces our scenario. Section 8.2 shows our experiments. Finally, section 8.3 presents and discusses our results.

8.1 Scenario Description

In this section we show experiments created to validate our system in a domestic scenario, where the robot can help a human partner to achieve a joint task. This experiment was previously presented in Fiore et al. (2014).

We present a scenario where the robot, a PR2 by Willow Garage, and a human have a joint goal: cleaning a set of furniture. The environment is composed by two different furniture, a $TABLE$ and a $SHELF$, and three tapes, $TAPE1$, $TAPE2$ and $TAPE3$. The goal of the agents is throwing each tape in a BIN. We will present different examples, where the items will be placed differently, depending on our needs. The scenario is shown in figure 8.1.

In this scenario, human detection and handling was done using a depth camera, mounted on the ceiling. the scenario uses both the capacities of the robot coworker, described in this part, and the belief management and action inference capacities of the robot observer, shown in part II.

8.2 Experiments

We will describe different experiments conducted in our laboratory.

- **Equal Partners**. In this scenario (figure 8.2) the user is asked to clean the table, without being explained what is the shared plan that will be used. The user is just informed that the robot will try to help as it can to perform the task. The robot is already aware about the joint goal, but at

FIGURE 8.1: The robot coworker scenario example, with a PR2 robot and a user.

the start of the scenario limits itself to monitor its partner. The user moves to the table and takes the *TAPE2*. At this point, the robot infers that the user has completed an action.

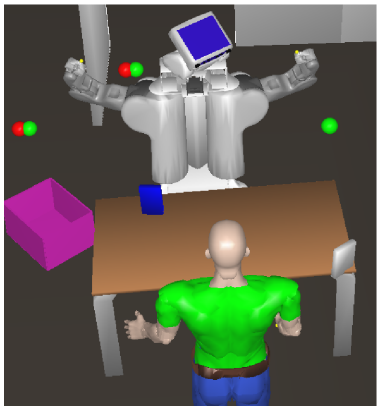
The robot creates a plan and executes its part of it while monitoring the human, who executes his part without deviating from the plan calculated by the robot.

- **Modality switch and user plans**. In this scenario (figure 8.3) the robot is the only agent able to reach both tapes, but it can not reach the bin, which can instead be reached by the human. We tested this scenario in two different runs. In the first run, the current plan management modality is *robot leader*. After exploring the environment, the robot produces a plan and starts its execution.

  While the robot is taking the *TAPE1* the human moves to take the *TAPE2*. This deviates from the robot's plan, so it switches to the *equal partners* modality, communicating the change to the user. The user throws the *TAPE2* in the *BIN* while the robot takes the *TAPE1* and handles it to the user. The user takes the *TAPE1* and throws it in the *BIN*, completing the task.

  In the second run the current modality is *human leader*. The user is asked to clean the table as he wishes. The user asks the robot to take each tape and give it to him, throwing them in the trashbin.

FIGURE 8.2: Robot adapts. This figure shows the robot's representation of the scenario. The white tape is the *TAPE2*, while the blue one is the *TAPE1*. The spheres represent the agents' rechabilities, with red spheres representing robot reachabilities and green spheres human reachabilities. In this case only the human can reach the *TAPE2* while both agents can reach the *TAPE1* and the *BIN*. After the human takes the *TAPE2* the robot produces a plan where the human must throw the tape in the bin while the robot can take the *TAPE1* and throw it in the bin.



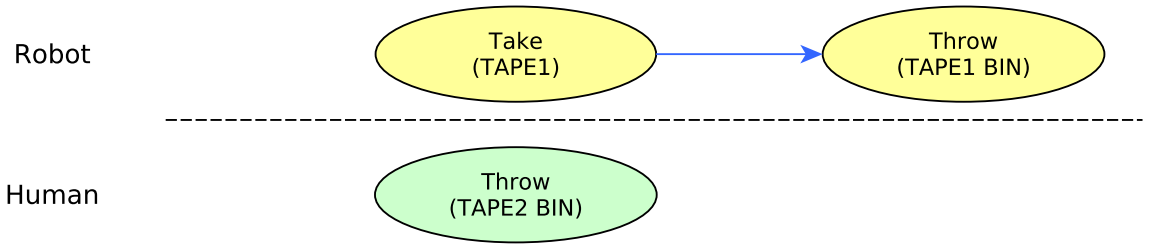
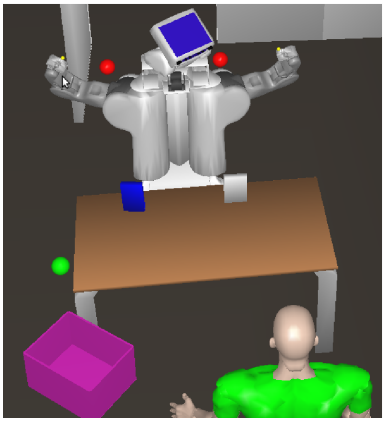


FIGURE 8.3: Modality switch and user plans. Another configuration of the environment, where the robot can reach the two tapes and the human can reach the bin. The robot generates an initial plan from this situation. The block surrounding the Give and Receive actions means that they are considered a single joint action.



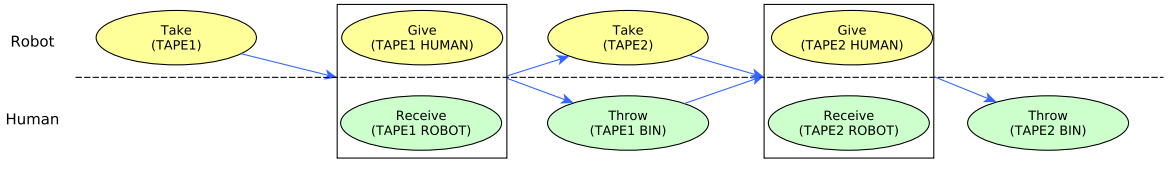
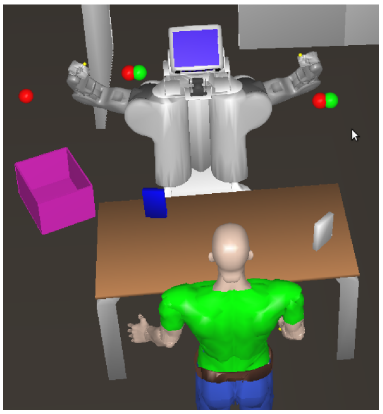
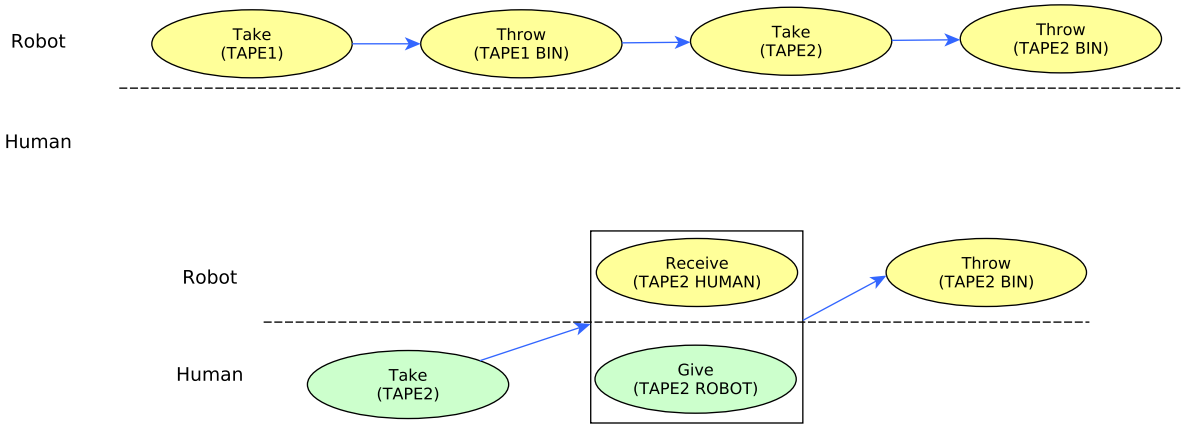


FIGURE 8.4: Replanning after failed action. Here we can see a first plan, produced at the start of the scenario, and a second, produced after the robot fails to take the *TAPE2*.



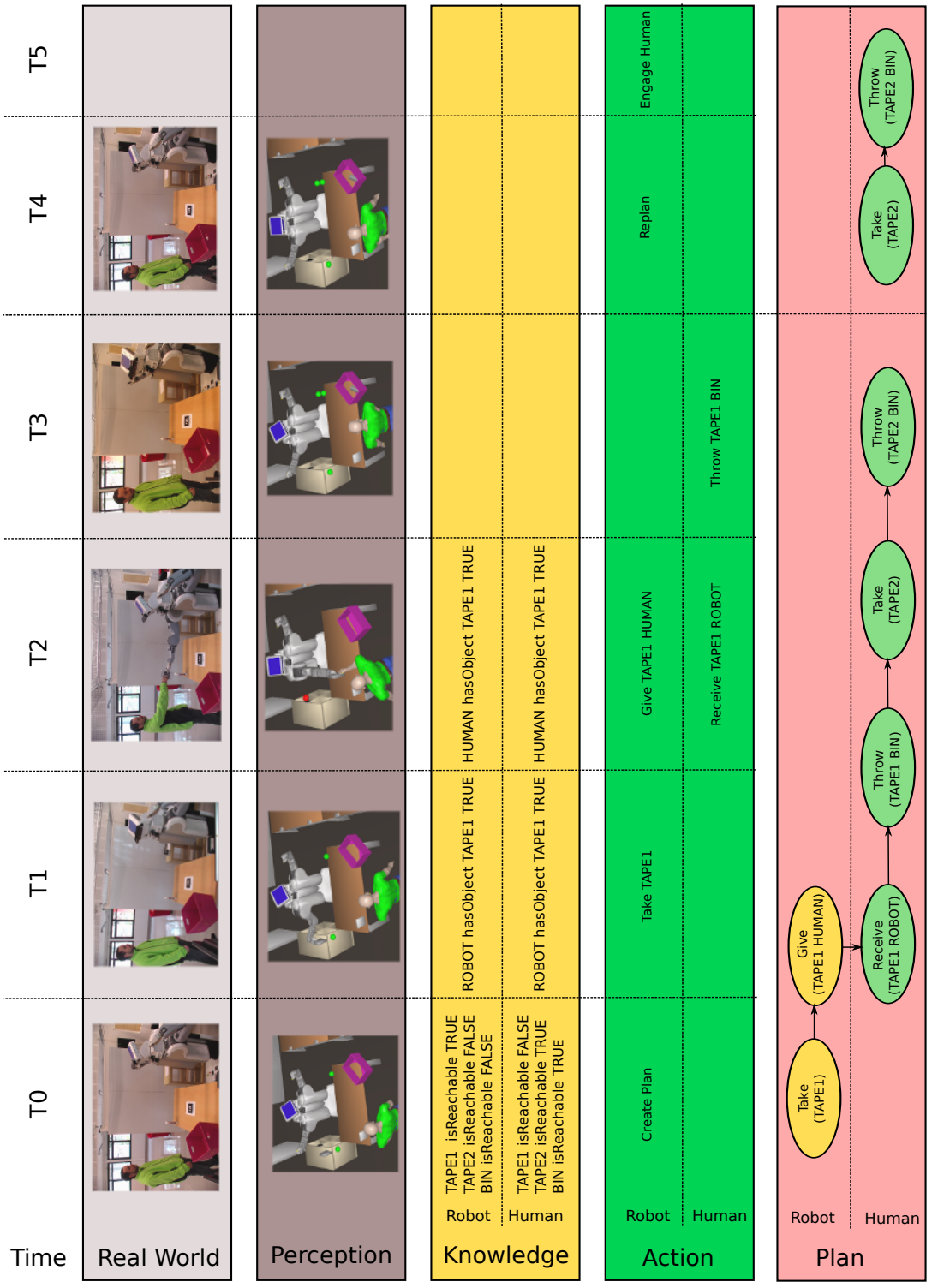


- **Replanning after failed action**. In this scenario (figure 8.4) the robot is the only agent able to reach the bin, while both agents can reach the two tapes. The robot is in the *robot leader* modality and, after examining the environment, produces a plan.

  After taking and throwing the *TAPE1*, the robot tries to take the *TAPE2*, but fails because it is too far. The robot informs the user and replans. The agents execute the plan, completing the task.

- **Replanning after human inactivity**. In this run the robot computes that the *TAPE3* and *BIN* are reachable only by the human, while the *TAPE2* is reachable only by the robot. The robot computes a plan and starts executing it, observing the human reactions. After an initial stage when the human is commited to the task, he does not execute a part of the plan (taking the final tape and throwing it), so the robot looks for another plan. The only solution to the problem is the one already computed at the beginning, so the robot decides to ask the human to take the tape and throw it. A run of this scenario is shown in figure 8.5.

FIGURE 8.5: The picture shows a run of our *replanning after human inactivity scenario*. The different rows show, starting from top to bottom: the real world picture, the world state representation built by the robot, symbolic facts introduced in the knowledge base at each time step, action taken by the agents at each time step, the current plan calculated by the robot.



8.3 Discussion

We review some of the main results of our experiments in this scenario:

- **The system is able to handle joint goals**. The system is able to create shared plans with different users, taking into account the capabilities of each agent. When unexpected changes in the world or task status arise, the system is able to quickly replan, adapting to new scenarios. The system is able to execute this joint goal in a human aware way.
- **The system is able to handle joint actions**. The system is able to estimate user intentions in collaborative tasks and to choose appropriate actions, using a set of MOMDP models.
- **The system is able to handle user preferences**. The system is able to adapt itself to user preferences, allowing the human partner to give commands or to be more passive in its role and switching from one modality to the other.
- **The system is able to handle each agent beliefs**. The system is able to represent different belief states for different agents and to take into accout what users can see, reach and know when creating a plan.
- **The system is able to monitor human actions**. The system was able to understand when the human performed action such as taking or throwing objects.

Chapter 9

Human-Aware Probabilistic Planning

In this section we present a recent work where we developed a Multi-Agent Markov Decision Process (MAMDP). Section 9.1 presents the problem. Section 9.2 briefly introduces our approach, which is explained fully in section 9.3 and section 9.4. Section 9.5 discusses the characteristics of this model, while section 9.6 show an example of creation of a MAMDP, starting from single-agent models. Finally, section 9.7 proposes an extension to the plan monitoring algorithm previously introduced, that would allow our system to monitor tasks and evaluate human engagement in collaborative activities.

9.1 Introduction

Multi-Agent planning is an important and studied topic in the AI community (Durfee et al., 1999). There are several approaches to this problem, using classical or probabilistc planning. There are several issues to consider:

- Distributed vs Centralized. A multi-agent planner might be distributed, meaning that separate systems plan independently and then communicate to build a shared plan (an idea investigated, for example in Guestrin and Gordon (2002), Nikolaidis and Shah (2013)); or centralized, meaning that a single system plans for all the agents.
- Coordination. Agents need to coordinate their plans, in particular in the presence of shared resources. Imagine, for example, two agents, Max and Bob, that are using a tool to repair a set of cars. If Max is proceeding faster than Bob and the two do not coordinate, Max might take the tool and leave, starting to repair another car, ignoring the fact that Bob still needs the tool. This example shows that it is important to reason on the duration of actions performed by agents. At the simplest level, agents need to know the advancement of the sub-plan of other agents. More complex reasoning might take into account how long an agent needs to perform a certain sub-task, in order to refine a plan.

- Cooperation. Even when performing different sub-tasks of the same plan, agents can help each other, for example by passing items, thus improving the efficiency of the plan. Multi-Agent problems can be loosely or tightly coupled, depending on the quantity of interactions between agents. Some works are not focused specifically on tightly or loosely coupled problems, and try to present a generic approach. Torreño et al. (2015) proposes a cooperative refinement planning approach, based on the partial-order planning paradigm. In this work each agent refines a centralized plan. Agents are able to exchange information, since the complete world-state might not be observable by everyone. Refined plans are then analyzed by each agent, and voted with a democratic leadership approach. Results show that this approach is very efficient at solving loosely coupled problems, but also competent on tightly coupled situations.

- Communication and Knowledge. In a multi-agent environment each agent might have an incomplete or incorrect belief on the world, which might lead to wrong or sub-optimal actions. Agents may communicate to progressively build a correct belief model on the world state.

Several approaches has been studied to bring the multi-agent planning problem in a probabilistic framework. Boutilier (1999) create a centralized MDP, able to select at each time step actions for every present agent. Dec-POMDP (Bernstein et al., 2002) and I-POMDP (Gmytrasiewicz and Doshi, 2005) are more complex frameworks, that take into account the belief models of agents. The complexity of these models makes them difficult to use in even moderately difficult scenarios. A solution to this problem is considering simpler problems, where the agents mostly work independently and interact only in limited situations, such as in Melo and Veloso (2013). Since we are more interested in tightly coupled scenarios, where the robot and human interact often, these kind of approaches are not suitable.

9.2 Overview

In some situations, the environment, or other agents' actions, can be very unpredictable, and the system needs to constantly adapt its plans to the current state of the world, by replanning or repairing processes, which can be expensive. To deal with this issue we developed a Human-Aware Probabilistic Planner, based on MDPs. Our goal is replicating the characteristics of HATP, which we presented in section 6.4, in a probabilistic domain. We designed this planner with the following ideas:

- Centralization. We will use a single planner, which chooses actions for all the involved agents.

- Hierarchical. The domain will be split in different modules, which interact to achieve the goal. Hierarchical models allow us to speed up the computation of the MDP policy and to reuse models in different domains and tasks.

- Tightly Coupled. We will focus on problems where the agents' interactions are frequent.
- No Communication. The planner will assume that all the agents have perfect knowledge of the world state. The rest of the system will have to take care of maintaining users' knowledge, ensuring the correct execution of the plan. Current planners that try to include communication issues often focus on loosely coupled interactions between the agents, in order to simplify the domain and be able to compute a policy for the problem. Since we choose to focus on tightly coupled problems, we can not use this solution, and so prefer to avoid the issue at this level.

This work is a very recent development, and has not yet been integrated in the rest of the system. We will explain the main characteristics of the MAMDP in the following sections.

9.3 Single-Agent MDP

The starting point for this planner is the single agent MDP. We start with the classical model (S, A, T, C, G, S_0), where S is the system space, A is the set of actions, $T(s_i, a, s_j)$ is the probability to transition from state s_i to state s_j after taking action a, $C(s, a)$ is the cost of taking action a in state s, $G \in S$ is the set of goal states, and $S_0 \in S$ is the set of starting states. We express the state space S of the system as a set of variables var, each one with a possible range of values $values(v)$, where v is a variable.

We develop this model in the following way.

9.3.1 Parameters

We can define parameters in our MDP, and assign them to different values. We will call the list of parameters of a MDP *par* and their current instantiation the *parameter_instance* of the model. Parameters can be linked to variables and values. We call *par_var* the variables associated to a parameter.

For example, let us imagine a scenario of a room, with different furniture and objects. Let us define the state space of a generic MDP to take an object in this room. We can define two variables: the location of an agent, *agent_isAt*, which can assume values in the set f_1, f_2, f_3 (where f_i represent a furniture in the room); and an object variable, *object_isAt*, which can assume the same values as the *agent_isAt* variable. If we define two parameters, *agent* and *object*, and link them to the *agent_isAt* and *object_isAt* variables, we can create a generic MDP which can be used to plan for any agent (with similar capacities) to take an object in this room.

The MDP receives as input, when consulted to select an action, the state of the world, which we will call *real_space*. The *real_space* will be converted, using the *parameter_instance*, to the *parameter_space*, which will be actually used by the model in its computations.

For example, in the real world, we have two agents, *Bob* and *Greg*, and three objects *glue*, *book*, *smartphone*. If we assign the *agent* parameter to *Bob* and the *object* parameter to *book*, when receiving the complete world state the MDP will assign as *agent_isAt* the location of Bob, and as *object_isAt* the location of the book, discarding unneeded variables.

Parameters allow us to create smaller, generic MDPs, which can be reused easily.

9.3.2 Actions and Macro Actions

We define actions as a tuple $(subject, action\_name, object, target)$, where each element of the tuple is called an *action_part*. We represent the action as a string, composed by its *action_parts*, joined by the character '_' as delimiter. For example, an action where Greg places a book on a table can be represented as *Greg_place_book_table*. We define the function $convertPar(a)$ and $convertReal(a)$ to convert action a to the *real_space* or the *parametrized_space*.

We add to the normal set of actions of a MDP *macro actions*, linked to other MDPs, which allow us to create a hierarchy of MDPs. We call M the set of *macro actions* of the MDP, and $sub(a)$ the sub-MDP linked to the *macro action* a. For example, we might create, in a MDP whose goal is cleaning a room, a macro action to reorder the room's objects and one to clean the floor. Each of these macro actions will be linked to a specific MDP, who will refine the problem in more details.

The cost of executing a macro is directly computed from the sub-MDP. There are several strategies to compute this cost, like in Dietterich (2000), Hauskrecht et al. (1998).

9.3.3 Name and Parameter Name

We define for each MDP model a *name*, which identifies it, and a *parametrized_name*, which substitutes parameters using the *parameter_instance*. The *name* is defined using the same tuple as an action.

For example, a MDP whose goal is to obtain an object could have as *name* *agent_get_object* and as *parametrized_name*, in a certain moment, *Greg_get_book*.

We define a function $assignParameterFromActionName(a)$ which creates the *parameter_instance* of the MDP based on an action name. This function can be very useful when using *macro actions*, so that the system can assign parameters to the sub-MDP from the *macro action* string and then consult it.

For example, if the *macro action* *Greg_get_book* is linked to the *agent_get_object* MDP, this function would assign values to the *agent_get_object* model's parameter. The system would assign the value *Greg* to the *agent* parameter and the value *book* to the *object* parameter.

Each *name* can be divided in a number of *name_parts*, with the same procedure as actions. The *parametrized_name* can be divided as well in *parametrized_parts*.

9.3.4 Abstract States

In some situations, a model might not need to base its planning choices on all the possible values of a variable in the *real_space*. Imagine, for example, the case where an agent needs to perform a series of operations on the furniture f_1 in the room. In this case, we could model the values of the *agent_isAt* variable as $\{f_1, other\_location\}$, greatly reducing the state space. In this situation we say that *agent_isAt* is an *abstract variable*.

We build a map $abstract\_values_v$ for each abstract variable v, which links real world values to model values (e.g. $f_2 \rightarrow other\_location$). This map will be used to convert a state expressed in the *real_space* to the *parameter_space* of the model.

9.4 Multi-Agent MDP

Now we will explain how we build a Multi-Agent MDP (MAMDP), starting from n single-agent models, one for each agent. We will call the single-agent models MDP_i, where $1 \leq i \leq n$ is the index of the agent. In the following paragraphs, we will use the notion $S_i, var_i, values_i, A_i, T_i, C_i, G_i, S_{0i}, M_i, par_i, par\_var_i$ when referring to components of the single-agent MDP, adding the i index to differentiate them from the MAMDP.

9.4.1 Name and Parameter Name

The *name* and *parameter_name* of the MAMDP are built by concatenating those of its agent MDPs, adding the '-' character to separate the single agents.

For example, the *name* of a MAMDP with single-agent models *agent_get_object* and *agent_clean_table* is *agent_get_object-agent_clean_table*.

9.4.2 State Space and Parameters

$par = \bigcup_{1 \leq i \leq n, p \in par_i} p + \text{'p'} + i$

$par\_var = \bigcup_{1 \leq i \leq n, v \in par\_var_i} rename(v)$

$var = \bigcup_{i=1}^{n} (var_i \setminus par\_var_i) \cup par\_var$

$$\forall_{v \in var}\ values(v) = \bigcup_{i | v \in var_i} \begin{cases} values_i(v) & \text{if } v \notin par_i \\ p + \text{'p'} + i & \text{if } v \in par_i \end{cases}$$

To create the set of parameters of the MAMDP we will rename each parameter in the single-agent MDPs, adding an identifier composed by 'p' and by the index of the MDP. We make this choice because different MDPs could share the same parameters, but they could be assigned to different values, and so need to be treated as separate entities, even if they have the same semantic meaning. Parameter variables are created by renaming the parameter variables of the sub-MDPs to account for this newly added identifier.

For example, if the MDP_1 model has a parameter *object*, linked to a variable called *object_isAt*, we will rename the parameter as *objectp1* and the variable as *objectp1_isAt*.

We defined S as the union of the variables of the single-agent MDPs excluding their parameter variables. We will instead take the parameter variables of the MAMDP, in the way that we just defined.

The MAMDP variable can assume all the possible values that are available in the corresponding variables of the sub-MDPs. As before, we will change values that are parameters to account for our renaming procedure.

We define the function $single_i(s)$, which converts the MAMDP state to state of the single-agent MDP i.

9.4.3 Actions

$A = (\prod_{i=1}^{n} A_i) \cup JointActions \cup WaitActions$, where $JointActions$ is the set of collaborative actions, and $WaitActions$ is a set computed by enumerating all the possibile instances where one or more agents do not act, and simply wait, while others are acting.

The actions of the MAMDP are a concatenation of the actions of the single MDP, adding the separator character '-' between the different agents' actions. We will refer to the single agent's actions of action a as $a_i \quad \forall\, 1 \leq i \leq n$.

For example, the MAMDP will contain actions such as 'agentp1_move_surface1-agentp2_move_surface2'.

We add to A the special set of $JointActions$, which are actions that the agents can use to cooperate. For example, if there is a resource in a single agent MDP which two agents can possess, we introduce a *handover* action between them.

For each $a \in A$, if there is a sub-action a_i which is a macro action for MDP i, we create a new MAMDP and assign it as sub-MAMDP of a. This sub-MAMDP will be created from n MDP models, as for its father, in the following way. Let f be the father MAMDP, c the sub-MAMDP, and a the macro-action of f.

$\forall_{\text{MDP } m_i \in f}$ we assign a MDP m_j in c where:

$$m_j = \begin{cases} m_i & \text{if } a_i \notin M_i \\ sub_i(a_i) & \text{if } a_i \in M_i \end{cases}$$

9.4.4 Transition Function

$$T(s_a, a, s_b) = \begin{cases} \prod_{i=1}^{n}(T_i(single_i(s_a), a_i, single_i(s_b))) & \text{if } !isIncongruent(s_a) \wedge !isIncongruent(s_b) \\ 0 & \text{else} \end{cases}$$

The transition function is computed as the product of the transition functions of the single agent MDPs, on the converted states and actions.

We set the probability as zero if either the starting or ending states state of the transition are incongruent. This is done by converting the states to the *real_space* and checking the values of the different variables. We say that a state is incongruent if two non abstract variables in the *parameter_space*, that are assigned to the same *real_space* variable, have different values. If one or both the variables are abstract we check if their *abstract_values*, for the current variable values, have a value in common. If not, we consider the state incongruent.

For example, consider the state ($objectp1\_isAt = `surface', objectp2\_isAt = `table', agentp1\_isAt = `table', agentp2\_isAt = `table'$). If in the current *parameter_instance* both *objectp1* and *objectp2* are assigned to the same variable, this state will be incongruent.

Consider now the state ($objectp1\_isAt = `agentp1', objectp2\_isAt = `other', agentp1\_isAt = `table', agentp2\_isAt = `table'$). If $objectp1$ is a parameter assigned to $Greg$, $objectp1\_isAt$ is an abstract variable, and $abstract\_values_{objectp2\_isAt}(`Greg') = `other'$, this state will not be considered incongruent.

9.4.5 Start and Goal States

$S_0 = s \in S \mid \forall_{i=1}^{n} s_i \in S_{0i}$. A state is a starting state in the MAMDP only if it is a starting state in all the single agent MDPs.
$G = s \in S \mid \exists_{i=1}^{n} \mid s_i \in G_i$. A state is a goal state in the MAMDP if it is a goal state in any one of the single agent goal states. The idea of this choice is that a MAMDP terminates when one of the agents has achieved its goal. If there is an agent that has not completed its task, and the MAMDP was a sub-model in a hierarchy, its father will select another action, perhaps assigning one or more agents to the incompleted task.

In some situations, we might try creating a MAMDP composed of opposing goals, like for example, a MAMDP where both agents need to get the same object. Since we are interested in cooperative and

not competitive scenarios, we will not create these kind MAMDPs. We can recognize this situation by checking if there is any state in G that is incongruent, as previously defined in subsection 9.4.4.

9.4.6 Cost Function

$$C(s_a, a, s_b) = \begin{cases} max_{i=1}^{n}(C_i(single_i(s_a), a_i, single_i(s_b)) & \text{if } s_b \notin G \\ \sum_{i|single_i(s_a) \notin G_i} expectedCost_i(single_i(s_a)) & \text{if } s_b \in G \end{cases}$$

where $expectedCost_i(s)$ is a function that computes the expected cost for agent i to reach a goal state by starting in state s and following the action policy. In this calculation, we allow from the other agent only *JointActions* that are necessary to solve the plan (e.g. handovers if the agent does not have a needed resource).

The cost function of the MAMDP is chosen by taking the maximum cost of the single agent actions that are executed, if the state s_b is not a goal state. If it is a goal state, we take as cost an estimation of the time required by any agent that has not reached his goal to complete their tasks, with only minimal cooperation. In this way, we encourage the MAMDP to choose plans where the agents cooperate and do not only try to achieve their goal.

9.5 Discussion

The result of this fusion is a MDP, that can be solved with well-known algorithms, like value iteration. By fusing all the possible MDPs, the resulting model can be very hard to solve but, using macro actions, parameters, and abstract states we can greatly reduce its complexity.

The action space of the joint model is calculated as the cross product of the action spaces of the single model. When there are several macro actions in the agent models, this could result in the creation and computation of the policy of several sub-models, which can significantly slow down the computation of a solution for the model. We can reduce the complexity of this process by making several considerations.

First of all, created MAMDPs can be reused in different branches of the hierarchy, if needed. When creating a new MAMDP, we will look to see if there is already an existing MAMDP with the same *name* and, if so, we will use it.

Second, parameters can greatly reduce the number of sub-MAMDPs to compute. Let a be a *macro action*, c the sub-MAMDP linked to a, $1..n$ the indexs of the single-agent MDPs of c, which are selected as previously explained.

We create the *parametrized_name* of c by concatenating the *parametrized_name* of its single-agent MDPs. To set the *name* of c we process each name of its single-agent MDPs $name_i$, by modifying all of

its $name\_parts_{i,k}$, with $1 \leq k \leq m$ in the following way:

$$name\_part_{i,k} = \begin{cases} name\_part_{i,k} & \text{if } !parametersInCommon(name\_part_{i,k}) \\ parametrized\_part_{i,k} & \text{else} \end{cases}$$

where $parametersInCommon(name\_part_{i,k})$ is a function that returns true if $name\_part_{i,k}$ is a parameter in sub-MDP i, and there is a sub-MDP with index j, where $name\_part_{i,k}$ is a parameter or variable.

The idea behind this choice is the following. If the sub-MDPs do not have parameters in common, we can use a generic MAMDP to represent this instance. For example, if we create a MAMDP for the macro action $agent1\_clean\_table - agent2\_clean\_shelf$, we can use the MAMDP $agent\_clean\_furniture - agent\_clean\_furniture$ since the sub-MDP models do not have parameters in common, and their actions will not conflict.

If instead we create a MAMDP for the macro action $agent1\_clean\_table - agent2\_clean\_table$ we create a specialized MAMDP, since there are parameters in common (*table*) and if we treat them as different objects there could be incongruences in the MAMDP planning.

9.6 Example

In this section we will show an example of creation of a MAMDP. We start by presenting a cooperative scenario. The robot and a human are working together to assemble three brackets on three differents work surfaces. To assemble a bracket the agents need to clean the surface, apply some glue to it, and fix the bracket. A possible set-up for this scenario is shown in figure 9.1.

9.6.1 Single-Agent MDP

We start by defining a single-agent MDP for this scenario. We will use a hierarchical architecture, as shown in figure 9.2. This architecture is composed by three different modules: AssembleAllBrackets, which controls the flow of the scenario; AssembleBracketSurface, which assembles a chosen bracket on a chosen surface; and GetObject, which obtains an object. We will create some simplifications in these models, as they are chosen just to explain how we build a MAMDP, and not for a realistic use.

9.6.1.1 AssembleAllBrackets

We will now show the *AssembleAllBrackets* model, whose goal is directing the flow of the scenario, by choosing which bracket should be assembled on which surface.

- $name$: $agent\_assembleallbrackets$.

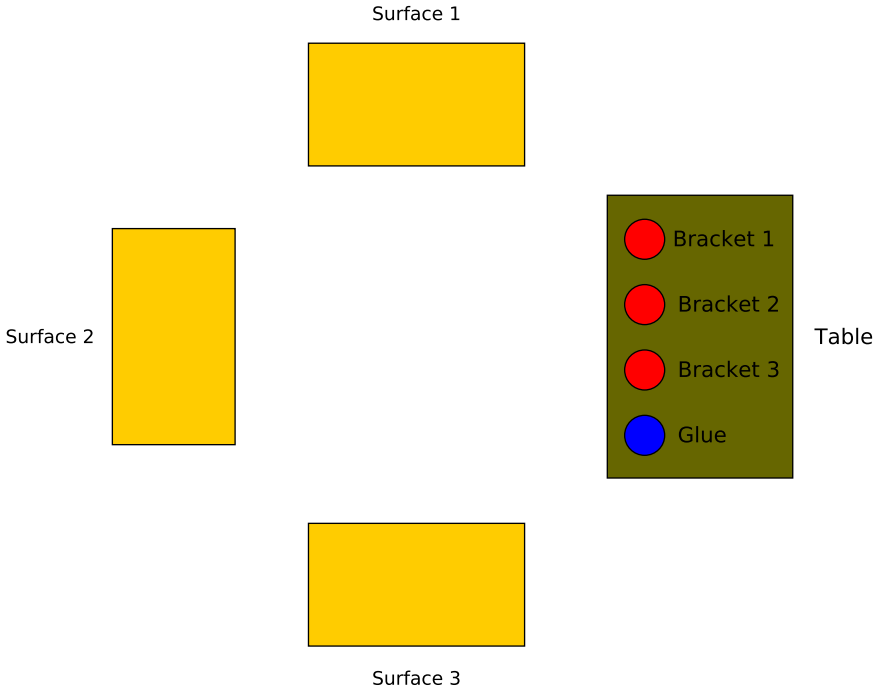


FIGURE 9.1: The image shows the set-up for this example scenario. The four locations are represented as different rectangles and the objects as circles.

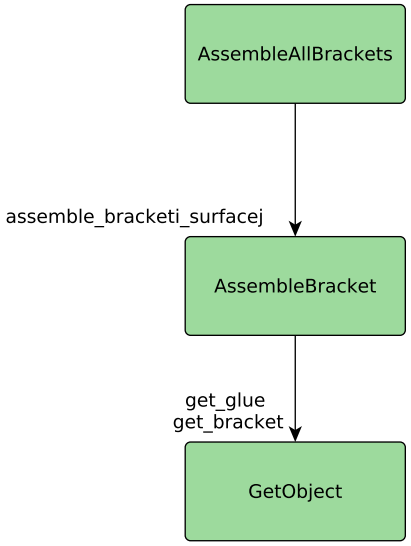


FIGURE 9.2: The image shows the architecture for the single agent MDP of the MAMDP example scenario. The various MDPs are represented as rectangles. The arrows show the links between hierarchical modules in the architecture. The label of the arrow shows the macro action corresponding to the link. The label *assemble_bracketi_surfacej* actually correspond to all the combinations to assemble a bracket on a surface. We grouped them since, as the AssembleBracketSurface MDP uses parameters, it can be used for all these macros.

- $par : agent$.
  - $par\_var(agent) = agent\_isAt$.

  In this example, we decided to create generic models which can plan either for Greg or for the robot.

- S : $\{agent\_isAt, bracket1\_isAt, bracket2\_isAt, bracket3\_isAt, surface1\_status, surface2\_status, surface3\_status\}$.
  - $values(agent\_isAt) : \{table, surface1, surface2, surface3\}$.
  - $values(bracketi\_isAt) : \{table, surface1, surface2, surface3, agent, other\_agent\} \forall_{i=1}^{3}$.
  - $values(surfacei\_status) : \{completed, other\_status\}\ \forall_{i=1}^{3}$.
  - $abstract\_values\_surfacei\_status$:
    - $none : other\_status$.
    - $cleaned : other\_status$.
    - $glued : other\_status$.
  - $abstract\_values\_bracketi\_isAt$
    - $greg : other\_agent$.
    - $robot : other\_agent$.

  The state set of the model contains the location of the agent, of the brackets, and the status of the surfaces. The agent and the objects can be located at the table or at one of the surfaces. In addition, objects can be possessed by another agent. The status of a surface can be set to none, meaning that the fixing process as not started; cleaned, meaning that the agent has already cleaned the surface; glued, meaning that the agent has applied glue to the surface; and completed, meaning the a bracket ha been fixed on it.

  To simplify the model we consider the states $none, cleaned, glued$ as abstract states. We also consider *other_agent* as an abstract value, assigned to both agents present in the scenario. Let us consider, for example, that the instance of the *agent* parameter is *greg*, and that in the *real_space* *bracket1_isAt=greg*. It could seem that, when converting from *real_space* to *parameter_space*, the system would assign *bracket1_isAt* to *other_agent*, since *other_agent* is an abstract value for *greg*. In reality, the system will give priority to the current parameter, and execute the conversion properly, by setting *bracket1_isAt=agent*.

- A : $\{agent\_assemble\_bracketi\_surfacej\}\ \forall_{i=1}^{1} \forall_{j=1}^{3}$.
- $M : A$.

  All of this model's actions are actually *macros*.

- G : $\cup_{s:S} \mid \forall_{i=1}^{3}\ surfacei\_status = completed$ AND
  $forall_{i=1}^{3}\ \exists_j \mid bracketi\_isAt = surfacej$ AND
  $\forall_{i,j=1}^{3}\ bracketi\_isAt \neq bracketj\_isAt$.

The goal of this model is to fix all the brackets on the surfaces. We set as goal states every state in which the brackets are located on different surfaces and the status of each surface is *completed*.

- $S_0 : \cup_{s:S} \mid \exists 1 \leq i \leq 3 \mid surfacei\_status \neq completed.$

Note that we did not need to specify a cost function for this model. Since all of its actions are *macros*, the cost function will be directly derived from its sub-models.

9.6.1.2 AssembleBracket

We will now show the *AssembleBracket* model, whose goal is assembling a specific bracket on a specifc surface.

- $name : agent\_assemble\_bracket\_surface.$
- $par : \{agent, bracket, surface\}.$
  - $par\_var(agent) = agent\_isAt.$
  - $par\_var(bracket) = bracket\_isAt.$
  - $par\_var(surface) = surface\_status.$
- $S : \{agent\_isAt, bracket\_isAt, surface\_status, glue\_isAt\}.$
  - $values(agent\_isAt) : \{surface, other\_location\}.$
  - $values(bracket\_isAt) : \{agent, surface, other\_location, other\_agent\}.$
  - $values(glue\_isAt) : \{agent, other\_location, other\_agent\}.$
  - $values(surface\_status) : \{none, cleaned, glued, completed\}.$
  - $abstract\_values\_bracket\_isAt$:
    - $surfacei = other\_location \ \forall_{i=1}^{n}$.
    - $table = other\_location.$
    - $greg = other\_agent.$
    - $robot = other\_agent.$
  - $abstract\_values\_glue\_isAt = abstract\_value\_bracket\_isAt$
  - $abstract\_values\_agent\_isAt$:
    - $surfacei = other\_location \ \forall_{i=1}^{n}$.
    - $table = other\_location.$

This state set will contain, other than the variables of *AssembleAllBrackets*, also the location of the glue bottle. In this situation, *surface_status* will not be an abstract variable, since the model needs to know what is the exact state of the fixing process. We greatly simplified the number of locations where agents and objects can be present by using abstract variables. This model is only interested in knowing if the agent has an object (glue or bracket) or not. In the case where the agent does not have an object, it can invoke the *get_object macro*, which will take charge of obtaining it, using a more complete state space.

- $A:$ $\{agent\_move\_surface, agent\_clean\_surface, agent\_glue\_surface,$ $agent\_fix\_bracket, agent\_get\_bracket, agent\_get\_glue\}$.
- $M:$ $\{agent\_get\_bracket, agent\_get\_glue\}$.
- $G:$ $\cup_{s:S} \mid surface\_status = completed.$
- $S_0:$ $\cup_{s:S} \mid surface\_status \neq completed.$

9.6.1.3 GetObject

We will now show the *GetObject* model, whose goal is obtaining an object in the environment.

- $name$: $agent\_get\_object$.
- par : $\{agent, object\}$.
  - $par\_var(agent) = agent\_isAt$.
  - $par\_var(object) = object\_isAt$.
- $S:$ $\{agent\_isAt, object\_isAt\}$.
  - $values(agent\_isAt) : \{table, surface1, surface2, surface3\}$.
  - $values(object\_isAt) : \{table, surface1, surface2, surface3, agent, other\_agent\}$.
  - $abstract\_values\_object\_isAt$:
    - $greg = other\_agent$.
    - $robot = other\_agent$.

  The state set of this model contains a more detailed representation of the possible locations of the scenario, since the goal of the agent is obtaining an object.

- $A:$ $\{agent\_move\_table, agent\_move\_surface1\, agent\_move\_surface2,$ $agent\_move\_surface3, agent\_take\_object\}$.

- M : $\emptyset$.
- G : $\cup_{s:S} \mid object\_isAt = agent$
- S_0 : $\cup_{s:S} \mid object\_isAt \neq agent$

9.6.2 MAMDP

We will now show how our MAMDP is created. The MAMDP hierarchy, shown in figure 9.3 will contain different models. At the highest level lies the model *AssembleAllBrackets-AssembleAllBrackets*, a duplication for two agents of the highest module in the single agent MDP hierarchy. The lowest modules include different combinations of the single agent sub-MDPs, in order to account for the possible task allocations. We will start by showing the top MAMDP model.

9.6.2.1 AssembleAllBrackets-AssembleAllBrackets

- $name$: $agent\_assembleallbrackets - agent\_assembleallbrackets$.
- par : $\{agentp1, agentp2\}$.
  - $par\_var(agentp1) = agentp1\_isAt$.
  - $par\_var(agentp2) = agentp2\_isAt$.

  The name of the parameters are changed, adding pi at the end, where i is the index of the agent. This allows to assign them in an unique way.
- S : $\{agentp1\_isAt, agentp2\_isAt, bracket1\_isAt, bracket2\_isAt, bracket3\_isAt,$ $surface1\_status, surface2\_status, surface3\_status\}$.
  - $values(agentp1\_isAt) : \{table, surface1, surface2, surface3\}$.
  - $values(agentp2\_isAt) : \{table, surface1, surface2, surface3\}$.
  - $values(bracketi\_isAt) : \{table, surface1, surface2, surface3, agentp1, agentp2\}\ \forall_{i=1}^{3}$.
  - $values(surfacei\_status) : \{completed, other\_status\}\ \forall_{i=1}^{3}$.
  - $abstract\_values\_surfacei\_status$:
    - $none$: $other\_status$.
    - $cleaned$: $other\_status$.
    - $glued$: $other\_status$.

  The state set is not very different, simply accounting for the presence of two agents.

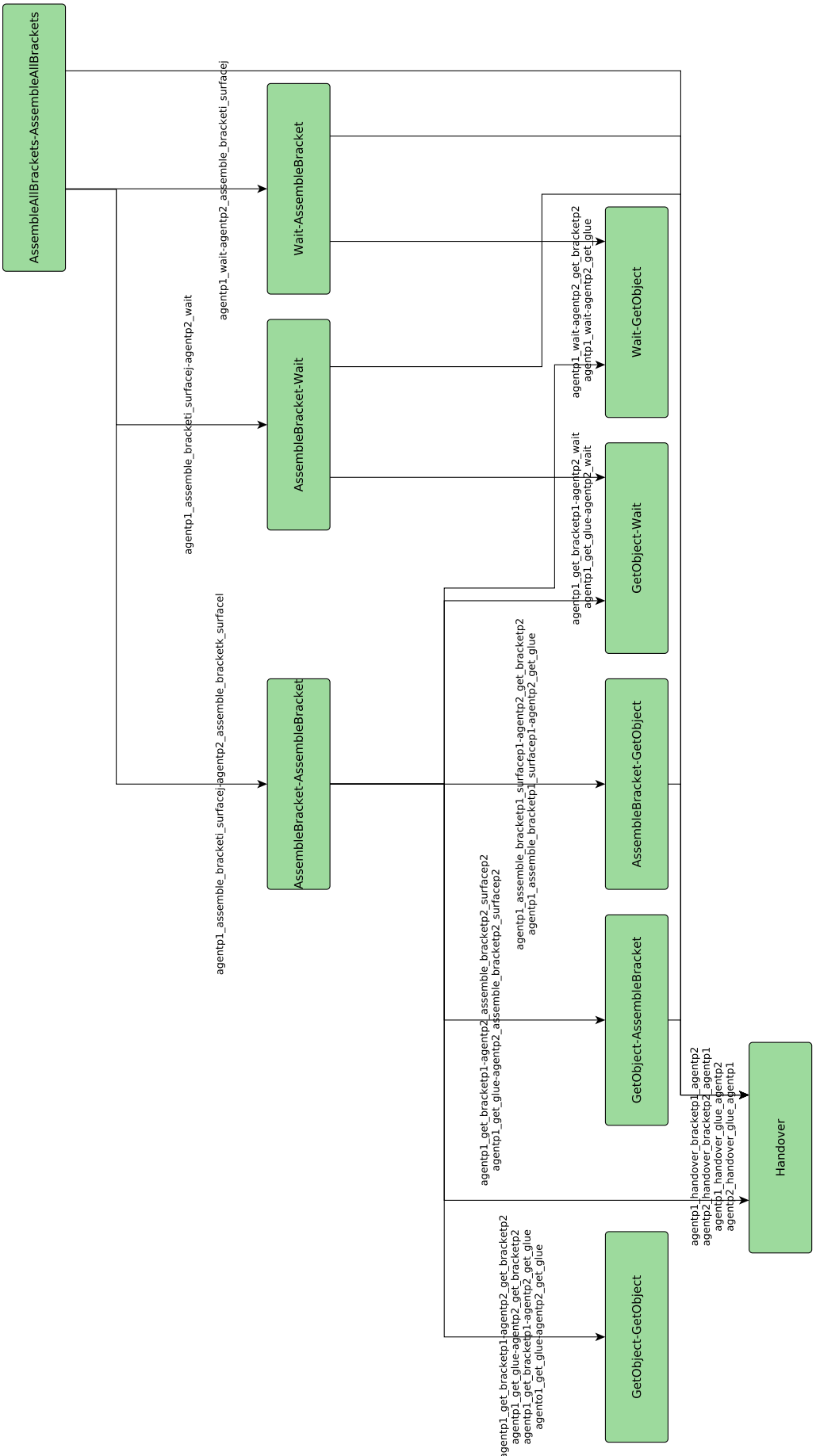


FIGURE 9.3: The image shows the architecture for the MAMDP of the example scenario. The various MAMDPs are represented as rectangles. The arrows show the links between hierarchical modules in the architecture. The label of an arrow shows the macro actions corresponding to the link.

- A : $\{agentp1\_assemble\_bracketi\_surfacej - agentp2\_assemble\_bracketk\_bracketl\}$ $\forall_{i=1}^{1}\forall_{j=1}^{3}\forall_{k=1}^{3}\forall_{l=1}^{3} \cup$ $JointActions \cup WaitActions$.

- $JointActions = agentpi\_handover\_bracketj\_agentpl$ $\forall_{i=1}^{2}$ $\forall_{j=1}^{3}$ $\forall_{l=1;l!=i}^{3}$.

- $WaitActions = (agentp1\_assemble\_bracketi\_surfacej - agentp2\_wait \cup$ $agentp1\_wait - agentp2\_assemble\_bracketi\_surfacej)$ $\forall_{i=1}^{3}$ $\forall_{j=1}^{3}$.

  The action set is the cartesian product of the actions of the single models, plus actions to exchange brackets, plus actions where only one agent is acting.

- $M : A$.

- G : $\cup_{s:S} \mid \forall_{i=1}^{3}\ surfacei\_status = completed$ AND
  $\forall_{i=1}^{3}\ \exists_j \mid bracketi\_isAt = surfacej$ AND
  $\forall_{i=1}^{3}\ bracketi\_isAt \neq bracketj\_isAt$.

- S_0 : $\cup_{s:S}\ \exists 1 \leq i \leq 3 \mid surfacei\_status \neq completed$.

  The starting and goal states are the same as the single-agent model. Since the two models used to create the MAMDP are the same, the intersection and union of the states will be the same.

If we would create and solve a new MAMDP for each of these macros, we would create a great quantity of models. Fortunately, as previously said, we can reduce the number of models by creating generic sub-models for a macro that can be used when there are not parameters in common. In this case, we actually need to create a specifc sub-MAMDP for every action of the type *agentp1_assemble_bracketi_surfacej-agentp2_assemble_bracketk_surfacel*, where $i = j$ or $j = l$ (since they would have variables in common), plus a generic sub-MAMDP for all the other cases.

Also, in this example, all the MAMDPs for actions where the same bracket is assembled by two agents on different surfaces would not be created, as their goal states contain incongruent states (since a bracket can not be on two surfaces at the same time and a surface can not contain more than one bracket).

9.6.2.2 AssembleBracket-GetObject

Let us consider the AssembleBracket-GetObject MDP, deeper in the hierarchy. In this case, we will need to create a generic sub-MAMDP, plus specific MDPs for the macros *agentp1_assemble_bracketi_surfacej-agentp2_get_bracketi* and *agentp1_assemble_bracketi_surfacej-agentp2_get_glue*, since they share resources. We will show parts of this sub-MAMDP.

- $name$: $agent\_assemblebracket - agent\_getobject$.

- par : $\{agentp1, bracketp1, surfacep1, agentp2, objectp2\}$.

- $par\_var(agentp1) = agentp1\_isAt$.
- $par\_var(bracket1) = bracketp1\_isAt$.
- $par\_var(surfacep1) = surfacep1\_status$.
- $par\_var(agentp2) = agentp2\_isAt$.
- $par\_var(object2) = objectp2\_isAt$.

- S : $\{agentp1\_isAt, bracketp1\_isAt, surfacep1\_status, glue\_isAt, agentp2\_isAt, objectp2\_isAt\}$.
  - $values(agentp1\_isAt) : \{surfacep1, other\_location\}$.
  - $values(bracketp1\_isAt) : \{agentp1, surface, other\_location, other\_agent\}$.
  - $values(glue\_isAt) : \{agentp1, other\_location, other\_agent\}$.
  - $values(surfacep1\_status) : \{none, cleaned, glued, completed\}$.
  - $values(agentp2\_isAt) : \{table, surface1, surface2, surface3\}$.
  - $values(objectp2\_isAt) : \{table, surface1, surface2, surface3, agentp2, other\_agent\}$.
  - $abstract\_values\_objectp2\_isAt$:
    - $greg = other\_agent$.
    - $robot = other\_agent$.
  - $abstract\_values\_bracketp1\_isAt$:
    - $surfacei = other\_location \forall_{i=1}^{n}$.
    - $table = other\_location$.
    - $greg = other\_agent$.
    - $robot = other\_agent$.
  - $abstract\_values\_glue\_isAt = abstract\_value\_bracketp1\_isAt$
  - $abstract\_values\_agentp1\_isAt$:
    - $surfacei = other\_location \forall_{i=1}^{n}$.
    - $table = other\_location$.

- G : $\cup_{s:S} \mid surfacep1\_status = completed$ OR $objectp2\_isAt = agentp2$
- S_0 : $\cup_{s:S} \mid surfacep1\_status \neq completed$ AND $objectp2\_isAt \neq agentp2$

The goal and starting states of the model are, respectively, the union and intersection of the single models.

9.6.2.3 Handover

Finally, to conclude, we show the special handover MAMDP, used for cooperative actions. Since this is a special model, not created by joining two single-agent MDPs, it will not follow the same rules, and instead it will be treated in a similar way as a single agent MDP.

- $name : agent1\_handover\_object\_agent2$.
- $par : \{agent1, object, agent2\}$.
  - $par\_var(agent1) = agent1\_isAt$.
  - $par\_var(agent2) = agent2\_isAt$.
  - $par\_var(object) = object\_isAt$.
- $S : \{agent1\_isAt, agent2\_isAt, object\_isAt\}$.
  - $values(agent1\_isAt) : \{surface1, surface2, surface3, table\}$.
  - $values(object\_isAt) : \{agent1, agent2, other\_location\}$.
  - $values(agent2\_isAt) : \{surface1, surface2, surface3, table\}$.
  - $abstract\_values\_object2\_isAt :$
    - $table = other\_location$.
    - $surface1 = other\_location$.
    - $surface2 = other\_location$.
    - $surface3 = other\_location$.
- $A : \{agent1\_move\_location - agent2\_move\_location, agent1\_move\_location - agent2\_wait,$
  $agent1\_wait - agent2\_move\_location,$
  $agent1\_give\_object\_agent2 - agent2\_receive\_object\_agent1\}$
  where $location \in \{table, surface1, surface2, surface3\}$
- $G : \cup_{s:S} \mid object\_isAt = agent2$
- $S_0 : \cup_{s:S} \mid object\_isAt = agent1$

9.7 Enhancing Task Monitoring

In section 6.6 we introduced the problem of task monitoring. We presented three different problematics: monitoring actions, monitoring tasks, and evaluating human engagement.

We presented a solution only for the problem of monitoring actions. In this section, we propose a strategy to monitor tasks and evaluate the human's engagement, based on the MAMDP model developed in this chapter. We are just starting to investigate this idea, and have not yet completed an implementation.

In general, a MAMDP proposes only the next action in the plan, which is not sufficient to manage plans and ensure synchronism. To deal with this problem, we can compute an horizon of h actions from the MAMDP, building causal links between the nodes to create the stream structure introduced in chapter 6.

9.7.1 Evaluating Human Engagement and Monitoring Tasks

Understanding if a human is engaged in a task is equivalent to infer if its current intention is to achieve the task. As previously said, in section 4.2.5, normally, our Intention and Action Recognition module is not able to monitor joint goals. To solve this issue, while performing a shared plan, we create, in the Situation Assessment layer, a new intention for each MAMDP model in the domain. We associate to these intentions the linked MAMDP and the context node *have a shared plan*, which is treated as evidence with a true value. We call these intentions *plan intentions*.

We associate to each known intention a precomputed *expected length*, which is the expected time to accomplish the linked goal.

Using the Intention and Action Recognition module, the robot can infer which is currently the most likely human intention. If the current intention equals the task to monitor, the robot infers that the human is actively working to complete its task.

We consider the task, and the monitor procedure, as *completed* when the linked MAMDP reaches a *goal state*.

If the human is currently not involved in the monitored task there are three possibilities:

- The human has momentarily interrupted the task. This can be inferred if the human is currently involved in another intention, which does not belong to the *plan intentions*, and whose expected length is *short*. In this case, the monitor procedure will not return an error until a predefined *allowed time*.
- The human has abandoned the task. This can be inferred if the human is currently involved in another intention, which does not belong to the *plan intentions*, and whose expected length is *long*. The monitor will return an error.

- The human is performing another task in the shared goal. This can be inferred if the human is currently involved in another intention, which belongs to the *plan intentions*. In this case the monitor will return an error.

Part IV

The Robot Teacher

We can imagine robots in several ways. The more realistic persons will imagine the robot as a tool, able to execute repetitive, high-precision tasks, reducing humans' effort. The more imaginative persons will instead think of the kind of robots that we can see in science fiction literature or movies: autonomous agents, able to reason and to act almost as humans. In general, we would think of the robot as some sort of 'programmable' entity. We teach the robot, through our code and algorithms, to execute its task.

In the last years, actually, there is an increasing interest in 'inverting this role' by having robots act as teachers, particularly with children. Some studies, like Movellan et al. (2009), have shown that a robot teacher can actually positively impact on the learning process of children.

In this part, we introduce the problem of the *robot teacher*, where the robot needs to explain a user how to execute a task and then guide him in its achievement. We are particularly interested in adapting this process to the *knowledge* of users in the tasks to perform, by explaining more in-depth processes where the user has less *expertise*, and quickly tasks that user has already accomplished several times. While we are calling this problem *robot teacher*, these mechanisms can benefit any cooperative application where the robot is acting as leader, and needs to explain a cooperative plan to its partners.

Chapter 10 will explain how we model the knowledge of a user in a task to adapt the plan explanation and management processes. Chapter 11 shows a user study performed to validate our approach.

Chapter 10

Adapting Plan Management to Human Knowledge

In this chapter we show how we use human knowledge to adapt the plan explanation and managent capacities of our system. Section 10.1 introduces the subject, while section 10.2 explains the basics of our approach to model knowledge. Section 10.3 shows how human knowledge can be used to adapt the planning capacities of HATP. Finally, sections 10.4 and 10.5 respectively describe how the robot can explain plans in a human-aware way, by taking into account the knowledge of the human, and how it can guide the human in the execution of the task, by modifying our plan management algorithm.

10.1 Introduction

In section 6.3.1 we introduced a simple algorithm to explain a plan to a user. The problem of explaining a plan in a 'naive way', by simply verbalizing all of its actions, is that the explanation can be long and not interesting.

Let us imagine a robot explaining a human how to change the tires of a car. The robot could start by telling the human how to loosen the nuts of a wheel. It could tell the human to take a wrench, to apply it to a nut, and to turn it counter-clockwise. The human will need to loosen several nuts. If for each nut the robot will explain the user again to apply the wrench and turn it counter-clockwise, the explanation will be much longer than what is needed. Imagine the robot explaining every single small operation of a large plan. The result could be a nightmare for the human! A more natural way to teach this task would be explain how to loosen the first nut, and then just ask the human to loosen the others, without going in-depth in each operation.

This idea is validated by research on Intelligent Tutoring Systems (Brusilovskiy, 1994) and on e-learning (Brusilovsky et al., 2005), which has proven the necessity of keeping and updating a model of the learner's knowledge to efficiently teach a task.

This idea has still not been widely investigated in robotics. Sorce et al. (2015) show a system where the robot can learn shared plans from a human, and then transfer this knowledge to non-expert humans. The explanations of the robot are joined by a video, representing what the robot perceived when it learnt the task, to give more information to the human.

Even when the robot is a coworker, and not a teacher, research (Lallee et al., 2013) suggests that collaborative plans should be fully communicated in order to sustain effective cooperation. The preferred method of communication in complex tasks is usually language (Warneken and Tomasello, 2007, Warneken et al., 2006).

In the following sections, we will present our approach, where we adapt plan explanation and management to the knowledge of users.

10.2 Overview

10.2.1 Modeling Knowledge

Before explaining our algorithms, we introduce a new attribute (as explained in 3.2) among our symbolic facts: the knowledge of a human in a task.

This attribute is represented as a tuple $(human, task, parameters, value)$, where $human$ is a string identifying the subject of this attribute, $task$ is the name of the related task, $parameters$ is a list of relevant parameters used to describe the knowledge in the task, and $value$ is the level of knowledge in the task. Knowledge values can be assigned from the set $[new, beginner, intermediate, expert]$. For example, we can represent the fact that Bob knows very well how to fix a wheel to a car with the tuple $(Bob, fix\_wheel, car, expert)$.

Parameters can be deeply linked to the knowledge of a task. In general, we can divide task parameters in two categories:

- Class-Link. In some situation, knowledge of a task is not linked to the specific instance of a parameter, but to a whole class. For example, we can imagine that if Bob knows how to paint the living room, he will know how to paint every room in a house. We could represent this attribute as $(Bob, paint, house\_room, expert)$. We can notice that the parameter, in this case, is $house\_room$, representing the class of rooms belonging to a house, and not $living\_room$.

- Instance-Link. In other cases, instead, knowledge of a task is deeply linked to specific instances of parameters. For example, knowing how to fix the motor of a specific car, would not necessary translate in knowing how to fix the motor of every car.

We can also consider some tasks as *common knowledge*, expecting every human to be able to perform them, like, for example, putting ingredients in a bowl. These tasks will be tagged as *common knowledge* and considered as known by any user, no matter the parameters.

10.2.2 Maintaining Human Knowledge in a Task

We define four task-knowledge levels for humans, that will lead to different behaviors from the robot.

- *New*. This value will be used for tasks which have never been performed by the user. If the user observes the task being executed, with an explanation of it, or if he performs it himself, the value will be changed to *beginner*. However if the user observes the task being executed without any explanation, we keep the level as *new* since we consider that he might not have received enough information to link the observed movements to the task.
- *Beginner*. This value will be used for users who have already achieved the task but may still need explanations to perform it again. If the user successfully performs the task a second time, without asking for explanations, the value is changed to *intermediate*, otherwise it is changed to *new*.
- *Intermediate*. This value will be used for users who are able to perform the task without guidance. If a *beginner* user successfully performs the task without guidance again, the value is changed to *expert*. In case of failure, it is downgraded to *beginner*.
- *Expert*. This knowledge level will be used for users who are able to perform the task without guidance and are experienced enough to explain it to a third party. If the user fails in performing the task, he will be downgraded to *intermediate*.

Using the knowledge of a task of a human, the system is able to adapt plan generation, explanation, and monitoring to that user.

10.2.3 Architecture

This process was implemented by integrating a new module in the Plan Management layer's architecture, presented in chapter 6, which we called 'Plan Explanation'. This module receives as input a shared plan, from the Plan Manager, and the current knowledge of the humans involved in the plan, from the Situation Assessmen layer. With these information, it will adapt the knowledge of users to explain the shared plan.

Task	Knowledge Level
Cook(ApplePie)	New
PrepareDough(Bowl Mould)	New
PrepareMixture(Bowl Mould)	Intermediate
PrepareFruits(Apple Bowl Mould)	Intermediate
Bake(Mould)	Intermediate

TABLE 10.1: Mental model for a human in the Cook Apple scenario in figure 10.2

Information about human's knowledge will be used also by the Plan Manager module to adapt the plan management algorithm, as we will explain in the following sections.

The modified architecture of the Plan Management layer is shown in figure 10.1. This work was presented in Milliez et al. (2016).

10.3 Adapting Plan Generation to Human Knowledge

When interacting with humans, it is important to take into account others' knowledge and capacities when planning. We consider two different policies for our planner: *teaching*, where the planner will look for plans not known to the human, in order to teach him different ways to achieve a task; and *efficiency*, where the robot will try to maximize the number of human tasks known when creating a plan. These ideas were represented as social rules.

To illustrate these new social rules let us consider an example where a human and a robot have to cook an apple pie. We can consider all the basic actions of this domain (pick, place, cut, etc.) as *common knowledge*, but the human might not know how to perform all the higher level tasks involved. If our policy favors *teaching*, the plan should should choose a decomposition with tasks where the human has a low knowledge level. On the other hand, if our policy is *efficiency* the planner should allocate to the human tasks where he is more competent, so that explanations and mistakes are reduced. Using this rule, the robot is able to adapt its plan generation to the knowledge of the user concerning tasks contained in the shared plan. In figure 10.2 we show an example of plan adaptation for this task with the two policies, using the mental model in table 10.1

To properly compute the cost of a plan, the planner will also upgrade accordingly the knowledge of a task once it is added to the plan. This allows the *efficiency* policy to prefer plans with repetitive decompositions, that are assigned to the same user, over plans with more variable decompositions, where repetitive tasks are assigned to different agents.

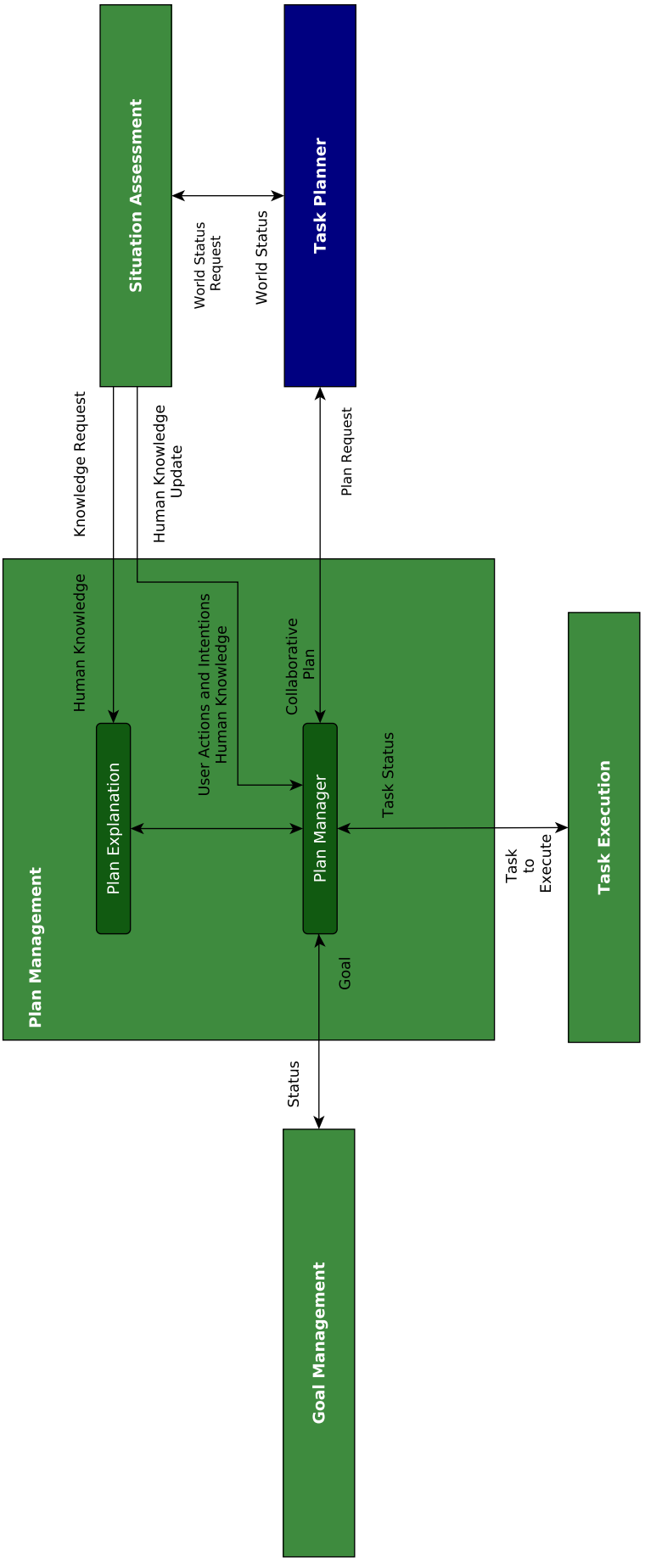


FIGURE 10.1: The architecture of the Plan Management layer integrated with the Plan Explanation module, and with human's knowledge information. Light green rounded rectangles represent modules, while dark green rectangles layers. The Task Planner is shown as a blue rectangle to indicate that it is a module external to the system. Arrows represent message exchanged between components, with the label detailing the message.

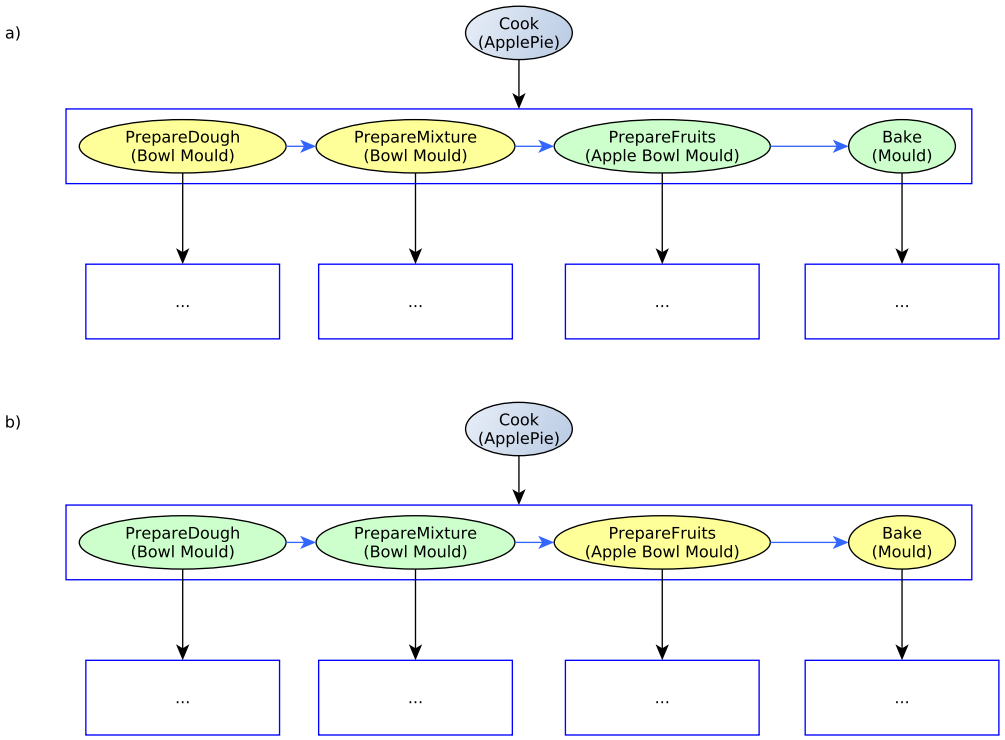


FIGURE 10.2: Allocation of tasks of a plan to cook an Apple Pie for two agents, a robot and a human, using the knowledge values in table 10.1. A) shows an *efficiency* policy for this task, while b) shows a *teaching* policy for the same task. Blue ellipses represent tasks involving both agents, green ellipses represent tasks assigned to the human, and yellow ellipses tasks assigned to the robot. Each decomposition is grouped in a blue rectangle. Black arrows link a method to its decomposition, while blue arrows represent casual links. Squares with a '...' label represent decompositions that are not shown in this picture.

agents(root) + have_to + root	"We have to cook an apple pie."
introduce_presentation	"I will tell you the steps."
agents(child[0]) + first + child[0]	"You will first fetch the ingredients,"
then + agents(child[1]) + child[1]	"Then I will assemble the apple pie,"
finally + agents(child[2]) + child[2]	"Finally, you will bake the apple pie in the oven."

TABLE 10.2: Presentation of a plan to cook an apple pie. Root is the root of the HTN tree and child is a list with its children.

10.4 Plan Presentation

Before executing the plan, the robot will present the goal and the proposed allocation of high-level tasks to give a global view of the chosen strategy. Standard natural language generation is used, as shown in table 10.2. In some situations, the plan will be too long to explain it in one step to a user, who could then be confused or annoyed. For this reason, when presenting the plan, the robot will verbalize only the first N highest level tasks. We have chosen N=3 based on our own experience during tests with the system, but we can imagine to adapt it to the complexity of the domain, or even to specific users. The robot will present the first steps of the plan, and then execute them. Once these tasks have been executed, the robot will repeat the present/execute process until the plan is completed or aborted.

10.5 Plan Management and User Knowledge

We start this section by showing how we adapted our plan management algorithm to take into account the human's knowledge in the task. After showing the algorithm, we will explain it. During this algorithm we will sometimes discuss about the consequence of a user *failing* at executing a task. We deduce that a user fails if he executes a different action than the one the robot is expecting, or if he does not execute any action for a long period of time (determined by setting a threshold).

```
1:  for n:=nodes.start to n:=nodes.end do
2:      if agents(n) = {robot} then
3:          if children(n) ≠ ∅ ∧ user_kn(n) = new
              ∧ teachPolicy then
4:              execute_tree(children(n))
5:              user_kn(n) := beginner
6:          else
7:              execute(n)
8:          end if
9:      else if user_kn(n) = new then
10:         explain(n)
```

```
11:         if children(n) ≠ ∅ then
12:             execute_tree(children(n))
13:             user_kn(n) := beginner
14:         else
15:             monitor(n)
16:         end if
17:     else if user_kn(n) = beginner then
18:         if propose_explain(n) then
19:             user_kn(n) := new
20:             (...)                                   ▷ Same process as new
21:         else
22:             monitor(n)
23:         end if
24:     else if user_kn(n) = intermediate
              ∨ user_kn(n) = expert then
25:         monitor(n)
26:     end if
27: end for
```

- *execute_tree*(n) is the main plan management function. The function receives as argument *nodes*, a list of nodes initially filled with the root's children.
- *teachPolicy* is a boolean that defines if we are in teaching or efficiency mode.
- *agents*(n) returns the agents involved in the node n.
- *verbalize*(n) will verbalize the current task, using the node context to present it (e.g. using sequential relations such as first, then or finally according to the node position in the list).
- *user_kn*(n) returns the knowledge level of the user concerning the task n.
- *propose_explain*(n) will lead the robot to propose an explanation for the current task. If the user accepts the explanation it will return true, and otherwise false.
- *explain*(n) launches a procedure to explain the current task to the user. This procedure could be implemented as a script to launch a video, an explanation speech or even to ask an expert to explain the task.
- *monitor*(n) starts monitoring the proper execution of the current node. If the request returns a success, the function will upgrade the user's knowledge and the *execute_tree* function will continue.

In case of failure, the function will downgrade the user's knowledge, exit the *execute_tree* function, and return a failure that will result in a replan request and a new execution if a plan is found.

- *execute*(n) works in a similar way to the monitor but sends a request to execute the node by the robot.

10.5.1 Explanation of the plan management algorithm

This plan management algorithm is based on the structured plan tree computed by the task planner, and on the tasks' knowledge values in the user models. We explore this tree with a pre-order strategy. The algorithm will analyze each node, with several possible outcomes:

10.5.1.1 Only the robot is involved (lines 2- 8)

If the robot is the only agent in charge of the current node, the collaborator has a knowledge level equal to *new* for the current task, and the chosen policy for the interaction is teaching, then the robot will execute the subtasks in 'demonstration mode, meaning that it will verbalize each child task before performing it.

Once the task has been executed, the robot updates the human's knowledge on the current node to *beginner*. The same process will be applied to the tasks' children. Using this process, the robot will verbalize each (and only) task that needs to be learned by the collaborator.

If the robot is in charge, but the human collaborator's knowledge on the task is different from *new*, or the current policy is efficiency, the robot will verbalize only the high-level task it performs.

If the human is involved in the current node, the robot's behavior will depend on the human's knowledge level for the task, since he might need explanations. Explaining a task could be done in several ways: showing a video, asking an expert to explain the it or simply verbally guiding the user, step by step.

10.5.1.2 The collaborator's knowledge level on the task is *new* (lines 9- 16)

If the human has the level *new* for the current task, we explain it. When verbally guiding the user, if the current node has only one child, we go deeper in the tree and apply again the corresponding behavior according to the knowledge level. If the current node is actually an operator (a leaf), the system monitors the current action execution. In case of success, the knowledge level for the task is upgraded to *beginner*.

10.5.1.3 The collaborator's knowledge level on the task is *beginner* (lines 17- 23)

If the human has the level *beginner* for the current task, we ask if he needs explanations. If so, we downgrade his knowledge level to *new* on the current task and apply the same process as the previous

paragraph. If the user refuses explanations, we simply monitor the execution of the current node. In case of success, the knowledge level for the current task is upgraded to *intermediate*. This knowledge level will also be used as default. This way, when the robot does not know the knowledge level of an agent concerning a task, it will just ask him if he needs an explanation and adapt its behavior accordingly.

10.5.1.4 The collaborator's knowledge level on the task is *intermediate* (lines 24- 26)

If the human has the level *intermediate* for the current task, we verbalize it without proposing explanations, since he has already succeeded with the plan at least once without help. Also, we do not go deeper in the tree and directly monitor the current task. If the user fails, we downgrade his knowledge to *beginner*, otherwise we upgrade it to *expert*.

10.5.1.5 The collaborator's knowledge level on the task is *expert* (lines 24- 26)

In case of an *expert* knowledge level on the current task, we proceed as for the previous knowledge level, downgrading the level to *intermediate* if the user makes a mistake and keeping the *expert* level if he performs the task as expected.

10.5.2 Replanning

When the plan manager reports a failure, the system needs to look for a new plan. If possible, we would like to create a plan which contains the same high-level task allocation. Sometimes, when the Plan Manager fails, there is no need to completely change the plan. Instead, it might be sufficient to repair only a part of it. For example, we can imagine a scenario where a human might execute task t_1 using resource r_1 or r_2. Perhaps the robot computed a plan where the human will use resource r_1, while itself will use r_2 to achieve another task.

If the human does not follow this plan, and uses r_2 to compute its task, it might be faster to just repair the plan, looking for a solution where the task allocation is the same, but the robot uses r_1 instead to achieve its task. This way, we avoid starting a new explanation phase, which could look useless and not natural to the human collaborator.

In HATP, we introduce a new social rule. When presenting a plan, we record which nodes have been presented and assigned to each agent. While planning, we penalize plans with a different task allocation. This way, the planner will prefer to use the same task allocation, but will change it if needed, for example because, using the previous task allocation, the goal is no longer achievable.

Chapter 11

Experiments and Results

In this chapter, we present a user study performed the capacity of our system to adapt plan explanation and management to the human's knowledge in the tasks to perform. Section 11.1 presents our scenario. Section 11.2 explains how we created and conducted our user study. Finally, section 11.3 presents and discusses our results. This study was presented before in Milliez et al. (2016).

11.1 Scenario

To test our system, we have chosen a scenario where a human is trying to prepare two desserts, an apple pie and a banana pie, without knowing their recipes. The robot's goal is to teach the human how to prepare the two dishes, by explaining him the tasks that he has to execute, and by guiding him during this process, as shown in figure 11.1.

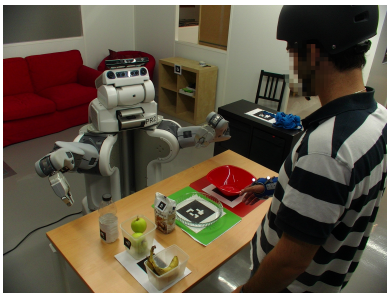


FIGURE 11.1: Illustration of the cooking pies scenario

We created a domain to represent this scenario, and used it with the previously explained *efficiency policy* and HATP as task planner. The process of cooking an apple pie involves five main tasks. We will now present an example of a simulated run in this scenario, to show how our algorithms would perform in this domain. In this example, we set the world state and chose which activities the human performs at each time.

We imagine that, in our set-up, the robot is not able to execute the *PrepareDough* and *PrepareFruits* tasks, since it can not reach the needed ingredients. HATP produces a plan, allocating the tasks as follows:

1. The human will prepare the dough, kneading it and putting it in the mould.
2. The robot will prepare a mixture with butter and sugar, putting the ingredients in the mould.
3. The human will then prepare the required fruits, cutting them and adding them to the mixture..
4. Then, the human will prepare the dough for the top of the pie.
5. Finally, the robot will bake the pie, by putting it into the oven and setting a timer.

After completing task 1, the human's knowledge on how to prepare the dough will be improved. Consequently, during the execution of task 4, the robot will ask the user if he needs help to prepare the second dough. We imagine he answers "no". The robot does not explain the task and the system will upgrade the human's knowledge level for the *PrepareDough* task to *intermediate*, after he completes its execution. The human's knowledge of task 3, *PrepareFruit*, will be represented as *human1 PrepareFruits [fruit] VALUE*. We consider the parameter of this task as *class-link* , since the process will be the same for any fruit (cutting and putting in the mould). This way we use, as parameter, the class *fruit* instead of the actual instances used in the task, *apple* or *banana*.

After cooking the first pie, the robot generates a plan to cook the banana pie. This time, we set the environment in order to allow each agent to perform all the tasks. Preparing a banana pie is a similar process to the apple pie. The parameters will, of course, be different, involving bananas and not apple. Also, the banana pie does not have a second dough on its top, and it has a different baking time from the apple pie. The plan generated is presented in figure 11.2. We can observe that the planner took into account the experience acquired by the human when preparing the apple pie, by assigning him the tasks (*PrepareDough* and *PrepareFruits*). During execution, since the user has an *intermediate* knowledge level on *PrepareDough*, the robot will not explain it. When the human is about to execute *PrepareFruits*, the robot will propose to explain him the task, since he only performed it once, and has a *beginner* knowledge level on it.

11.2 User Study

We have conducted a comparative user study in order to have a first evaluation of our system's adaptability by users. Two groups of users were asked to participate in the two-pies scenario. The first group interacted with a simulated robot equipped with a basic system (BS). BS has the same behavior as our

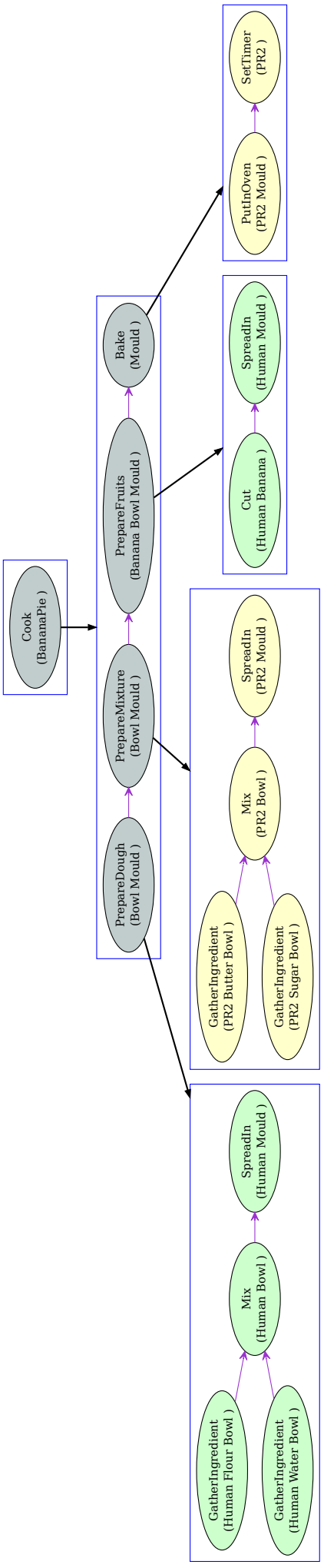


FIGURE 11.2: Shared plan generated to collaboratively make a banana pie.

system, excluding the agent knowledge awareness mechanisms. The second group interacted with another system, which we will call knowledge system (KS), that exhibits similar characterstics to the presented system.

The hypothesis of our study is that, on average, users will express a preference on KS over BS. More formally, we set our null H_0 and alternative H_A hypotheses as follows:

- H_0: $\mu_{KS} - \mu_{BS} = 0$
- H_A: $\mu_{KS} - \mu_{BS} \neq 0$

where μ_{KS} and μ_{BS} are the average preference of users for the KS and BS systems.

In both systems, we generated a plan for the users which uses the same task allocation presented in section 11.1 to make an apple pie. For cooking a banana pie, KS will use the plan presented in figure 11.2, where the human performs tasks he has already executed while preparing the apple pie. In BS, instead, we generated a plan that does not take into account human knowledge, and allocated tasks differently, by making the human prepare the mixture instead of the dough.

Two groups of 19 participants, from 18 to 60, interacted with each system in an online user study[1], where we presented pictures of the task state and recordings of the robot's speech, in French, for each step of the interaction (as shown in figure 11.3). At some steps, the user could choose the action to perform, allowing him to execute a wrong action, leading to a replan from the robot. For simplicity, the replan just corrected the wrong action, before resuming the previous plan. At the end of the simulated interaction, we have asked the same questions to both groups, concerning the adaptability of the system and the robot partner itself. The users gave marks along a Likert scale from one (disagree) to five (agree) to express their agreement with several statements (as shown in figure 11.3).

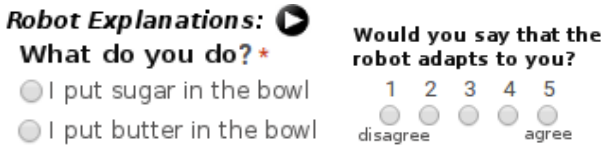


FIGURE 11.3: *Left*: The user listens to a recorded robot explanation and chooses the action to take. *Right*: At the end, the user evaluates the interaction using a Likert scale.

11.3 Results and Discussion

We collected the answers from each form, and computed the mean, along with the standard deviation and p-value to evaluate the system. The p-value was computed using a t-distribution with 18 degrees of freedom and evaluated using a significance value $\alpha = 0.05$. Figure 11.4 summarizes the results.

[1]User study for KS http://goo.gl/forms/qvbtu4vcFW, and BS http://goo.gl/forms/ZSvGcCi5le

Comparing users' answers, we can see that users appreciated the capacity of the system to explain the plan while adapting to their knowledge, with a mean of 3.74 for KS against 2.05 for BS. The users interacting with KS globally noticed that the task distribution took their knowledge into account, by giving a mean rating of 3.42 for KS and 2.58 for BS. The last question concerned the freedom to choose how to perform the task. In this case, the calculated mean was 2.58 for KS and 1.89 for BS. In all these cases, the p-value was lower than the α.

With KS, the users attributed a mean of 3.11 for the global adaptability of the system against 1.89 for the basic one. We also asked how the robot partner was perceived. While in KS the robot is not perceived as more verbose (2.53 for KS against 2.47 for BS), people found the interaction slightly more natural (2.74 against 2.42) and the robot appeared smarter (2.79 against 2.26). Even if these last two results look favorable, since their p-value is higher than α we do not have enough evidence to prove a difference between the two systems. We believe that other aspects might have been taken into account by the users, such as the speech itself, which conditioned their perception on the naturalness of the interaction. Improving the robot's verbalization process with a synonym dictionary could be a first step to get more significant results.

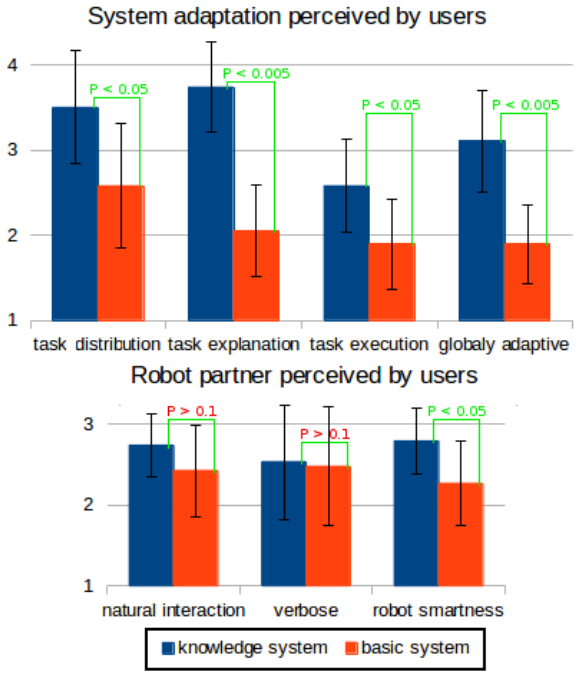


FIGURE 11.4: Average users' rating of the interaction on several criteria. Blue for KS and red for BS.

This study sheds light on how users were able to perceive the robot adaptation to their knowledge concerning task distribution, task explanation and monitoring. In addition, the robot partner was perceived as smarter and the interaction seemed a bit more natural to the users. However, these first results need to be confirmed with study on a larger population. Also, as the scenario was simulated, results on a real robot might differ. In both studies, we asked the participants how the system could be improved.

Several users suggested they would like to be able to choose which action to perform, showing the importance of negotiation. A user suggested he would like to be informed about the progress of the task from time to time. Other comments concerned suggestions about the robot's speech capacities, like its voice, intonation, and the chosen words. These aspects were not the aim of our experiment but are indeed an important part of the interaction process.

Part V

Case Study: The SPENCER Project

In this part, we present an application for a human-aware robot guide, where we used parts of the system shown in the previous chapters, adapted for this scenario. This study was done as part of the european project SPENCER[2].

SPENCER was born with the idea to study social awareness issues in human-robot interaction, and particularly in the application of a robot guide. This project aimed at joining complex robotic algorithms with social signal processing, by modeling people not as objects, but as entities with relationships, social rules, and culturally diverse backgrounds. The final goal of the project was deploying a robot guide, in collaboration with the KLM airline[3], in the Schiphol airport of Amsterdam, performing user studies on real passengers to evaluate the validity of the system.

Chapter 12 will explain how we built a human-aware robot guide, starting from our system, while chapter 13 will present our experiments in a laboratory setting and in the airport.

[2]http://www.spencer.eu/
[3]https://www.klm.com

Chapter 12

Human-Aware Robot Guide

In this chapter we present a case study for our system: the human-aware robot guide. Section 12.1 presents the subject and the motivations of our application. Section 12.2 shows how we adapted our architecture to scenario. Section 12.3 introduces a new intention recognition algorithm developed for this task, based on environmental informaton. The scenario required a different approach to task planning than the one we developed, and we show our solution in section 12.4. To guide humans, we developed a new collaborative planner, which is shown in section 12.5.

12.1 Introduction

12.1.1 Overview on the topic

One interesting problem in human-robot interaction is developing robots able to guide humans, by offering a tour of attractions in an area or simply by helping to reach a destination. A generic mobile robot platform should possess a vast set of skills, which includes advanced perception, motion planning, and task planning. These skills are not enough for a robot guide, which is deployed in highly dynamic human environments, and need to be complemented with human-aware behaviors.

Different robot guides have been studied and developed, starting with pioneers like Rhino and Minerva (Thrun et al., 2000). Few systems have actually been deployed for long period of time in human environments. Rackham (Clodic et al., 2006), a museum guide with human-aware behaviors, is an example of such system, and has been deployed in a science museum for several months. Rackham's algorithms, like voice and face recognition, enable it to estabilish a form of relationship with his users, and to consider the task as a collaborative activity. For example, Rackham will stop if the user goes away and waits for a period of time, resuming the task if it recognizes the user coming back.

Another example of robot guide was developed in Bueno et al. (2011), where the robot was integrated in a smart museum environment, using virtual avatars with human-aware interfaces, associated to exhibitions, in order to convey information to users.

After these first experiments, several researchers have tried to focus on the social aspects of the problem, which are especially important if the robot needs to offer information. Studies like Evers et al. (2014), Yousuf et al. (2012) focus on how the robot should address humans, concentrating on spatial relationships and on how to convey information. Jensen et al. (2005) present a robot guide able to convey emotions to user, by joining perceptual information with the internal state of the robot.

Building and using mental models of users is important in these scenarios, particularly if we are developing a proactive robot, which approaches people in order to offer its services, while monitoring their level of interest in the interaction. In Rashed et al. (2015), a robot guide is able to infer people's intentions by studying their trajectories. The authors conducted experiments in a museum, managing to classify user trajectories in three categories: users interesting in looking at a collection of paintings, users looking for a particular painting, and users visiting the museum by chance. By recognizing the trajectory of a user the robot infers if he could be interesting in a guided tour.

Recently, there has been emphasis on robot navigation algorithms that reason about human beings in the environment differently from other static or dynamic obstacles. Starting from proxemics, researchers have investigated explicit social signals, based on human-posture and the affordances of the environment, to improve the legibility of the robot's motions. For a detailed discussion on human-aware navigation algorithms we refer the readers to Kruse et al. (2013), Rios-Martinez et al. (2014). Human-Aware navigation in a museum was studied in Samejima et al. (2015), where the authors built environmental maps, which included information learnt from human trajectories and postures, in order to plan safe paths that do not disturb humans present in the area.

The robot guide scenario can become more complex if we consider not only single humans, but groups. Humans, in fact, tend to naturally form groups, which can be classified in different types, based on their size, on their level of cohesion, on their duration, and other factors (Forsyth, 2009). Studying group formations is a complex problem, since the robot must be able to detect humans, which can be occluded in populated environments; and to model their social interactions, which can be ambiguous. One of the most powerful and expressive social cues is distance, used in different works to track social relationship, like Luber and Arras (2013).

12.1.2 Motivations

We believe that most robot guide systems are focusing on the social aspects of the problem, and on human-aware navigation, without fully considering the fundamental aspects of joint actions. Guiding is

a collaborative task, where the robot does not need only to reach a destination, but also to ensure that its followers reach it, while providing a socially acceptable experience to them. In order to achieve this goal, the robot needs to constantly monitor its users, to adapt to their behaviors and to be ready to proactively help them.

We applied our system to this problem, creating a human-aware robot guide which is able to lead a group of people to a destination. More particularly, the originality of our approach is that the robot is able to show both adaptive and a proactive behaviors. The robot will try, while guiding, to select a speed that pleases its users, when adapting, or to propose a new speed, using environmental and task related stimulus. Finally, our system will proactively try to engage members of the group if it detects they need assistance.

12.2 Building a Robot Guide

In order to adapt our system to the robot guide scenario we had to enhance several modules, as shown in figure 12.1:

- The user is able to interact with the robot by using a tactile interface on its front. Partners in the project developed an interface that connects this tablet to requests for the robot.
- In this scenario we are not really interested in the intention recognition skills that we developed in chapter 4. In an airport there are no complex sequence of actions that humans need to perform and we will evaluate users' intentions by their trajectories and the surrounding environment. We developed an Environment-Based Intention Recognition module to enhance our Situation Assessment layer, presented in section 12.3. The geometrical reasoning skills of the Situation Assessment layer will, instead, be necessary to properly guide the robot's followers.
- Similarly, task planning and plan management is quite simple in this scenario. The robot will mostly vary its paths in the environment, using similar plans. We developed an A*-based task planner, which is able to find a path in a semantic map, to guide users to their destination, presented in section 12.4. The Plan Management layer will receive this plan and communicate with the Task Execution layer to move the robot toward the goal.
- The main work, in this application, has been creating a Collaborative Planner to guide users in a human-aware way. Using this planner, our robot is able to adapt itself to users' actions, or to proactively propose new behaviors. This planner is presented in section 12.5.

The system has been tested with different configurations, in laboratory experiments and in the Schipol airport, presented in chapter 13. Parts of this section were presented in Fiore et al. (2015).

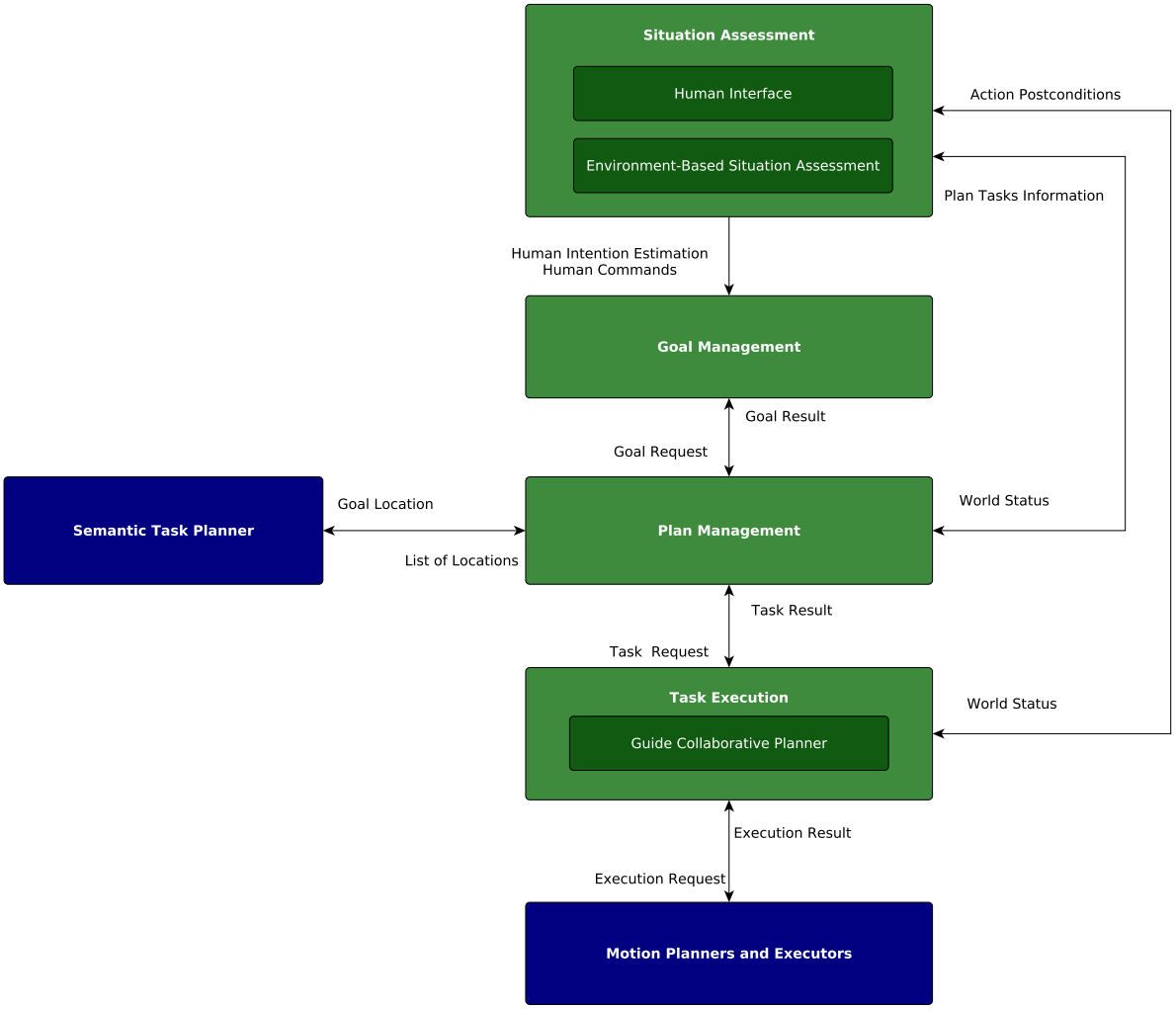


FIGURE 12.1: This image shows the main layers of our architecture, represented as dark green rectangles, as shown in chapter 2. In each layer, represented as light green rectangles, we show the modules that we modified or introduced in the robot guide scenario. Blue rectangles represent external modules. The Semantic Task Planner was specifically created for this scenario.

12.3 Environment-Based Situation Assessment

There are many situations where, to properly reason on humans, we should link their movements and actions to the current environment. Imagine for example the case where we see a person oriented toward a screen. In an airport, we could infer from this observation that the person is looking at the screen, and perhaps in need of information.

We introduced, in the Situation Assessment layer, the ability able to create activity areas in the environment and link them to different kind of computations. An activity area is a polygonal or circular area, which can be fixed or linked and updated with an entity's (object, human or robot) position. We studied and experimented three activity areas:

- Information Screen Area. This area, shown in figure 12.2, is linked to information screens present in the environment. Using this information, the robot can recognize that humans are looking at

the screen, and start a proactive behavior, like approaching to offering help or information.

- Touristic Point Area. These areas are linked to interesting sights and attractions in the environment. We can imagine, for example, to associate these areas to paintings, statues, or even rooms in a museum. Knowing that a human is in a touristic point area, and looking at an attraction, the robot could approach the human and tell him some interesting information about it.

- Robot Area. We assumed, at a first step (wrongly, as explained in subsection 13.2.3), that users would follow the robot mainly from behind, and sometimes from its sides. We created a polygonal area and linked it to the robot. This area is a trapezoidal figure, as shown in figure 12.3. We will explain its use in subsection 12.5.1.

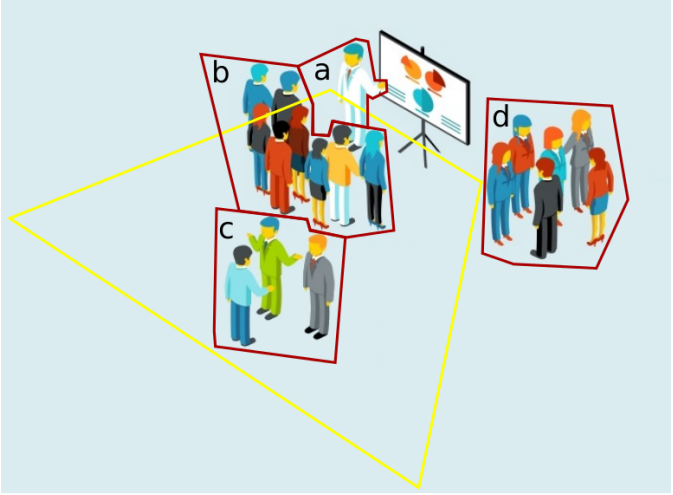


FIGURE 12.2: The Information Screen area used for environment-based intention recognition. The yellow polygon represents an area linked to the screen in the figure. Red polygons represent groups of persons in the area. The person in group a is in the screen area, but the robot will infer that he is not looking at the screen, since he is oriented in another direction. The persons in group b are in the screen area and oriented toward the screen, so the robot infers that they are looking at it. The persons in group c are not looking at the screen since they are either oriented in another direction or outside its area.

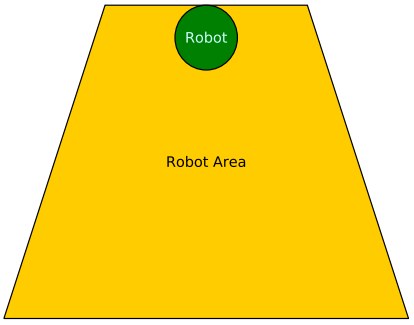


FIGURE 12.3: The activity area associated to the robot, used as a basis for understanding which users are following.

12.4 Task and Motion Planning Problems

To guide people in environments like museums or airports, the robot will need to navigate very large areas, which are often too big to efficiently perform motion planning. A solution to this problem is reducing the size of the area where the motion planner will compute its paths, splitting the navigation problem in a list of sub goals. The problem, with this approach, is choosing the correct list of sub-goals to reach the final position.

To deal with this issue we have implemented an A*-based task planner, which will choose a high-level path to be followed by the robot from a semantic map. This map is a hand-crafted graph, composed by different nodes, that represent parts of the environemt (corridors, elevator entrances, gates, etc.). Each node will be linked to a point in the real world. This task planner will, so, produce a list of coordinates usable by the motion planning layer.

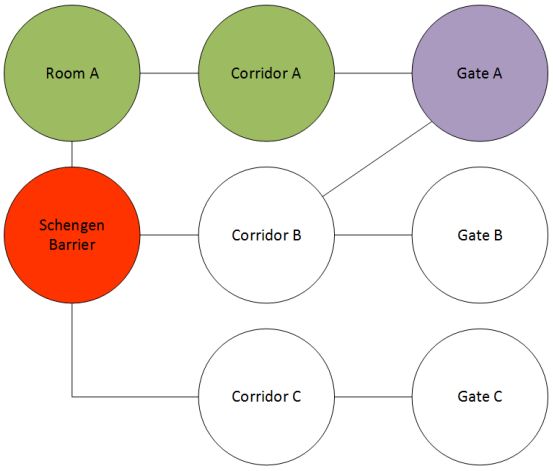


FIGURE 12.4: An example of plan computed in the robot guide scenario. The circles represent nodes in the semantic map. The purple node is the starting node of the plan. The red node is the goal. The green nodes represent intermediate nodes in the plan.

A first way to manage this kind of plan is simply travelling to each calculated point, sending a goal to the motion planner for the next point every time the robot reaches its sub-destination. The problem with this approach is that the path followed by the robot might be very inefficient and not look natural.

Our solution was using a rolling window approach, shown in figure 12.5, where the motion planner will compute paths on a grid map centered on the robot, which will be updated with its position. In this way, the robot will send the next sub-goal in the calculated plan as soon as it is present in the rolling window, without waiting to reach the previous sub-goal. It is very important, with this approach to carefully select the nodes in the semantic maps so that the rolling window will contain at least two semantic nodes. If not, the motion planner would have to plan a path to a goal outside its grid map, which would generate an error.

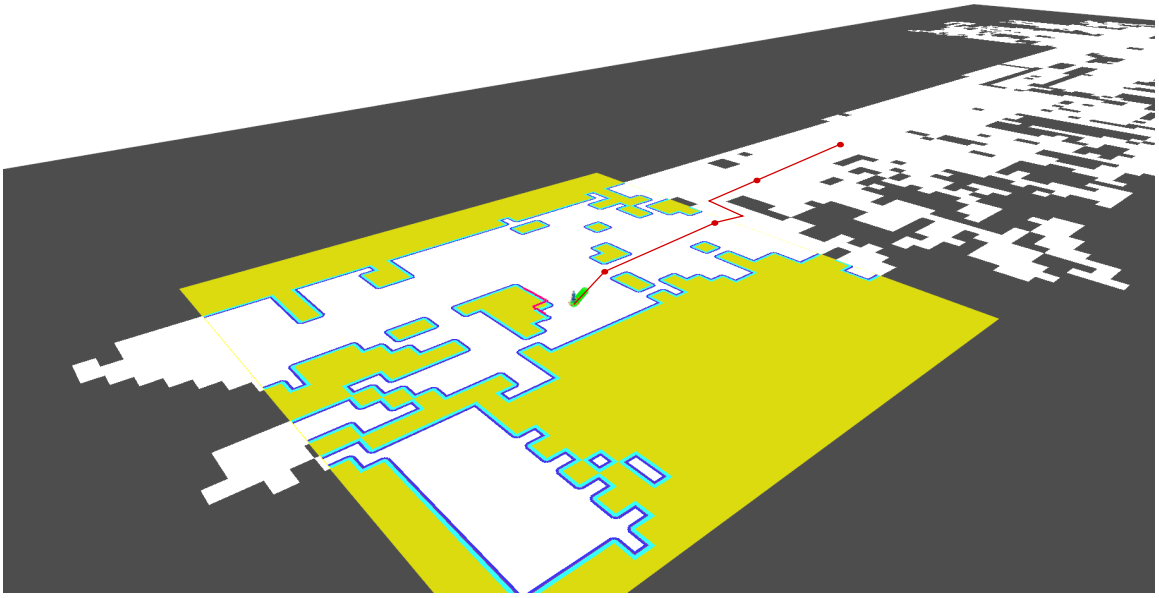


FIGURE 12.5: This figure shows the rolling window approach. The motion planner will compute robot paths in the grid represented by the yellow area, which is always centered on the robot. The red line and the red dots represent the sequence of goals calculated by the task planners. Each goal from this path is sent to the motion planner as soon as it is inside the window. The robot path does not pass through the closest node, since there is a further one in the rolling window.

12.5 Collaborative Guide Planner

We have developed a new collaborative planner to guide users to a destination. The planner is organized as a set of hierarchic modules, as shown in figure 12.6 and is composed by several models:

- Guide Group Model. The main MOMDP of the hierarchy, responsible of choosing the main action performed by the robot.
- Speed Adaptation Model. This model chooses if the robot should accelerate, decelerate, or keep the current speed.
- Suspend Model. This module chooses which actions to execute when the users has stopped following.

12.5.1 Representing a group

Our system has been built with the assumption that the robot would navigate in a crowded environment, where its followers could be often occluded. In addition, after speaking with our partners, we understood that the perception components would not be able to maintain a stable identifier for members of the groups, meaning that the robot would not be able to reliably understand if a group of users is following it. We tried to maintain a good balance between accuracy, trying to *guide* effective users and not people

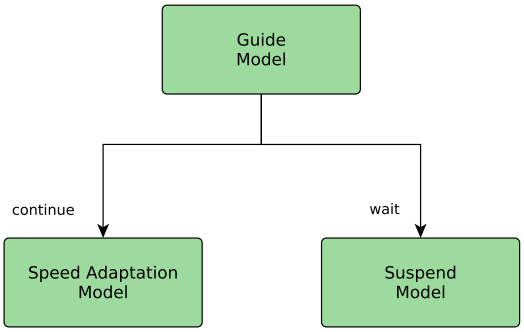


FIGURE 12.6: This figure shows the hierarchical collaborative planner used to guide a group. The arrows show the macro-actions of the root of the hierarchy.

which are simply going in the same direction as the robot, and robustness, copying with unreliable perception.

We built our collaborative planner following these ideas:

- The robot will record a list g composed by the identifiers of the group at the start of the scenario. During this scenario, it will maintain a list f, composed by users that are still tracked whose identifier is in g, and another list o, composed by all the tracked people whose distance from the robot is less than a constant d.

- As long as f is not empty, the robot will choose a *best* follower b from this list, and try to guide him, using the collaborative planner. The *best* follower is chosen by considering the user whose behavior is the most consistent with the robot, by following a set of rules:

  1. If there is only one agent a in f so that $a\_isIn = robot\_area$ and $a\_isMoving = true$ AND $a\_isTowardRobot = true$ he is the *best follower*. If there are more agents that satisfy this condition, the *best follower* is the closest to the robot. Our preferable condition for best follower is somebody who is moving toward the robot and staying behind it, as we assume that he is following it.

  2. If no agent satisfies condition 1, but there is an agent a in f so that $a\_isIn = robot\_area$ AND $a\_isTowardRobot = true$ he is the *best follower*. If there are more agents that satisfy this condition, the *best follower* is the closest to the robot. If there is no one that is moving toward the robot, we choose an agent that is not moving but still behind it and oriented toward it, as he could just have stopped for some moments before resuming following.

  3. If no agent statisfies condition 1 or 2, but there is an agent a in f so that $a\_isIn = robot\_area$ AND $a\_isMoving = false$ AND $a\_isTowardRobot = false$ he is the *best follower*. If the previous conditions are not statisfied, we choose an agent that is not moving, behind the robot, and not oriented toward it. There could be humans that have stopped following and are speaking between themselves, or looking at a screen.

4. otherwise, *the best follower* is the closest to the robot.

- If f is empty, but o is not, the robot will choose a *best* follower b from o, by following the rules that we just introduced (substituting f for o). If the robot has lost track of its group, it will try guiding other persons that have a behavior consistent with that of a follower.
- If both f and o are empty, the robot will abandon the task.

Our idea is considering that the robot's followers will *act as a group*, staying together as much as they can. The robot will try to guide members of the group, but if it has lost tracking of all of them, and there is still somebody behind it which is acting in a consistent way with the guiding task, it will think that its users are still following it, but their identifiers have changed in the perception layer. We represent this idea by tracking every user, even those whose identifier was not in g, that are closer to the robot than a constant d. This constant was set as 8 meters after performing experiments in three different environments: our laboratory, a university hall, and the airport.

12.5.2 Guiding a User

The main problem of the robot is choosing if it should still guide the group, suspend temporarily the task, or abandon it. The Guide Model is the root MOMDP of our architecture and will make this decision. We will describe this model, which is based on the Collaborative Planners basic framework introduced in section 7.4.

- $S_h : \{human\_commitment\}$. The hidden variable, and its values, follow the basic framework for the Collaborative Planners..
- $S_o : \{task\_advancement, timer\}$.
  - $values(task\_advancement) = \{not\_completed, completed\}$.
  - $values(timer) = \{not\_expired, expired\}$.

  The state variables are similar to the basic framework. The task advancement contains only two values: *not_completed* and *completed*. When this model is invoked we assume that the group will already have set a goal and *committed* to perform the joint task. The Task Executor will keep track of the location of the robot, and set the task as *completed* when the goal is reached. We introduce a timer variable, as used in the Handover Collaborative Planner (shown in section 7.5), that is controlled by the Task Executor. The idea behind this choice is that we do not want the robot to be stuck waiting for users for too long. This timer was set at 30 seconds in the Task Executor, after experimenting with different values in our tests.

- $A : \{continue, wait, abandon\}$.

  These three actions follow the same idea as the basic framework, but, in this case, the *continue* and *wait* actions are actually macros, which invoke, respectively, the Speed Adaptation Model and Suspend Model. There is no action to *engage* users. The robot will provide information (visualizing what it is currently doing and the time needed to reach the goal) at all time using routines in the system. The robot will *abandon* the task when it is completed or when all the users have left.

- $O : \{delta\_distance, distance, orientation, is\_moving\}$.

  - $values(delta\_distance) = \{unknown, decreasing, stable, increasing\}$.
  - $values(distance) = \{close, far, outOfRange\}$.
  - $values(orientation) = \{unknown, towardRobot, other\}$.
  - $values(is\_moving) = \{unknown, notMoving, moving\}$.
  - $values(in\_robot\_area) = \{true, false\}$.

  The group engagement is estimated through observations, obtained from the *best follower* (as explained in the previous section). Our observations are the *delta_distance* of the user (e.g. the variation of the distance) to the robot, his distance to the robot, his orientation, and if he is moving or still. When there is no *best follower*, these values are *unknown*, for the *orientation*, *is_moving*, and *delta_distance* observations; *outOfRange*, for the *distance* observation; and *false* for the *in_robot_area* observation.

  We have chosen the values *close*, *far*, *outOfRange* from experimental studies. We consider the user as *close* if his distance from the robot is less than 4 meters, *far* if it is less than 8 meters, and *outOfRange* otherwise.

  We imagine that a user is engaged if we detect that he is following the robot, meaning that he is moving in its same direction, he is behind the robot, and he is not too far from it.

- R: the robot receives a reward for completing a task, for waiting for not enaged users, and for continuing if the user is engaged and the task is not finished.

12.5.3 Adapting the Robot's Speed

We believe that to be socially acceptable, the robot should adapt its speed to his follower. By setting its own pace at the start of the scenario the robot would risk of being too slow, annoying users, or too fast, which would lead the robot to constantly stop to wait for users, producing an awkward behavior. Adapting to users might not be enough in some tasks, and, depending on the situation, the robot might desire to influence the behavior of its follower, to respect social rules or to accomplish its task in a more

efficient way. It is important to find a balance between these two behaviors, which is the goal of the Speed Adaptation Module.

To adapt to the speed of the group, the robot defines a desired range of distances $[dr_1, dr_2]$ from the best follower b. The distance of b from the robot, $d(b, r)$, will influence the chosen action:

- if $d(b, r) < dr_1$ the model will influence the robot to *accelerate*.
- if $d(b, r) > dr_2$ the model will influence the robot to *decelerate*.
- if $dr_1 < d(b, r) < dr_2$ the model will influence the robot to maintain its current speed.

In our study, dr_1 and dr_2 were predefined values, but they could be learnt and adapted to the users during the task, since different people could prefer following the robot at different distances and positions. During our experiments at the airport, we set these values as 1.5 meters, and 4 meters.

The robot should also not constantly change speed, in order to give time to users to adapt to its new chosen speed, and so we have defined a temporal threshold in which we do not allow the robot to repeat an *accelerate* or *decelerate* action.

The robot changes its speed in increments or decrements of a constant s. In our preliminary studies at the airport, we tested several constants for t and s, and found that increments of 0.1 m/s and a threshold of 1.5 seconds showed the best behavior for the robot. The starting, minimum and maximum speed of the robot where selected, again, with experimental studies, based on the confort and interest of users surrounding the robot, and of the opinions of the research team. The starting speed was selected as 1 m/s, the maximum as 1.3 m/s, and the minimum as 0.7 m/s. The Task Executor will ignore any command to accelerate when it has already reached the maximum speed, and any command to decelerate when it has reached the minimum speed.

We have studied two different situations where the robot can proactively try to influence the speed of the group.

- There is a time limit to reach the destination. In this case the robot must balance the desire to satisfy the group with the task urgency. Different situations will require different policies. For example, in an airport scenario, the robot could prioritize arriving on time, warning users if their speed would render the goal not achievable, while in other situations the robot could try to arrive in time but still avoid to adopt speeds that are uncomfortable for the group.
- The rules of the current environment limit the robot's speed. In this case the robot will avoid accelerating over a set speed even if it detects that its current velocity is considered too slow for the group. For example, the robot could be navigating in a construction zone.

We present a MOMDP model to implement these ideas:

- $S_h : \{human\_intention\}$.
  - $values(human\_intention) = \{slow\_down, maintain\_speed, accelerate\}$.
- $S_o : \{task\_urgency, environment\_limit\}$.
  - $values(task\_urgency) = \{normal, high\}$.
  - $values(environment\_limit) = \{not\_reached, reached, passed\}$.

  The hidden variable of this model is the human intention, which can be to slow down, to maintain the current speed, or to accelerate.

  We introduce the observed variable representing the urgency of the task. Depending on this value, the system might decide to propose a new speed without simply adapting to the human's desires.

  The *environment_limit* variable also informs the robot, when *true*, that it has reached the speed limit for the current environment (i.e. costruction zone, has explained before) and should not accelerate. When the value is *passed*, the robot is going too fast for the speed limits of the current environment. When the value is *false* there is no speed limit in the area or, if there is, the robot's speed is lower than this limit.
- $A : \{accelerate, decelerate, continue\}$.
- $O : \{highest\_density, in\_slow\_area\}$.
  - $value(highest\_density) = \{behind, accelerate\_area\}$.
  - $values(in\_slow\_area) = \{false, true\}$.

  When we designed this model, we envisioned the possibility of guiding with a realiable perception, knowing the identifiers of the members of the group, and thinking that these identifiers would be stable over the course of the scenario.

  We introduced an observation related to the distribution of the locations of the members of the group. This observation can assume the value *accelerate_area* if the majority of the members of the group are closer than d_1 to the robot, or *behind* otherwise. We also introduced the observation *in_slow_area*, which assumes the value *true* if any member of the group is at a distance from the robot higher than d_2.

  The idea of this choice is the following: if the majority of the members of the group would prefer a higher speed, the group intention is more likely to be *accelerate*. If even a single person would prefer to slow down, we consider the group intention as *decelerate*. This choice was made because

we believe that the robot's priority, when adapting to users' needs, should be to guide the *whole* group to the end. So, if a member of the group would prefer slowing down, because perhaps he is having trouble following, the robot will think that the whole group wants to *decelerate*.

In reality, as explained, we have used the idea of *best follower* to cope with perception limits. In this case, the *highest_density* observation will simply depend on the distance of the *best follower*.

- R: We can imagine different policies for this problem. In the airport scenario, we chose to have our robot simply adapt to to users' needs. In this case, the robot will obtain a reward for respecting the *best follower*'s intention (e.g. *accelerate* when the intention inferred is *accelerate*, etc.).

  If we would like the robot to have a proactive behavior and propose a speed, depending on the task needs, the robot would receive a reward if he would *accelerate* when the task priority is *high* and the group's intention is different from *slow_down*. Also, the robot should not *accelerate* if $environment\_limit = true$, and it should *decelerate* if $environment\_limit = passed$.

12.5.4 Suspending the task

In some situations, the robot needs to suspend the task, because the group has stopped following it. In this case, the robot should estimate if this suspension of the collaborative scenario is temporary or permanent, and in the latter case abandon the task. We estimate this information using the Suspend Model and the activity areas from Situation Assessment. We link activity areas to the maximum time we expect that the group will be involved in the linked activity, and with a set of proactive actions that the robot can choose to execute.

In our work, we have investigated a single possible proactive behavior: giving information. In this case, if we detect that one or more members of the group has stopped following because it is looking at a touristic sight, or at an information screen, the robot can try to engage him and offer related information. At the moment, we just propose a simple routine-based framework for this behavior, and plan to further study it in the future. We believe that the solution of this problem could be rich, and that the robot should estimate the reaction of the group during the execution of its proactive behavior, in order to be able to interrupt if the group does not want to be helped or to resume the original task if they are satisfied by the robot's actions.

We do not want the robot to be inactive for a long time waiting for the group. If there is a small amount of time to reach the destination, or the group is engaged in the activity for a longer period of time than the one predicted, or the robot can not estimate the reason why the group stopped following, the Suspend Model can issue a warning action, and eventually abandon the task if the group does not start following it again.

Chapter 13

Experiments and Results

In this chapter, we presents our experiments and results for the human-aware robot guide case study. Section 13.1 shows preliminary experiments performed in our laboratory, while section 13.2 show the real case study in the airport.

13.1 Laboratory Experiments and Analysis

To have a first validation of our system, we performed experiments in a laboratory. In this setting, we used motion capture to track users and a navigation software based on Kruse et al. (2012), Sisbot et al. (2007a). This software add proxemics based costs to static humans in the environment, continuously predicting and avoiding future collisions with moving persons, while simultaneously keeping the robot as close as possible to the planned path.

We performed experiments with a single user following a robot on a predefined path. Data from these experiments are shown in table 13.1 and in figure 13.2. We start by showing speed adaptation tests:

- *Adapting slow and fast.* In these two tests (figure 13.1) we asked a user to follow the robot at a pace that he considered *slow*, and at a pace that he considered *fast*. The robot had to guide this user by adapting to his speed.
- *No adaptation.* In this experiments we asked a user to follow the robot at the speed that he prefers. The robot did not adapt to the speed of the user, setting its own pace and stopping if the user is too far.

Looking at the data we can see that our system shows lower values for the variance of speed and distance, which means that after a certain time it is able to find a condition of equilibrium with the human follower. The *no adaptation* system shows a significantly higher variance for both values, since

FIGURE 13.1: This figure shows the SPENCER robot guiding a user in a laboratory.

the robot stopped several times to wait for a user. We will now show some tests regarding the proactive behaviors of the robot:

- *Proactive slow and fast.* During the task, the robot proactively chooses to change pace, in the first case by slowing down and in the second by accelerating. In our tests the user adapted after some seconds to the robot's pace, but this behaviors should be studied in-depth in user studies.

- *Suspend with screen and with no reason.* In these tests we asked a user to stop during the task. In the first case the user stopped near an information screen. After detecting this event, the robot approached the user to offer information, which lead to the resumption of the task. In the second case the user stopped at a different point of the path. The robot was not able to detect the reason for the suspension of the task and so simply issued a warning to the user, abandoning the task after some seconds.

TABLE 13.1: Experiment results: d is the distance between the robot and the user, s_r is the robot's speed, s_h is the human's speed, μ is the average and Δ is the variation of the quantity. Distances are expressed in meters, velocities in meters for seconds.

test name	μ *distance*	μ *speed difference*	Δ *distance*	Δ *speed difference*
adapting slow	2.82	-0.03	0.64	0.02
adapting fast	1.38	0.00	0.29	0.01
no adaptation	3.08	-0.09	1.04	0.07
proactive slow	1.45	-0.06	0.04	0.10
proactive fast	2.66	-0.11	0.63	0.01

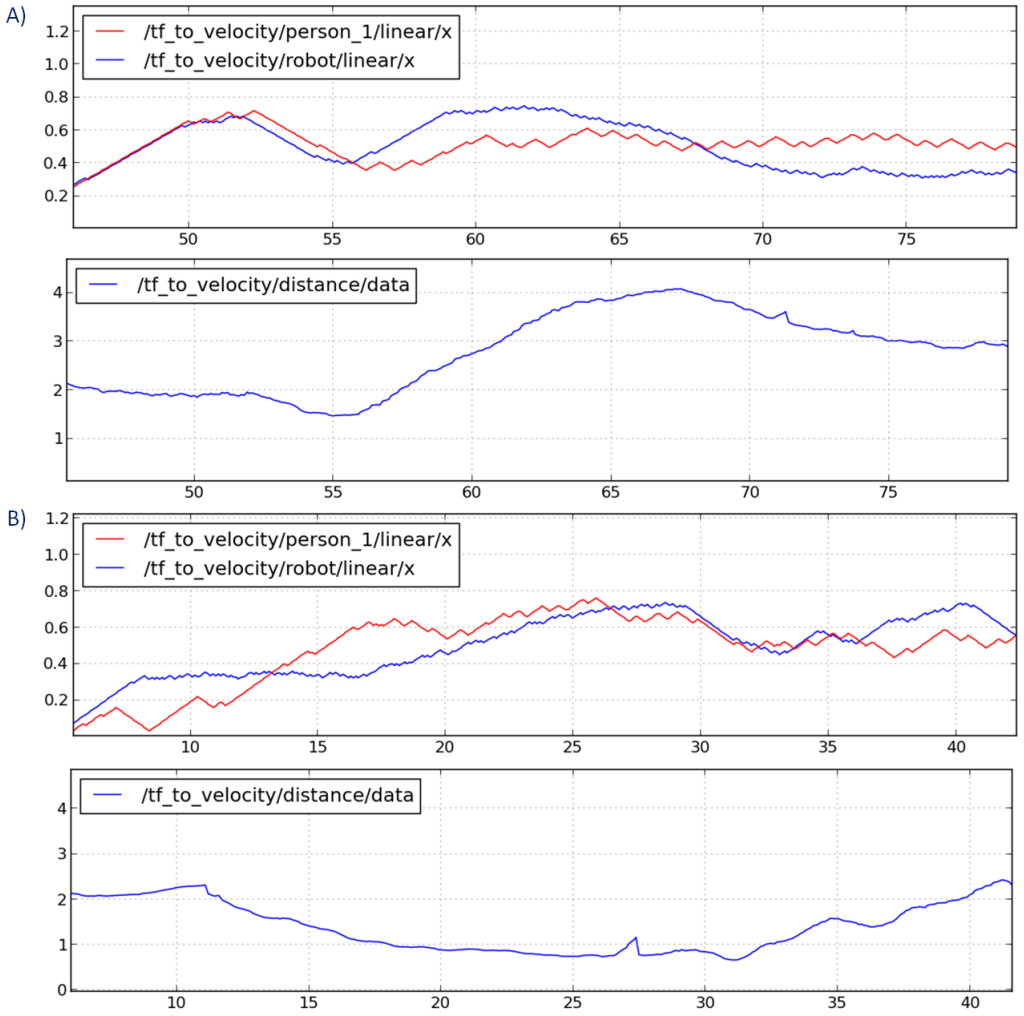


FIGURE 13.2: Robot Guide Laboratory Experiments: a) Adapting robot speed to a slow user. The first figure shows the speed of the user ($tf\_to\_velocity/person\_1/linear/x$) and of the robot ($tf\_to\_velocity/robot/linear/x$), and the second their distance. The robot starts slowing down at $t = 60$, when the distance from the user is growing, until it finds an equilibrium with the user's speed. Notice that there is a turn in the path, at $T = 50$, that causes the robot and the user to slow down. Distances are expressed in meters, velocities in meters for seconds. b) Adapting robot speed to a fast user. As before, the figures show the robot and user's speed and their distance. The robot starts accelerating at $t = 15$ when the distance from the user becomes small.

13.2 Results on Airport Deployment

13.2.1 Integration with Other Components

The system was deployed in the Schiphol Airport for two different sessions, respectively of a week and of two weeks. In those weeks the team of the SPENCER project adapted the robot to the complex environment of the airport. Our system was integrated with several softwares from our partners, such as a combined laser-RGB people tracker, developed in Linder et al. (2016), a novel localization approach, shown in Kucner et al. (2015), a RTT based motion planner, shown in Palmieri et al. (2016), and cost-based social rules, introduced in Okal and Arras (2016). More details on the whole SPENCER system are available in Triebel et al. (2015).

FIGURE 13.3: The robot moving in the Schipol airport

13.2.2 User Study

The system was tested in a user study, performed by researchers of the University of Twente[1], which participated in the project. The study was conducted in two different days, with 18 participants, 11 males and 7 females, aged between 26 and 54. Ten participants indicated that the purpose of their journey was business, while eight indicated pleasure. Two different tests were performed:

- In the first test the robot would pick up users at a chosen point, in Lounge-1 or in the Starbucks coffee shop of the airport, and guide them to a prefixed gate, B18. Users interacted with the robot by scanning special boarding passes created for this test on the robot's board pass reader, which prompted the start of the mission. To reach the gate, the robot had to pass through a crowded area, composed by several shops and other flight gates.

[1] https://www.utwente.nl/en/

FIGURE 13.4: The team that worked on the SPENCER project.

- In the second test the robot met users at the gate B18 and guided them to Lounge-1. In this situation users initiated the mission by using the touchscreen display of the robot to select the destination.

At the end of each test, we collected three different kind of measures:

1. An individual feedback questionnaire where the users evaluated several aspects of the robot, like its behavior and aspect, on a 7-point Likert scale. In the second day of testing two new questions where introduced about the robot, and a question on the participant's opinion on the robot in general.

2. A group interview, where users could discuss about their first impression of the robot, their experience in the test, and their thoughts on how to improve the systemm.

3. Notes taken by researchers during the test, that helped to contextualize the results of users and to record specific events that occurred during the guidance.

Figure 13.5 shows the results of the feedback questionnaire. Two questions were reformulated between the two days of testing, and so we included means for both of them. While the number of tests conducted were too limited to generalize, we can affirm that the participants had in general a positive impression of the robot. We list the main results of this test.

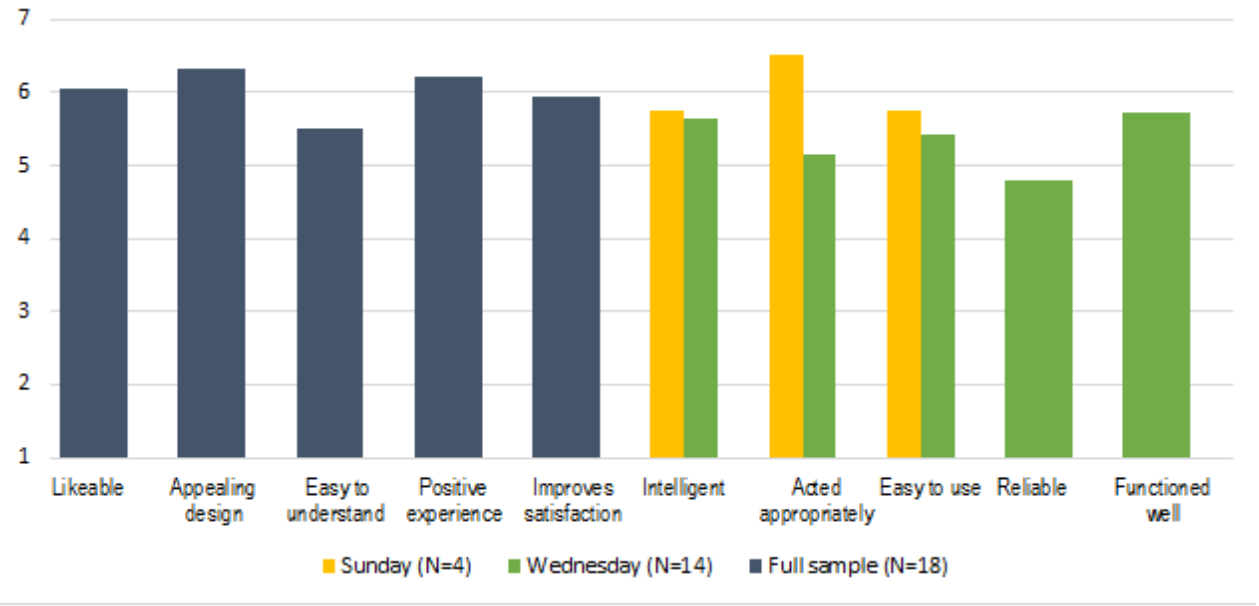


FIGURE 13.5: Answers to feedback questionnaire

- Participants were generally happy with the robot's guiding performance.
- The robot can be described as friendly-looking, easy-to-use and reliable.
- The main issues encountered by users where related to the boarding card reader of the robot, and to its frequent abrupt stoppings, caused often by false positive obstacles by the perception software. These false positives where caused by the complex conditions of light, and by the many reflective surfaces in the airport.
- Some users considered the speed of the robot too fast, or too slow, for their experience.
- Users thought that the robot guide could be useful for people inexperience with flying, or new to a particular airport.
- Users thought that the robot improved their customer satisfaction.
- The main improvements for the robot include technical improvements, in particular related to the frequent stopping, and the addition of new functionalities, like carrying luggage or guiding to shops and restrooms.

13.2.3 Discussion

The experience in the airport showed the complexity of deploying a robot in such a complex scenario, interacting with inexperienced users. From our experiments we noticed several things:

- **It is important to not make too many assumption about how users will interact with the robot**. We expected users to follow the robot, staying mostly on its rear or side regions. This

was not always true. In fact, several users were more interested in moving ahead of the robot, often concentrating on its behaviors more than on following it. The robot should adapt to these behaviors and not consider them as errors.

- **Human-Aware motion in complex environments is still a hard problem**. Adapting the robot's speed was not simple, because the robot had stop or slow down often to avoid obstacles. Some times false positives generated by perception can lead to abrupt stops by the robot, which are seen as annoying and unnatural by users, and could even be dangerous.

- **Users are fascinated by robots**, and are willing to ignore errors generated by the systems in short-term experiences.

- **It is important to perform long deployments in similar scenarios with real users**. In the three weeks where we worked at the airport, many days were spent adapting the robot to the scenario. Longer deployments would allow for an iterative process of design-implementation-study.

Part VI

Conclusions

This part concludes the thesis. Chapter 14 reviews the main points of this work, discussing possible future extension. We also present appendix A that introduces the mathematical models used in our work.

Chapter 14

Conclusions

Human-Robot Interaction is a very complex subject, that involves different problematics. In order to be socially acceptable the robot needs to be able to interact with a human in a simple, efficient, and natural way. In this thesis we presented a framework that allows a robot to perform cooperative tasks with humans. We will now review the main aspects of our system:

- Our system is able to model and maintain a mental belief model for the robot and for all the present agents, using geometrical reasoning to compute relationships between entities. These ideas are inspired by psychological subjects, like perspective taking.
- By using belief models, contextual information, planning with MDPs, geometrical reasoning, and a BN model, the robot is able to infer humans' actions and intentions. Our algorithm has been compared to users' skills in a user study.
- The robot is able to build plans for all the present agents and to take into account their knowledge and expertise. Our system is interfaced with two different planners: an HTN-based human-aware planner, and a novel multi-agent MDP planner.
- Plans can be explained to users based on their expertise on tasks, avoiding to convey too many useless information. This algorithm has been tested in a user study, where it was well perceived by users.
- The robot is able to manage plans in a flexible way. While managing a plan the robot can execute its part while monitoring humans to evaluate if their actions are consistent with the shared plan.
- Our robot is able to execute actions in a robust and safe way. The system is able to take into account humans when planning its motions and to stop if one of its actions could endager them.

- The system is able to execute joint actions with humans. We presented a special framework based on MOMDPs that we used to execute cooperative activities. We presented two of these activities: handover and guiding.

- We presented three possible faces of this system, using different parts and components: the robot observer, whose goal is reasoning on the environment and on user activities and beliefs to provide help; the robot coworker, whose goal is achieving a joint goal with a human partner in a natural and efficient way; and the robot teacher, whose goal is explain users how to accomplish a task and guide them in the operations to execute, while adapting these processes to their knowledge about the problem to solve.

- The system is able to work in complex scenarios in the real world. We presented the case study of the human-aware robot guide, which was conducted in the airport of Schiphol, a very complex environment. In this scenario the robot was evaluated positively by users.

14.1 Perspectives

In this work, we tried to built an architecture that includes several novel and experimental systems. This architecture is just a starting point for further studies, and there are several developments that could be done.

- Introduce learning algorithms. We believe that learning algorithms could enhance several aspects of our system. For example, the robot could learn the users' habits and capacities, to be more able to understand his behaviors. Similarly, learning algorithms could enhance the execution of motions of the robot, adapting them to the current scenario and user.

- More user studies. During the development of this work, we managed to perform several preliminary user studies, of which the most important was the one conducted in the Schipol airport with a robot guide. It would be interesting to perform more in-depth user studies, on several sub-systems and on the architecture as whole, to guide our future developments

- Introduce Dialog. Dialog is a very important part of interaction. In our system, the robot is able to perform very simple and not natural dialogue. While natural dialogue is a very complex problem, we believe that if we would introduce more communication capacities in our robot it would be perceived better by users.

- More complex models. While developing our system we tried to find a balance between the quality of our representations and the overall complexity of the architecture. Improving the complexity of the model and studying more efficient algorithms could improve the quality of our system.

Appendix A

Methods

In this appendix, we introduce the main methods used in our work. Section A.1 gives a short introduction to the subject of MDPs, while A.2 shows an extension to link MDPs in hierarchies, and A.3 shows how Markov Models are used in partially observable domains. Finally, section A.4 introduces the Bayesian Network model. The purpose of this appendix is only to give a short overview in these topics. For more details, the reader can consult specific books and surveys, such as Mausam and Kolobov (2012).

A.1 Markov Decision Processes

A MDP models the decision process of an intelligent agent that needs to act in an environment where the results of its actions is partly random. MDPs are a well known and actively researched topic in artificial intelligence, with several applications, like reinforcement learning.

Formally a MDP is a tuple (S, A, T, R), where:

- S is the system state.
- A is the set of actions that the agent can execute.
- $T(s, a, s')$ is the transition functions, that models the probability that taking action a in state s will lead to state s'.
- $R(s, a, s')$ is the reward function, giving a finite numeric reward obtained by executing action a in state s and reaching state s'.

The solution of a MDP is a policy $\pi(s)$ that links to each state the best action that the agent should execute to maximize his reward.

There are several ways to represent this problem:

- Finite-Horizon. In this case the system assumes that the MDP will terminate after a known number of time steps. Picture for example, a student that has one week to prepare for an exam, and must choose the best course of actions to execute in this limited time.

- Infinite-Horizon. In this case the reward obtained by the agent is accumulated over an infinite sequence of time steps. There are many scenarios that can be represented with this framework, like for example the managing of a business. In this case, to force a policy to converge to a stationary value, we introduce a discount factor γ, set to a value $0 \leq \gamma \leq 1$. The discount factor allows us to privilege shorter term rewards.

- Indefinite-Horizon. In this kind of scenario the process of the agent has a known end, but the number of time steps necessary to reach this goal is unknown. Imagine, for example, the process of an agent who is trying to build a house. The goal will be reached when the house is fully built, but the process might take more or less time depending on different factors. A possible representation of this kind of problems introduces a set of goal states $G \in S$ and substitues the reward function R with a cost function $C(s, a, s')$, that models the cost of executing an action in state s and reaching state s'. Usually, $C(s, a, s') > 0 \; \forall s \notin G$ and $C(s, a, s') = 0 \; \forall s \in G$. The goal of the system becomes minimizing the cost obtained by the agent to reach a goal state.

In some situations, it can be also convenient to specify a set of starting states $S_0 \in S$ which effectively allow us to exclude 'unreachable states', simplifying the computation of the solution of the model.

There are several well known algorithms, like value iteration, to compute the policy of the model. Computing the policy also allows us to build the action value function $Q(s, a)$ that associates to a state s and an action a the expected reward (or cost) that will be received by executing action a and then following the policy π.

A.2 Hierarchical Markov Decision Processes

It can be natural to divide problems in a set of smaller subproblems, which can be handled more easily, and whose solutions can be combined to solve the original scenario.

We can represent this idea in MDPs, by enhancing our model in the following way:

- $M \in A$ is the set of macro actions. Each macro action invokes a sub-MDP on execution, and can possibly terminate in several time-steps.

- $T(s, m, s')$ is the hierarchical transition function, which models the probabilty of transitioning from state s to state s', after executing macro action m.

- $R(s, m, s')$ is the hierarchical reward function, giving a finite numeric reward obtained by executing macro action m in state s and reaching state s'.

Sub-Tasks can only be executed in a subset S_t of the state space S, and terminate when reaching a goal state $g_t \in G_t$.

Special care must be taken to compute the hierarchical transition and reward functions, since the sub-tasks can take several time-steps to be completed. Several ways have been studied to compute these functions, like the MAXQ algorithm, presented in Dietterich (2000), and solving a set of linear systems of equations, as shown in Hauskrecht et al. (1998).

If the state space of the sub-MDPs is limited, hierarchical models can be very efficient. In hierarchical models, the flow of information is bottom-up, and the decisions taken at lower levels of the hierarchy are independent of the higher-level goal to be achieved. This leads the policies to be recursively optimal, and not hierarchically optimal, meaning the sub-models might choose actions that are not optimal to achieve the global goal.

A.3 Partially Observale Markov Decision Processes

POMDPs are an extension of MDPs that model a much more complex scenario, where the agent is not able to observe the world state fully. At each time steps the agent receives a list of observations, which he can use to infer the world state.

Formally, the POMDP is a tuple $(S, A, T, R, \Omega, O, \gamma)$. We will describe the new aspects of this model:

- Ω is a set of observations.
- $O(o, s, a)$ is the probability of observing observation o while being in state s and executing action a.

In this kind of model the agent can take actions that do not immediately lead to reward, but help him receive observations to have a better understanding of the world state. At each moment, the system will maintain a belief $b(s)$, which is the current probabilty of being in state s.

POMDPs are much harded to solve than MDPs, because they have a continuous and infinite state space, given by the probability distributions over the states. Approximate algorithms, like point-based methods, are very popular in these models.

A middle-ground between POMDPs and MDPs is the MOMDP, where part of the state is observed, and part is hidden.

A.4 Bayesian Networks

A BN is a probabilistic model, composed by a directed acyclic graph with random variables as nodes. Edges between the nodes represent conditional dependencies between the associated variables. We can associate a probability function to each node, depending on its parent variables, that produces the probability distribution of the variable represented by the node.

Bayesian Networks can be a very powerful instrument. When we acquire information we can consider a part of the nodes as 'evidence', fixing their values and using them to have a better estimation of the other nodes' probabilities.

Bayesian Networks support both 'top-down' reasoning (i.e. causal, from causes to effects) and 'bottom up' reasoning (i.e. diagnostic, from effects to causes).

There are several well-known algorithms that can be used to perform probabilistic queries on the model, to learn parameters in the conditional dependencies of the nodes, or even to learn the structure of a model.

DBNs are a powerful extensions of BNs, which is able to represent sequences of variables. DBNs have several applications, like speech and gesture recognition.

Bibliography

Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999. 31

Chris L Baker and Joshua B Tenenbaum. Modeling human plan recognition using bayesian theory of mind. *Plan, activity, and intent recognition: Theory and practice*, pages 177–204, 2014. 28, 47

Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009. 28

Frith U Baron-Cohen S, Leslie AM. Does the autistic child have a 'theory of mind'? *Cognition*, 21(1): 37 – 46, 1985. 17

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002. 85

Sarah-Jayne Blakemore and Jean Decety. From the perception of action to the understanding of intention. *Nature Reviews Neuroscience 2, 561-567 (August 2001*, 2:561 – 567, 2001. ISSN 1471-003X. doi: http://dx.doi.org/10.1038/35086023. URL `http://www.nature.com/nrn/journal/v2/n8/suppinfo/nrn0801_561a_S1.html`. 31

Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI*, volume 99, pages 478–485, 1999. 85

Michael Bratman. Two faces of intention. *Philosophical Review*, 93:375 – 405, 1984. 10

C. Breazeal, M. Berlin, A. Brooks, J. Gray, and A. Thomaz. Using Perspective Taking to Learn from Ambiguous Demonstrations. *Robotics and Autonomous Systems*, pages 385–393, 2006. 19

Cynthia Breazeal, Jesse Gray, and Matt Berlin. An embodied cognition approach to mindreading skills for socially intelligent robots. *I. J. Robotic Res.*, 2009. 9, 19, 29

Jerome S. Bruner. Intention in the structure of action and interaction. *Advances in Infancy Research*, 1: 41 – 56, 1981. 10

PL Brusilovskiy. The construction and application of student models in intelligent tutoring systems. *Journal of computer and systems sciences international*, 32(1):70–89, 1994. 107

P. Brusilovsky, S. Sosnovsky, and O. Shcherbinina. User modeling in a distributed e-learning architecture. In *User Modeling 2005*, volume 3538 of *Lecture Notes in Computer Science*, pages 387–391. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-27885-6. doi: 10.1007/11527886_50. URL http://dx.doi.org/10.1007/11527886_50. 107

Diana R Bueno, Eduardo Viruete, and L Montano. An autonomous tour guide robot in a next generation smart museum. In *5th International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI 2011)*, 2011. 125

Hung Hai Bui. A general model for online probabilistic plan recognition. In *IJCAI*, volume 3, pages 1309–1315. Citeseer, 2003. 27

Antoine Bussy, Pierre Gergondet, Abderrahmane Kheddar, François Keith, and André Crosnier. Proactive behavior of a humanoid robot in a haptic transportation task with a human partner. In *RO-MAN, 2012 IEEE*, pages 962–967. IEEE, 2012. 69

Lindsey J Byom and Bilge Mutlu. Theory of mind: mechanisms, methods, and new directions. *Frontiers in human neuroscience*, 7, 2013. 27

Aurélie Clodic, Sara Fleury, Rachid Alami, Raja Chatila, Gérard Bailly, Ludovic Brethes, Maxime Cottret, Patrick Danes, Xavier Dollat, Frédéric Elisei, et al. Rackham: An interactive robot-guide. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 502–509. IEEE, 2006. 124

Aurélie Clodic, Hung Cao, Samir Alili, Vincent Montreuil, Rachid Alami, and Raja Chatila. Shary: a supervision system adapted to human-robot interaction. In *Experimental Robotics*, pages 229–238. Springer, 2009. 9

Yiannis Demiris. Prediction of intent in robotics and multi-agent systems. *Cognitive processing*, 8(3): 151–158, 2007. 29

Yiannis Demiris* and Matthew Johnson†. Distributed, predictive perception of actions: a biologically inspired robotics architecture for imitation and learning. *Connection Science*, 15(4):231–243, 2003. 9

Daniel Clement Dennett. *The intentional stance*. MIT press, 1989. 31

Sandra Devin, Grégoire Milliez, Michelangelo Fiore, Aurélie Clodic, and Rachid Alami. Some essential skills and their combination in an architecture for a cognitive and interactive robot. *arXiv preprint arXiv:1603.00583*, 2016. 30

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303, 2000. 87, 150

Prashant Doshi and Piotr J Gmytrasiewicz. Monte carlo sampling methods for approximating interactive pomdps. *Journal of Artificial Intelligence Research*, pages 297–337, 2009. 28

Anca D Dragan, Kenton CT Lee, and Siddhartha S Srinivasa. Legibility and predictability of robot motion. In *Human-Robot Interaction (HRI), 2013 8th ACM/IEEE International Conference on*, pages 301–308. IEEE, 2013. 69

Edmund H Durfee, Charles L Ortiz Jr, Michael J Wolverton, et al. A survey of research in distributed, continual planning. *Ai magazine*, 20(4):13, 1999. 84

Mica R Endsley. Toward a theory of situation awareness in dynamic systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 37(1):32–64, 1995. 15

Vanessa Evers, Nuno Menezes, Luis Merino, Dariu Gavrila, Fernando Nabais, Maja Pantic, and Paulo Alvito. The development and real-world application of frog, the fun robotic outdoor guide. In *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 281–284. ACM, 2014. 125

Fabio-Valerio Ferrari and Abdel-Illah Mouaddib. Hierarchical factored pomdp for joint tasks: Application to escort tasks. In *2015 AAAI Fall Symposium Series*, 2015. 74

Emmanuel Ferreira, Grégoire Milliez, Fabrice Lefevre, and Rachid Alami. Users' belief awareness in reinforcement learning-based situated human-robot dialogue management. In *IWSDS*, 2015. 23

Michelangelo Fiore, Aurélie Clodic, and Rachid Alami. On planning and task achievement modalities for human-robot collaboration. In *The 2014 International Symposium on Experimental Robotics*, 2014. 61, 71, 78

Michelangelo Fiore, Harmish Khambhaita, Grégoire Milliez, and Rachid Alami. An adaptive and proactive human-aware robot guide. In *Social Robotics*, pages 194–203. Springer, 2015. 126

John H Flavell. The development of knowledge about visual perception. In *Nebraska symposium on motivation*. University of Nebraska Press, 1977. 17

T. W. Fong, C. Kunz, L. Hiatt, and M. Bugajska. The Human-Robot Interaction Operating System. In *2006 Human-Robot Interaction Conference*. ACM, March 2006. 8

Donelson Forsyth. *Group dynamics*. Cengage Learning, 2009. 125

Andrea Frick, Wenke Möhring, and Nora S Newcombe. Picturing perspectives: development of perspective-taking abilities in 4-to 8-year-olds. *Frontiers in psychology*, 5, 2014. 18

Christopher W Geib and Robert P Goldman. Partial observability and probabilistic plan/goal recognition. In *Proceedings of the International workshop on modeling other agents from observations (MOO-05)*, 2005. 68

Piotr J Gmytrasiewicz and Prashant Doshi. Interactive pomdps: Properties and preliminary results. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1374–1375. IEEE Computer Society, 2004. 27

Piotr J Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, pages 49–79, 2005. 85

Carlos Guestrin and Geoffrey Gordon. Distributed planning in hierarchical factored mdps. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 197–206. Morgan Kaufmann Publishers Inc., 2002. 84

Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 220–229. Morgan Kaufmann Publishers Inc., 1998. 87, 150

Trong Nghia Hoang and Kian Hsiang Low. Interactive pomdp lite: Towards practical planning to predict and exploit intentions for interacting with self-interested agents. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2298–2305. AAAI Press, 2013. 28

B. Jensen, N. Tomatis, L. Mayor, A. Drygajlo, and R. Siegwart. Robots meet humans-interaction in public spaces. *Industrial Electronics, IEEE Transactions on*, 52(6):1530–1546, Dec 2005. ISSN 0278-0046. doi: 10.1109/TIE.2005.858730. 125

Matthew Johnson and Yiannis Demiris. Perceptual perspective taking and action recognition. *International Journal of Advanced Robotic Systems*, 2(4):301–308, 2005. 9, 19

A-B Karami, Laurent Jeanpierre, and A-I Mouaddib. Human-robot collaboration for a shared mission. In *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 155–156. IEEE, 2010. 33

Erez Karpas, Steven J Levine, Peng Yu, and Brian C Williams. Robust execution of plans for human-robot teams. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015. 58

Hema S Koppula and Ashutosh Saxena. Anticipating human activities using object affordances for reactive robotic response. *Robotics: Science and Systems, Berlin*, 2013. 27

Hideki Kozima, Cocoro Nakagawa, and Yuriko Yasuda. Children–robot interaction: a pilot study in autism therapy. *Progress in Brain Research*, 164:385–400, 2007. 8

T. Kruse, P. Basili, S. Glasauer, and A. Kirsch. Legible robot navigation in the proximity of moving humans. In *Advanced Robotics and its Social Impacts (ARSO), 2012 IEEE Workshop on*, pages 83–88, May 2012. 137

Thibault Kruse, Amit Kumar Pandey, Rachid Alami, and Alexandra Kirsch. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12):1726–1743, 2013. 125

Tomasz Piotr Kucner, Martin Magnusson, and Achim J Lilienthal. Where am i? an ndt-based prior for mcl. In *Mobile Robots (ECMR), 2015 European Conference on*, pages 1–6. IEEE, 2015. 140

S. Lallee et al. Cooperative human robot interaction systems: Iv. communication of shared plans with humans using gaze and speech. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 129–136, 2013. doi: 10.1109/IROS.2013.6696343. 107

Raphaël Lallement, Lavindra de Silva, and Rachid Alami. HATP: An HTN Planner for Robotics. In *2nd ICAPS Workshop on Planning and Robotics, PlanRob 2014*, 2014. 61, 64

Steven James Levine and Brian Charles Williams. Concurrent plan recognition and execution for human-robot teams. In *Proceedings of the Twenty-fourth International Conference on Automated Planning and Scheduling (ICAPS-14)*, 2014. 58

Timm Linder, Stefan Breuers, Bastian Leibe, and Kai O Arras. On multi-modal people tracking from mobile platforms in very crowded and dynamic environments. *ICRA*, 2016. 23, 140

Rui Liu, Xiaoli Zhang, and Songpo Li. Use context to understand user's implicit intentions in activities of daily living. In *Mechatronics and Automation (ICMA), 2014 IEEE International Conference on*, pages 1214–1219, Aug 2014. doi: 10.1109/ICMA.2014.6885872. 28, 51

Matthias Luber and Kai Oliver Arras. Multi-hypothesis social grouping and tracking for mobile robots. In *Robotics: Science and Systems*, 2013. 125

J. Mainprice, E.A. Sisbot, L. Jaillet, J. Cortes, R. Alami, and T. Simeon. Planning human-aware motions using a sampling-based costmap planner. In *IEEE International Conference on Robotics and Automation*, 2011. 71

Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012. URL http://dx.doi.org/10.2200/S00426ED1V01Y201206AIM017. 148

Francisco S Melo and Manuela Veloso. Heuristic planning for decentralized mdps with sparse interactions. In *Distributed Autonomous Robotic Systems*, pages 329–343. Springer, 2013. 85

Grégoire Milliez, Raphaël Lallement, Michelangelo Fiore, and Rachid Alami. Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring. In *The Eleventh ACM/IEEE International Conference on Human Robot Interation*, pages 43–50. IEEE Press, 2016. 109, 116

Grégoire Milliez, Mathieu Warnier, Aurélie Clodic, and Rachid Alami. A framework for endowing interactive robot with reasoning capabilities about perspective-taking and belief management. In *ISRHIC*, 2014. 22

Javier Movellan, Micah Eckhardt, Marjo Virnes, and Angelica Rodriguez. Sociable robot improves toddler vocabulary skills. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, pages 307–308. ACM, 2009. 105

T. Nagai, K. Abe, T. Nakamura, N. Oka, and T. Omori. Probabilistic modeling of mental models of others. In *Proceedings of the 24th IEEE International Symposium on Robot and Human Interactive Communication*, Aug 2015. 28

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000. 28

Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pages 33–40. IEEE Press, 2013. 84

Billy Okal and Kai O Arras. Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2016. 140

Elisabeth Pacherie. The phenomenology of joint action: Self-agency vs. joint-agency. *Joint attention: New developments*, pages 343–389, 2012. 3

Luigi Palmieri, Sven Koenig, and Kai O Arras. Rrt-based nonholonomic motion planning using any-angle path biasing. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016. 140

A.K. Pandey and R. Alami. Mightability maps: A perceptual level decisional framework for co-operative and competitive human-robot interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. 71

Joelle Pineau, Nicholas Roy, and Sebastian Thrun. A hierarchical approach to pomdp planning and execution. In *Workshop on hierarchy and memory in reinforcement learning (ICML)*, volume 65, page 51, 2001. 73

David Premack and Guy Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, 1(04):515–526, 1978. 17

Miquel Ramırez and Hector Geffner. Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc*, pages 1778–1783. Citeseer, 2009. 27, 29

M Golam Rashed, R Suzuki, A Lam, Y Kobayashi, and Y Kuno. Toward museum guide robots proactively initiating interaction with humans. In *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction Extended Abstracts*, pages 1–2. ACM, 2015. 125

Jorge Rios-Martinez, Anne Spalanzani, and Christian Laugier. From Proxemics Theory to Socially-Aware Navigation: A Survey. *International Journal of Social Robotics*, 7(2):137–153, September 2014. 125

Raquel Ros, Séverin Lemaignan, Emrah Akin Sisbot, Rachid Alami, Jasmin Steinwender, Katharina Hamann, and Felix Warneken. Which one? grounding the referent based on efficient human-robot interaction. In *RO-MAN, 2010 IEEE*, pages 570–575. IEEE, 2010. 19

Ippei Samejima, Yuma Nihei, Naotaka Hatao, Satoshi Kagami, Hiroshi Mizoguchi, Hiroshi Takemura, and Akihiro Osaki. Building environmental maps of human activity for a mobile service robot at the" miraikan" museum. In *Field and Service Robotics*, pages 409–422. Springer, 2015. 125

Matthias Scheutz. Computational mechanisms for mental models in human-robot interaction. In *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, pages 304–312. Springer, 2013. 20, 29

Natalie Sebanz, Harold Bekkering, and Günther Knoblich. Joint action: bodies and minds moving together. *Trends in cognitive sciences*, 10(2):70–76, 2006. 3

Julie Shah, James Wiken, Brian Williams, and Cynthia Breazeal. Improved human-robot team performance using chaski, a human-inspired plan execution system. In *Proceedings of the 6th international conference on Human-robot interaction*, pages 29–36. ACM, 2011. 58

Parag Singla and Raymond J Mooney. Abductive markov logic for plan recognition. In *AAAI*, 2011. 27

E. A. Sisbot, A. Clodic, R. Alami, and M. Ransan. Supervision and Motion Planning for a Mobile Manipulator Interacting with Humans. In *Human-Robot Interaction (HRI), 2008 3rd ACM/IEEE International Conference on*, 2008. 71

E.A. Sisbot, L.F. Marin-Urias, R. Alami, and T. Simeon. A human aware mobile robot motion planner. *Robotics, IEEE Transactions on*, 23(5):874–883, Oct 2007a. 137

E.A. Sisbot, R. Ros, and R. Alami. Situation Assessment for Human-Robot Interaction. In *20th IEEE International Symposium in Robot and Human Interactive Communication*, 2011. 23

Emrah Akin Sisbot, Luis F Marin-Urias, Rachid Alami, and Thierry Simeon. A human aware mobile robot motion planner. *IEEE Transactions on Robotics*, 23(5):874–883, 2007b. 8

M. Sorce et al. Proof of concept for a user-centered system for sharing cooperative plan knowledge over extended periods and crew changes in space-flight operations. In *IEEE RO-MAN*, 2015. 107

Kartik Talamadupula, Gordon Briggs, Tathagata Chakraborti, Matthias Scheutz, and Subbarao Kambhampati. Coordination in human-robot teams using mental modeling and plan recognition. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2957–2962. IEEE, 2014. 29

Sebastian Thrun, Michael Beetz, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Haehnel, Chuck Rosenberg, Nicholas Roy, et al. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *The International Journal of Robotics Research*, 19(11): 972–999, 2000. 124

Alejandro Torreño, Eva Onaindia, and Óscar Sapena. An approach to multi-agent planning with incomplete information. *arXiv preprint arXiv:1501.07256*, 2015. 85

Greg Trafton, Laura Hiatt, Anthony Harrison, Frank Tamborello, Sangeet Khemlani, and Alan Schultz. ACT-R/E: An Embodied Cognitive Architecture for Human-Robot Interaction. *Journal of Human-Robot Interaction*, 2(1):30–55, 2013. 8

J. Trafton, N. Cassimatis, M. Bugajska, D. Brock, F. Mintz, and A. Schultz. Enabling Effective Human-robot Interaction Using Perspective-taking in Robots. *IEEE Transactions on Systems, Man, and Cybernetics*, Part A:460–470, 2005. 19

Rudolph Triebel, KO Arras, Rachid Alami, Lucas Beyer, Stefan Breuers, Raja Chatila, Mohamed Chetouani, Daniel Cremers, Vanessa Evers, Michelangelo Fiore, et al. Spencer: A socially aware service robot for passenger guidance and help in busy airports. *10th Conference on Field and Service Robotics, FSR.*, 2015. 140

B. Tversky, P. Lee, and S. Mainwaring. Why do speakers mix perspectives? *Spatial Cognition and Computation*, 1(4):399–412, 1999. 17

F. Warneken and M. Tomasello. Helping and cooperation at 14 months of age. *Infancy*, 11(3):271–294, 2007. ISSN 1532-7078. doi: 10.1111/j.1532-7078.2007.tb00227.x. URL http://dx.doi.org/10.1111/j.1532-7078.2007.tb00227.x. 107

F. Warneken, F. Chen, and M. Tomasello. Cooperative activities in young children and chimpanzees. *Child Development*, pages 640–663, 2006. 107

Heinz Wimmer and Josef Perner. Beliefs about beliefs: Representation and constraining function of wrong beliefs in young children's understanding of deception. *Cognition*, 13(1):103 – 128, 1983. 18

Mohammad Abu Yousuf, Yoshinori Kobayashi, Yoshinori Kuno, Akiko Yamazaki, and Keiichi Yamazaki. Development of a mobile museum guide robot that can configure spatial formation with visitors. In *Intelligent Computing Technology*, pages 423–432. Springer, 2012. 125