Geometric reasoning and planning in the context of human robot interaction

Mamoun Gharbi

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *16/09/2015* **par :**

Mamoun Gharbi

Résonement et plannification géometrique dans le context de l'interaction homme robot

---

JURY

Premier MEMBRE	Thierry Siméon	Président du Jury
Second MEMBRE	Rachid Alami	Membre du Jury
Troisième MEMBRE	Michael Beetz	Membre du Jury
Quatrième MEMBRE	Raja Chatila	Membre du Jury
Ciquième MEMBRE	Adolfo Suarez	Membre du Jury

École doctorale et spécialité :

MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :

Laboratoire d'analyse et d'architecture des systèmes

Directeur(s) de Thèse :

Rachid Alami

Rapporteurs :

Michael Beetz et *Raja Chatila*

LAAS-CNRS

Abstract

Institut National des Sciences Appliquees de Toulouse

Doctor of Philosophy in Robotics

Geometric reasoning and planning in the context of human robot interaction

by Mamoun GHARBI

In the last few years, the Human robot interaction (HRI) field has been in the spotlight of the robotics community. One aspect of this field is making robots act in the presence of humans, while keeping them safe and comfortable. In order to achieve this, a robot needs to plan its actions while explicitly taking into account the humans and adapt its plans to their whereabouts, capacities and preferences.

The first part of this thesis is about human-robot handover: where, when and how to perform them? Depending on the human preferences, it may be better, or not, to share the handover effort between him and the robot, while in other cases, a unique handover might not be enough to achieve the goal (bringing the object to a target agent) and a sequence of handovers might be needed. In any case, during the handover, a number of cues should be used by both protagonists involved in one handover. One of the most used cue is the gaze. When the giver reaches out with his arm, he should look at the object, and when the motion is finished, he should look at the receiver's face to facilitate the transfer.

The handover can be considered as a basic action in a bigger plan, the second part of this thesis reports about a formalization of these kind of "basic actions" and more complex ones by the use of conditions, search spaces and restraints. It also reports about a framework and different algorithms used to solve and compute these actions based on their description.

The last part of the thesis shows how the previously cited framework can fit in with a higher level planner (such as a task planner) and a method to combine a symbolic and geometric planner. The task planner uses external calls to the geometric planner to assess the feasibility of the current task, and in case of success, retrieve the state of the world provided by the geometric reasoner and use it to continue the planning. This part also shows different extensions enabling a faster search. Some of these extensions are "Geometric checks" where we test the infeasibility of multiple actions at once, "constraints" where adding constraints at the symbolic level can drive the geometric search, and "cost driven search" where the symbolic planner uses information form the geometric one to prune out over costly plans.

Version française

Au cours des dernières années, la communauté robotique s'est largement intéressée au domaine de l'interaction homme-robot (HRI). Un des aspects de ce domaine est de faire agir les robots en présence de l'homme, tout en respectant sa sécurité ainsi que son confort. Pour atteindre cet objectif, un robot doit planifier ses actions tout en prenant explicitement en compte les humains afin d'adapter le plan à leurs positions, leurs capacités et leurs préférences.

La première partie de cette thèse concerne les transferts d'objets entre humains et robots : où, quand et comment les effectuer? Dépendant des préférences de l'Homme, il est parfois préférable, ou pas, partager l'effort du transfert d'objet entre lui et le robot, mais encore, à certains moments, un seul transfert d'objet n'est pas suffisant pour atteindre l'objectif (amener l'objet à un agent cible), le robot doit alors planifier une séquence de transfert d'objet entre plusieurs agents afin d'arriver à ses fins. Quel que soit le cas, pendant le transfert d'objet, un certain nombre de signaux doivent être échangés par les deux protagonistes afin de réussir l'action. Un des signaux les plus utilisés est le regard. Lorsque le donneur tend le bras afin de transférer l'objet, il doit regarder successivement le receveur puis l'objet afin de faciliter le transfert.

Le transfert d'objet peut être considéré comme une action de base dans un plan plus vaste, nous amenant à la seconde partie de cette thèse qui présente une formalization de ce type d'"actions de base" et d'actions plus complexes utilisant des conditions, des espaces de recherche et des contraintes. Cette partie rend aussi compte du framework et des différents algorithmes utilisés pour résoudre et calculer ces actions en fonction de leur description.

La dernière partie de la thèse montre comment ce framework peut s'adapter à un planificateur de plus haut niveau (un planificateur de tâches par exemple) et une méthode pour combiner la planification symbolique et géométrique. Le planificateur de tâches utilise des appels à des fonctions externes lui permettant de vérifier la faisabilité de la tâche courante, et en cas de succès, de récupérer l'état du monde fourni par le raisonneur géométrique et de l'utilisé afin de poursuivre la planification. Cette partie montre également différentes extensions de cette algorithme, tels que les "validation géométriques" où nous testons l'infaisabilité de plusieurs actions à la fois ou "les contraintes" où l'ajout de contraintes au niveau symbolique peut dirigée la recherche géométrique ou encore "recherche dirigé par coût" où le planificateur symbolique utilise les informations fournies par la partie géométrique afin d'éviter le calcul de plans moins intéressants.

Acknowledgements

I want to thank all the people I know who supported me during this adventure that was my PhD student years. First of all, I want to thank my close family, for their unconditional support, whatever the difficulties and the hardships I went through. For all that, thanks mum, thanks dad, and thanks brother! and more recently, thanks sister-in-law! For the help during my thesis writing, I want to give special thanks to my brother and my aunt Myriem for reading and correcting everything more than once. I also thank the rest of my family for their support and cheers during all those years. The next person I would like to thank is my supervisor Rachid Alami, for the support and the help especially during the deadlines and the rushing times. I would also like to thank all the collaborators without whom this manuscript wouldn't be what it is, the first of them is Raphael Lallement, and others such as Jules Waldhart, Jim Mainprice, Séverin Lemaignan, Lavindra Da Silva, and Amit Kumar Pandey. During those years, some co-workers with whom I shared desks, homes, and/or good times became my friends and I cannot be more thankful to them for those wonderful years. To cite just a few of them, Ellon Mendes, Elena Stumm, Michelangelo Fiore, Artur Maligo, Gregoire Milliez, Wuwei He, Didier Devaur, Renaud Viry, Alexandre Ravet, Alexandre Boeuf, Pierrick Koch, Nicolas Staub, and so on. Finnally, I want to thank my reviewers Raja Chatila and Michel Beetz for accepting the task and making me finally a PhD.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

This thesis enters into the so-called human-robot interaction (HRI) field. As defined by Goodrich and Schultz (2007), HRI tries to understand and shape the interactions between one or multiple humans and one or multiple robots. In other words, how exactly an autonomous robot (or multiple autonomous robots) needs to behave when brought in the vicinity of humans, and more specifically when they need to cooperate or help one (or more) of these humans.

One interesting part of this field is to provide robots with enough autonomy to let them perform and execute tasks and actions in this human environment. In order to achieve this, they need to "think" and infer from the geometric properties of the real world. The main focus of this thesis is to equip robots with geometric reasoning enabling them to plan their actions. This planning is done while taking into account the environment properties but also the human preferences and social rules.

In the beginning of this thesis we propose an approach to let a mobile manipulation robot (such as the PR2 from Willow Garage (2008)) handover small objects to a human, which is extended to a multiple agent case, and where the interaction cues at the exchange moment are studied. A generalisation of this kind of geometric reasoning is also proposed and then coupled with higher level planning in order to provide the robot with even more autonomy.

1.1 Human-Robot Interaction

Bringing autonomous robots into our houses and work places rises a number of challenges that need to be tackled in order to achieve this integration. Among these challenges two categories can be isolated: hardware and software challenges. The hardware challenges cover the design of the robot shapes, such as the skin, the face, or the eyes. Specific actuators and sensors also belongs to this category.

The software challenges, this work belongs to, cover a number of fields such as task planning, supervision, belief management, human-aware motion planning, situation assessment and so on. Among these challenges, the one interesting us concerns planning the geometric actions needed by the robot to perform tasks in a human environment. In other words, we want to endow the robot with actions enabling it to interact with its environment in general and with the humans in particular.

Integrating autonomous robots in human environments can serve multiple purposes. For example, a service robot can assist and help elderly people in their houses for everyday life. A guiding robot, can detect and find lost people and help them to reach their destination goal. Another example is the robot co-worker, as depicted in one of the use cases of the SAPHARI project (`http://www.saphari.eu/`). The human interacting with this kind of robot are "expert", which means they should be accustomed to work with the robot in contradiction with the previous examples where the users are more likely naive ones. The robot co-worker can be used for multiple tasks such as helping tidying a workspace, deliver objects, lift heavy objects or perform precision tasks.

1.2 Geometric planning

Planning the motions of a robot in a human environment brings a number of issues: the first one concerns the security of the human. When the robot plans its motions close to the human, it needs to take into account the possibility of the human moving in an unexpected or expected way. For example, when walking, a human will most likely continue walking. Taking this motion into account helps to ensure his safety. Another important issue concerns the comfort of the human, even when insuring this safety, some motions can create uncomfort or lead to a misunderstanding of the robots behaviour. Planning while taking this, and other social rules that humans use in a daily basis, into account, is called Human-Aware Planning

The supervision asks geometric plans following the plan provided by the task planner which (should) deals with symbolic knowledge. A number of researches, such as Ghallab et al. (2004), focus on this task planning while in others, such as LaValle (2006), the main interest lies in the motion planning area. Between these two research topics, there is a gap where the symbolic knowledge should be transformed into information usable by the motion planner. This is especially important in human-robot interaction, as the information about the human are both symbolic and geometric and need to be dealt with at both level successfully. This work tries to bridge the gap between the high level symbolic planning and the low level motion planning.

The first milestone in this path was to design and improve a unique task which is the handover (Chapter 2). A second contribution consisted in proposing a global framework able to communicate with both symbolic reasoning and motion planning where multiple actions such as pick, place and navigate where implemented (Chapter 3). The last part consists on a tighter interleaving between the symbolic layer and this framework (Chapter 4).

1.3 Contributions

The main contribution of my thesis are:

- Giving the robot the ability to choose where and how to perform a handover with a human while taking into account his safety, comfort, capabilities and preferences. When planning this handover, the robot computes both its path and the human path in order to assess the feasibility of the task using a combination of grid-based and sampling-based method. This work is presented in Section 2.3.

- Extending the previous work to a multi-agent task where the goal is to bring an object from an agent to another one (agents can be either robots or humans) through a sequence of handovers. This approach also takes into account the HRI constraint related to the possible humans in the environments, and is graph-based using a Lazy Weighted A*. This work is presented in Section 2.4.

- Studying the gaze behaviour of both givers and receivers during a handover (or assimilated tasks), in order to define gaze pattern allowing a more understandable and human-aware task execution. This work is presented in Section 2.5.

- Proposing a framework (and its formalization) in order to create and plan actions at geometric level. This framework, called Geometric Reasoner and Planner (GRP), is able to compute, based on symbolic information (such as the agent performing the task, the object to manipulate or the support table), complex actions such as pick, place or navigate. This work is presented in Chapter 3.

- Developing the Symbolic Geometric Action Planner (SGAP) which interleaves task and geometric planning and produces plans that contains, in addition to the classical symbolic plan, the geometric information, such as the trajectories, relative placement, grasps and postures, needed to execute the plan in the real word. This framework, by using facts computed at geometric level and backtracks, is able to tackle the ramification problem. This work is presented in Section 4.3.

- Adding a number of powerful heuristics enabling SGAP to decrease the combinatorial explosion resulting from the complexity of the geometric world, or the symbolic models. This heuristics use constraints, different level of actions, cost computation and specific request to the GRP framework to enhance the search efficiency. This work is presented in Section 4.4.

List of publish papers

- Jim Mainprice, **Mamoun Gharbi**, Thierry Siméon, and Rachid Alami. "Sharing effort in planning human-robot handover tasks." In the International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 764-770. IEEE, 2012. Mainprice et al. (2012)

- Lemaignan Séverin, **Mamoun Gharbi**, Jim Mainprice, Matthieu Herrb, and Rachid Alami. "Roboscopie: a theatre performance for a human and a robot." In Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction (HRI), pp. 427-428. 2012. Lemaignan et al. (2012a)

- Lavindra de Silva, Amit Kumar Pandey, **Mamoun Gharbi**, and Rachid Alami. "Towards combining HTN planning and geometric task planning." In Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications, Robotics Science and System (RSS). 2013. De Silva et al. (2013)

- **Mamoun Gharbi**, Séverin Lemaignan, Jim Mainprice, and Rachid Alami. "Natural interaction for object hand-over." In Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction, pp. 401-402. IEEE Press, 2013. Gharbi et al. (2013)

- Lavindra de Silva, **Mamoun Gharbi**, Amit Kumar Pandey, and Rachid Alami. "A new approach to combined symbolic-geometric backtracking in the context of human-robot interaction." In International Conference on Robotics and Automation (ICRA), pp. 3757-3763. IEEE, 2014. De Silva et al. (2014)

- Rachid Alami, **Mamoun Gharbi**, Benjamin Vadant, Raphaël Lallement, and Adolfo Suarez. "On human-aware task and motion planning abilities for a teammate robot." In Workshop on Human-Robot Collaboration for Industrial Manufacturing, Robotics Science and System (RSS). 2014. Alami et al. (2014)

- Benjamin Vadant, **Mamoun Gharbi**, Rachid Alami, and Thierry Siméon. "Being safe around the robot." In Workshop on Algorithmic Human Robot Interaction, international conference on Human-Robot Interaction (HRI). 2014. Vadant et al. (2014)

- **Mamoun Gharbi**, Raphaël Lallement, Rachid Alami. "Combining Symbolic and Geometric Planning to synthesize human-aware plans: toward more efficient combined search." to appear in the International Conference on Intelligent Robots and Systems (IROS). IEEE/RSJ, 2015. Gharbi and Alami (2015)

- **Mamoun Gharbi**, Pierre-Vincent Paubel, Aurélie Clodic, Ophélie Carreras, Rachid Alami, and Jean-Marie Cellier. "Toward a better understanding of the communication cues involved in a robot-human object transfer". to appear in the International Symposium on Robot and Human Interactive Communication (RO-MAN). IEEE, 2015. Gharbi et al. (2015)

- Jules Waldhart, **Mamoun Gharbi**, Rachid Alami. "Planning handovers involving humans and robots in constrained environment" to appear in the International Conference on Intelligent Robots and Systems (IROS). IEEE/RSJ, 2015. Waldhart et al. (2015)

Chapter 2

Human-robot handover

Contents

2.1 Introduction

A robot acting and "living" in the same environment as humans needs to be able to perform a set of actions and tasks that will make it a helpful and a useful companion for the humans. One key action, in this set, is giving and receiving objects from the human. This action, called handover, is not innate for humans but is performed naturally, many times a day, every day, by most people. A robot equipped with this feature would be more than helpful: it is a perfect example of a service task. It can be used in various scenarios, for example, in a work space, a robot would be able to fetch objects from the environment and bring them to a co-worker. In an elderly house, a robot would be able to bring items to the residents. The requirements for the robot –to make handovers usable– are to perform it as flawlessly and naturally as a human would.

The handover is an interesting and complete example of a physical joint action where both involved agents need to signal their intentions and reach an agreement while physically collaborating in order to achieve the task. One possible definition of a joint action is proposed by Sebanz et al. (2006): "joint action can be regarded as any form of social interaction whereby two or more individuals coordinate their actions in space and time to bring about a change in the environment". They also propose a list of abilities enabling a successful joint action: the ability to "(i) share representations, (ii) predict actions, and (iii) integrate predicted effects of own and others' actions".

The challenges to tackle in order to let the robot perform handovers are various and diverse. We propose a classification of these challenges into phases: Figure 2.1 shows these phases represented in a time-line where, in each phase a number of actions need to be performed in order to enter the next phase. Note that the successive actions and phases can overlap each other depending on the situation, for example, reaching out with the arm can be done while still navigating. These phases concern both involved agents, the giver and the receiver, even if some actions seem more related to one of them only (Getting the object for example, is done by the giver, although it needs to take the receiver abilities into account).

FIGURE 2.1: During a handover, the agents[1] go through four phases, with, in each one, a number of actions to perform. This time-line concern both agents.

The rest of this chapter is structured as follows, Section 2.2 presents the different handover phases and their main contributions. Section 2.3 shows the work done about choosing a handover position and an arm configuration for the robot at this position. Section 2.4 presents the problem where a sequence of handover is needed to perform a task, and finally Section 2.5, shows a user study concerning gaze cues during a handover.

2.2 The different handover phases and their main contributions

Figure 2.1 shows the four different handover phases: taking the decision, approaching the agent[1] while preparing the handover, giving/taking the object and disengaging. These phases are composed of two parts, a timeline of the different actions the agents need to perform and a sequence of communication cues to make the handover as fluent as possible, the next subsections present in more details these four phases and their components.

2.2.1 Taking the decision

Three different reasons might support the decision of a robot to perform a handover:

- The other agent (a human or a robot) asks for an object, and the robot performs a handover to give it to him.
- As a proactive behaviour, the other agent needs a handover and the robot proposes it to him.
- The robot needs an object and asks the other agent to give it to him.

This phase is related to the more generic field of task planning (analysed in Ghallab et al. (2004)) which is more detailed in Chapter 4.

2.2.2 Approaching the other agent while preparing the handover

In this phase, the agent needs to perform three actions:

Getting the object Optional for the giver as he may already have the object in hand, and doesn't exist for the receiver. This action is about fetching the object, and the most important part here, is the grasp: it may have a direct impact on the future handover. Various studies have been done in the field of grasping with robots, part of them depicted by Bicchi and Kumar (2000) and some of them, detailed in the next paragraph, focused on the particular problem of grasping in order to handover the object.

Numerous contributions concern this particular topic, among them, Berenson et al. (2008), where the authors choose the grasps for the objects accounting for the future actions, for example, when placing a glass in a dishwasher, some grasps will not work while others will. Pandey et al. (2012) extended this idea to the handover, where a grasp is chosen in order to leave enough space to allow another grasp. Aleotti et al. (2012) and Aleotti et al. (2014) also choose the grasps accounting for the receiver: they segment the object, find the "handles" of this object (a hammer handle for example), and perform the handover while presenting this part to the receiver. Kim et al. (2004) have a similar approach but introduce the notion of dangerous features concerning objects such as a knife sharp edge, and plan a change of grasp, in case the robot needs to grasp the object

[1] An agent is someone or something able to act and change its environment, it can be a human or a robot.

with the first hand, take it with the other hand and then give it to a human while presenting the non-dangerous part.

While these papers focus on finding an algorithmic solution to this problem, Chan et al. (2013a) and Chan et al. (2014) propose a learning approach where the robot learns how to grasp the objects that are exchanged by the surrounding humans.

Choosing a handover posture We believe that this choice should be made while taking explicitly the human and the environment in general into account. The position should favour the human comfort while enabling a great latitude (for example, handovers above a counter or through a window should be possible). In this phase, the choice concerns the general posture if the agent is a human, but the complete configuration if it is a robot.

Walters et al. (2007) show that humans with no prior interaction with a robot will prefer to receive an object from the sides if they are sitting or standing against a wall, but will prefer a frontal approach when standing in the middle of a room (the reason given is that while sitting, a robot might be intimidating and standing against a wall might restrain movements inducing an uncomfortable situation). Koay et al. (2007) argue that the intimidation feeling disappears when the human gets used to the robot, thus making the frontal approach the most preferred one in all the cases. Sisbot et al. (2007a) and Sisbot et al. (2010) base the robot placement on a list of parameters such as the distances between the robot and the human (*proxemics* theory, Hall (1966)), the visibility of the robot by the human (not going behind him or behind an obstacle) but also the human arm comfort and the robot navigation distance.

On a slightly different direction, Shi et al. (2013) propose an approach where the robot perform a handover with an already walking pedestrian, handing them flyers. They propose a model based on analysing humans distributing flyers and implement this model to perform a user study on a real robot.

Navigating to the handover position Human aware navigation is a widely studied field and complex as shown by the survey of Kruse et al. (2013). When navigating among humans, a robot needs to take into account various parameters such as the humans' comfort and safety, the dynamics of the environment such as the humans' future movements. At the end of this action, the agents need to be at the handover position (or in a close enough location) to be ready to perform the handover.

Communication cues

The communication cues (CC) for this phase (Approaching the other agent while preparing the handover) are depicted in Figure 2.2, it consists in the joint action signals. One of the agents signal to the other one his intention of performing a handover with him. If the other agent gets the signal, his attention will focus on the agent and both of them should agree on this intention. If the other agent does not get the signal or does not agree to perform the handover, the first agent will try again until it seems clear that no handover is possible than he aborts the task. In the other case, both agents establish the intention sharing[2] and communicate about the details of the task (where, when and how). If in the meantime, one

[2]The intention sharing means mean that both have the intention to perform a handover with each other. Loosing this intention sharing means that at least one of the agents does not want to perform the handover any more.

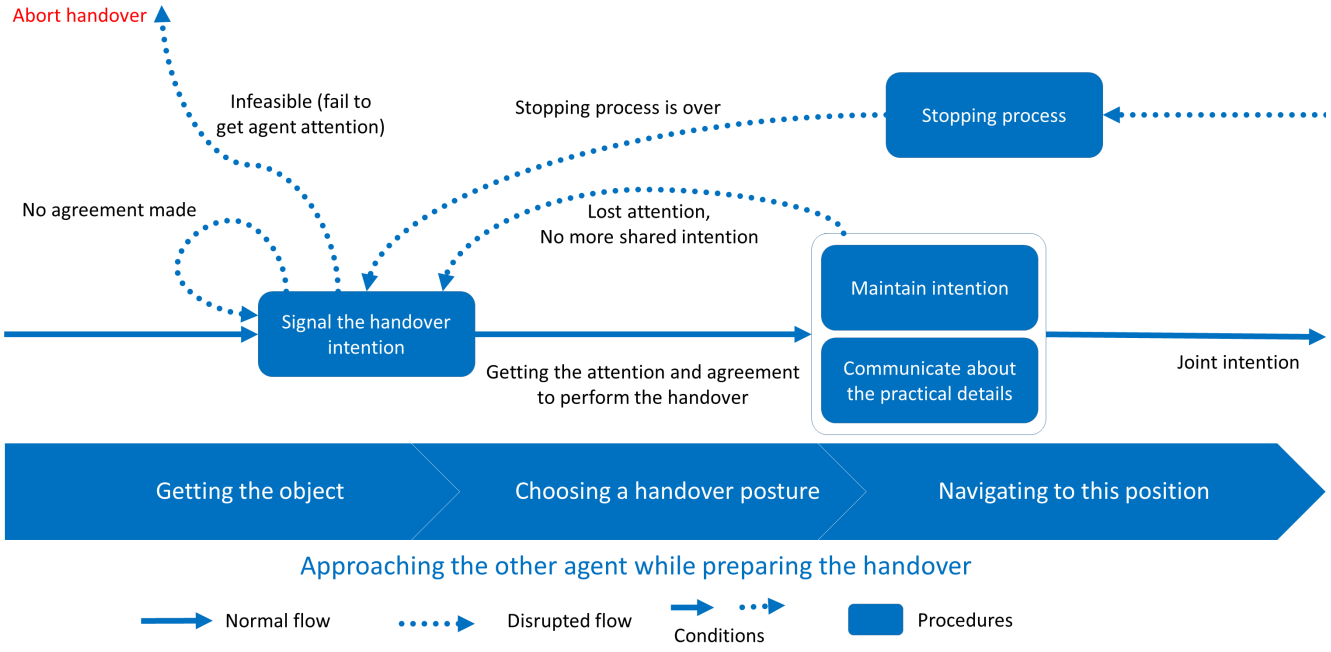


FIGURE 2.2: During the phase "Approaching the agent while preparing the handover", a number of communication cues are exchanged, depicted as a sequence in this figure

of the agents loses the intention, the other one loops back to the first step where he signals his intention again. If both agents continue to have the shared intention and agree on the details, the phase is finished and next phase can begin.

Note that this phase is considered as finished only when the CC are achieved: if one agent navigates to the handover position and the other agent still didn't get the signals for the handover intention, the first agent must wait until all CC are done to begin the next phase (giving the object).

In some situations, some of these CC can be avoided, for example, Strabala et al. (2012b) showed through a user study that the receiver has to look in the giver's direction but there is no requirement for the giver to look back. Such gaze cues are used in Kirchner et al. (2011) where the robot looks at a particular person in a group to communicate about his intention to give an object. Beside gaze cues, other handover indications can be used such as shifting the object from a two hand grasp to a one hand grasp or reaching out while still navigating as depicted by Lee et al. (2011).

2.2.3 Giving the object

This phase can be divided into two different actions: the first one is to reach out with the hand and the second one is to release/grab the object. These two actions happen in the sequence, reach out then grab/release. There are some events that can to interrupt the first one, as: if the giver begins to reach out and the receiver get distracted, the giver needs to stop the action, and to engage in the stopping process. In this case, the giver goes back to the previous phase to establish the intention sharing again, and in case of success (going through all the communication cues with success) he can continue/restart (depending on the stopping process specified) the motion. Another interrupting event might be provoked by the receiver who wants to grasp the object before the giver has finished his motion, on this case the giver releases the object and go to the next phase. This behaviour is depicted in Figure 2.3.

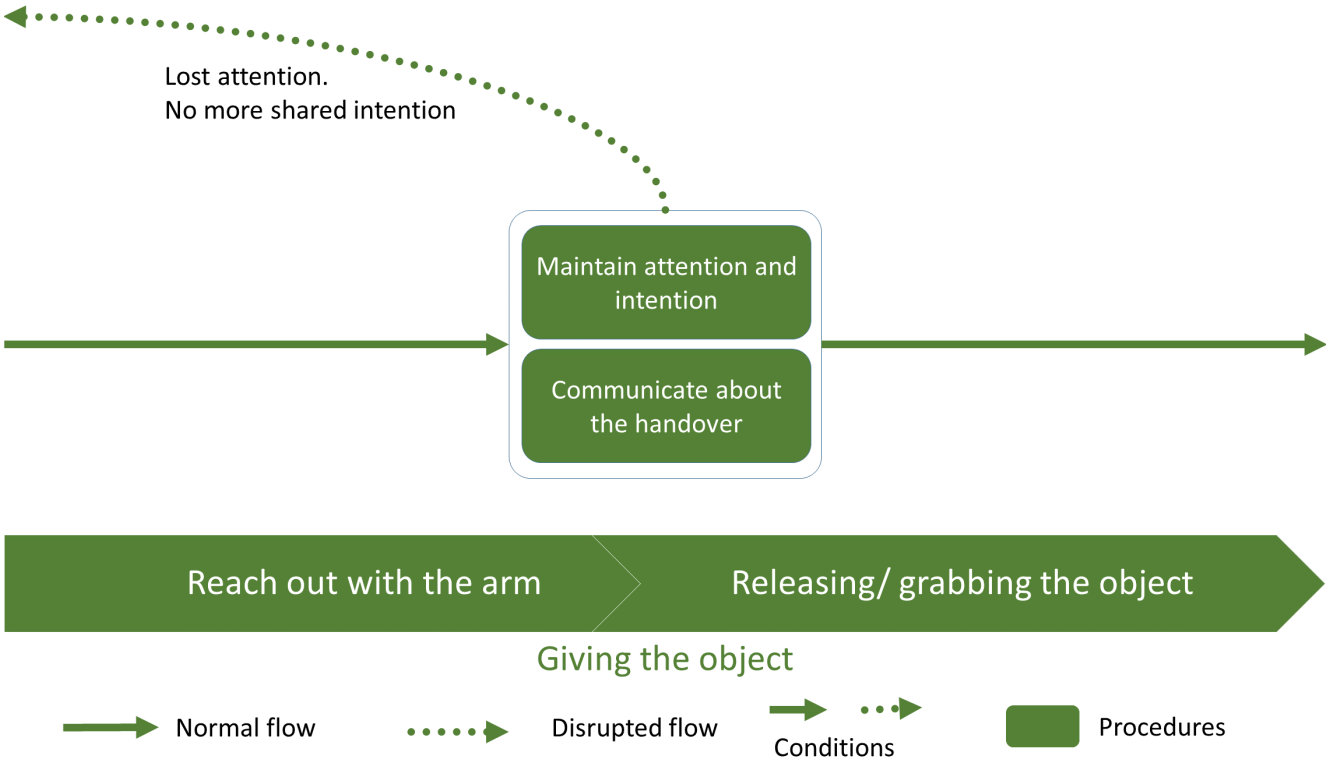


FIGURE 2.3: During the phase "giving the object", a number of communication cues are exchanged, depicted as a sequence in this figure

A number of researchers worked on how to reach out with the arm, and this work can be divided into three categories:

The final position Of both agents arms and the object. Huber et al. (2009), Hart et al. (2015) and Basili et al. (2009) agree that during a handover, humans tend to position the object halfway between the giver and the receiver. For the arm position, Prada et al. (2014), Koene et al. (2014) and Micelli et al. (2011) argue that, when the human is not focused on the handover, it is essential to adapt the giver end effector position to the human hand using a closed loop, including perception. When the focus of the human is on the handover, Edsinger and Kemp (2007) show that the human will easily adapt to the robot.

The final position of the robot hand should also be impacted by the human presence: Cakmak et al. (2011a) show the importance of the robot arm position (especially the elbow) and the position of the object handed over (e.g. when handing a plate, it is better if the plate is in horizontal position). Moreover, Sisbot et al. (2007a) show, among other parameters (visibility, comfort, and so on), the importance of taking into account the *proxemic* theory, Hall (1966) while handing over an object.

Yamane et al. (2013) adopt a different approach where they learn from a human motion database how to perform a handover (either the final position of the arms or the timing or the motion legibility.)

The timing of the action Each agent needs to execute his motion at the right timing. Huber et al. (2008) differentiate between three time phases: the reaction time (the time the receiver takes to begin its motion after the giver started his) the manipulation time (while both agents manipulate the object) and the post-handover time (the disengagement phase). They find that a handover mean time is less than 2 seconds, the reaction time is around 0.35 seconds and the manipulation time is around 1.2 seconds. Koene et al. (2014) show through a user study the importance of respecting the temporal precision over the spatial one: if the robot is too fast, the human may think the robot is upset, while if it is too slow he may feel bored or frustrated.

The motion legibility As shown in a number of researches, such as Dragan et al. (2015), a legible motion brings more comfort and safety feeling for the human, and the human-robot handover also follows this rule. Both Micelli et al. (2011) and Huber et al. (2008) show that a motion where the end effector executes straight lines is preferred over a motion in the joint space. Moreover, Dehais et al. (2011) and Mainprice et al. (2010) assert that straight lines need to be combined with a higher level motion planner taking into account the human comfort and safety. Palinko et al. (2014) go as far as adapting the arm trajectory to the object weight in order to give to the receiver more information about the object weight.

The arm control also plays an important role in human-robot interactions, the usage of motions with limited jerk such as the one proposed by Broquère et al. (2008) enables for a better comfort while extending the arm. Different control strategies have been tested by different teams: Prada et al. (2014) formalised and implemented a Dynamic Movement Primitives control strategy on a robotic arm, Kajikawa et al. (2002) use the human-human handover analysis done by Shibata et al. (1995) to define a control strategy and Erden et al. (2004) use a fuzzy controller to execute the motion.

The second action during the **giving object** phase is to release the object at a good timing. Mason and MacKenzie (2005), Endo et al. (2012), Chan et al. (2013b), Jindai et al. (2015) and He and Sidobre (2015) developed a force detection object in order to record data from human-human handovers and apply it to a controller when performing a human-robot handover. Cabibihan et al. (2013) achieve the same detection with a glove like sensor worn by both participants. The purpose of all these approaches is to make the robot release the object at the moment it detects the particular forces, applied to the object, corresponding to a firm grasp from both agent. This moment triggers the object release which is an issue: it is often ambiguous (even for humans) and it can be done during the agents motions.

Communication cues

During this phase, a number of signals needs to be exchanged and the first one is the starting signal for reaching out with the arm. Cakmak et al. (2011b) claim that the robot reaching out with the arm (moving the arm from a rest position to the handover position) is in itself the starting signal. Micelli et al. (2011) consider data (hand position and orientation) from the kinect to detect when to start the handover process. In addition to that, Boucher et al. (2012) established that the gaze effect has a great impact in the perception of naturalness during a human-robot face-to-face cooperation, which was tested in a handover set-up by Moon et al. (2014) and Zheng et al. (2014) who find that a turn taking signals

(the giver looks first at the object and midway through the handover looks at the receiver) give the best results (better time performance). Admoni et al. (2014) also show that a little delay before beginning the reach out enables the receiver to give more attention to the giver non-verbal cues such as the gaze (in the paper they use a one second delay and find it sufficient for a handover).

2.2.4 Disengaging

To our knowledge, no one worked on how a robot should disengage itself from a handover, even if Huber et al. (2008) mention it as the *post-handover-phase* and show its mean time (around 0.15 seconds), no real study has been held. This phase is about how and when the robot should retract itself from the physical space of the human and how to finish the joint action.

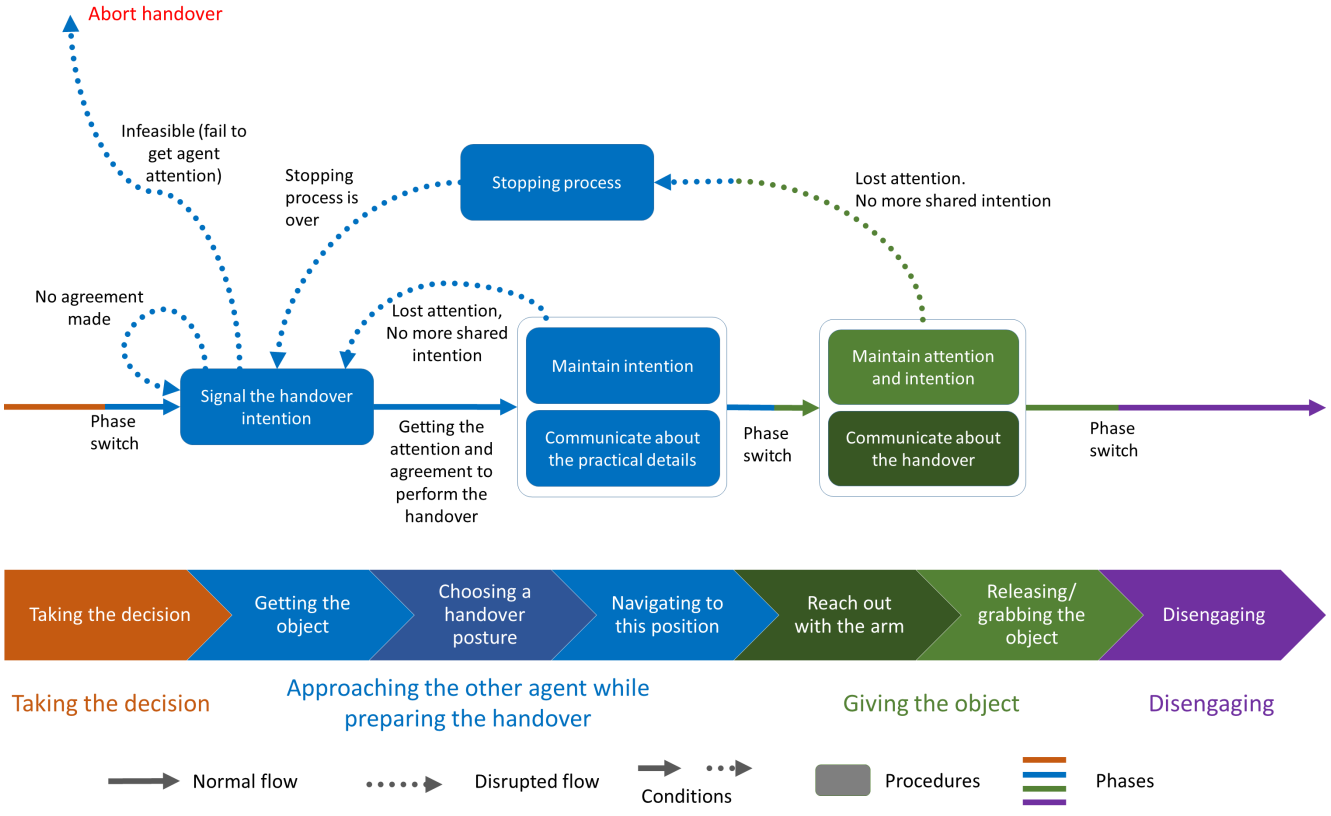


FIGURE 2.4: A synthesis of all the handover phases. The dark colours indicate where our contributions belong to.

2.2.5 Synthesis

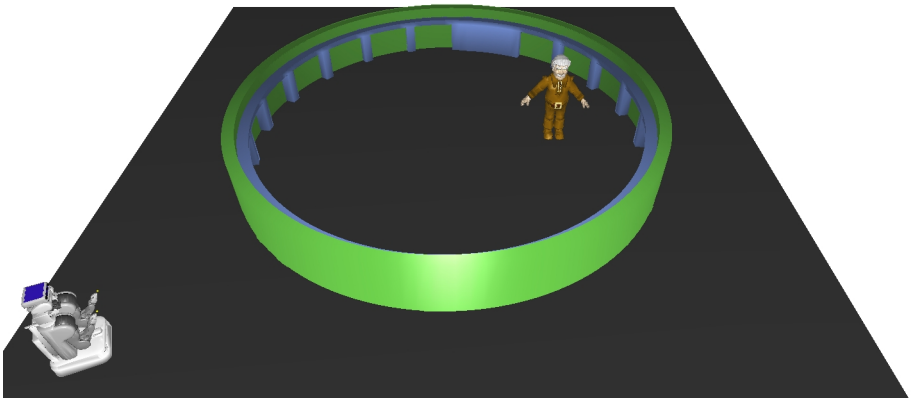
The handover phases are synthesised in Figure 2.4 where the bottom part refers to the actions and the top parts to the CC. Strabala et al. (2012a) show a similar diagram where the bottom part is called "*Physical*" processes and the top part "*Social-Cognitive*" processes. Figure 2.4 shows an extended approach to their work which incorporate the CC to the timeline. Inside a phase, the CC are not linked to the timeline, but each phase needs its own signals to be achieved before its end. This chapter contributions can be categorized following the different phases illustrated in the figure with the dark coloured parts.

Some contributions tackle the problem of executing handovers while taking into account all or part of these phases, for example Sisbot et al. (2008) propose a state machine approach where they handle interruptions and suspension when the robot already start reaching out with the arm. In Fiore et al. (2016) and in Karami (2014), they use a Partially observable Markov decision process (POMDP) to choose which action to perform and when to perform it (for example, when reaching out with the arm, if the human attention is driven away from the robot, this one should enter in a stand by phase).

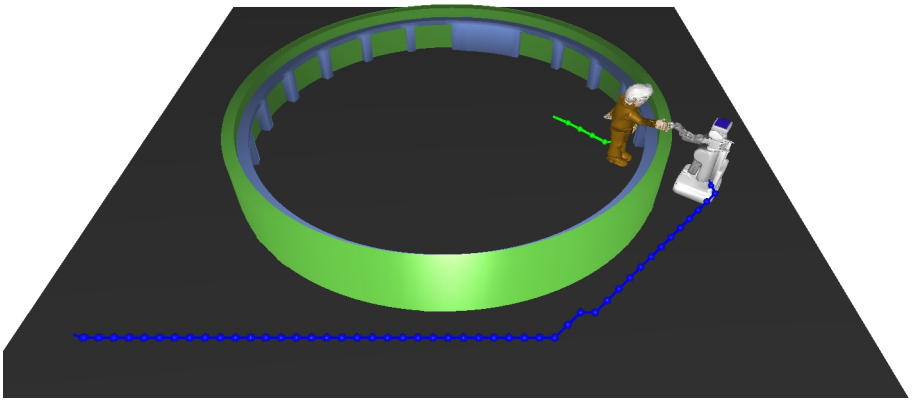
2.3 Sharing the effort with the human

The focus of this section is the particular problem of finding good object handover configurations, which is formulated as a special instance of the motion planning problem (Choset (1991), Choset et al. (2005), LaValle (2006)). We will consider mobile manipulators such as the PR2, exchanging object with a human and introduce the notion of "shared effort" in the handover plan. This work has been done in cooperation with Jim Mainprice and presented by Mainprice et al. (2012).

In this work, both agents (involved in the handover, robots or humans) are considered, and both their motions are computed, in a possibly cluttered workspace. Computing the human motion might seem meaningless, as the human will not follow the computed trajectory, but the reason behind this computation is to enable the system to find solutions to problems where the human cannot be reached by the robot. Figure 2.5 illustrates this problem: the human is in a workspace not reachable by the robot who still wants to hand him over an object. By computing the human motions, the algorithm can find solutions to this problem and choose among them one compatible with the human preferences. For example, in Figure 2.5(b), the robot proposes pro-actively a solution to the human which reduces his displacement.



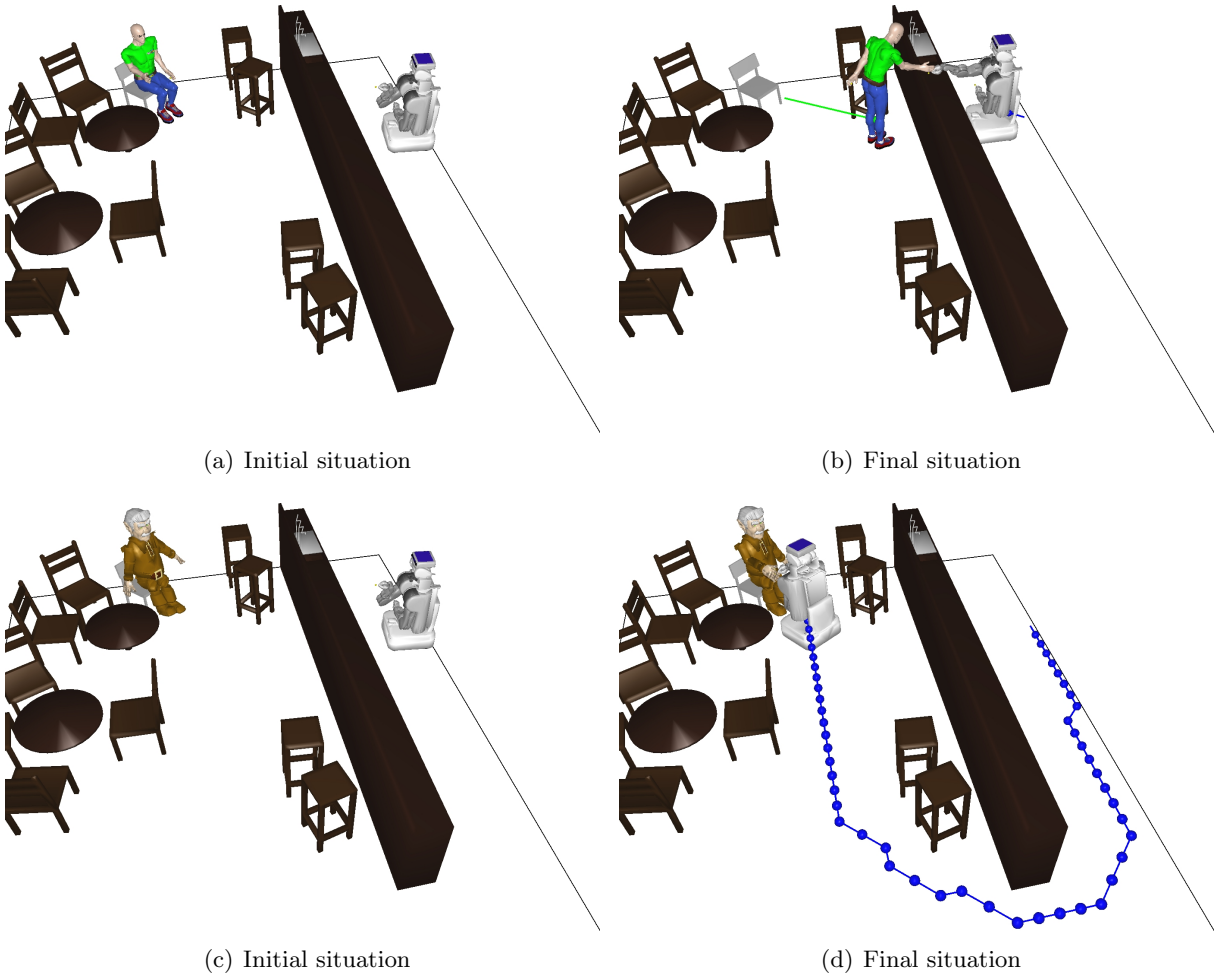
(a) Initial situation



(b) Final situation

FIGURE 2.5: The human cannot be handed directly the object, the robot needs to plan a path for both of them in order to achieve the task

The human preferences may vary depending on the context: he may prefer not standing up, or not moving from his actual location (as he may be buzzy) or, in contrast, he may be eager and in a hurry to get the object and prefer moving toward the robot than waiting for it to come closer. One of the criteria introduced and used here is the *mobility*, which is a representation of this contrast (between the least movement possible for the handover and the fastest possible handover) and enables the system to balance between "shared effort" and comfort.



(a) Initial situation (b) Final situation

(c) Initial situation (d) Final situation

FIGURE 2.6: A young person who is in a hurry to get his drink will express more comfort getting a glass above the counter (a & b) while an older person may be more comfortable while waiting for the robot to navigate to him even if the task will take more time (c & d). The blue path is the robot navigation path, and the green one is the human walking path.

We propose a formulation of the underlying planning problem and an efficient algorithmic solution. Figure 2.6 presents an example of a handover task solved by our planner with different settings of the *mobility* parameter.

This section is organized as follows: Subsection 2.3.1 gives a formal definition to the handover planning problem. Subsection 2.3.2 introduces a simple but yet computationally efficient algorithm based on a combination of grid-based and sampling-based methods. Subsection 2.3.4 presents the simulation and experimental results obtained using this approach. Subsection 2.3.5 shows an implementation on a real robot of the algorithm and a user study done to test the relevance of the *mobility* parameter.

2.3.1 The human-robot handover planning problem

In this subsection, a formal definition of the handover planning problem is proposed. First, inputs and outputs are presented, then the search space is defined along with the feasibility and interaction constraints to be taken into account.

2.3.1.1 Inputs and outputs

The inputs of the problem can be summarized into: the initial configurations of the giver q_g^{init} and the receiver q_r^{init} (q refer to the configuration), the kinematic model of both agents, and the environment (and the position of every object in it).

In the rest of this section, as the robot and the human can both be either the giver or the receiver (It is also possible to consider two robots or two humans), we will refer to the two handover protagonists as the giver (noted g) and the receiver (noted r). The human needs to be taken into account explicitly, two paths (noted τ) will be computed: the first one, τ_g, the path taking the giver from its initial position to the final handover position, and the second one, τ_r, that brings the receiver to his final handover position. Those paths are represented as parametric curves in their respective configuration space.

2.3.1.2 The handover search space

Let's consider the configuration space C formed by the Cartesian product between the giver configuration space C_g and the receiver configuration space C_r:

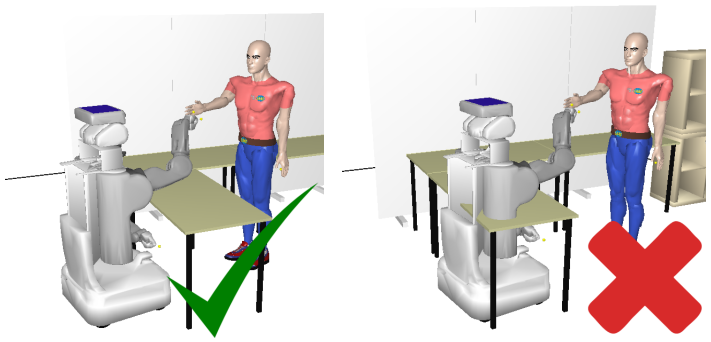
$$C = C_g \times C_r$$

The configuration space of an agent consists on all the configurations allowed by the kinematic of said agent (more details are available in Section 3.2). Thus C contains all configurations allowed for both agents involved in the handover. Finding the solution for this kind of problems implies to find a *handover configuration* $q_{hand} = (q_g^{hand}, q_r^{hand}) \in C$ (q_g^{hand} is the giver configuration and q_r^{hand} the receiver one) which belongs to a subspace $C_{feasible} \in C$ restricted by the constraints listed below:

Collision free both agents configurations, at the handover configuration q_{hand}, must be collision free regarding self-collision, collision with obstacles and with each other. This subspace is named C_{free} and is illustrated in Figure 2.7

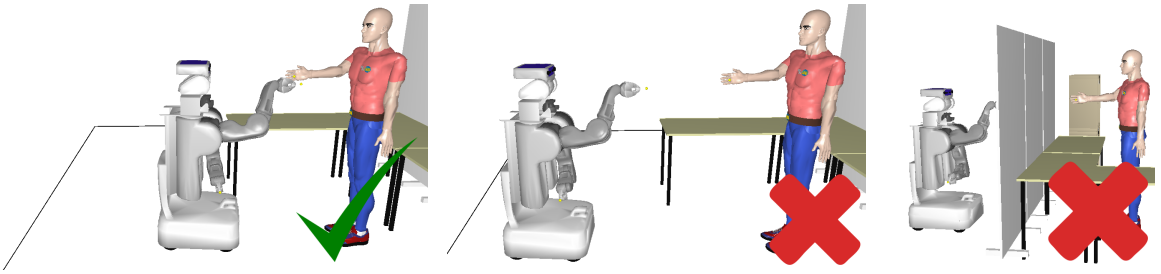
Reachability both agents, at the handover configuration q_{hand}, must be able to reach the exchanged object i.e. the gripper of the robot and the hand of the human must grasp the object at the same time. This subspace is named C_{reach} and is illustrated in Figure 2.8.

Stability both agents, at the handover configuration q_{hand}, have to be stable regarding newton law of mechanics. This subspace is named C_{stab} and is illustrated in Figure 2.9.



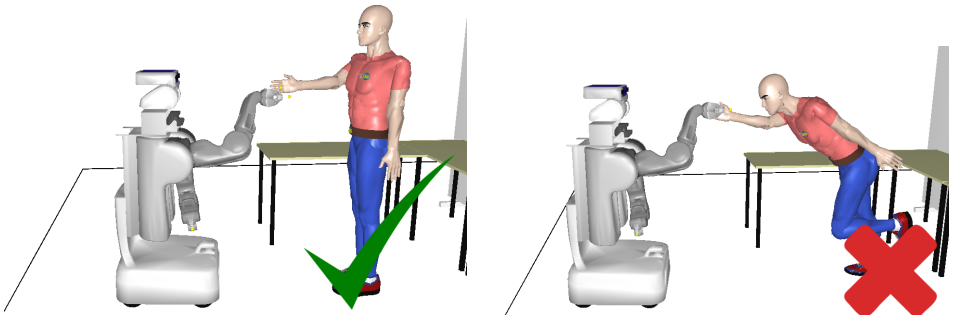
(a) $\in C_{free}$ (b) $\notin C_{free}$

FIGURE 2.7: A collision free solution and a state that is not part of C_{free}



(a) $\in C_{reach}$ (b) $\notin C_{reach}$ (c) $\notin C_{reach}$

FIGURE 2.8: One configuration in C_{reach} and two not in it. In (b) the agents are too far from each other and in (c) a wall separates them.



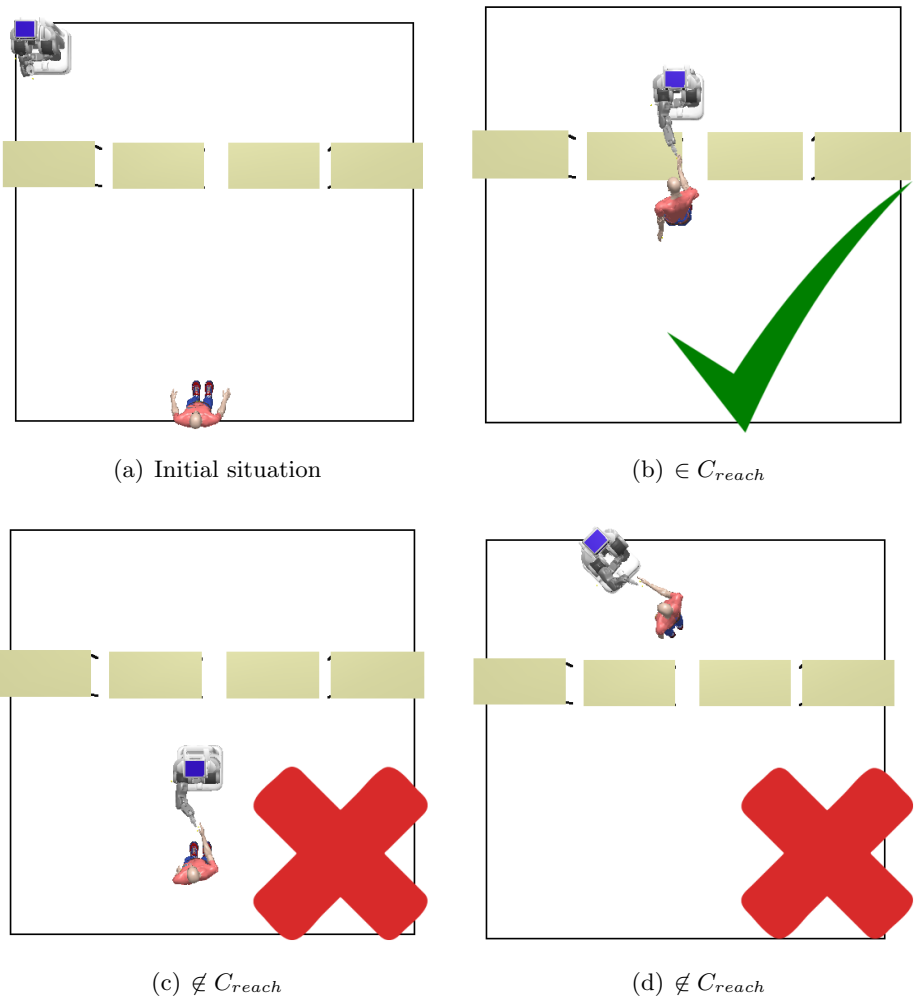
(a) $\in C_{stab}$ (b) $\notin C_{stab}$

FIGURE 2.9: A situation with both agents in stable configuration (a) and a situation where the human is in an unstable position.

Accessibility both agents must be able to navigate from their initial configurations (noted as $q_{init} = (q_g^{init}, q_r^{init})$) to the handover configuration. Let C_{access} be the subset where this constraint is respected. A configuration $q = (q_g, q_r)$ is in C_{access} only if a collision free path exists between q_g^{init} and q_g and another one exists between q_r^{init} and q_r. This is illustrated in Figure 2.10.

The set of all feasible handover configurations $C_{feasible}$ is the intersection of these four subspaces:

$$C_{feasible} = C_{free} \cap C_{reach} \cap C_{stab} \cap C_{access}$$



(a) Initial situation (b) $\in C_{reach}$

(c) $\notin C_{reach}$ (d) $\notin C_{reach}$

FIGURE 2.10: For the initial situation shown in (a), if one of the agents cross the tables, the position is not accessible (c and d): each agent needs to stay in his side (b)

$C_{feasible}$ is the handover search space as it restricts the configuration space to feasible handover configurations. Among the configuration belonging to $C_{feasible}$, some are to be excluded as they may endanger the human, others should be avoided, if possible, as it may not respect social protocols and cause discomfort for him, and some should be preferred as they consider explicitly the human preferences. In the next subsection, we detail some important properties that have to be accounted for in order to generate valid human-robot handovers regarding intuitive social rules and the more formal *proxemics* theory.

2.3.1.3 HRI constraints

In order to account for the interaction safety and the robot intentions legibility, Sisbot et al. (2007b) and Mainprice et al. (2011) introduced a set of HRI constraints that rely on notions from the *proxemics* theory, Hall (1966), as well as user studies such as the one held by Koay et al. (2007). These constraints have been introduced and used by Sisbot et al. (2007b) to generate human-aware navigation motions and used also by Mainprice et al. (2011) to generate comfortable handover motions in cluttered environments. These constraints can be summarized in three sets.

First, the comfort constraint that can be divided into three criteria. Let c_{comf} be their corresponding cost value:

The robot proximity (Hall (1966)) in this theory, the human is considered at the centre of three co-centric circles as depicted in Figure 2.11. The first one, the smallest, marks the limit between the intimate area where very close friends and lovers can enter, and the personal area, the normal area for friends and family to be. The next circle, bigger than the first one marks the limit between this last area and the social/consultive area, where acquaintances and work colleagues might interact. The last zone –Public–, delimited by the biggest circle, is used to interact with a stranger (in speech for example).

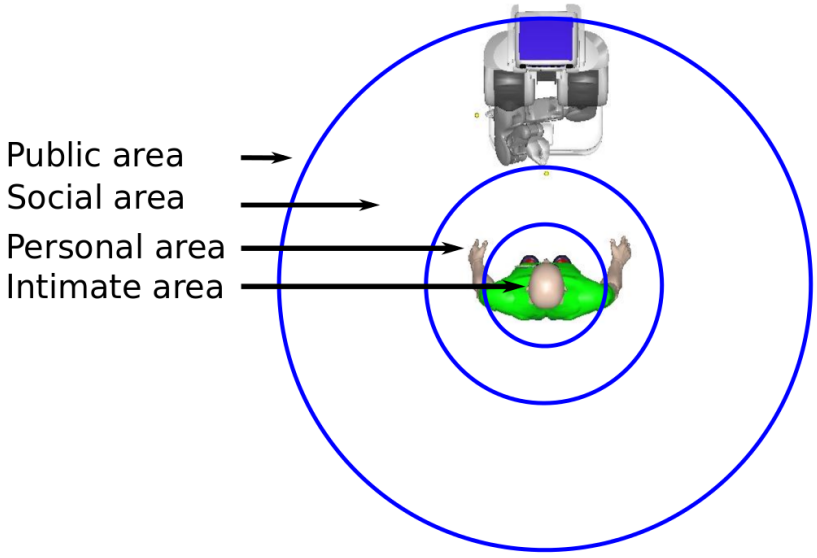
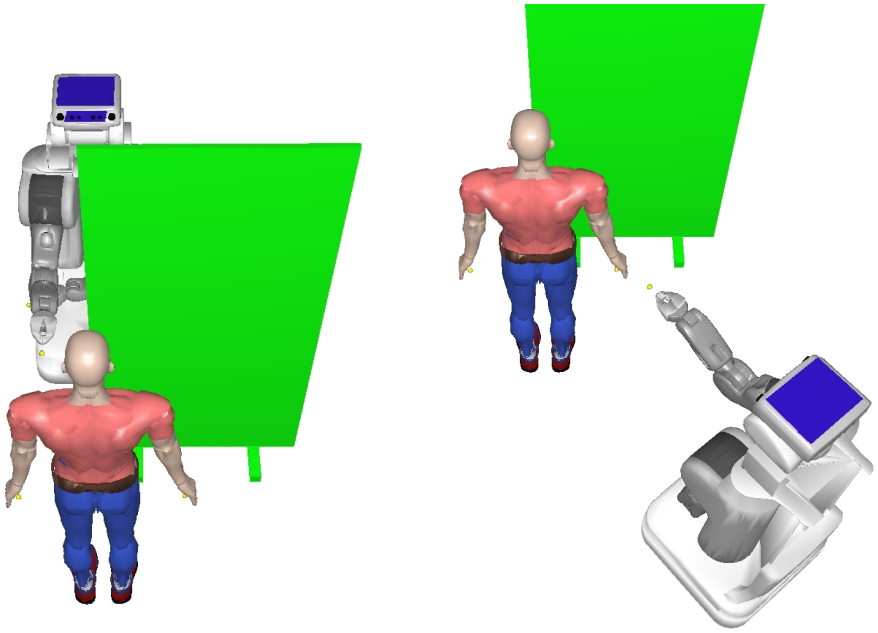


FIGURE 2.11: The *proxemic* theory areas around a human.

The robot visibility (Koay et al. (2007)) in order to limit the effect of surprise, the robot should avoid the areas behind and close to the humans, but also areas where part or all the robot is hidden by an obstacle. Figure 2.12 shows two situations where this constraint is not respected.

The musculoskeletal comfort of a given posture (Marler et al. (2005)) this criterion represents the effort the human must dedicate to move and stay in the handover configuration from a neutral posture. Figure 2.13 shows a series of position going from a comfortable one to less comfortable ones.

Second, the motion constraint which preserves the humans from great displacement efforts, depends on the sum of two parameters: the navigation, and the standing up motion. The navigation parameter can be inferred from the length of the human path (starting at his initial position and finishing at the handover configuration). The standing up motion parameter represents the need for the human to stand up before the navigation can start (the cases where the human is sited) and is simply added to the navigation cost as an offset (when relevant) to for the motion cost c_{mot}.



(a) Behind an obstacle (b) Behind the human

FIGURE 2.12: Two social uncomfortable positions where the robot is partially hidden to the human

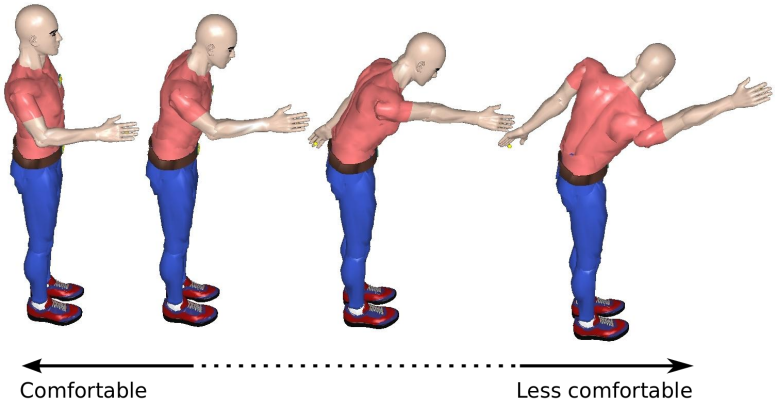
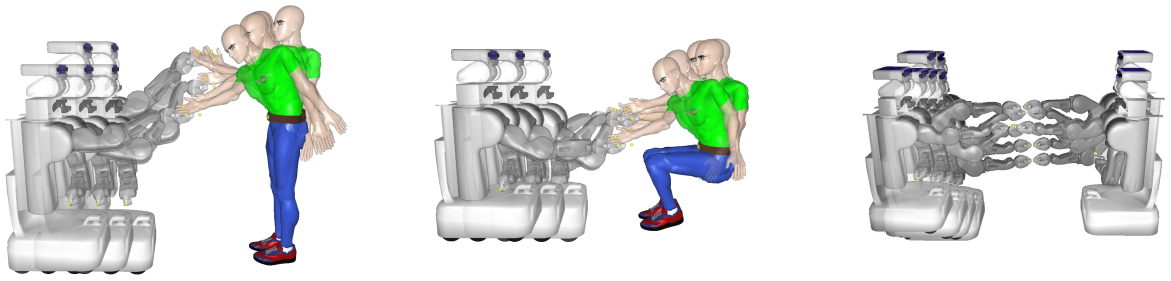


FIGURE 2.13: The musculoskeletal comfort for a human in a handover configuration (as shown in Marler et al. (2005))

Finally, the efficiency constraint that limits the total duration of the handover and favours efficient plans. This cost, c_{len}, is computed based on the maximum value of the time taken by either agent to reach the handover configuration (we consider only the fastest path to compute this cost).

Some of these desired constraints such as the human displacement and the action duration may contradict one another (if both agents share the work load, the task will be done faster but if we minimize the human displacement, the task will take longer as the robot must do most of the work). To balance the impact of the different properties on the output plan, we use the *mobility* parameter reflecting the human's physical capabilities and his eagerness to obtain the object.

Indeed, the handover duration may generate discomfort if it does not match with the human possible eagerness or *urgency* to get the object. High mobility values will balance motion and comfort constraints to favour quicker plans, resulting in the final cost defined as:



(a) With a standing human (b) With a sitting human (c) With a robot

FIGURE 2.14: Some handover configurations for human-robot handovers and robot-robot handovers (these are just informative, more positions are available).

$$c = (c_{mot} + c_{conf}) * (1 - mobility) + c_{len} * mobility$$

Where $mobility \in [0, 1]$. As explained in next section, these interaction constraints and their corresponding cost functions are evaluated during the planning process and are combined together according to the human preferences modelled by the *mobility* parameter.

2.3.2 The proposed algorithm

This section presents the handover planner that was developed to compute human-robot handovers while accounting for the interaction constraints introduced earlier. The approach relies on a combination of grid-based and sampling-based algorithms that consider the workspace obstacles and the kinematics models of both agents. After some grid based pre-processing, the method consists of iteratively sampling feasible handover configurations, evaluating their cost and finally returning the minimal cost plan obtained.

The main steps of the handover planner are sketched in Algorithm 1. The initialization phase, called `initGrids`, consists on computing a planar grid where each cell contains information about the agents accessibility (if they can reach it or not) and the navigation distances from each agent initial position to this cell (Figure 2.15). In this phase, a set of preselected handover configurations between the two agents is loaded (in some cases, multiple sets are loaded Figure 2.14)

After the initialization, the algorithm enters a loop (from line 4 to 23), where each iteration consists on finding a feasible handover configuration $q_{hand} \in C_{feasible}$, where the exact values of both agents degrees of freedom are encoded. After the loop, the algorithm chooses the best q_{hand} according to the cost presented in Section 2.3.1.3.

In order to find a q_{hand}, the algorithm goes through 6 steps:

- Sampling a random position, $p = (x, y, \theta)$, in the receiver accessible space (line 5)
- Computing the navigation path τ_r of the receiver from his initial position to this sampled position (using a standard technique, Choset (1991), consisting of descending the distance gradient in the pre-processed receiver grid) (line 6)

- Transforming p into a fully specified q_{hand} by iterating through the set of preselected handing over configuration and choosing a collision free configuration. (line 7)
- Extracting the givers position from q_{hand} (line 11)
- Computing the navigation path τ_g of the giver from his initial position to this position (same technique as τ_r) (line 12)
- Computing q_{hand} cost, as explained in the previous subsection. (line 16)

If one of the steps fails (such as failing to find a collision free configuration or a path) the algorithm loops over the steps until a stopping criterion is satisfied (Line 17 to Line 21). When all the steps succeed, the computed cost is compared to the stored handover configuration cost. If it is lower, the new q_{hand} is stored and the loop continues. In the current implementation the stopping conditions combine two criteria, and break out of the loop as soon as one of them is reached:

Maximum time Set by the user, and checked after each loop, it stops once the maximum time is reached.

A minimal improvement of the best current solution The user sets a threshold: if after a fixed number of iterations (also set by the user) the algorithm does not improve the cost with a difference bigger than the threshold, it breaks out of the loop.

The final paths τ_g and τ_r consist of a set of way points corresponding to the traversed cells centres interpolated by straight lines. The orientation θ along these paths is selected implicitly by facing the agent to the next way point.

The next subsections describe in further details the processing done during the initialization phase and the main steps of the algorithm. Some additional pre-processing that can be done to speed-up the sampling of constrained handover solutions are also described.

2.3.2.1 Distance propagation and initialization

In order to speed up the computation of feasible handover configurations and the cost evaluation of those solutions, the method integrates a precomputing phase in which 2D grids are constructed and processed. Two grids, depicted on Figure 2.15, one for the giver (referred to as the *giver grid*) and one for the receiver (referred to as the *receiver grid*), provide an approximation of the free-space and the navigation distance to the initial position. This enables to find, at a very low computational cost, the regions of the workplace accessible for each agent.

The free-space grids are computed using bounding cylinders of the agents in resting postures. The resting postures, also depicted in Figure 2.15, correspond to navigation configuration of the arms. A cell is marked as free if when placed at its centre, the corresponding bounding cylinder is not in collision with the environment.

The accessible space and the navigation distance to the initial position of a cell are simultaneously computed with a standard wave propagation technique: for each cell, a collision free test is done with

Algorithm 1 Computing handover plans

```
1: function ComputePlan(p_g,p_r,mobility)        ▷ Giver and robot receiver position and the human mobility
2:     cost^best ← ∞
3:     G ← initGrids(p_g, p_r, mobility)
4:     while StopCondition( ) do
5:         p ← SampleReceiverPos
6:         τ_r ← DescendOnReceiverGrid(p)
7:         q_hand ← BestFeasibleConf(p)
8:         if q_hand == NULL then
9:             continue looping
10:        end if
11:        p_giv ← GetGiverPos(q_hand)
12:        τ_g ← DescendOnGiverGrid(p_rob)
13:        if τ_g == NULL then
14:            continue
15:        end if
16:        cost ← ComputeCost(mobility,τ_g,τ_r,q_hand)
17:        if cost > cost^best then
18:            continue
19:        else
20:            cost^best ← cost
21:            StoreBest(q_hand,τ_g,τ_r)
22:        end if
23:    end while
24:    return (q_hand,τ_g,τ_r)        ▷ agents handover configuration and their navigation path
25: end function
```

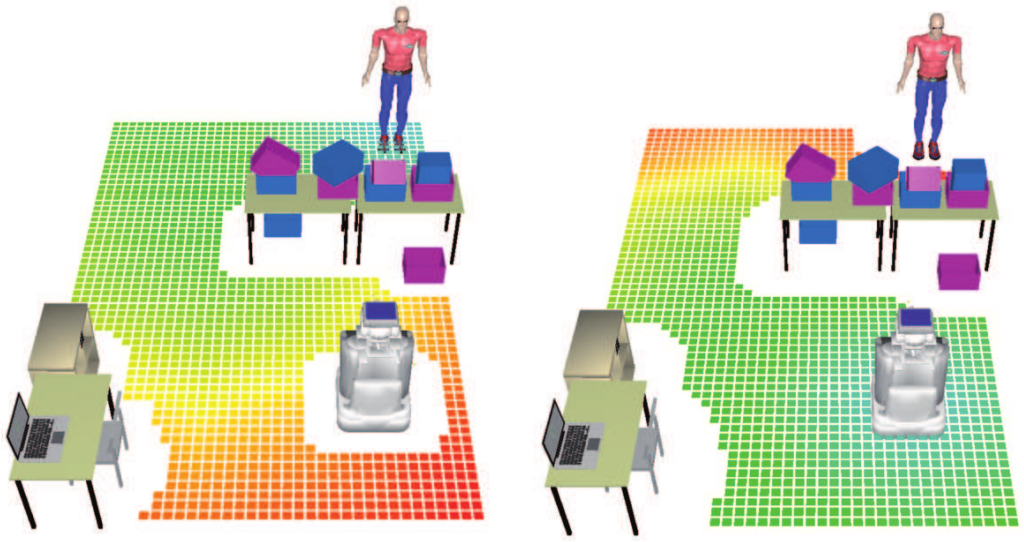
the bounding cylinder, then its distance to the initial positions is computed (not the Euclidean, the navigation distance). Figure 2.15 shows the free space and the propagated distance of a robot and a human from their initial position, where green cells are close to the initial position and red cells are far.

As mentioned earlier, during the initialization phase, this method loads a set of predefined handing configurations as illustrated in Figure 2.16. These handover configurations are named Q_{HR} in the rest of the chapter. They are selected offline and do not depend on the workspace nor on the absolute position of the agents. Thus, each configuration is defined relatively to the receiver position and consists of the receiver and the giver arm Degrees of freedom.

2.3.2.2 Sampling the receiver positions

The first step of each iteration consists of sampling the receiver position and orientation $p = (x, y, \theta)$ inside the accessible space stored in the pre-processed grid (*receiver and giver grid*). In order to sample this triplet a cell is selected then a point is sampled inside the cell and finally an orientation is randomly sampled.

For each position p chosen, we find only one handover plan, that we consider as the best given our criteria, thus, it is important to sample the positions that yield better solutions. Subsection 2.3.3 provides two enhancements of the pre-processing phase to bias both the selection of the cell and the orientation of the receiver.



(a) Human distance grid (*receiver grid*) (b) Robot distance grid (*giver grid*)

FIGURE 2.15: The distance propagation (a) is human centred and (b) is robot centred. The green cells correspond to nearest positions, and the red the farthest.

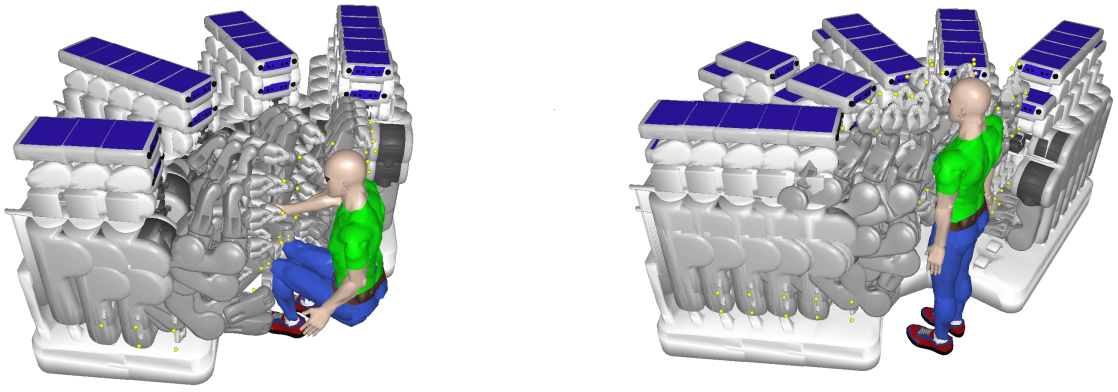


FIGURE 2.16: The preselected configurations of a robot relative to a human standing and sitting.

2.3.2.3 Returning the best feasible configuration

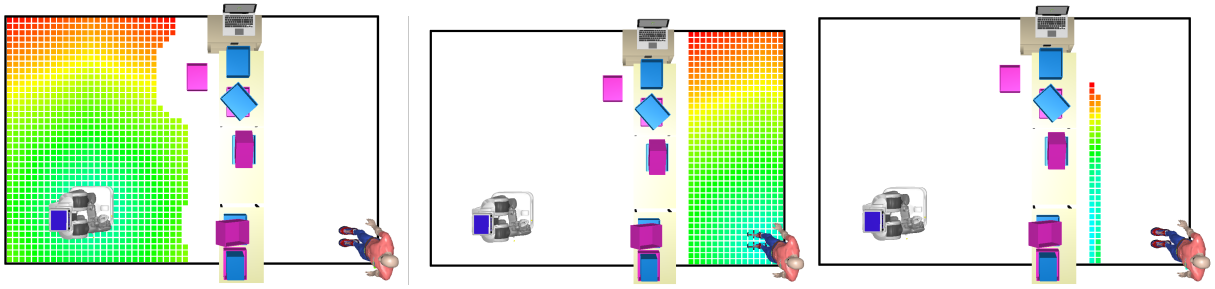
The configurations Q_{HR} illustrated in Figure 2.16 are sorted according to the c_{comf} cost (see Subsection 2.3.1.3). Here, this cost is computed independently from the environment and is used as a heuristic. Later, during the real cost computation, the obstacles are considered for a better estimation of the cost. When searching for the best feasible handover configuration at the receiver position p, the first feasible configuration is selected (i.e. collision free and accessible to the giver).

This process enables the method to find solutions in constrained environment (e.g. a handover through a small windows connecting separated workspaces) while saving the computation time as the cost does not need to be recomputed (due to the sorting).

2.3.3 Reducing the search space and biasing the sampling

One of the most tricky cases that the method has to address consists of finding handover plans when the human and the robot do not lie in connected parts of the workspace, as depicted in Figure 2.17, by sharing the effort in the plan. In general, but especially in this case, the algorithm samples the receiver accessible space without considering the giver accessibility and reaching capacity, which results in a high number of failures when testing the accessibility of the receiver. The same problem arises when randomly sampling the receiver direction.

In order to speed-up the generation of feasible handover configurations, we propose two enhancements to the basic version. First, we construct a smaller grid by merging the two others: it contains only cells were the receiver is close enough to cells accessible to the giver and will be referred to as the *combined grid*. Then, a value which approximates the total cost of the candidate handover solution is precomputed and stored in this *combined grid*. The receiver position sampling is then biased to low cost regions of this accessible space. The sampling of the receiver direction is also biased to an estimate of the giver approaching direction.



(a) *Giver grid* (b) *Receiver grid* (c) *combined Grid (based on the receiver grid)*

FIGURE 2.17: The human and the robot lie in separated part of a workspace, parted in two by a table. The accessible space of the giver (a) and the receiver (b) and the *combined grid* (c) are computed.

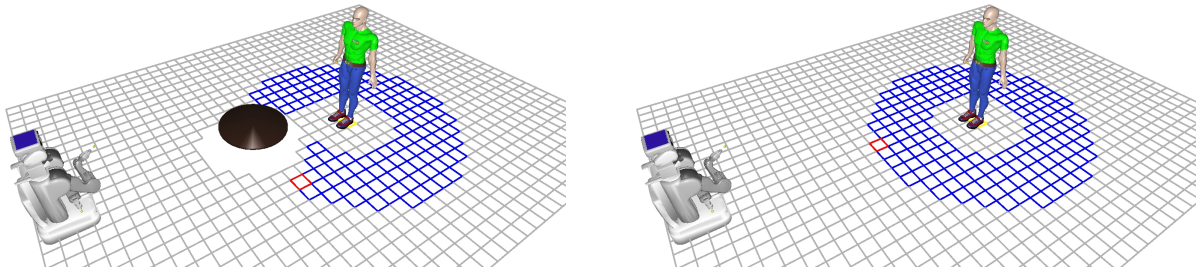
Combined grid

To generate the *combined grid* of the agents accessible space, the minimal and maximal giver-receiver distances in Q_{HR} are computed. These *min* and *max* values are used to define a ring region around the receiver such as illustrated in Figure 2.18. Then each accessible cell in the *receiver grid* which ring region does not overlap any cell in the *giver grid* (i.e. that cannot yield a valid handover configuration) are removed from the *combined grid*. As the algorithm first step is to sample the receiver position, the choice was made to use the receiver grid as the base to compute the *combined* one.

Hence, by discarding the cells that are not within reaching distance from the giver accessible space, a smaller *combined grid* is obtained, such as depicted in Figure 2.17(c). Note that the *combined grid* does not account for 3D collision checking which is performed in the exploration phase.

Bias sampling

In order to bias the sampling of the receiver position (line 5) to candidates which will yield better handover configurations, each cell in the *combined grid* is associated with a cost:



(a) Ring region (b) Solution on the side

FIGURE 2.18: The estimated handover positions of the robot, feasible (in blue) and closest feasible $cell_{min}$ (in red) for a given human position according to the reaching capabilities of the human and the robot.

$$c_{Bias} = c_{mot} * (1 - m) + c_{len} * mobility$$

This cost approximates the final c presented in Subsection 2.3.1.3 and is used to evaluate the quality of the candidate handover solution.

In order to bias the human direction sampling, the valid cell that minimizes the robot motion from its initial position to the ring region (in red in Figure 2.18) is stored in the *combined grid*. When sampling θ, directions facing this cell are favoured.

Next section provides simulation results of this algorithm with different settings of the *mobility* parameter.

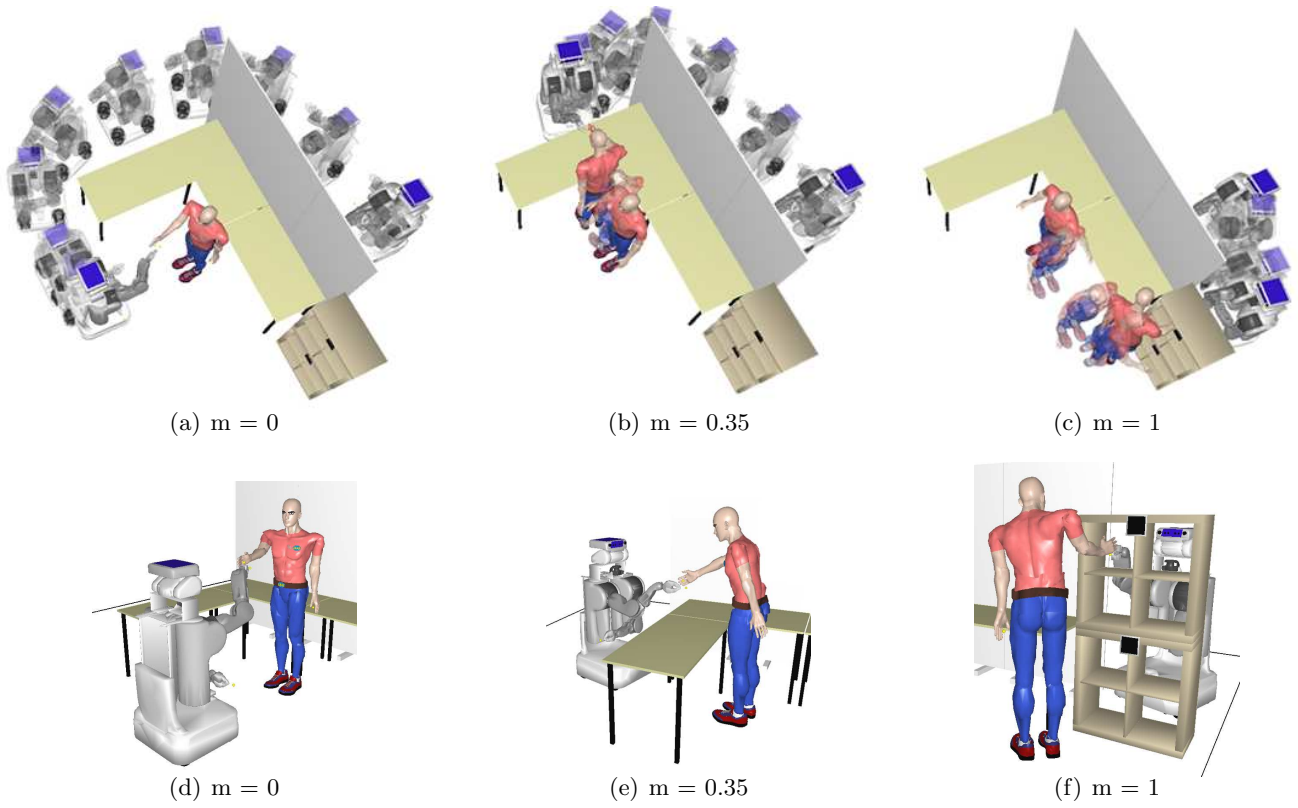
2.3.4 Results

This section reports the algorithm ability to find handover plans between a robot giver and a human receiver in workspaces containing sparse obstacles, and the strategies it produces using different values of *mobility*, with it convergence rate when using different pre-processing variants and their sampling schemes.

In order to assess its performance, the algorithm has been implemented, along with test environments, in *Move3D* Siméon et al. (2001) and simulations were performed on a computer equipped with a 2.26GHz INTEL processor running on one core only.

2.3.4.1 The mobility parameter

Figure 2.19 shows three handover strategies that have been computed for the same problem using three values of *mobility*. For low values of *mobility*, the human (receiver) is supposedly less involved, asked as little effort as possible, on the contrary high values of *mobility* require more effort and participation from him, resulting in faster handover strategies. This time enhancement results from the parallelisation of the navigation (both agents navigate at the same time, making the global time needed to achieve the task smaller) and from the human navigation speed, which is higher than the one of the robot.



(a) m = 0 (b) m = 0.35 (c) m = 1

(d) m = 0 (e) m = 0.35 (f) m = 1

FIGURE 2.19: Three values of the *mobility* parameters are used to generate three handover strategies. The first three pictures depict the resulting trajectories while the three bottom pictures show the final handover configuration that accounts for the 3D obstacles.

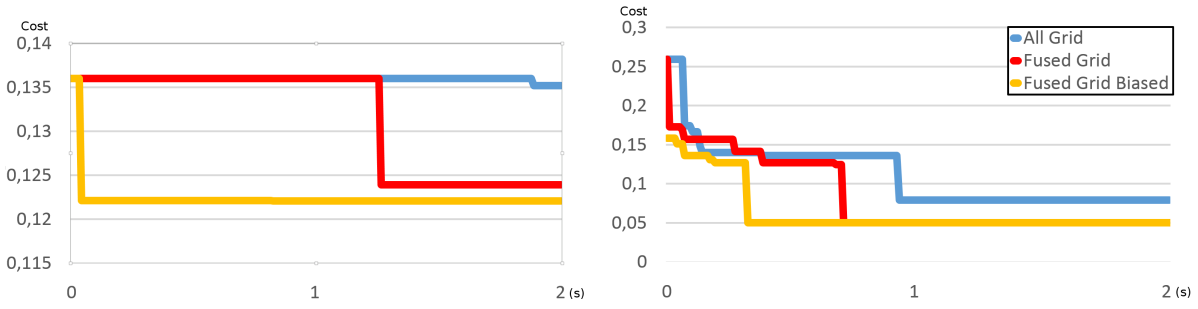
- *mobility = 0*: Generates a long path for the robot to reach the handover position but the human does not move.
- *mobility = 0.35*: A shorter robot path to a feasible handover position over the table is allowed by a small displacement of the human.
- *mobility = 1*: Evenly shared effort between the robot and the human enables a constrained handover position through the shelves.

Note that depending on *mobility* the solution proposed by the planner can be radically different. The resulting plan accounts for the feasibility of the handover position and motion using the 3D models of both agents even though planning of navigation motion is performed in 2D Cartesian space.

2.3.4.2 Performance of pre-processing variants

Figure 2.20 shows the cost improvement over two seconds on a single run corresponding to approximately one thousand sampled positions and five hundred fully tested handover strategies on the example of Figure 2.19. The graphs illustrate the interest of the proposed *combined grids* and bias variants.

The simplest case (*mobility*=0) is not shown in the figure since all variants converge to the displayed solution after a couple of iterations. However, for the two more complex cases, the basic pre-processing shows difficulty to find handover above the table (*mobility*=0.35) or through shelves (*mobility*=1). In

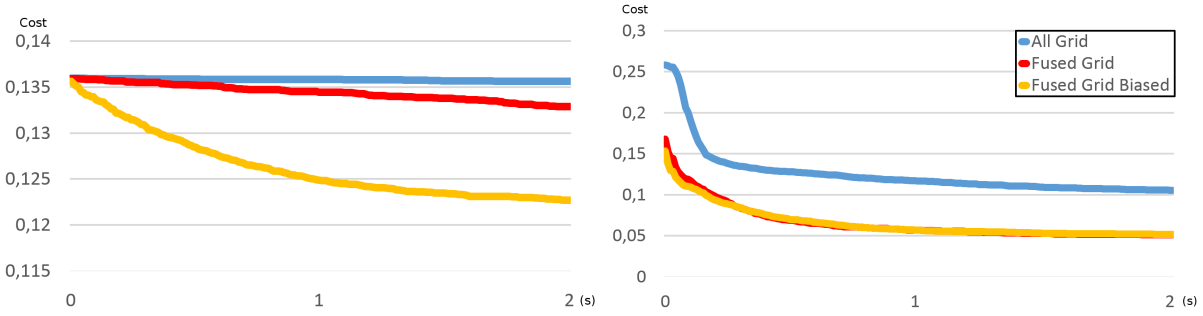


(a) *mobility*=0.35 (b) *mobility*=1

FIGURE 2.20: Unique convergence curve, with two settings of *mobility*, using the three pre-processing variants yielding three sampling strategies over the scenario depicted in Figure 2.19

order to generate more direct (and lower cost) handovers, the basic version requires much more iterations than the other variants. The basic method keeps sampling handover configurations in the free space similar to the ones shown in Figure 2.19(a) and 2.19(d), but placed elsewhere in the free-space.

Thanks to the *combined grids* the algorithm generates more samples on the boundary of the free space (close to the tables) and thanks to the bias it discovers more easily the solution through the shelves. Figure 2.21 shows similar results averaged over 300 runs of the planner which confirms these results. In average a good quality solution is obtained in less than 1 sec with the *combined grid* and bias sampling pre-processing.



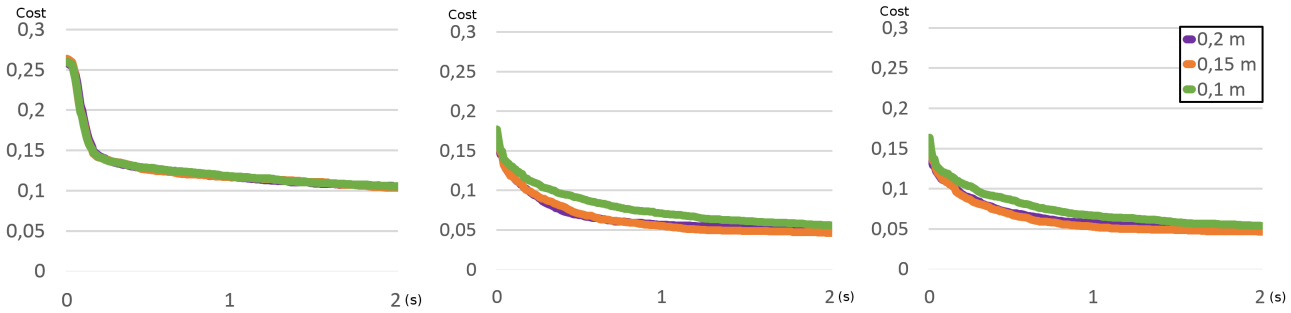
(a) *mobility*=0.35 (b) *mobility*=1

FIGURE 2.21: Averaged convergence curve over 300 runs, with the same settings as shown in Figure 2.20 (*mobility*=0.35 right, *mobility*=1 left)

2.3.4.3 Influence of discretization

Figure 2.22 shows for the case *mobility*=1 that the grids size has little influence on the performance of the handover sampling stage. The influence is limited to the pre-processing stage for which distance propagation and *combined grid* computation depend on the resolution. Note that the pre-processing is problem dependent (initial position of the robot and the human) and has to be performed for each query.

For the scenario in Figure 2.19, the pre-processing requires 200$msec$ (resp. 660$msec$) for a resolution of 20cm (resp. 10cm) and the additional cost of computing the *combined grid* is 68$msec$ (resp. 1109$msec$). Thus, using a resolution of 20cm, the pre-processing stays below 1sec, however the time grows exponentially as the discretization gets smaller.



(a) Normal gird (b) combined grid (c) combined grid Biased

FIGURE 2.22: Averaged convergence over 300 runs on the scenario of Figure 2.19 with m=1. Three distinct grid sizes 10, 15 and 20 cm are superposed

The performance above tends to indicate that this approach could be used to dynamically adapt the computed handover motion plan during execution in order to account for possible human motions or to adapt the *mobility* factor if the human motion does not look compatible with the proposed handover strategy.

2.3.5 Implementation

This work has been implemented on a PR2 robot (from Willow Garage (2008)) in the context of a demo as presented by Gharbi et al. (2013) (the video is available at: `https://www.youtube.com/watch?v=8ZFGsnDmH08`) and a user study as depicted in this section.

2.3.5.1 The demonstration

The demonstration consists of a human asking the robot to bring him a can. The robot is provided with the models of every object in the world, and uses his sensors (such as classical cameras, RGB-D cameras, and lasers) to place objects, humans, and himself in his internal 3D environment. Based on this environment, it builds, in real-time, a symbolic model stored as an ontology, Lemaignan et al. (2010), which holds spatial relations between objects and agents along with a set of affordances (such as reachability, visibility and so on) Lemaignan et al. (2012b).

The human verbal input (such as "Bring me a can", "take this object", or "put the can on the table") is parsed, then semantically grounded in tight interaction with the symbolic model. It allows for multi-modal and perspective-aware communication: affordances (like visibility) and human gestures that are recognized (like pointing) and stored in the ontology are used during the grounding process.

Starting from this point, the robot go fetch the object (the object position is known as a fact: the can is on the big table) by localising it precisely and picking it, then it plans and executes a handover as depicted in this section.

Figure 2.23 shows different scenarios where the robot successfully brought me the can.

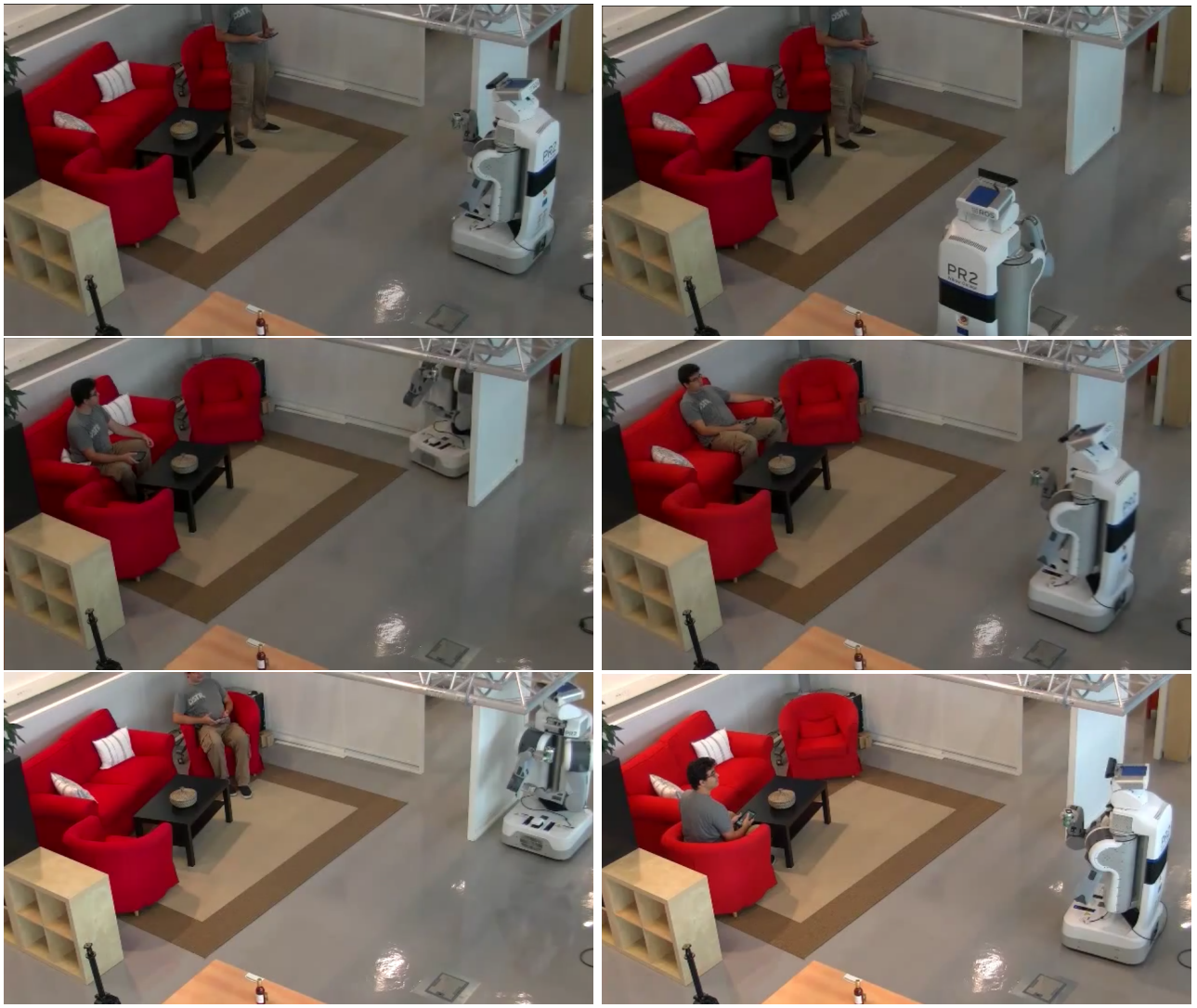


FIGURE 2.23: The robot executes a handover of a can to the human, all these situations have been tested with success.

2.3.5.2 The user study

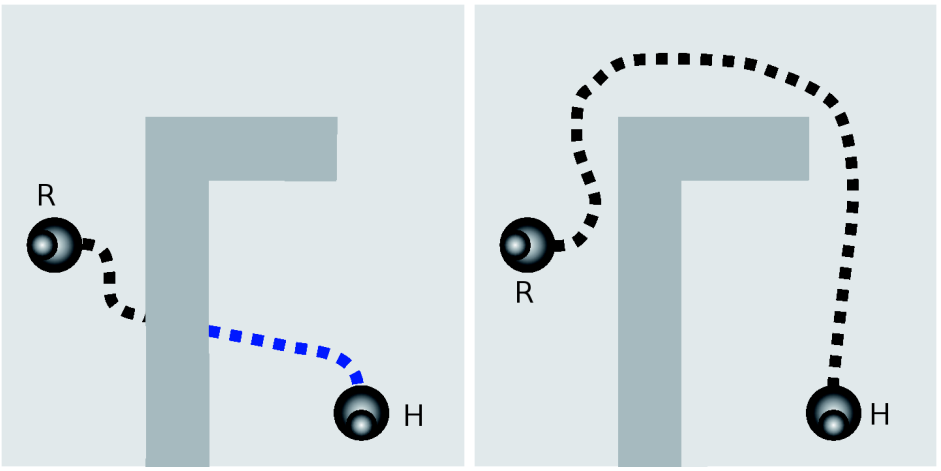
We ran a human-robot interaction experiment confronting the participants to choices of hand-over configurations provided by our planner: the shortest-time feasible plan at the cost of substantial effort asked from the human, or the plan that minimizes the human effort, at the cost of low global time performance. Objective and subjective measures are discussed to validate our hypothesis.

Hypothesis

- The *mobility* of the human receiver depends on the task and intrinsic parameters associated to the receiver such as physical capacities or involvement in another task.
- Accounting for the receiver *mobility* leads to more efficient hand-overs, especially when it matches the context.

Experiment Design

We have designed an experiment consisting of collecting objective and subjective data on human-robot



(a) Shortest plan (b) Less effort plan

FIGURE 2.24: Two handover solutions: In the first case the robot (R) proposes the object between the walls. The motion is shorter but requires the human (H) to stand up to grasp the object. In the second case the robot comes closer to the human with a large detour.

handovers. Each one of the 34 subjects was confronted to one handover only, executed by the PR2 robot Willow Garage (2008). At their arrival, the subjects were briefed before the interaction, the handover was video recorded and the participants were finally handed a survey.

Scenarios

Two distinct handovers were planned using high and low values of the *mobility* parameter resulting in one plan which prioritize the time and one plan which prioritize the human-effort, Figure 2.24 depicts the two handovers motions. In both cases the participants are sitting on a comfortable chair on the other side of a wall, where the robot starts by picking a ball. This wall presents a slot that can be used to handover the object.

In the first handover, planned with the high value of the *mobility* parameter, the robot makes use of the direct access to the human through the wall slot. When handed the object between the walls the human sitting on the chair is unable to reach it, requiring him to stand up and walk to grasp the object. In the second handover, planned with a low value of the *mobility* parameter, the robot makes a detour motion over the L shaped obstacle. Thus, the robot comes closer to the human who only requires to reach for the ball while staying seated. In both scenarios, the robot announces verbally when it is ready to give the ball.

Tasks

The participants are briefed to sit down on the chair and are made aware that the PR2 robot is going to hand them a small ball in order for them to put it in a plastic tube. They are also told that the PR2 cannot put it itself inside a plastic tube that is placed next to the chair and that they have to put the ball themselves. Two additional tasks are assigned to the participants in order to act on the human involvement in the handover task. The first task, corresponding to a high *mobility*, consists of watching a chronometer while trying to achieve the task as fast as possible. The second task, corresponding to a low *mobility*, consists of playing a sudoku game. The first task should stimulate the participant to feel

eager to get the object while the second task should force the participant to pay less attention to the handover.

Four types of interactions are then possible, referred to in the rest of this section as:

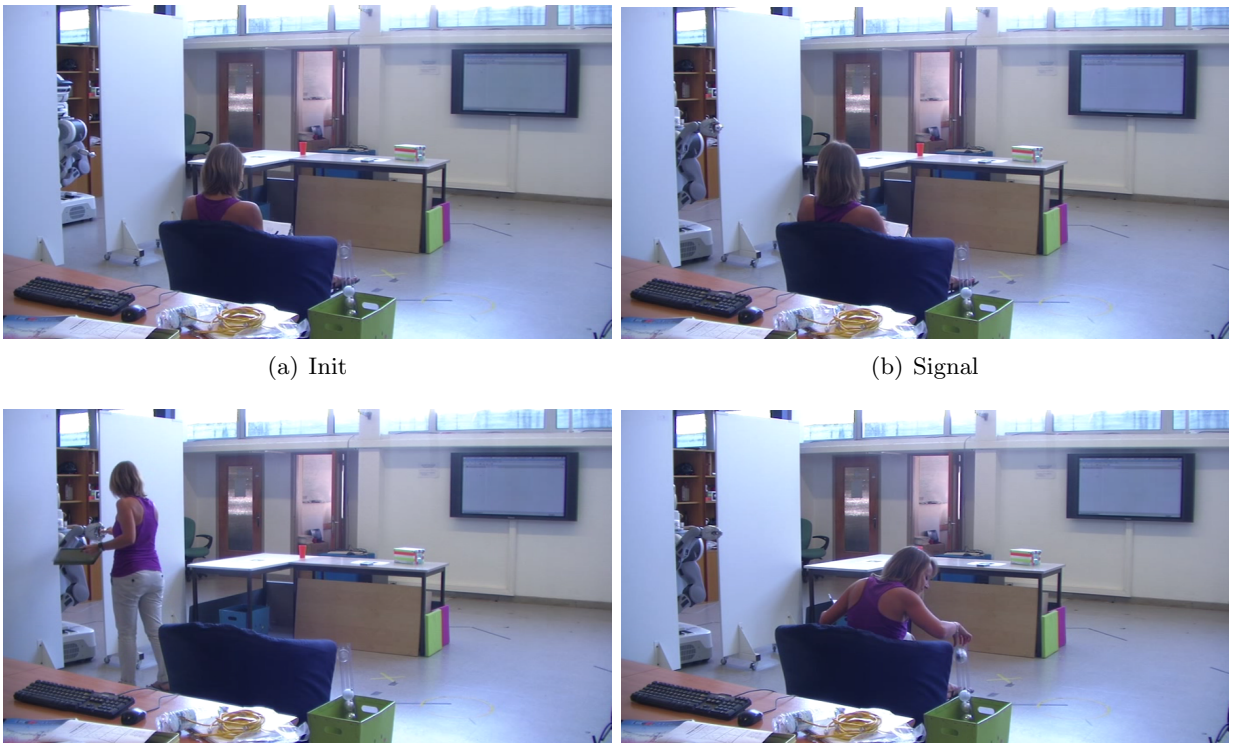
A : Shortest handover plan with chronometer

B : Shortest handover plan with 'sudoku'

C : Less effort handover plan with chronometer

D : Less effort handover plan with 'sudoku'

Figure 2.25 and Figure 2.26 illustrate type B and D scenarios.



(a) Init (b) Signal

(c) Handover (d) End

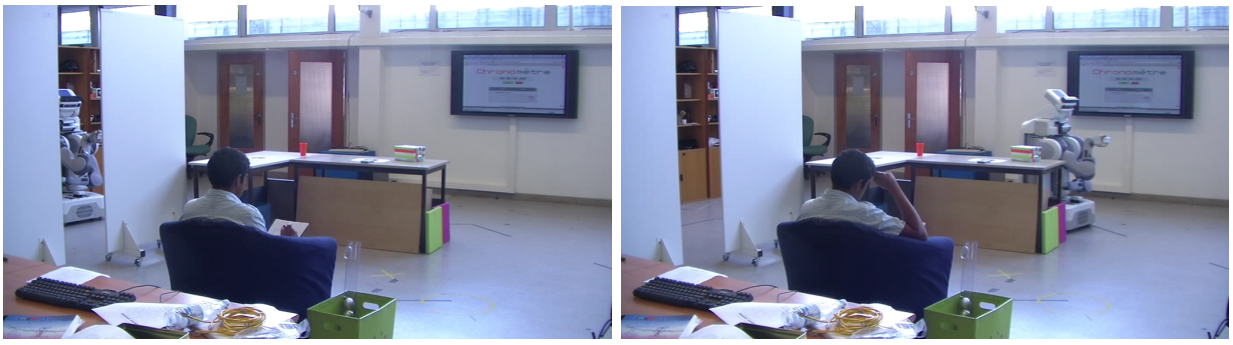
FIGURE 2.25: Type B scenario: the robot hands-over an object through the walls

Evaluation

In order to evaluate the fluency and the efficiency of the interaction, two measures were extracted from the videos recordings: the reaction time (time between the participant first motion and the robot releasing the ball) and the total time (between the robot starting motion, and the ball entering the tube). The quality of the interaction is evaluated with a set of subjective criteria collected by compiling the survey's answers.

The survey

The form combines three types of questions; *open*, *closed* and *evaluation*. Eight multiple choice questions were asked, five of which enable the participants to evaluate one of the interaction criteria with a five-point Likert scale. The evaluation questions concern: the comfort of the handover, the distance appropriateness, the scariness of the robot, the surprising factor, the eagerness of the participant to get the object and the timing appropriateness. The closed questions aim to determine if the participants understood the



(a) Init (b) Signal

(c) Handover (d) End

FIGURE 2.26: Type D scenario: the robot handover the object with a large detour

location where the object was going to be exchanged and if they found the location natural or not. Finally, participants were asked if they would have preferred to be handed the object in one of the alternative solution as depicted in Figure 2.24.

Results and discussion

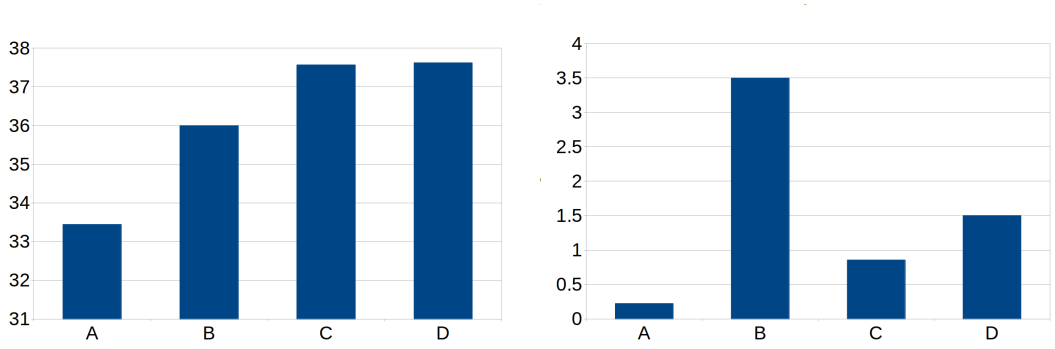
In this part we compare the times measured on the video recordings and the answers of the survey to study the validity of our hypothesis. The results are reported in Figure 2.27 and Figure 2.28.

Times

The total times of the task reported in Figure 2.27(a) indicate that the handover was realized faster with motions of type A and B, which is normal as the handover chosen here prioritizes the handover global time. It is actually faster in A than in B because of the reaction times reported in Figure 2.27(b): it is shorter for the participants given a time constraint and longer when they are given a game (for both kinds of priority). This suggests that participants were more aware and prompt to accomplish the handover when given a time constraint. We believe this particular observation of the subjects' behaviour corroborates the second part of our hypothesis that postulates that the current task modulates the *mobility* of the human receiver.

Subjective measures

Concerning the subjective measures, they are summarized in Figure 2.28. The scariness of the robot affects the users only in type D interaction, which is quite normal as they are focused on the sudoku and suddenly the robot reaches out near them with the object. The distances are felt less appropriate in case A and B, the cases where the robot stand behind the wall. This can be explained by the fact that we



(a) Total time (s) (b) Reaction time (s)

FIGURE 2.27: The total time and the human reaction time to fulfil the goal

couldn't give the subjects the eagerness feeling (also in the figure, the difference between the cases is not significantly interesting) and thus, they didn't understand why the robot was so far. For the same reason, the comfort was better appreciated in the two last cases. However, nearly all the subjects from case B would have preferred the other path for the robot, and while the subjects from D liked unanimously the path, some in C would have preferred the other one. This shows that when the context didn't correspond to the robot actions, the subjects didn't like it, which corroborates the first part of our hypothesis.

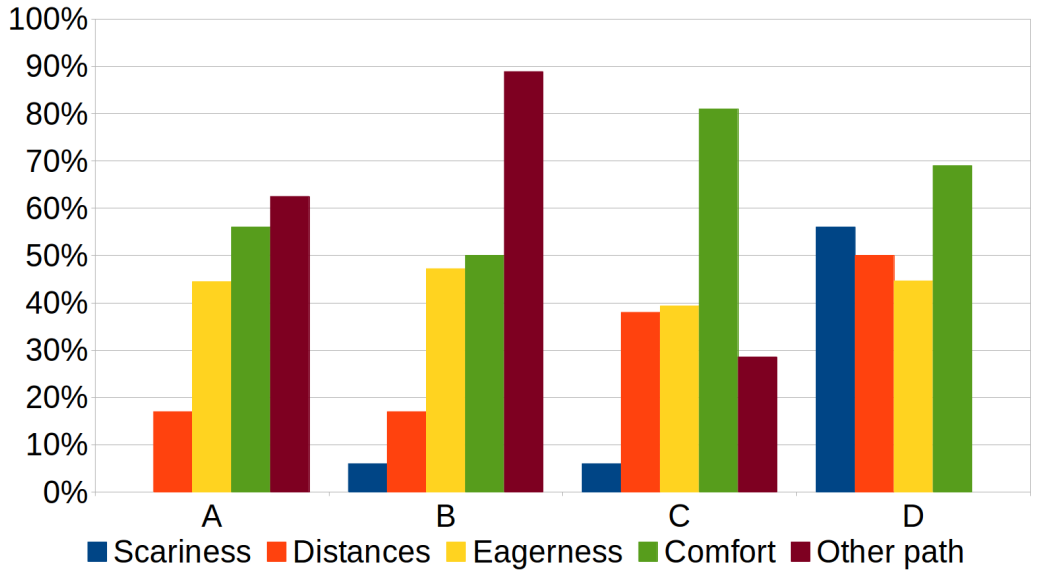


FIGURE 2.28

Conclusion on the user study

The user study confirms partially our hypothesis, but another one is needed where the eagerness parameter is handled more carefully. The videos of this user study are available at https://www.youtube.com/playlist?list=PLJeAfn0C8Ci3DMyLG3Q1KzXgeCMIBg1fW

This section was about how an agent can handover an object to another agent, in the next section, we address a more global problem of where and how to do a sequence of handovers in order to bring an object from an agent (robot or human) to another one.

2.4 Multiple agent handover problem

This section presents the work done, in cooperation with Jules Waldhart, concerning the computation of a handover sequence, where multiple agents (humans and robots) are involved in a task with a predefined goal which is to bring an object from a starting agent to a target one. This work extends the human-robot handover approach presented in the previous section, as it generalizes it to multiple humans and multiple robots exchanging an object to achieve a goal. Figure 2.29 illustrates this problem: different agents are distributed into various zones, separated by walls, where counters allow them to exchange objects in order to fulfil the given goal. As the previous section, this one falls in the handover parts concerning the choice of the handover position and the reach out with the arm (Figure 2.4), and it is presented by Waldhart et al. (2015).

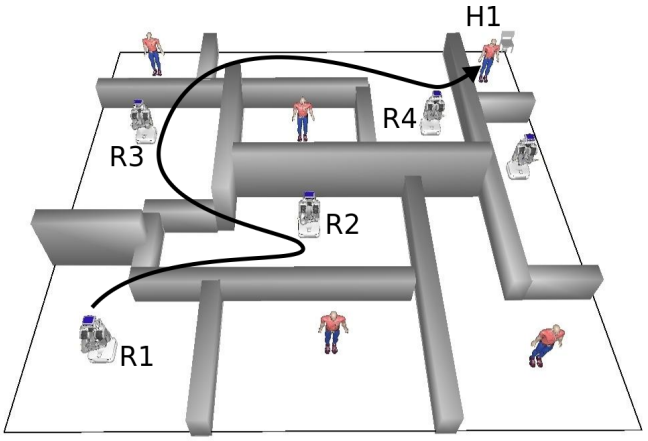
This work was done in the context of the SAPHARI project (`http://www.saphari.eu/`) where one of the use cases is linked to the robot co-worker. For example, in a workshop, one or multiple robots might be asked to help and support the human workers by bringing them the tools and objects they need. To achieve such a task, we developed a kernel algorithm for task allocation taking into account various criteria such as the humans' comfort and preferences, and the agents general availabilities.

The multiple agent handover problem involves computing which agents sequence to use and where handovers should be performed, ensuring the plan is feasible while preserving humans' comfort. Various criteria are taken into account such as the human efforts, the time, the energy and so on. In Figure 2.29(b) even if a handover is possible between initial and target agents (both humans), the algorithm chooses to use a robot to do most of the navigation, in order to reduce their effort.

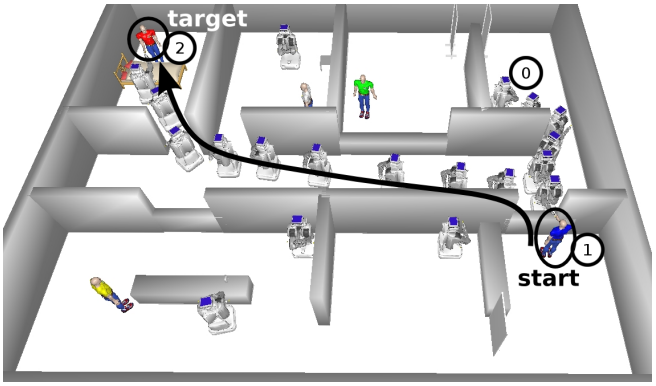
This kind of problem can be solved using a combination of symbolic and geometric planning, Dornhege et al. (2012), Kaelbling and Lozano-Pérez (2011), Karlsson et al. (2012): these approaches will solve the problem, but does not enable to find, efficiently, an optimal solution based on the parameters cited earlier (note that using a task planner alone will be under efficient as the problem is geometrically complex as demonstrated by Lagriffoul et al. (2013)). The problem is tightly linked to the more general pick-up and delivery problem (PDP). Savelsbergh and Sol (1995) present a survey of the PDP with its different types, and solution methods. More recently the link between PDP and handovers has been stressed out by Coltin and Veloso (2014) where they present an algorithm where robots transfer objects to optimize a PDP plan. For Cohen et al. (2014) the problem is to find needed handovers between manipulators arms (no base motion) to bring the object from a position to another one. They find a path for the object and compute for each position of the object on the path, the inverse kinematic of at least one arm which is grasping it and then deduce the trajectory of every arm involved. Their resolution is search-based in a discretized environment, using a lazy variant of weighted A*.

The main contribution of this section is the elaboration of a planner able to solve a multiple agent handover problem by finding an optimal solution based on social rules and humans' comfort. This planner is based on a graph using various models, from geometric computation to more abstract high level reasoning. It has been implemented in simulation and in two PR2 robots Willow Garage (2008).

This section is structured as follows, Subsection 2.4.1 defines the problem and proposes a formalization, and Subsection 2.4.2 presents the algorithms used to find solutions while Subsection 2.4.3 shows the



(a)



(b)

FIGURE 2.29: (a) An example where, to bring the object from R1 to H1, the agents R2, R3 and R4 perform a sequence of handovers. In this example every agent is in a separated zone. (b) An example in an environment where agents are separated into two navigable zones linked by windows and counters. The blue human is in possession of an object needed by the red one. The solution found by the algorithm is: robot 0 navigates to the blue human, takes the object through a handover over the counter, and then navigates to the red human and gives it to him.

different examples used to test the algorithm. Finally, Subsection 2.4.4 shows the results and discusses them.

2.4.1 Problem definition and formalization

The problem tackled here, is to bring an object, held by a starting agent, to a target agent, by making agents carry the object or hand it over to other agents. Note that more than the start and target agent can be involved in the task and the object can be carried by one agent at a time.

The problem inputs are the agent list, the starting and target agents, the initial state, consisting of all agents and objects positions, and agents specific information about speed and availability. The last input is a parameter to balance between the task urgency and care given to the humans' comfort. This parameter is inspired from the *mobility* parameter from the previous section.

A solution to the problem is a scheduled sequence of actions (navigation and handovers), that brings the object from the starting agent to the target agent.

The search space is the full configuration space, as described by LaValle (2006), of the whole problem. As it involves several agents, it can be written as the cross-product of the configuration spaces of each agent: $C = C_0 \times C_1 \times \cdots \times C_n$. We assume the object is sufficiently small to not influence the problem (otherwise, its configuration space should be added to the full one). The problem high dimensionality results in an extremely high computational cost while using classical solutions and algorithms: Figure 2.29(b) shows an example with 5 humans and 5 robots, which results in roughly $card(C) \simeq 300$ degrees of freedom (37 for each human[3] and 22 for each robot[4]) to plan for, which is not suitable for on-line solution search.

2.4.1.1 Global approach and simplifications

The problem is decomposed into two distinct subsets of lower dimensionality: the navigation between the handovers positions and the handovers themselves.

The navigation phase is based on a path finding in a discrete 2D grid built using the input environment, the agents, the objects geometries and the agents initial positions. The grid is computed off-line in order to not affect running-time.

Based on the assumption of a large environment with sparse obstacles and few narrow passages, inter-agent collisions are ignored during this phase. Following this statement, the model considers only one agent at a time for the navigation (Section 2.4.2.3 explains how the system deals with these collisions when they occur).

A Handover involves two agents. The full dimensionality of their models and their positions is needed to ensure the feasibility of the task (for example, to test if a handover through a window or above a counter is feasible or not). As the computational cost of this test is high, fast specific tools are used to prune out candidate solution with no chance of success.

The problem also have a higher level discrete component: finding a sequence of agents to perform successive handovers which guides the search toward relevant handovers, limiting the number of time-consuming evaluations.

In this subsection, we present the models used, and the algorithms involved in the solution search.

2.4.1.2 Representation

The problem is solved as a path finding problem in a graph G. Each node of this graph can be represented as a pair $[a_{id}, \{x, y\}]$ where a_{id} is the id of an agent and $\{x, y\}$ a 2D position: a node represents an agent holding the object at a defined position (all the other agents are considered at their initial position). The edges of G represent one of the two actions:

- *Navigation*: an agent a holding the object can travel to a neighbour position: $[a, \{x, y\}] \rightarrow [a, \{x + \delta_x, y + \delta_y\}]$;

[3] Each human has 2 arms and 2 legs with 7 degree of freedom each (DoFs) in addition to 3 DoFs for the head, 3 Dofs for the torso and 3 navigation DoFs

[4] Each robot has 2 arms with 7 DoFs each, in addition to 2 DoFs for the head, 1 DoF for the torso and 3 Navigation DoFs

- *Handover*: the holding agent a can give the object to another agent b: $[a, \{x, y\}] \rightarrow [b, \{x', y'\}]$. This requires b to move from its initial position to its new one ($\{x', y'\}$), and causes a to go back to its own initial position.

A solution of the problem can be represented as a path in this graph. Figure 2.30 shows an example of this graph, where four agents are involved, and the source agent is A1 and the target one is A4. The agent A1 needs to first navigate to reach the position (0,2) then the agent can handover the object to the agent A2 which is in (5,5). A2 tries and fails to find a solution when going to (5,7), then it goes to (6,5) where it can handover the object to A3 which navigates until reaching a position (13,5) where a handover is possible with A4.

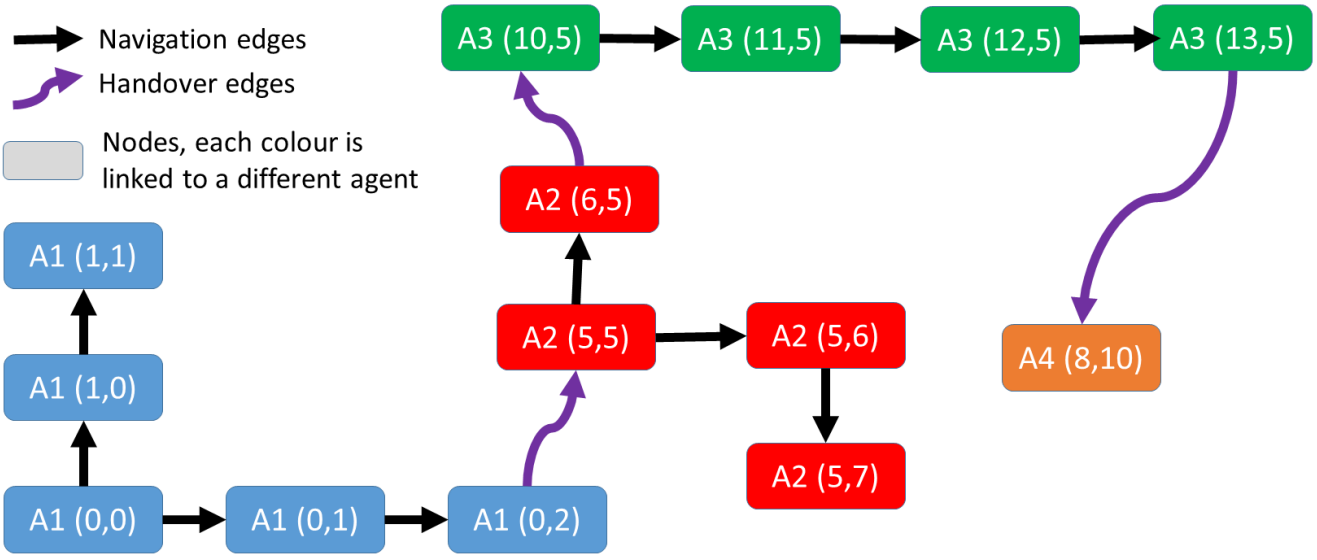


FIGURE 2.30: An example of the graph G.

The edges are weighted with the cost of the actions (computed based on the time to execute and the effort needed). Note that the cost of a handover action is high compared to a navigation action, as it involves two auxiliary navigation tasks along with the handover itself.

The search algorithm relies on other tools, at different steps of the process: high level representation, 2D model for navigation, full geometric representation for handover posture search and check, collision checking and motion planning. These models descriptions are presented here bellow, in a top down order.

High-level representation The problem is represented at the highest level of this model as a graph able to guide the search through all possible handovers. It is referred to, in the rest of this section, as *agent graph* G_A (different from graph G). In this graph, the nodes represent the agents and the edges, the handovers between them. At the initialisation, all the edges are created, with the supposition that any handover is possible, and each edge is weighted with an optimistic estimation of the cost (based on the time needed to perform the handover and the optimal human-related cost expected), independently of the environment. During the search, the costs will be adjusted with the real ones, and if a handover is proven not feasible, the edge is removed. Note that during the search, this graph is used as a heuristic to guide the search, and it does not allow an agent to get the object twice. We chose to not consider

this possibility as the cases where this might be pertinent is when big objects that change the topology of the world are handled, which does not fall into the scope of this work.

2D navigation grid To plan the navigation tasks, the environment is discretized and projected in a two dimensions grid. This grid can be transformed into a navigation graph G_n (a node is a cell and each cell is linked by edges to its neighbours) and used to find agents paths from a position to another one using classical graph search algorithm such as A*. Some nodes in this navigation graph are obstructed with obstacles making them unreachable by the agents, and some of these obstacles surround some areas, disconnecting them from the rest of the navigation graph which creates multiple connected components. This graph is used as the base for the graph G but the various connected components might be linked together using the handover edges.

Geometric environment model Geometric algorithms (e.g. collision checking, inverse-kinematics, motion planning) are used to find valid handover positions and to compute their costs, by taking into account social rules, the humans' comfort, the motions legibility and so on. If the handover is feasible, the computed cost is used to update the agent graph G_A, otherwise the edge corresponding to this handover is removed from G_A. A handover is considered valid if a collision free position where both agents can grasp the object at the same time exists and a motion linking the starting position of both agents to the handover configuration also exists (more details available in Subsection 2.3.1.2). All the process of finding and evaluating a handover will be referred to as the handover search tool.

The cost function defined to evaluate a solution considers three weighted parameters:

The agents involvement duration is the time each agent is involved during the task (let A be all the agents in the environment), weighted with its level of availability ($f_{av}(a \in A)$). It can be expressed as: $Inv = \sum_{a \in A} t_{inv}(a) \cdot f_{av}(a)$.

The global execution time is the total time Tot between the moment when the first agent starts moving until the one when the last agent stops its motion.

The human comfort is related to the human-robot distances, the visibility of the robots by the humans and the postures of the humans during the handovers (more details available in Section 2.3.1.3). These parameters enable us to compute the comfort cost $c_{HO}(i)$ of the agents involved in the i^{th} handover of the plan (a plan with n handovers). The comfort cost can be expressed as: $Confo = F(c_{HO}(0), ..., c_{HO}(n))$ where F is either a Maximum or a Sum function.

A solution cost can be expressed as follows:

$$cost = w_{inv} * Inv + w_{time} * Tot + w_{HRI} * Confo$$

Where w_{mob}, w_{time} and w_{HRI} are the weight of the different parts, and by increasing or decreasing them, the priority can be given either to the global time execution or the humans' comfort or humans' involvement during the task (Those are the parameters replacing the *mobility* parameter from the previous section).

2.4.2 Resolution

The most time-consuming search the planner needs to perform, in order to find a solution in these models, concerns the graph G which represents a simplified form of the problem. The search time is directly related to the connectivity of this graph which is itself related to the number of neighbours a node can have. In the rooms environment (Figure 2.29(b)) this value reaches 3000. This number can be approximated as follows:

$$\frac{1}{d^2} \sum_{p \in N} \left(\pi \cdot R(a,p)^2 - \pi \cdot r(a,p)^2 \right)$$

where the elements are:

- d: the discretization step of the 2D navigation grid.
- a: the agent holding the object in a node of G.
- p: a node of G_A (p represents also the agent this node is linked to).
- N: the neighbours in G_A of the node linked to the agent a.
- $R(X,Y)$: the maximal distance for a handover between agents X and Y
- $r(X,Y)$: the minimal distance for a handover between agents X and Y

The rooms environment has multiple agents in the same zones, making the number of possible actions for each cell in this zone very high, and thus increasing the connectivity of G.

The algorithm implemented in order to perform the search in this graph is a Lazy weighted A* (LWA*) introduced by Cohen et al. (2014) as it is able to postpone the most time consuming searches (the handover feasibility) to the moment these handovers seem relevant. This algorithm has proven bounds of sub-optimality inherited of Weighted A*, Likhachev et al. (2004), and can perform faster when it involves computationally expensive evaluations.

LWA* algorithm is based on A* algorithm, which searches for the shortest path using a heuristic to guide the search. When a node is expanded, three values are given to children nodes: the g value is the distance (cost) between the origin and the child node, the h value is the heuristic, *i.e.* the estimation of the distance remaining to reach the target, and the f value is the sum $g + h$. In the next iteration, the unexpanded node with the smaller f value will be expanded, until the target node is reached.

In the weighted variant, the h value is increased by a factor, f becomes $g + \epsilon.h$ with $\epsilon \geq 1$, thus adding a depth-first flavour to the search, but decreasing the quality of the solution of at most ϵ (the path found is at most ϵ times as long (expensive) as the optimal path).

The lazy variant uses a temporary g value attributed to expanded node children. This temporary cost is optimistic and is faster to compute than the real cost. The real cost is computed only when the node is selected to be expanded, *i.e.* is the one with the smaller f value. Its g and f values are updated and it is put back in the list of nodes to be expanded. Figure 2.31 shows an exploration example were there are unexplored nodes, expanded ones, and a solution.

The choice of the search algorithm is open, any other graph search algorithm would work, but the specificities of LWA* makes it a good candidate for this problem.

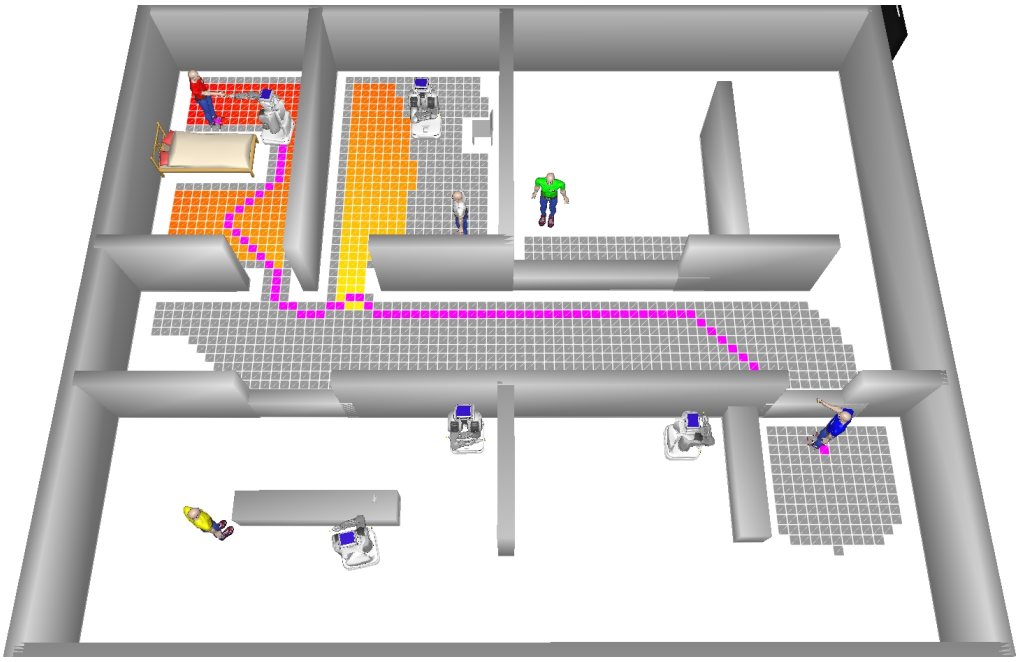


FIGURE 2.31: Pink is the actual object path, the grey is the cells to be extended (from the lazy part of LWA*) and the coloured cells are the explored ones. The two indentations of the pink path are due to the sub-optimality of the weighted variant of A*, it is close to the heuristic

Heuristic and cost The costs in the graph G are evaluated depending on the action they model: a navigation or a handover. The navigation cost is computed from the distance related parameter such as energy consumption and navigation time. The handover cost includes both agents motions, the interaction duration and the navigation cost of the receiver agent, who needs to navigate to the handover position.

The heuristic function (algorithm 2) guides the search in the graph G and through the possible handovers. It is based on the agent graph and the navigation grid: It first searches for an agent sequence that can bring the object to the target agent. This search is made in the agent graph and takes into account minimal handover costs with no navigation (line 2). Then, based on this sequence, it searches the minimal cost related to the navigation. In this model, it is using the cheapest agent for the whole distance (in Euclidean sense, line 4). At this step, it is not known yet if the handover is possible or not, but it guarantees that the heuristic is admissible. It then adds the estimation of handover costs, which must be computed with an admissible heuristic too (line 7).

2.4.2.1 Handover tests

In order to further reduce the computation time, the system can postpone the usage of the handover search tool (highly time consuming) and use simpler process to detect infeasible handover as early as possible:

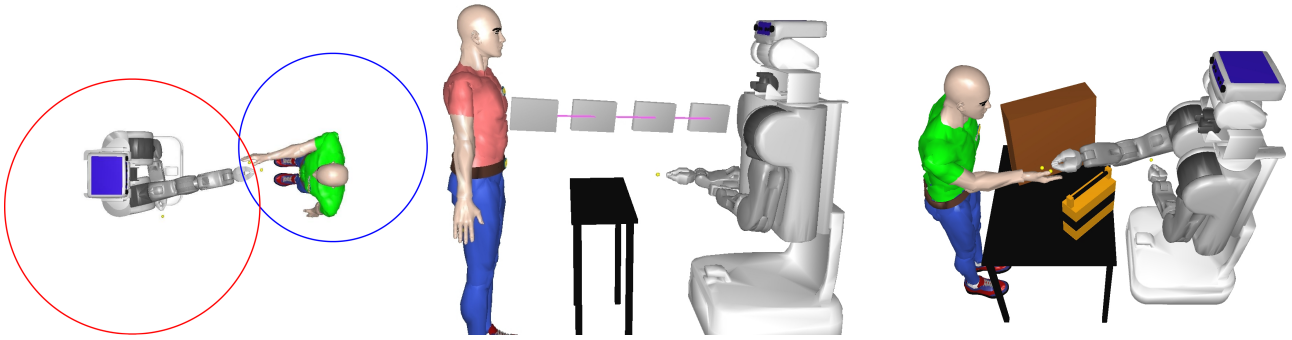
Algorithm 2 Heuristic function for the main search algorithm

```
1: function HEURISTIC(N,N_goal)                          ▷ current and goal nodes
2:     path ← SHORTESTPATH(G_A,N,N_goal)                 ▷ G_A is the agent graph (task level)
3:     for each agent a of path do
4:         d ← min(d, DISTANCECOST(a) )                  ▷ cost for a to go from N to N_goal (Euclidean distance)
5:     end for
6:     for each handover HO in path do
7:         h ← h+ HANDOVERHEURISTIC(HO)                  ▷ estimation of each handover
8:     end for
9:     h ← h + d
10:    return h
11: end function
```

Distances test checking if the agents are within reach of each other (based on arm length) Figure 2.32(a).

Object collision test testing if a path exists for the object to go from an agent to the other one (just the object) Figure 2.32(b) shows a test example where a straight path exists.

Inverse-kinematics test testing if the agents can both reach the object when this one is in between them, Figure 2.32(c).



(a) Distances test (b) Object collision test (c) Inverse-kinematics test

FIGURE 2.32: Examples of the handover tests. (a) Shows a distance test: if the two circles (one for the robot one for the human) intersect, the distance test is validated. (b) Shows a collision test for the grey book: there is a direct path from an agent to the other one. (c) Shows an inverse kinematic test, where the whole body positions of the agents are tested.

The inverse-kinematics test uses the same pre-defined configuration and their costs as in Section 2.3.2.1 and update the edge cost in G. Optionally, the full motion can be computed, but unless the environment is highly cluttered, it is preferable to avoid it, as it is even more time-consuming and in most cases the inverse-kinematics test is enough to ensure the feasibility of the handover (when not computed during the search, the motion plan can be computed later, just before the execution).

2.4.2.2 Schedule

The time information, missing from the path provided by the search algorithm (LWA*), need to be computed after the search, in a post processing step. This process retrieves all the missing information

(such as the receiver navigation to go from initial position to a handover position), adds them to the unordered list of tasks and then schedules all these tasks using

Simple Temporal Networks (STN), Dechter et al. (1991)

The following assertions are applied to build the STN:

- A handover can start only when both agents are at their handover position.
- After a handover, agents can leave only when the handover is finished.

Knowing the duration of each task, we can use the STN to compute all tasks optimal start and end dates for the plan to be executed as fast as possible. Figure 2.33 shows an example of a schedule in the example of Figure 2.36(b) using the STN assertions.

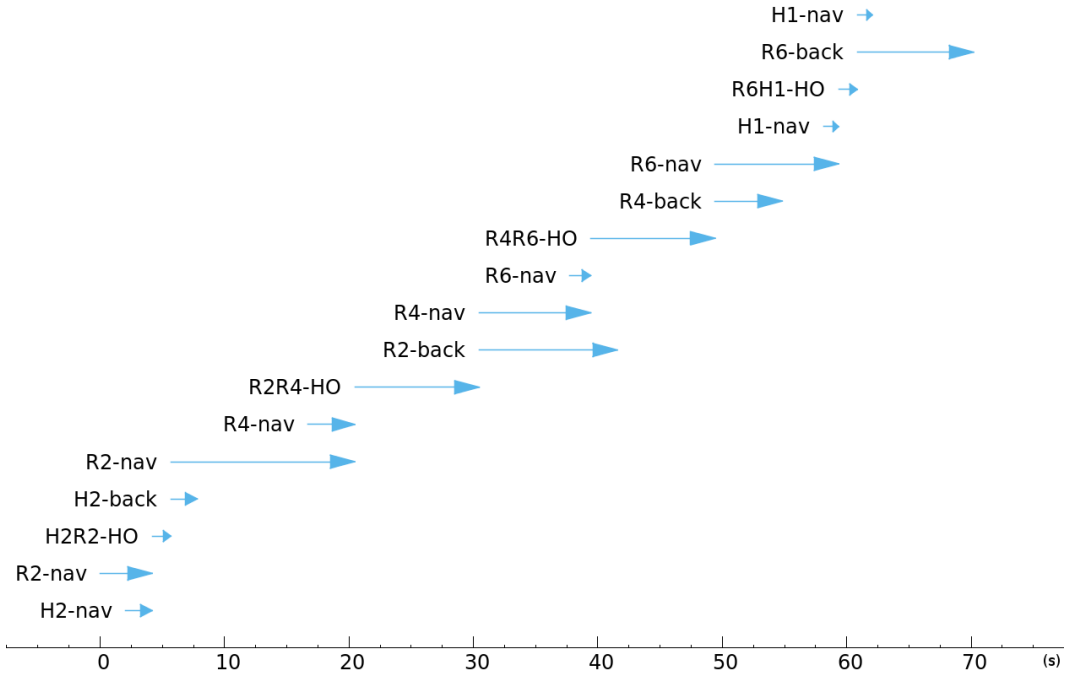


FIGURE 2.33: This is an example of a schedule computed for the example in Figure 2.36(b): nav means navigation, HO means handover and back, returning to initial position. H1 and H2 are humans, and R2, R4 and R6 are robots. We specified a longer duration for a robot-robot handover compared to a human-robot one: this is a reasonable assumption as humans adapt themselves easily, while between robots, a longer time is needed to achieve the mandatory synchronisation.

2.4.2.3 Collision post-process

The final step of the planning process is to guaranty the feasibility of the trajectories according to collisions between agents. Following the hypothesis formulated in section 2.4.1.2, the algorithm did not check for agent-agent collisions during the search. As it can still occur, this step verifies if every agent trajectory is feasible, and in the case of collision, tries to find a solution.

The schedule lets the system know the position of any agent at any time, and when a collision is found, paths are recomputed taking into account all the agents involved in that collision. The agent holding the object is supposed to have priority, safe places are found for the other agents, and the new

solution is scheduled. We iterate until all trajectories are valid. We assumed that the world is large with sparse obstacles, and with this loop, we assume that we will find a collision free path in most of the cases. When these assumptions and this loop are not enough to find a solution, we go back to the main algorithm and continue the search.

Figure 2.34 shows an example solution of this case, where an agent blocks the path to access the target agent, and the plan involves moving this agent out of the trajectory in order to reach the goal.

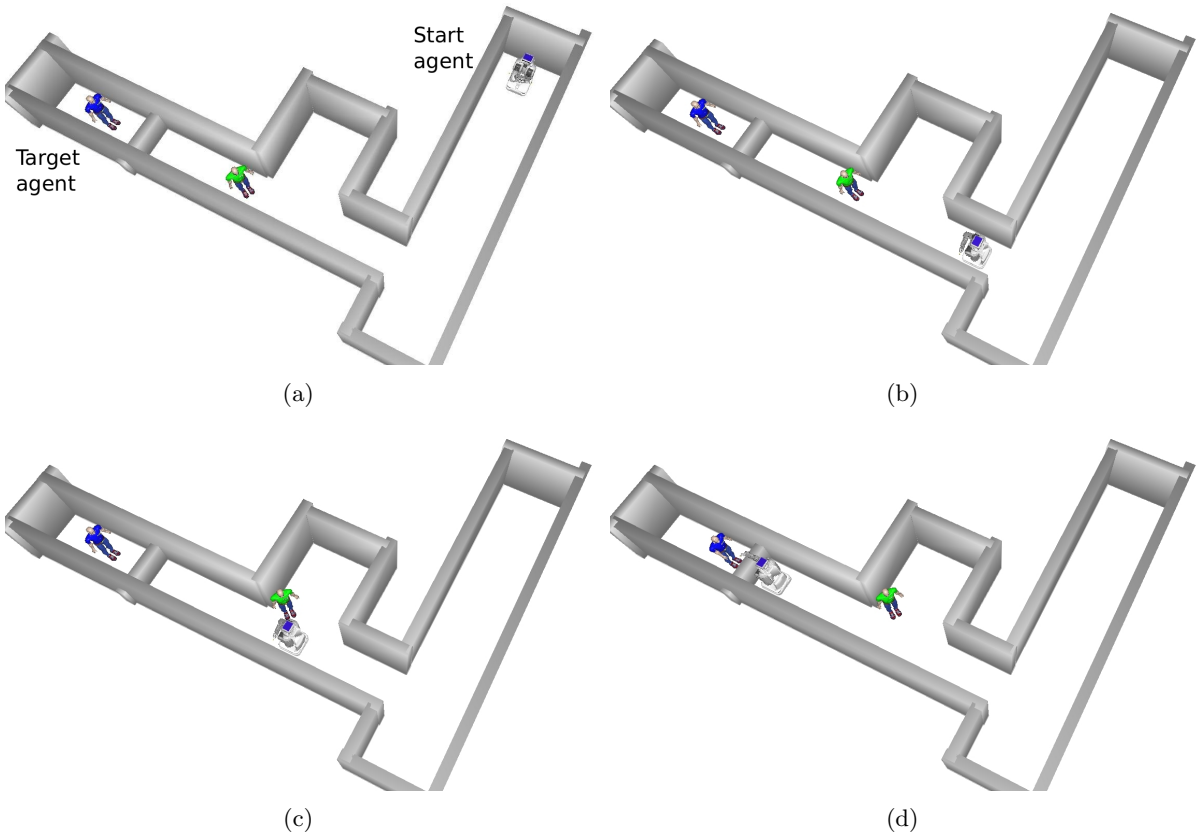


FIGURE 2.34: An example where the robot plan for a third agent (not involved in the handover) to move in order to avoid collision and let him access the target agent

2.4.3 Examples

This section provides results of the planner running on various scenarios with different sizes and agents number, of various complexities and likeliness [5], in simulation and real life (using two PR2 robots) and discuss the scalability and adaptability to other kinds of scenarios (the chosen scenarios concern small or medium buildings with a number of agents inferior to twenty).

2.4.3.1 Example 1 - rooms

The environment presented in Figure 2.29(b) is a scenario with medium complexity ($18 \times 16m^2$, 10 agents). The agents share common areas, where handover are possible almost everywhere, this property

[5]The simple scenarios with only one handover are not presented here, but has been tested with satisfying results (Figure 2.34 shows a solution with a unique handover)

increases the graph G connectivity and complexity. The node number in this example [6] is about 64000 (we use a discretization step for the navigation grid of 0.15cm in all the examples, and for this one, there is an average of five possible agents by cell). In this environment, the A* algorithm is efficient as the heuristic is close to the real cost.

Synthesis: $18 \times 16m^2$, 10 agents, 64000 nodes, efficient heuristic.

2.4.3.2 Example 2 - maze

Figure 2.35 is a maze where there is always a simple solution where the starting and target agents can meet to perform a single handover. But windows allow faster delivery if the object is handed over through them between intermediary agents. The A* heuristic gets trapped in this environment as a solution is rarely close to the straight line. There are 102400 nodes (8 possible agents per cell).

Synthesis: $18 \times 16m^2$, 8 agents, 102400 nodes, inefficient heuristic.

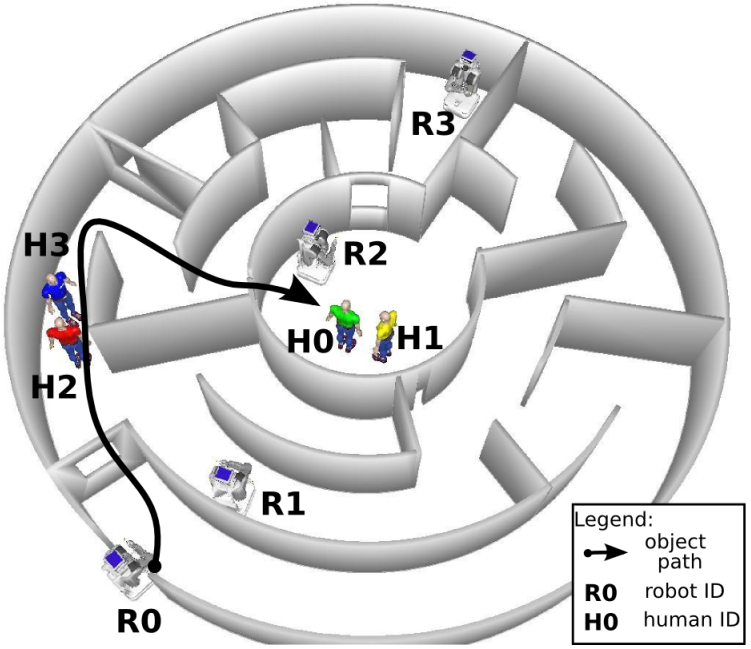


FIGURE 2.35: The maze example, with a possible path for the object shown by the black arrow. In this example, the object is successively held by R0, H2, H3, R3, and H0. The use of multiple agents saves time for the delivery while asking more work to humans. Their use is still limited as most of the navigation is done by robots. Under other settings, the whole navigation could be done by the robot R0, which is slower but does not disturb humans at all. Or the same path for the object could be done while being held mainly by R1, but it would cause the humans H2 and H3 to leave their place for the robot to go across.

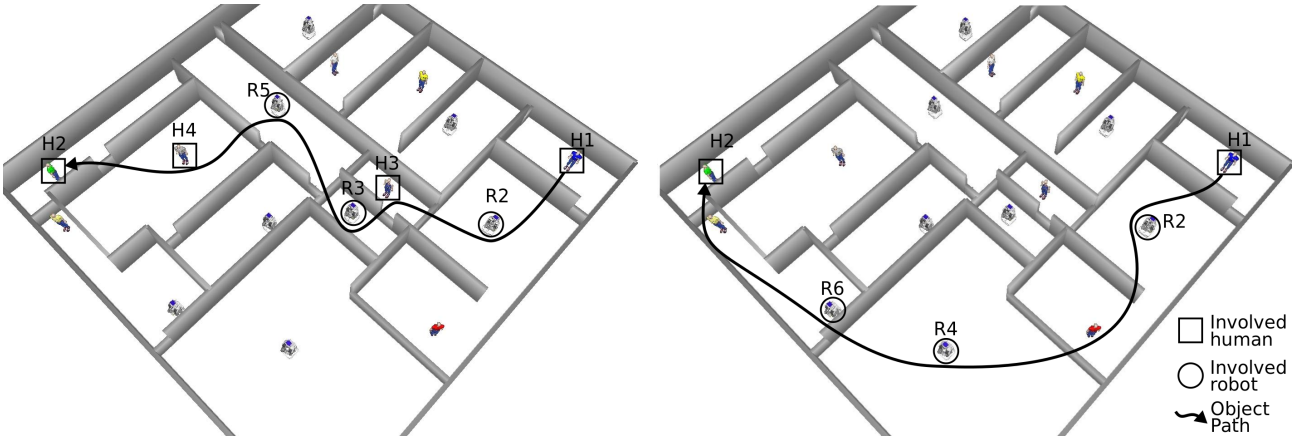
2.4.3.3 Example 3 - big room

The environment in Figure 2.36 is the largest example environment ($25 \times 25m^2$) where the 16 agents are in rooms connected by doors or windows. In this example all (or nearly all) agents are in separated areas

[6] In order to compute the node number we use the formula: $surface/(discr.step)^2 \times NAC$. NAC is the average number of agents able to reach any cell of the environment

(average of 1.5 agent per cell), causing the connectivity of the graph G to drop with the node number (41500 nodes). The A* heuristic does not get trapped as in the maze, but solutions are usually not straight lines.

Synthesis: $25 \times 25m^2$, 16 agents, 41500 nodes, normal efficiency for heuristic.



(a) (b)

FIGURE 2.36: The big rooms example, with the object path of two solutions represented by the black arrow. In 2.36(a) the time is prioritized, while in 2.36(b) the agent comfort is. Note that in 2.36(b) the solution chosen only involve the starting and target human, no other human is involved in the task.

2.4.3.4 Example 4 - apartment

The environment in Figure 2.37 is a small environment ($8 \times 15m^2$) with few agents (4 only, 2 humans and 2 PR2 robots), 16000 nodes and an average of 3 possible agents per cell.

This environment has been tested in simulation but also on the real robots[7]. In our implementation, one robot computes the solution (without the trajectories) and share it with the other robot, then each one of them computes its own trajectories based on the information included in the solution:

- The 2D handover position for each robot ((x, y, θ)).
- The object position at the handover configuration: (x, y, z).
- The 2D handover position to give the object to the next agent (x, y, θ).
- The handover configuration (arm configuration) to give the object.

Synthesis: $8 \times 15m^2$, 4 agents, 16000 nodes, normal efficiency for heuristic.

2.4.4 Results and Discussions

Two main results are interesting to discuss: the computation time and the solutions related to the assigned costs. In the Figures 2.29(b), 2.35, 2.36, 2.37 the default starting and target agent are displayed, unless stated otherwise in the rest of this subsection, the default starting and target agent will be used.

[7]The video https://youtu.be/OWfQGUeBPzw shows multiple solutions in simulation for this scenario and the execution of one of them on the robots.

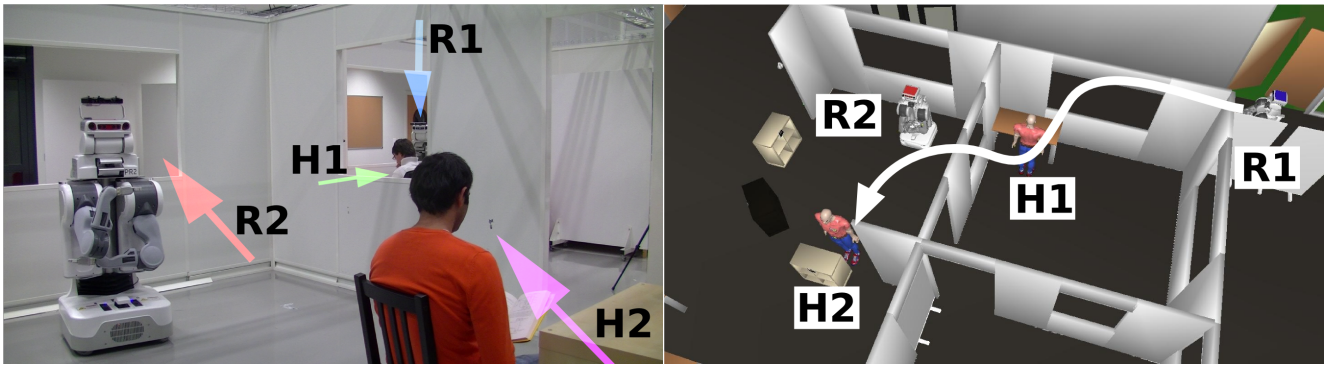


FIGURE 2.37: The real robot example. The right picture is the 3D model, with the object path as a white arrow. The solution involves successively R1, H1, R2, H2, and the object is handed over through the windows, minimizing H1 efforts, and brought directly to H2 so he doesn't have to move.

2.4.4.1 The computation time

The algorithm is able to compute in few seconds interesting solutions for the proposed examples. Table 2.1 shows the mean computation time of a solution in the four examples were the starting and target agent were randomized and two different cost priorities were used (prioritizing robots use, the human should participate the less possible in the action, or prioritizing the execution time, the task should be done as fast as possible). As expected, the computation time decreases when ϵ increases.

The lazy variant reduce dramatically a number of real expansions. Averaged on 80 random runs in the rooms environment (Figure 2.29(b)), the true cost is computed only for 0.46% of the explored nodes. The ratio of computation time for one true cost evaluation versus the one for the temporary cost estimation is almost of 200, with $67ms$ for the former and $0.334ms$ for the latter. Statistically, without lazy variant, evaluating explored nodes would be more than 100 times longer. That factor reaches 400 in the rooms environments, while still providing exactly the same solutions. This enforces the relevance of the lazy A* variant use in our case. Figure 2.31 shows an example of a resolution in the rooms environment, where most of the cells are explored but the true cost is computed just for a small part of them.

ϵ	10	4	1
mean time (s)	4.675	5.25	20.8

TABLE 2.1: The mean times computed over 160 run, in the four examples, with random starting and target agents, using two different cost priority (agent and time)

The connectivity of the environment does play a role in the computation time, but the most relevant factor (as in any A* search) is the accuracy of the heuristic: as depicted in Table 2.2, even though the connectivity of the rooms environment is very high, the heuristic is efficient; hence, the computation time is small.

	rooms	maze	big rooms	apartment
mean time (s)($\epsilon = 1$)	11.2	17.9	40.7	13.4

TABLE 2.2: The mean computation time for each example for $\epsilon = 1$

2.4.4.2 Solution quality

Table 2.3 presents the values of the main cost components (the execution time, the number of involved agents and the number of involved humans) for some algorithm solutions, running on Section 2.4.3 examples, with $\epsilon = 1$. The results show that when the priority is set to agent, no human is involved in the task (whatever the number of involved agents) but this results on a loss of efficiency as shown by the execution times: when the priority is set to time, even though humans are involved in the task, the execution time is faster than the agent priority execution time (up to 2 times faster).

Priority	Agent		Time	
Environment	time (s)	nb humans / nb agents	time (s)	nb humans / nb agents
Rooms	28.7	0 / 1	22.5	2 / 2
Maze	32.1	0 / 0	15.1	2 / 2
Big rooms	63.9	0 / 3	44.6	2 / 4
Apartment	17.6	0 / 0	8.4	1 / 1

TABLE 2.3: This table contains the main components used to compute the cost of a solution: the complete execution time and the number of humans (resp. agents) involved in the task, excluding the starting and target agents.

Adaptability This approach can find a solution for any kind of scenario, but the computation time will grow exponentially with the environment size and the agents number. Even though, if the connectivity of the graph G is low or the heuristic efficient, the approach would perform better. In larger and more complex scenarios (than those presented in this section) the algorithm will still find a solution, but the computational time would not allow on-line use for such situations. Though, such complex cases are supposed to be rare and do not enter under the scope of this work.

2.5 Where to look when handing over the object

In this section we present a contribution related to the communication about the handover part, coloured in dark in Figure 2.4, and presented by Gharbi et al. (2015). This work was done in collaboration with Pierre-Vincent Paubel, Aurélie Clodic, Ophélie Carreras and Jean-Marie Cellier.

The handover is a joint action involving physical interaction. As explained in the beginning of this chapter, a joint action, to be natural and efficient, needs to rely on a number of signals enabling a better communication between the involved agents. It is especially true when one of the agents is a humanoid robot that people tend to anthropomorphize and expect a hand-eye coordination making it uncomfortable when missing.

In order to achieve a better Human-robot interaction, the robot needs to take this behaviour into account and compute its head/eyes motion accordingly. In this section, the movements or pattern of movements that a giver and a receiver should perform during a handover to make it as understandable as possible are investigated.

A focus is made on the way a robot should use its gaze cues (in our case head movement) to support a joint action involving robot arm movements. It is based on multidisciplinary study, involving psychologists and roboticists, where we chose to study a simple task: a user (the giver) sets an object down on a table in the direction of another user (the receiver). This study explores two cases for the giver: a human and a robot (Figure 2.38).

The chosen movement to achieve the task, was the simplest possible (straight lines) in order to concentrate the study on non-verbal cues that should be produced by the giver to enhance receiver perception about movement understandability.

FIGURE 2.38: Snapshots of the videos used to evaluate the movement naturalness.

A brief subjective analysis of this kind of gesture between two humans shows the significance of two variables to judge this feature: the **gesture speed** and the **giver's gaze direction**.

The task is not a handover but a set down, but we believe that in this setting, the communication cues are the same for both actions. The aim of this study is to find which coordination smoothers (Becchio et al. (2010), Vesper et al. (2010)) should be added to the robot motion to be more understandable to the human it interacts with. Gaze cues usage have already been studied in HRI context Boucher et al. (2012), Moon et al. (2014), Mutlu (2009), Staudte and Crocker (2009), Strabala et al. (2012a) and this work should be considered as complementary to these studies. The purpose is to confirm, in a controlled

experimental context, results that could have been already exhibited, and the results tend to do so. In addition, what could be credited to this work is the study of gaze behaviour for **both** the giver (Human or Robot) and the receiver during a give action. The closest studies in the field to our knowledge are Boucher et al. (2012) and Moon et al. (2014). In the first one, the authors do a similar study over a different action, while the second does not study the receiver gaze.

The user study presented in this work tries to confirm these hypotheses:

The giver gaze cues are important In order to achieve an understandable and efficient handover, the giver gaze cues should not only change (a static gaze is not good) but follows a specific pattern.

The receiver gaze cues should not change when the giver changes When changing the giver (human or robot), the receiver gaze cues should be similar.

The gesture speed is important A conventional speed should be preferred over a slower or a faster one.

Related work

Gaze analysis allows the receiver to make hypothesis on the cognitive activity handled by the giver, and a number of researchers tried to codify and implement these cues on robots.

Mutlu (2009) studies gaze cues communication on several robotics platforms, and showed it importance in HRI and how a well-defined gaze patterns can enhance human-robot communication experience.

Boucher et al. (2012) observe that one of the current roadblocks in the elaboration of smooth and natural human-robot cooperation is the coordination of robot gaze with the ongoing interaction and tried to identify pertinent gaze cues in human-robot cooperation. When the gaze cues are well defined, the cooperating human can reliably exploit it and anticipate actions in the cooperative task.

Interestingly, in a study oriented toward gaze cues in human-human interaction, Furlanetto et al. (2013) show that eliminating gaze cues by blurring the actor's face did not reduce perspective-taking, suggesting that in the absence of gaze information, observers rely entirely on the action. Intriguingly, perspective-taking was higher when gaze and action did not signal the same intention, suggesting that in presence of ambiguous behavioural intention, people are more likely to take the other's perspective to try to understand the action.

Moon et al. (2014) exploited human-like gaze cues during human-robot handovers and found that the subjects' reaction time is faster with the appropriate cue (looking toward the handover position) but also that those subjects judge the handover more natural when accompanied with this cue.

These researches show the importance of gaze during human-robot interaction, the robot would be able to achieve the task without the gaze, but the cooperation would suffer from it. In order to have a better grasp on the importance of the gaze, let's situate this work in the global frame of joint action.

Vesper et al. (2010) established that a minimal architecture for joint action should be able to handle, next to a goal, tasks representation (possibly shared), monitoring and prediction processes and what they call coordination smoothers. They argued that "where joint action requires precise coordination

in time or space, there are often limits on how well X's actions can be predicted. One way to facilitate coordination is for an agent to modify its own behaviour in such a way as to make it easier for others to predict upcoming actions."

We suggest that gaze cues could hold the role of coordination smoother in helping the human in front of the robot to better understand robot behaviour and help the robot to achieve its movement in a more natural way.

2.5.1 Materials and methods

In order to confirm or deny the hypothesis, a user study has been designed where volunteers watched videos of a human or a robot (Figure 2.38) setting down an object in front of them, and were asked to judge the naturalness of the task. Different gaze cues were available in the videos. This subsection presents the participant and experimental set-up used to evaluate the task.

2.5.1.1 Participants

Thirty three volunteers participated in the experiment (age range 22–38 with mean value of 27 and a standard deviation of 3.5; 21 males, 12 females), among them, fifteen watched human videos and eighteen watched robot videos. All participants had normal uncorrected vision but two volunteers had to be excluded from subsequent analyses due to a technical problem that damaged eye-tracking data (unreliable calibration)

2.5.1.2 Experimental Set-up

The experimental situation implies watching a video where a giver (Human or Robot), seated behind a table, takes the object with his right hand, and put it on the table so that the receiver, behind the video camera, can reach it. The choice of using videos instead of a real interaction is supported by the need of isolating gaze cues and motions velocity to find some hints about the use of these factors in handovers. Moreover, it has been proposed by Kiesler et al. (2008) and Woods et al. (2006) that video-based scenario can enable us to infer such valuable results

The experiment took place in a room where temperature and luminosity (19 lux) were kept constant and the participants faced a computer screen where the video was presented. Eye movements were recorded using an EyeLink 1000 remote eye tracker (SR Research Ltd., Mississauga, Ontario, Canada) which possesses a spatial accuracy greater than 0.5° and a 0.01° spatial resolution with a sampling rate of 1000 Hz. The camera was placed at a distance of 20 cm from the screen (DELL 19″, refresh rate of 75 Hz, resolution of 1024x768 pixels) and the eye-camera distance was 60 cm maintained by a forehead rest. All eye tracking data were extracted using the SR Research default centroid algorithm.

In the experiment, we manipulated 3 variables: (1) the type of giver (Human or Robot), (2) the speed of the movement (normal, rapid and slow) and (3) the gaze behaviour. The Robot was a PR2 and the Human was a white man (65 years old). The videos were shot to be as similar as possible (see Figure 2.38) and were accelerated and decelerated to obtain different speeds while keeping the

same motion. The duration of the slow (3750 ms), normal (2250 ms), and rapid videos (1750 ms) were identical for the human and for the robot. Six different sequences were created for the gaze behaviour, their timelines are displayed figure 2.39, and they will be referred to later as patterns or conditions:

- **O** the giver looks only at the **O**bject, while the object is moving, the giver looks at the centre of the object
- **R** the giver looks only at the **R**eceiver
- **RO** the giver looks at the **R**eceiver at the beginning, then when its arm (and the object in it) is moving it starts to look at the **O**bject and stay on it until the interaction's end
- **ORO** the giver looks at the **O**bject at the beginning, then when its arm is moving it starts to look at the **R**eceiver until the end of the motion. When the motion is over it looks at the **O**bject again until the interaction's end
- **OR** the giver looks at the **O**bject at the beginning, then when its arm is moving it starts to look at the **R**eceiver and keep this position until the interaction's end
- **ROR** the giver looks at the **R**eceiver at the beginning, then when its arm is moving (and the object in it) it starts to look at the **O**bject until the end of the motion. When it is over, it looks at the **R**eceiver again until the interaction's end.

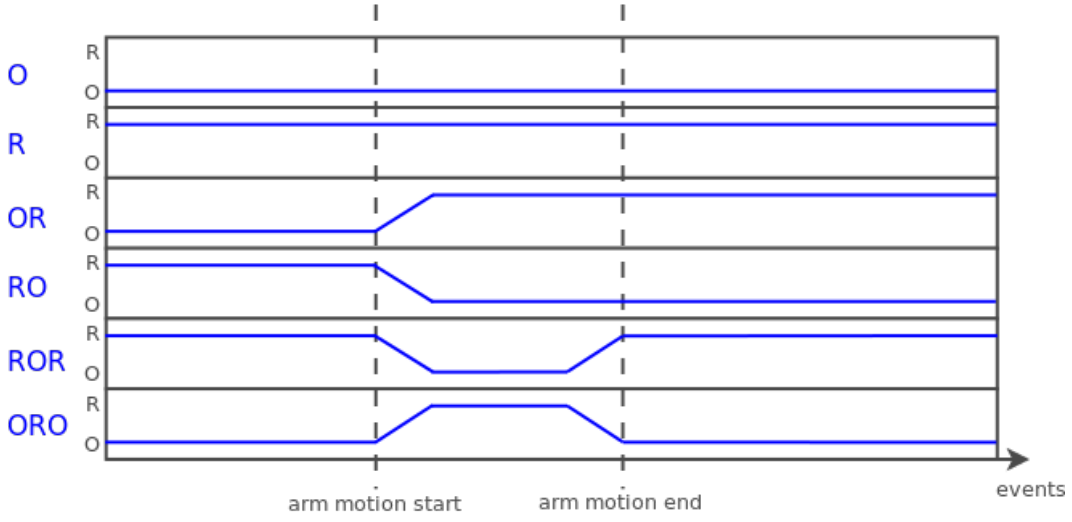


FIGURE 2.39: the timing of the different gaze behaviours according to were the givers looks.

The human moved only his eyes whereas the robot (PR2) moved its head to simulate a gaze. Imai et al. (2002) established that the perception of the robot gaze is coupled to the robot head orientation when this one doesn't have orientable pupils. Examples of those videos are available at https://www.youtube.com/playlist?list=PLJeAfn0C8Ci1XdR-_tCZi2ultZgE4FS7E

2.5.1.3 Procedure

Participants were familiarized with the situation by watching a first video, a bit different from the experimental ones. The experimental session started, and was composed of 18 trials: 3 (speed) x 6

(gaze), presented in a randomized order. In a trial, the participant pressed a button to begin the video and immediately after the video is finished, he/she was asked to rate the perceived naturalness of the movement on a 5 points Likert scale (5 for "perfectly natural", 1 for "not natural at all") presented on the screen. Between each trial, participants had to complete a digital logical suite, to break the dullness of the task. The session (18 trials) was repeated one time making each participant watching and judging 36 videos, in the rest of the section we will refer to each session as a video block.

Before the session started, the participants were told that their objective is to rate the naturalness of the videos they are going to watch. For methodological reason, the instructions and questions were very neutral (even though in the participant mother tongue –French–) in order not to influence the judgement of the subject. The judgement method is the same as one of the three that have been used by Moon et al. (2014), whereas Boucher et al. (2012) do not look for subjective evaluation.

2.5.1.4 Occulometric measurement

Classical dependent variables in eye-tracking studies include the number and duration of fixations on areas of interest. In this study, the areas of interest (AOIs) were (1) the giver face and (2) the object. Those AOIs were fix, as depicted in figure 2.40. As video duration changed between experimental conditions, we computed the percentage of dwell time spent on AOIs to study the distribution of the visual attention.

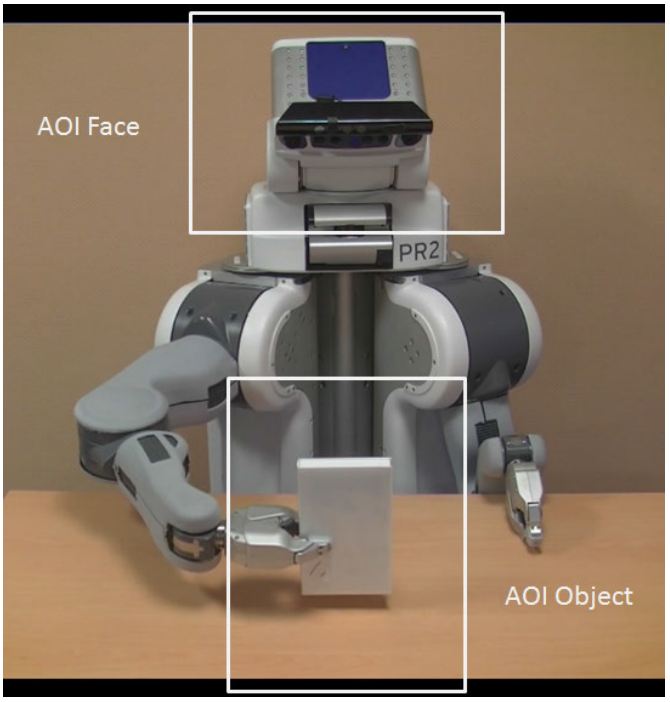


FIGURE 2.40: The AOIs used in the occulometric measurement

2.5.2 Results

We performed a mixed-design analysis of variance to examine the effects of (1) the gaze behaviour, (2) the speed movement, and (3) the type of giver on our subjective and oculometric dependent variables.

Subjective and eye tracking data have been analysed with the software package Statistica 8.0 (Statsoft, Tulsa, Ok, USA).

2.5.2.1 Subjective measurements

Gaze Behaviour

Results indicated a main effect of the gaze behaviour on the naturalness ratings, $F(5, 145)=15.034$, $p<.001$[8] (Figure 2.41). Post-hoc paired comparisons showed that OR and ROR gaze behaviour are significantly judged more natural than the four other conditions R, O, RO, ORO (highest p-value in the post hoc table equal to .003). No difference was found between (1) the two conditions OR and ROR ($p=.70$) and (2) the three conditions R, O and RO (lowest p-value equal to 0.48). Finally, the condition ORO is significantly judged more natural than the two gazes behaviour R and O (highest p-value equal to .03).

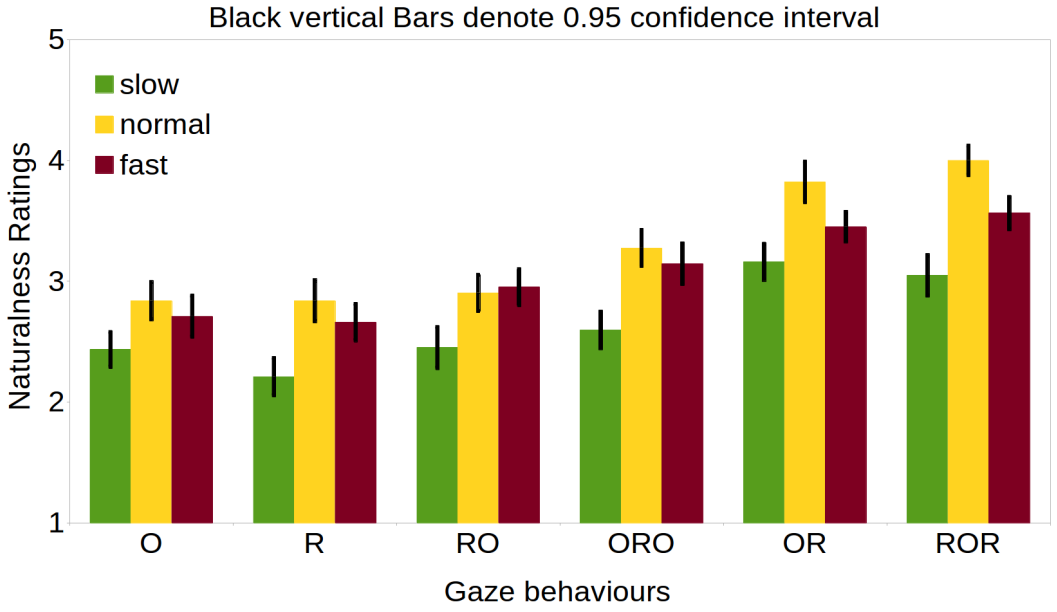


FIGURE 2.41: Naturalness ratings as a function of the gaze behaviour and the speed movement

Movement speed

Results indicated a main effect of the movement speed on the naturalness ratings, $F(2, 58)=10.354$, $p<.001$. Fisher's LSD post-hoc comparisons showed that the normal and the fast motion conditions are judged more natural than the slow one (highest p-value in the post hoc table equal to .004). No significant difference was found between the normal and the fast motion speed conditions ($p=.16$).

Type of giver (Human vs. Robot)

Interestingly, there was no significant difference in the results between the two types of givers ($F(1, 29)=1.988$, $p=.16$). Moreover, no interaction was found between the three main manipulated factors.

[8] F is the Fisher variable and combined with the p-value enable to establish the significance of a difference between two variables: the difference is significant when $p < 0.05$ and is not otherwise.

This result suggests that the effects of movement speed and gaze behaviour described above are not influenced by the type of giver.

2.5.2.2 Eye tracking measurements

Two main interesting behaviours has been noticed thanks to the eye tracker: a difference in the pupil size between human and robot giver, and a different distribution of the attention between them.

Distribution of the visual attention between the face of the giver and the object

Overall, results indicated a significant difference between the mean percentage of dwell time spent on the face of the giver and the mean percentage of dwell time spent on the object ($F(1, 29)=59.848$, $p<.001$): the participants tended to focus mainly their visual attention on the face of the giver (the dwell time spent on something means the time during which the eye focused solely on this something, and is obtained thanks to the eye tracker).

However, the detailed analysis displayed in Figure 2.42 shows an interesting correlation between the type of giver and the gaze behaviour:

Human giver: There is no effect of the giver gaze behaviour on the mean percentage of dwell time spent on the face of the giver ($F(5, 65)=0.807$, $p=.54$), nor on the mean percentage of dwell time spent on the object ($F(5, 65)=1.004$, $p=.42$).

Robot giver: Results indicated a main effect of the gaze behaviour on the mean percentage of dwell time spent on the face of the giver ($F(5, 80)=12{,}82$, $p=.001$), and on the mean percentage of dwell time spent on the object ($F(5, 80)=6.264$, $p=.001$).

When the giver is a human, the main conclusion is that participants focus mainly their visual attention on the face of the giver to provide a judgement concerning the naturalness of the task, independently of the giver gaze behaviour.

When the giver is a robot, Fishers LSD posthoc comparisons shows that participants focus more on the face of the robot for the three types of gaze behaviour ORO, OR and ROR than for the three other conditions R, O, RO (highest p-value in the post hoc table equal to .04). On the other hand, participants focus less on the object for the same three types of gaze behaviour ORO, OR and ROR than for the two other conditions R, O (highest p-value in the post hoc table equal to .05). However, gaze behaviours OR and ROR are not significantly different from the condition RO (lowest p-value equal to 0.26).

If we take the human giver as a reference, the occulometric pattern with a robot giver is identical to the one of a human giver only for the ORO, OR and ROR conditions.

Finally, no effect of the movement speed was found ($F(2, 58)=1.798$, $p=.43$), and the reader may find useful to know that there was no difference between the two blocks of video presentation ($F(1, 29)=0.947$, $p=.33$).

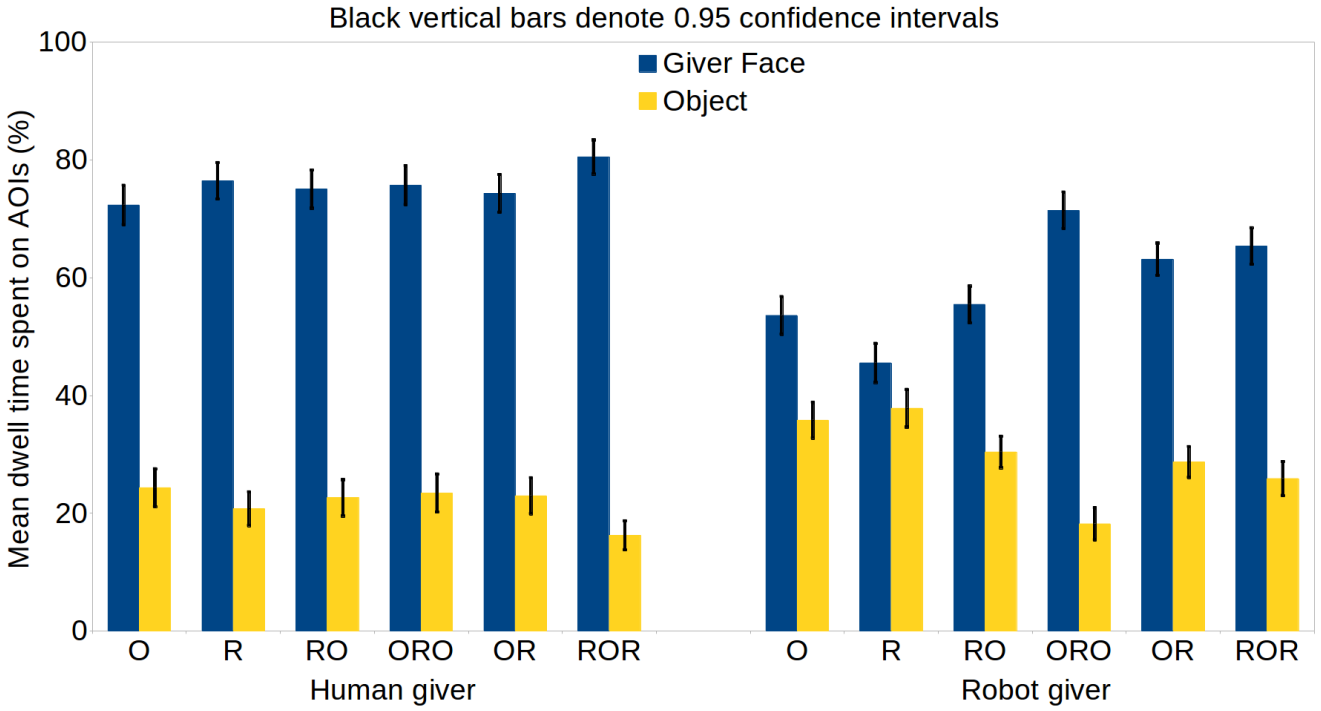


FIGURE 2.42: Distribution of the visual attention between AOIs as a function of the gaze behaviour and the type of giver

Pupil size

Results indicated a strong main effect of the type of giver on pupil size variations, with larger pupil diameters when the giver is a robot ($F(1, 29)=12.803$, $p<.001$).

Results also revealed a main effect of the gaze behaviour ($F(5, 145)=3.050$, $p<.001$), however, post-hoc paired comparisons showed only one significant difference, with smaller pupil size in the OR gaze behaviour condition (highest p-value equal to .03 in the post hoc table).

No effect of the motion speed was found on the pupil size ($F(2, 58)=1.798$, $p=.17$).

Finally, results indicated a significant difference between the two blocks of video presentation ($F(1, 29)=26.155$, $p<.001$), with smaller pupil size during the last block, what could be reasonably considered as a training effect.

2.5.3 Discussion

The first hypothesis (the giver gaze cues are important) is confirmed by the results: subjective measurement clearly shows the subjects didn't like videos with static giver gaze/head. Moreover, the head patterns are not to be neglected: the head final pattern **OR**[9] seems to be preferred over the rest of the patterns, the subjective measurement shows that the subjects preferred the patterns **OR**[9] and **ROR**[9] over the others. This final head pattern could be an acknowledgement/turn-taking signal from the giver to the receiver.

Note that the variable "type of giver (human or robot)" does not affect the subjective naturalness rating.

[9]refers to the type of patterns explained in Section 2.5.1.2

Concerning the distribution of visual attention between the giver face and the object, results are different according to the type of giver. When the giver is a human, there is no effect of the giver gaze behaviour on the distribution of the visual attention. The receiver focuses mainly its visual attention on the face of the human. We believe this behaviour is normal for humans as the face is the most expressive part of the body and humans are used to focus on the face to determine a number of features.

When the giver is a robot, we can distinguish two cases:

- **O**[9], **R**[9] and **RO**[9] cases: The receiver visual attention is shared between the face and the object. That means, receiver will not focus either on the face or on the object but may go from one to the other. We interpret this as the receiver being lost in this kind of situations. Further analysis of eye tracking data is needed to validate this interpretation.
- **ORO**[9], **OR**[9] and **ROR**[9] cases: The visual attention is mainly focused on the head of the robot. In those cases, we found the same pattern of visual attention as in the human giver.

Taken together, the results on the perceived naturalness of the movement and the ones on the oculometric pattern of the receiver seem to put forward two main conditions: **OR** and **ROR**. Those two conditions are not only perceived as more natural than the others (with a robot or a human giver) but they present a similar occulometric pattern of the receiver (with a human or a robot giver). It seems that the final **OR** is an important pattern. When the giver, at the end of the movement, moves the gaze from the object to the receiver, it may mark the end of the exchange. The fact that the receiver looks mainly at the face also in the **ORO** condition may be interpreted in the same sense: when the robot ends its movement on the object, the receiver seeks an acknowledgement on the robot head (our first look at more detailed eye-tracking results seems to corroborate this thought). These results partially corroborate the second hypothesis: the receiver gaze are similar between the robot and the human giver only when the perceived naturalness is high.

This study is more about the movement itself rather than its initiation, however, the preferred patterns meet the ones found by Strabala et al. (2012a). That is at the beginning of the action, the robot is looking at the object or at the receiver. We have also shown that the gaze pattern at the end of the exchange seems also to be important. Some patterns are considered as more natural than others, whereas Moon et al. (2014) did not find any difference on that aspect which was confirmed by objective measurements. These patterns tend to confirm the first intuition and findings about handover conditions: Moon et al. (2014), Strabala et al. (2012a) or Boucher et al. (2012) (a cooperative task) stated that a human exploit the gaze of the robot when it is present.

The difference in the pupil size between the two types of givers (human or robot) might have different explanations: more curiosity or cognitive load induced by the observation of non-familiar, unknown machine. In the general eye-tracking literature, pupil diameters have been found to increase along with cognitive demands Kahneman and Beatty (1966) and emotional load Bradley et al. (2008). In this context, the difference in the pupil size between the two types of givers (human or robot) might have different explanations: more curiosity or cognitive load induced by the observation of non-familiar, unknown machine.

The objective results didn't show any difference between the different speeds, although the subjective measurements show that the normal and fast speed is preferred over the slow one. Again, this partially confirms the third hypothesis.

2.6 Future work

Several contributions have been presented in this chapter. We see potential improvements and perspectives (they are categorized following the section they belong to):

Extending to multiple actions (Section 2.3) For now sharing the effort with the human is only about handing over objects. The approach can be extended to different actions such as taking a picture, talking.

Planning with the object (Section 2.3) It can also be more accurate if the real object form is considered for both manipulation and navigation (navigating with a big object will cause different problems).

Real time adaptation (Section 2.3) The fast convergence times of the results for a handover indicates that the algorithm can be used to dynamically adapt the solution to the human while he is moving.

Relaxed synchronisation constraint (Section 2.4) using a place then a pick (by different agents) sequence instead of a handover.

Agents involvement (Section 2.4) The planner, using STN knows the involvement duration of each agent, the rest of the time, those agents can be used to perform other tasks, but in order to do that, the task planner using the algorithms presented here must explicitly take the time into account Dvorak et al. (2014).

2D grid discretization (Section 2.4) This grid can be replaced by another kind of grid such as a quad-tree structure, Finkel and Bentley (1974), to reduce the number of nodes.

Further analyse the results (Section 2.5) More information are still available on the eye tracker data, and they need to be retrieved and analysed.

Integrating the results on the robot (Section 2.5) The pattern found can be implemented as the normal behaviour of the robot while doing a handover.

A real robot user study (Section 2.5) In order to compare the results with the first one and to ensure their validity.

2.7 Contribution to the human-robot handover in a nutshell

In this chapter three main contributions were presented to the human-robot handover community:

Sharing the effort with the human for a handover An algorithm computing handovers configurations for both the giver and the receiver (humans or robots) while taking into account the human comfort and preferences. We also presented a user-study concerning the sharing part where we proved that a *mobility* parameter (either the human wants or not to share the effort with robot) is relevant in the context.

Multi-agent handover An algorithm that computes an optimal sequence of handover to bring an object from an agent (human or robot) source to an agent target. This algorithm is able to compute, in addition to every motion plan, the exact schedule of every agent involved in the task.

The handover gaze cues A user study where we have shown the importance of gaze cues during a handover, and have shown the importance of the used pattern: the subjects preferred the two patterns **OR** (the giver looks at the **O**bject then at the **R**eceiver) and **ROR** (the giver looks at the **R**eceiver then at the **O**bject and then at the **R**eceiver again)

Chapter 3

Geometric reasoning and planning

Contents

3.1 Introduction

The goal of this chapter is to create a link and fill the gap between the task planning and the motion planning. In the context of autonomous robots, task planning is used to take decisions about what action to perform and when to perform it (as shown in the architecture presented by Alami et al. (1998)). Usually, task planning manipulates symbols and concepts, and tries to find symbolic plans able to achieve a given goal. In the same context, motion planning is used to compute the robot trajectories executable in the real world, trajectories that enable it to achieve tasks. As explained in LaValle (2006), motion planning is based on a geometric model of the world, and needs a full description of the initial and final position of the robot model. This full description is usually expressed with the numerical values related to the position of every part of the robot in the model.

To synthesize, task planning deals with symbols while motion planning requires specific numerical values to compute the trajectories. The gap between these two planners is the problem we are trying to solve in the context of manipulation and navigation planning, using fetch and carry examples in the presence and in interaction with humans. Let's refer to this problem as the Geometric Reasoning and Planning (GRP) problem.

The main goal of the GRP is to compute actions: Based on symbols, the GRP should compute trajectories that will achieve the goals specified with the symbols. In other words, it should be possible to plan for actions while specifying only the desired information: the desired property to achieve at a level of abstraction sufficiently high to be usable by the task planner. For example, giving an object to this person or putting an additional object on the table. This is even more important when other (human oriented) constraints have to be taken into account. The GRP can have another usage which is to compute Facts, based on the geometric model of the world, it is able to compute symbols describing the actual world state. We call these symbols facts. For example, it should be able to compute facts such as an object is in another one, an object is on another one or more human related ones such as an object is reachable by an agent. These links between agents and objects are called affordances.

The affordances were first introduced by Gibson (1977) to explain how agents directly perceive the inherent "values" and "meanings" of things, and how they can use this information to infer the possible actions offered by the environment. Sahin et al. (2007) propose various formalizations of these affordances in the domain of autonomous robotics. One of these formalizations, which will be used in this chapter, is: "Affordances, are relations between the abilities of organisms and features of the environment". In order to compute these affordances we base ourselves on previous work such as Marin-Urias et al. (2008) where they use perspective-taking to compute them.

A geometric reasoner and planner endow the robot with a number of abilities (such as pick, put, show ...). This is very close to the "task-level" planning problem, as defined in Lozano-Perez et al. (1987), as it extends it to more possible actions and includes the multiple agents (humans or robots) possibility. It is also close to the manipulation planning problem which was a focus on various work, such as Simeon (2004), but lately more and more researchers began to work on the GRP problem. Fedrizzi et al. (2009), for example, worked on finding a placement for the robot base, where it is able to grasp an object with an uncertain position. Cosgun et al. (2011) plan for placing an object on a cluttered table

by pushing the objects already on the table. Fraisse and Simeon (2012) developed a framework based on the "mightabilities maps" which are maps in the 3d model of the world where affordances for every agent are computed for each cell in these maps. Based on this, the framework computes where an object can be placed, and where it will be visible and reachable by a specific agent.

In this chapter, we define a framework for specifying actions in a sufficiently formal and flexible manner enabling us to plan and compute their different steps but also to build geometric plans: a sequence of actions where there is interferences and interdependences between the several actions and steps that compose this plan. For example, pick then place: the choices made during the pick might interfere with the place or navigate to an object then pick it up: the position of the robot needs to be close enough to pick the object, but far enough to enable the robot motions (no collisions)

This chapter is divided into 4 sections: the first one, Section 3.2, introduces the problem formalisation whereas, Section 3.3, presents the framework designed to handle this formalization with the simplification done. Section 3.4 addresses the possible future work while, the last section of this chapter, Section 3.5, synthesizes its contents.

3.2 Formalization

The Geometric reasoning and planning problem can be described with the 2-uplet $\{D_g, E\}$ where:

- D_g is the domain that contains all the available actions and
- E is the set of entities known to the robot.

The problem consists on solving queries where the goal of each one of them is to make an agent (or multiple agents) perform an action. The next subsections will present in details the actions, the entities, and other models used to formalize the problem.

3.2.1 Entities

Each entity e is defined by an identifier, a type and a description $\{id_e, t_e, g_e\}$. t_e refers to the entity type, which can be one (or more) of the followings: a human, a robot, a manipulable object, a support object or a virtual object. The agents (robot and human) are considered in order to compute their motions, and, when needed, some social rules. The objects can be from different types, such as manipulable and support at the same time, and as their type indicates, manipulable object can be moved around by the agents, and can be placed on the support objects. The virtual objects are special objects for which the collision can be ignored in certain occasions.

The description g_e follows the classical one in motion planning (see, for instance, LaValle (2006)), a kinematic graph, where the nodes are the entity joints and the edges the links. In this context, the links are the entity rigid bodies, which are defined by a frame and a representation of the geometric model in this frame. The joints are defined by a parametrized transformation matrix, where the independent parameters that characterize this transformation are the degree of freedoms (*DoFs*). Each *DoF* value belongs to a set $S_e \subset \mathbb{R}$, which can be infinite, or bounded by the entity geometry or by the world. If the robot has n *DoFs*, the set of transformations is usually a manifold of dimension n. This manifold is called the configuration space $Cspace_e$, and an instance of this $Cspace_e$ is called a configuration q_e; in other words, q_e is the value of every degree of freedom of every joint in the entity kinematic graph.

In addition to this $Cspace_e$, the kinematic graph is used to compute the entity forward kinematic, which consists on computing the relative (to the entity) and the absolute position of every entity rigid body. It is also used to compute the entity inverse kinematic: computing the *DoFs* values based on the position of one of the entity end effectors. Finally, it is also used to compute trajectories, using motion planners, which consists on computing a collision free path between a starting and a stopping configuration.

Let $Cspace$ refer to the configuration space of all the entities and $Cspace_{e1}$ the configuration space of the entity $e1$. In an environment where the entities are $\{e1, e2, ..., em\}$, the $Cspace$ is the Cartesian product of all the $Cspace_{ei}$:

$$Cspace = Cspace_{e1} \times Cspace_{e2} \times Cspace_{e3} \times ... \times Cspace_{em}$$

3.2.2 World States

A world state (ws) is the state of each entity from E at a given time t_i. The state of an entity e is defined by $\{q_e, t_e, pos\}$ where:

- q_e is the entity configuration at time t_i. This configuration can be fully known, or partially known (in case of uncertainties) or not known at all (no information regarding the entity position).
- t_e is the entity trajectory at time t_i. In other words, in addition to the position, it contains the future and previous configurations of the entity, in addition to its dynamic. The trajectory can also be fully known, partially known or completely unknown.
- pos represent the position of q_e on the trajectory t_e at time t_i.

3.2.3 Actions

An action $a \in D_g$ is defined by $\{aId, des, \text{IN}\}$:

- aId is the action identifier which is unique. It can be **Pick**, **Place**, **Give** and so on.
- des is the action description, explained later in this subsection.
- IN is the list of required inputs, which varies from an action to the other.

We can consider the example of the **Pick** action: the aId is **Pick**, the des will be defined later, and IN contains the agent performing the task, the object which needs to be picked and the initial world state ws_{init}. Note that among the various possible inputs an action can have, it will always need an initial world state.

D_g contains every action description mapped with it identifier aId, and when a query is made to such a system, only the aId and the IN are needed to solve the query. The result of a query links this initial world state to a final one ws_{final} corresponding to the end configuration of every entity in the world. Note that the link between these two world states is actually one or multiple trajectories, and, as for any trajectory, we can retrieve every entity configuration at any point of it, and by extension a corresponding world state.

In order to explain the action description des we first explain the expected result. The computation of an action consists on finding a geometric action solution (GAS) which is composed by a set of geometric sub-action solutions (GSAS) and a cost c. A GSAS is defined by $\langle e, t, gsasNexts \rangle$ where:

- e is an entity.
- t is the trajectory that should be performed by the entity.
- $gsasNexts$ is the list of all the GSAS that need this GSAS to be finished in order to begin (we are going to refer to it as *geometric causal link*).

Following the previous definitions, for each GSAS, we can retrieve an initial world state and a final one. The initial world state of all the GSAS with no previous (or which does not belong to any $gsasNexts$) is the same: ws_{init} the action initial world state. Figure 3.1 shows an example of a GAS, composed of 4 GSAS. Each GSAS has an initial and final world state, and by linking all the world state together, they form the link between the initial and the final world state.

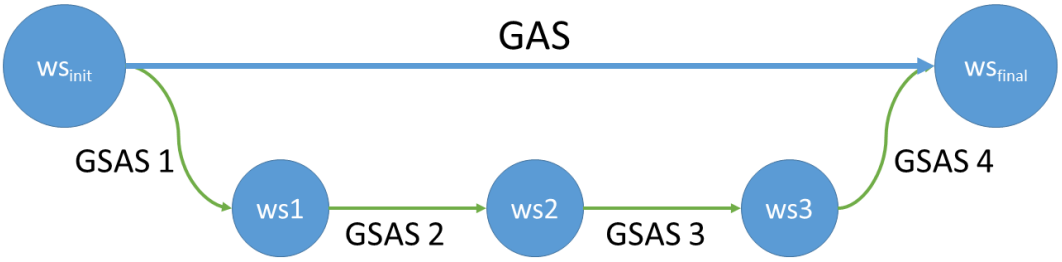


FIGURE 3.1: An example of a GAS. The four GSAS composing it links intermediate world state (ws1, ws2, and ws3) together to form the link between the initial world state ws_{init} and the final one ws_{final}

The cost c is a scalar computed based on each GSAS.

Each GSAS has its own description, referred to as a sub-action description. An action description is composed by a number of sub-action descriptions linked between themselves using temporal operators, such as $\|$ meaning that both GSAS begin at the same time, $\circ$ where the second GSAS begins after the first one is finished (*geometric causal link*), or / which is used when there is no synchronisation between both GSAS. These temporal operators are very basic ones and are enough for describing the actions so far, but we can also use more complex ones such as Linear Temporal Logic as proposed by Plaku (2012b).

The sub-action description can be written under the form of $(gp,\ ss,\ fc)$ where gp is the geometric pre-conditions, ss the search space and fc the final constraints:

Geometric pre-conditions Those are facts that can be computed based on the geometric model of the world (more explanations concerning facts are given in Subsection 3.2.4). Those facts need to be true in the GSAS initial world state in order to be able to compute said GSAS.

Search spaces a subset of $Cspace$ ($ss \subset Cspace$), obtained by fixing the $DoFs$ not useful to perform the GSAS. This subset is used to find the final GSAS world-state and to limit the search during the motion planning exploration (e.g. only the right arm of the agent a can move, all the other agents, are fixed, and the rest of the agent DoFs are also fixed).

Final constraints a subset of the Search Spaces ss used only to compute the final world state of the GSAS (e.g. the right arm end effector should be at 10 cm over the object).

In some cases, one of these parts might not be needed (represented as $\emptyset$ in the sub-action description). If the geometric pre-condition is missing, it means that no conditions need to be tested on the GSAS initial world state. If the Search Space or the final constraints are missing, this means the space that they define is not bounded (equal to $Cspace$ for the search space and equal to ss for the final constraints)

In order to link the description, some logical operators are needed, some of them can link two geometric pre-conditions, such as & for "and" or | for "or" while others link two configuration spaces (two search spaces or two final constraints) such as $\cap$ for "intersection" or $\cup$ for "union"

To synthesise, an action description is built by smaller sub-action descriptions, linked between them by temporal operators, and each one of these sub-action description is formulated using a logical linked geometric pre-conditions, search spaces and final constraints.

3.2.3.1 Example

Let's take as an example the action "**Pick**". In this action, the agent needs to grab an object then disengage itself from the support object[1]. As some definition are still missing, this example is not complete, its complete version is available in Subsection 3.3.2.1.

The **Geometric pre-conditions** used for a **Pick** are:

- $HFree_A$ no object in arm A end effector.
- $Reach_A(O)$ target object O is reachable by arm A.
- $HFull_A(O)$ target object O is in arm A end effector.

The **Search spaces** are:

- $Fix_A(O)$ the subset of $Cspace$ where all entities are fixed, apart from the DoFs corresponding to the arm A of the agent, and the object O. If O is omitted, only the arm is not fixed.

The **Final constraints** are:

- $HApp_A$ arm A end effector in approach position[2].
- $HGrasp_A$ arm A end effector grasping the object.
- $Free(O)$ object O disengaged from its support.

The description of the action **Pick** object O is:

$$\textbf{Pick}(O) = \exists a \in \{R, L\}, (HFree_a \& Reach_a(O),\ Fix_a(),\ HApp_a) \circ (\emptyset,\ Fix_a(),\ HGrasp_a), \\ \circ (HFull_a(O),\ Fix_a(O),\ Free(O)) \quad (3.1)$$

where R is for right arm and L for left arm. In this example we consider that there is only one agent (with two arms) who will perform the action. The list of inputs of this action contains only the object O and the initial world state ws_{init}.

This action solution is a GAS where there are three GSAS: approach, grasp, and disengage.

The rest of this section presents a number of definitions that complement the formalization.

[1] the support is the object or obstacle the target object is placed on or attached to at ws_{init}

[2] An approach position for an end effector is a position from where the end effector can reach a grasping position through a direct, straight line in the Cartesian space with a short motion.

3.2.3.2 Language discussion

The set of rules needed to write an action description can be considered as a language. This language, although very simple, can be used to describe more and more complex actions.

The advantage of such a language is the simplicity of describing new actions: the pre-conditions are symbolic facts that are intuitively understood easily with basic three dimensional logic. The search space and final constraints are subsets of the agents C_{space} that defines how the action should be done and what is its goal of it: the search space defines globally what is going to move during the action, while the final configuration needs to define a subset of this space where a number of properties should be true (such as having the object in the hand and so on).

The disadvantage of using such a language concerns the implementation: as each action has different pre-conditions, search spaces, and final constraints, each action will need a different set of functions to test, compute, and validate each part of the description. Although, as we work in a manipulation/navigation domains, the functions used are quite close and can be reused (as seen later in this chapter) for other actions too.

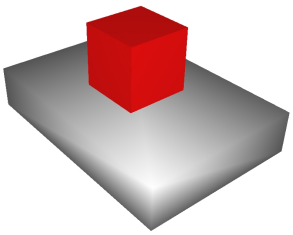
3.2.4 Facts & affordances

As seen in Subsection 3.2.2 a world state is a precise geometric description of the model of the world at a given time. In order to qualify this information and give it a symbolic meaning, to make it human readable and also usable by other models, facts are computed in these world states. A fact is a link between two entities: for example, object A **is on** object B or Object A **is in** Agent's X hand. A fact can also be defined as the relative configuration between two entities: if the polygon formed by the bottom of object A is included in the polygon formed by the top of object B, then object A **is on** object B. This relative configuration can enable us to define a space, related to a specific world state, where a fact is always valued to true.

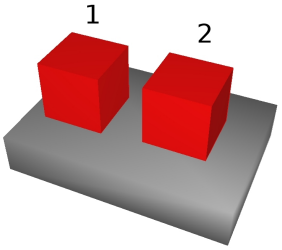
A fact can be represented under the form: $\{e1, type, e2, v\}$ where $type$ is the type and (if relevant) the sub-type of the fact, $e1$ and $e2$ are the entities involved in the fact (in this order) and v can be either a Boolean or a scalar, depending on the $type$, for example $\{Obj1,$ **is in**, $Obj2,$ $true\}$ means that $Obj1$ is located inside $Obj2$. The affordances are considered as facts here, as they are, following the definition of Sahin et al. (2007) (cited in Section 3.1), links between an agent (an entity) and an object (another entity). For example, object o **is reachable by** agent X or object o **is visible by** agent X are example of affordances that can be useful in a HRI context. Figure 3.2 shows examples of these affordances, in addition to two other facts (**is on** and **is next to**)

3.2.5 Additional definitions

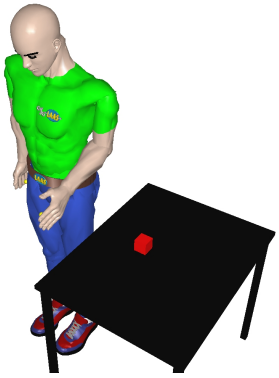
In this subsection, a number of additional definitions linked to the previous formalization are presented.



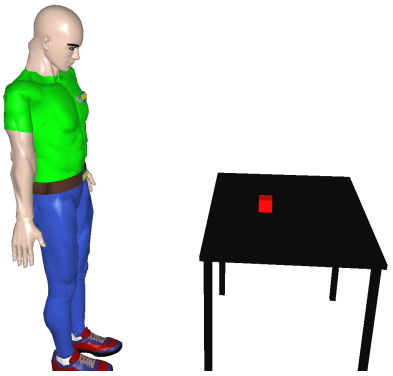
(a) {*RedCube*, **is on**, *GreyBook*, *true*} The polygon forming the bottom of *RedCube* is inside the polygon forming the top surface of *GreyBook*



(b) {*RedCube*1, **is next to**, *RedCube*2, *true*} The minimal distance between *RedCube*1 and *RedCube*2 is smaller than a given threshold



(c) {*RedCube*, **is reachable by**, *Human*, *true*} The inverse kinematic of the *Human* model enables him to touch the *RedCube*



(d) {*RedCube*, **is visible by**, *Human*, *true*} The *RedCube* is in the field of view of the *Human*

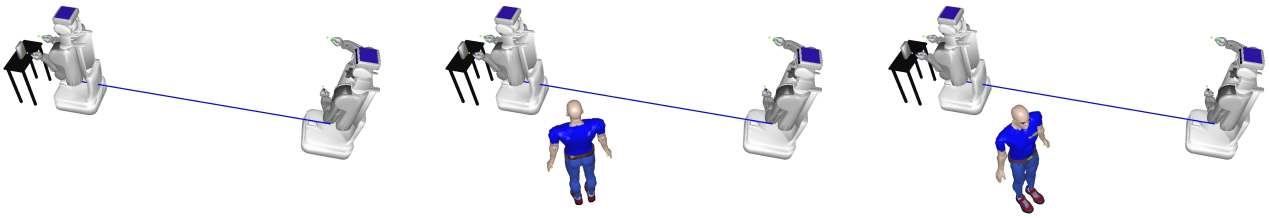
FIGURE 3.2: Various types of facts and affordances in different situations.

3.2.5.1 Costs

The function f_{cost} computes the cost of a GSAS based on its trajectories and its initial and final world states. This computation can be related to the geometry only (such as the trajectory length) or to a more complex notions such as human-aware considerations. Figure 3.3 shows an implemented example were the cost changes depending on the human position (in this example, the robot navigates to the table). It is computed based on the costs presented by Sisbot et al. (2007b), and taking into account the human-robot distance and the robot visibility by the human (going behind and close to the human is to be avoided).

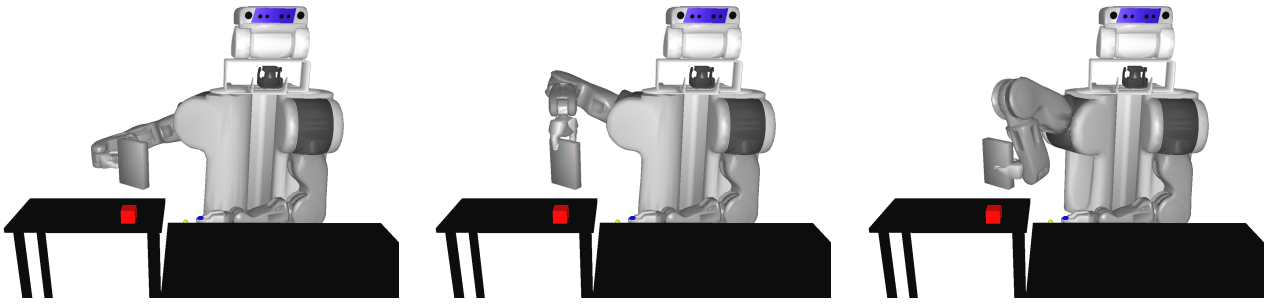
3.2.5.2 Alternatives

The space defined by the **Final constraints** for the final world state in each sub-action description is usually not a singleton, therefore multiple solutions for the same sub-action description (and by transition, the same GAS) may exist and are called alternatives. If the final constraints define a singleton, there is only one alternative, the one corresponding to the unique solution. In other words, alternatives are unique, and cannot have the same final world state. Figure 3.4 shows different alternatives for the **Pick** action, where the final constraints $HGrasp_A$ and $HApp_A$ have multiple solutions.



(a) **c = 3.7** (b) **c = 6.4** (c) **c = 8.9**

FIGURE 3.3: Three scenarios where the robot navigate in a human environment. The path doesn't change, but the cost does: it is low when the human is far away 3.3(a), it gets higher when the human comes closer 3.3(b), and if he is not facing the path 3.3(c), it goes even higher.



(a) First alternative (b) Second alternative (c) Third alternative

FIGURE 3.4: Different alternatives for the **Pick** action

3.2.5.3 A geometric plan

A geometric plan is a set of GAS linked between themselves through geometric causal links. These links are created based on the initial and final world state of each action: if the final world state of a GAS $g1$ is the initial world state of a GAS $g2$, then, in the geometric plan $g1$ is the previous of $g2$.

The geometric plan can be a simple sequence of GAS or a set of GAS assembled as Directed Acyclic Graph (for example in the case of multiple robot acting in parallel).

The GRP stores all the possibilities and the alternatives computed, and arrange them in a three, where a path from the root to the leave is the geometric plan.

3.2.5.4 Additional constraints

Constraints can be considered as spaces limiting the search spaces of an action. To be more precise, in our case, constraints will be applied to a sub-action description, either to the search space or to the final constraints. In other terms, to add a constraint we can simply add to the targeted search space or final constraints an *intersection* with the constraint space.

For example, the **Pick** action can be constrained by: $Hatop_a$ the approach position should be strictly above the object, which can be incorporated in the action description as:

$$\textbf{Pick}(O) = \exists a \in \{R, L\}, (HFree_a \& Reach_a(O),\ Fix_a(),\ HApp_a \cap Hatop_a)) \circ (\emptyset,\ Fix_a(),\ HGrasp_a), \\ \circ (HFull_a,\ Fix_a(O),\ Free(O)) \quad (3.2)$$

As they define spaces in specific world states, facts can also be used as constraints, by using the *intersection* operator again. For example, for an action where the agent R needs to place object O on the table S, a constraint can be set as $\{O,\ \textbf{is reachable by},\ H,\ true\}$. This constraint reduces the space defined by the final constraints of the action description to itself intersecting the space reachable by the human.

3.3 Framework

In order to put in practice this formalization, a framework was developed, able to compute GASs, while maintaining a plan and computing human-aware costs for each action. This section defines in details this framework by first presenting the choice and the simplifications made (Subsection 3.3.1) then it presents some examples of formal definitions available in the framework (Subsection 3.3.2). Next it shows three different algorithms able to find GASs based on the action description (Subsection 3.3.3), later it shows the results of this framework concerning one of these algorithms and finally, it states some possible future works concerning the framework.

3.3.1 Simplification and choices

We assume that in this framework the entities states are fully known (the configuration and the trajectories are fully known and defined at every moment). The second assumption concerns the sequentiality: only sequential actions are possible (no parallel actions) causing the plan to be a sequence and not a Directed Acyclic Graph.

Some simplifications were also done to facilitate the computations: they add to the models presented in the previous section a number of parts on which reasoning at symbolic level is easier than reasoning on the basic models.

Wrist manipulation joint (WMJ) Every agent is equipped with a WMJ. It is a virtual point fixed –with a transformation matrix– to the agent end effectors (the hands or the grippers) and is considered roughly at its centre when it is closed as shown in Figure 3.5. (Zacharias et al. (2006), among others, call it Tool Centre Point)

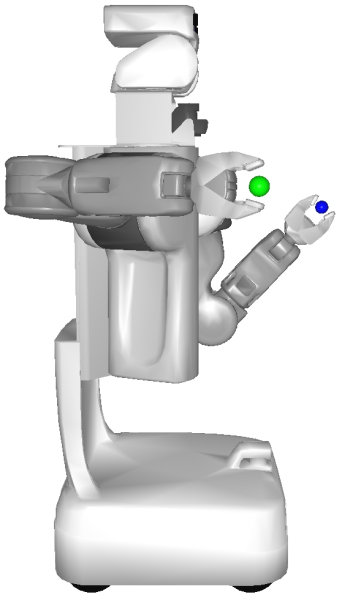


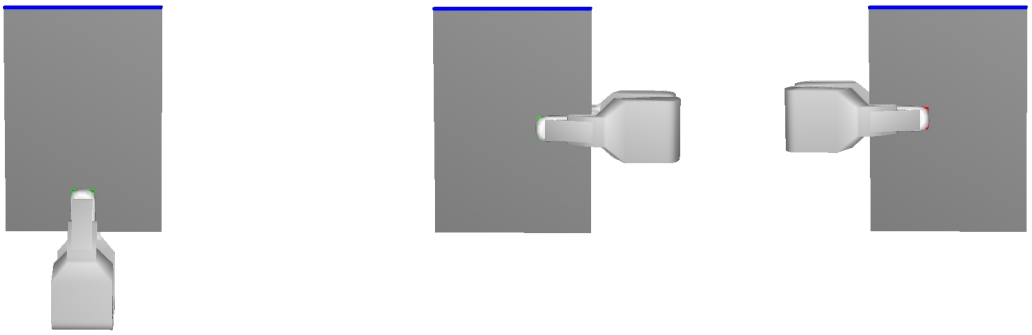
FIGURE 3.5: This is the PR2 robot, a two-arm (R and L) mobile manipulator, the green and blue points are respectively the WMJs of R and L end effectors.

Attachments An attachment is a transformation matrix between the WMJ and an object. It enables the system to keep track of the objects grasped by the agents and is stored in the world states. In the rest of this manuscript, an object attached to the end effector of an agent arm means that this transformation matrix is known.

Arms We consider that each agent is equipped with at least one arm, and at most two. When there are two arms they are noted R and L for right and left. Let A_{ag} be the set of agent ag arms. This is not a limitation of the system, adding robots with more arms is feasible, but for now, the implementation handles only up to 2 arms.

In addition to these simplifications, the framework needs a number of additional information. This information such as the possible grasps can be computed on-line using off-the-shelf methods such as Miller et al. (2003) for the grasps, but in this work, we have made the choice to pre-compute the following information in order to speed up the on-line computation time:

Grasps The grasps used in the framework are precomputed for each different end effector, in the form of a transformation matrix between the object and the WMJ, in addition to a direction from where the grasp is feasible. Figure 3.6 shows 3 different grasps for the grey book. If an object does not have grasps, it is not considered as a manipulable object. Let G_{o-ee} be the set of precomputed grasps of object o by end effector ee.



(a) (b) (c)

FIGURE 3.6: Different grasps for the grey book (this is just a sample from the available grasps).

Supports An object support is a geometrical form attached to an object face (by a transformation matrix) where other objects can be placed. An object can have multiple supports (such as a shelf) and when an object does not have any supports, it is not considered as a support object. Figure 3.7 shows different supports on various tables. Let S_o be the set of precomputed supports of object o in the environment.

Stable configurations These are rotations of a manipulable object that enable a stable placement when the object is on a horizontal support. An object without Stable configurations cannot be placed in any ways (but can, for example, be handed over). Figure 3.8 illustrate these configurations on the grey book. Let P_o be the set of precomputed Stable configurations of the object o.

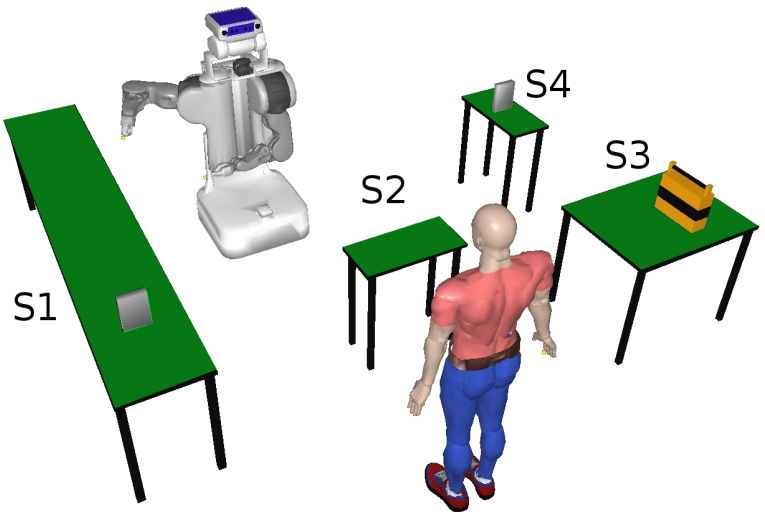


FIGURE 3.7: The supports of the tables are represented in green, each table has one support, which is a rectangle covering its top face. The objects on the tables are not support objects, hence, they don't have any supports.

In addition to this, the definition of a GAS was extended to include a unique GAS identifier ($gasNum$) that differentiates GASs from each other, and a unique alternative identifier ($gasAltNum$) that differentiates, within the same GAS, alternatives from each other. The GAS definition now contains the list of all the GSAS linked together through geometric causal link, its cost, the $gasNum$, and the $gasAltNum$.

3.3.2 Actions description and examples

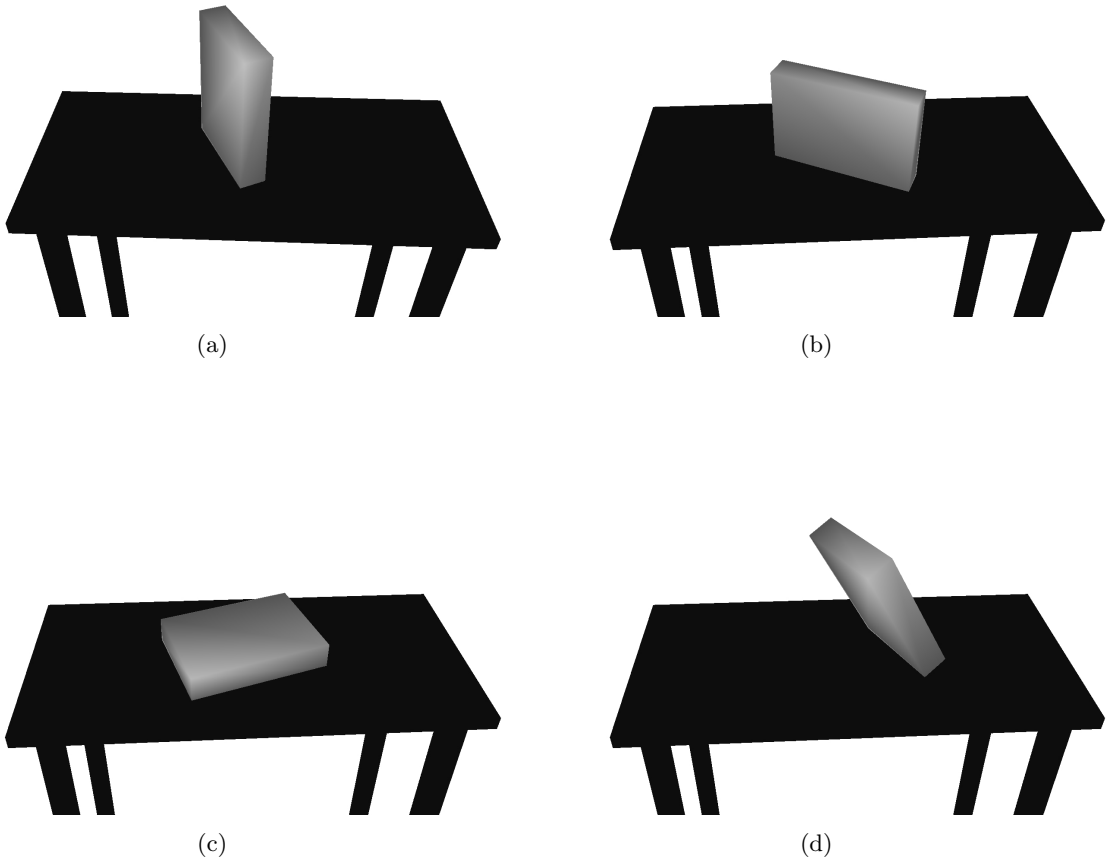
We have developed in this framework a number of actions. Some examples are described in details in this subsection. Some descriptions used in the following are common for different actions (in the rest of this chapter, ag refers to the agent, a refers to one of its arms, and $a.ee$ refer to arm a end effector).

Geometric pre-conditions

- HF$ree_a(ag)$: no object in $a.ee$.
- HF$ull_a(ag, o)$: o is in $a.ee$.
- OR$each_a(ag, o)$: object o reachable by arm a of ag while the robot base is fixed.
- EEO$pen_a(ag)$: $a.ee$ is open.
- EEC$lose_a(ag)$: $a.ee$ is closed.

Search spaces

- $Fix_a(ag)$: the subset of $Cspace$ where all entities DoFs are fixed, apart from the DoFs of the arm a of ag.
- $UpFix(ag)$: the subset of $Cspace$ where only ag displacement DoFs are not fixed.
- OF$ix_a(ag, o)$: the subset of $Cspace$ where all entities DoFs are fixed, apart from DoFs of the arm a, and the DoFs of object o which is attached to $a.ee$.



(a) (b) (c) (d)

FIGURE 3.8: (a), (b) and (c) are different stable configurations for the grey book (this is just a sample from the available stable configurations). (d) is not a stable configuration.

- EEF$ix_a(ag)$: the subset of $Cspace$ where all entities DoFs are fixed, apart from the DoFs of $a.ee$.

Final constraints

- EEC$_a(ag)$: the subset of $Cspace$ where $a.ee$ is closed (This subset contains also the cases when the end effector is not completely closed because it is grasping an object).
- EEO$_a(ag)$: the subset of $Cspace$ where $a.ee$ is open.

The inputs are expressed between the parentheses while the non-defined variables (such as the arm to use) are noted as subscripts. In the following descriptions, the inputs might be omitted when no ambiguity is possible.

3.3.2.1 Pick

The **Pick** action description presented in Subsection 3.2.3.1 was lacking some details defined in this section and is fully redefined here. The final constraints needed are:

- HA$pp_{a-g}(ag, o)$: the subset of $Cspace$ where WMJ of $a.ee$ is at a given distance[3] from the position defined by grasp $g \in G_{o-a.ee}$ in the direction defined by g. This position is an approach position in order to grasp the object.

- HG$rasp_{a-g}(ag, o)$: the subset of $Cspace$ where WMJ of $a.ee$ is at the position defined by g (this grasp must be the same as the one in $HApp_{a-g}(ag, o)$).

- OF$ree(o)$: the subset of $Cspace$ where the object o is at a given distance[3] above its initial support. This position allows to disengage the object o from the contact it has with its support.

The description of the action **Pick** is now:

$$\begin{aligned}\textbf{Pick}(ag, o) = \exists a \in A_{ag}, \exists g \in G_{o-a.ee}, (\text{HF}ree_a(ag)\&\text{OR}each_a(ag, o),\ Fix_a(ag),\ \text{HA}pp_{a-g}(ag, o)) \\ \circ (\emptyset,\ Fix_a(ag),\ \text{HG}rasp_{a-g}(ag, o)) \circ (\emptyset,\ \text{EEF}ix_a(ag),\ \text{EEC}_a(ag)) \\ \circ (\text{HF}ull_a(ag, o),\ \text{OF}ix_a(ag, o),\ \text{OF}ree(o))\end{aligned} \quad (3.3)$$

The input list of a **Pick** contains also the initial world state (as this input is mandatory for every action, it will be omitted in the rest of the action descriptions), the manipulable object o and the agent ag performing the action. The different parts of the description are explained in the followings and illustrated in Figure 3.9:

- $\exists a \in A_{ag}, \exists g \in G_{o-a.ee}$ means that at least one pair (arm, grasp) exists where the next parts of the description are fulfilled.

- (HF$ree_a(ag)$&OR$each_a(ag, o)$, $Fix_a(ag)$, HA$pp_{a-g}(ag, o)$) is the approaching sub-action description. The corresponding trajectory, should bring a free end effector from its initial position to a position where the object can be reached easily (a given distance[3] away from the object), Figure 3.9(a)[4].

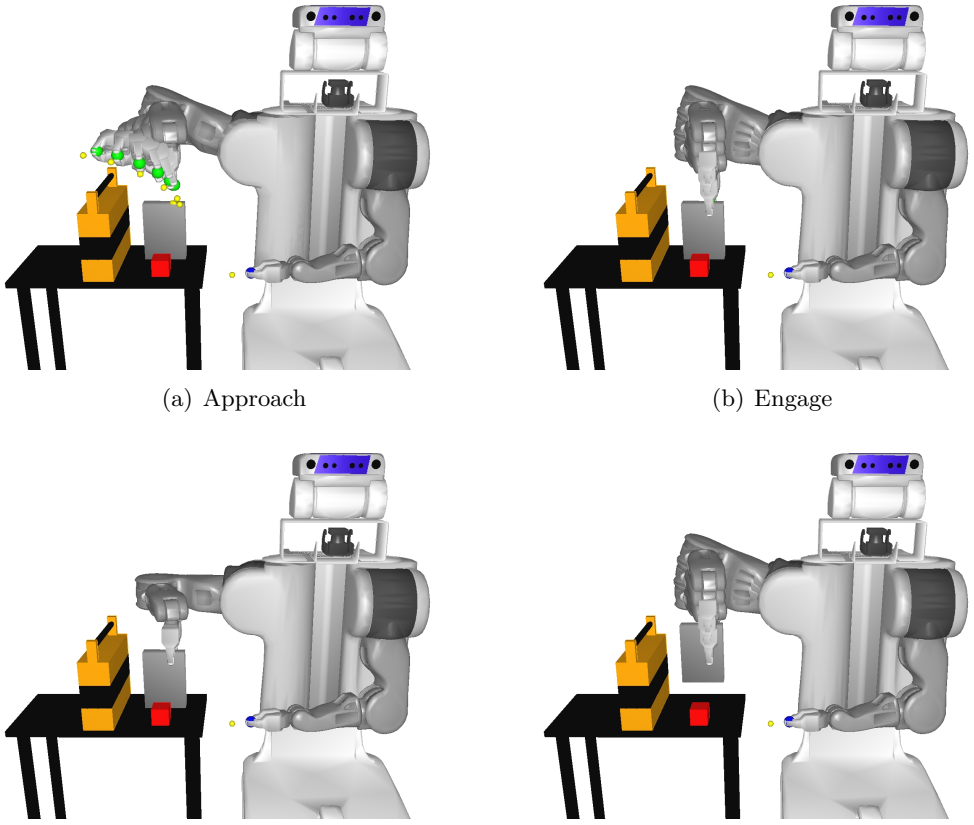
- ($\emptyset$, $Fix_a(ag)$, HG$rasp_{a-g}(ag, o)$) is the engaging sub-action description. The corresponding trajectory should be a simple straight line of the end effector from the previous position to a position where closing it will result on grasping the object, Figure 3.9(b).

- ($\emptyset$, EEF$ix_a(ag)$, EEC$_a(ag)$) is the grasping sub-action description. The corresponding trajectory closes the end effector to grasp the object, Figure 3.9(c).

- (HF$ull_a(ag, o)$, OF$ix_a(ag, o)$, OF$ree(o)$) is the disengaging sub-action description. The corresponding trajectory disengage the object from the contact it has with its support, Figure 3.9(d).

[3] In our implementation and for the scenarios we are using, this distance is set to 10 cm.

[4] The figures (and the ones after) are generated in simulation within the environment provided by move3d Siméon et al. (2001).



(a) Approach (b) Engage

(c) Grasp (d) Disengage

FIGURE 3.9: The different GSAS of **Pick**, a trace of the trajectories is shown.

3.3.2.2 Place

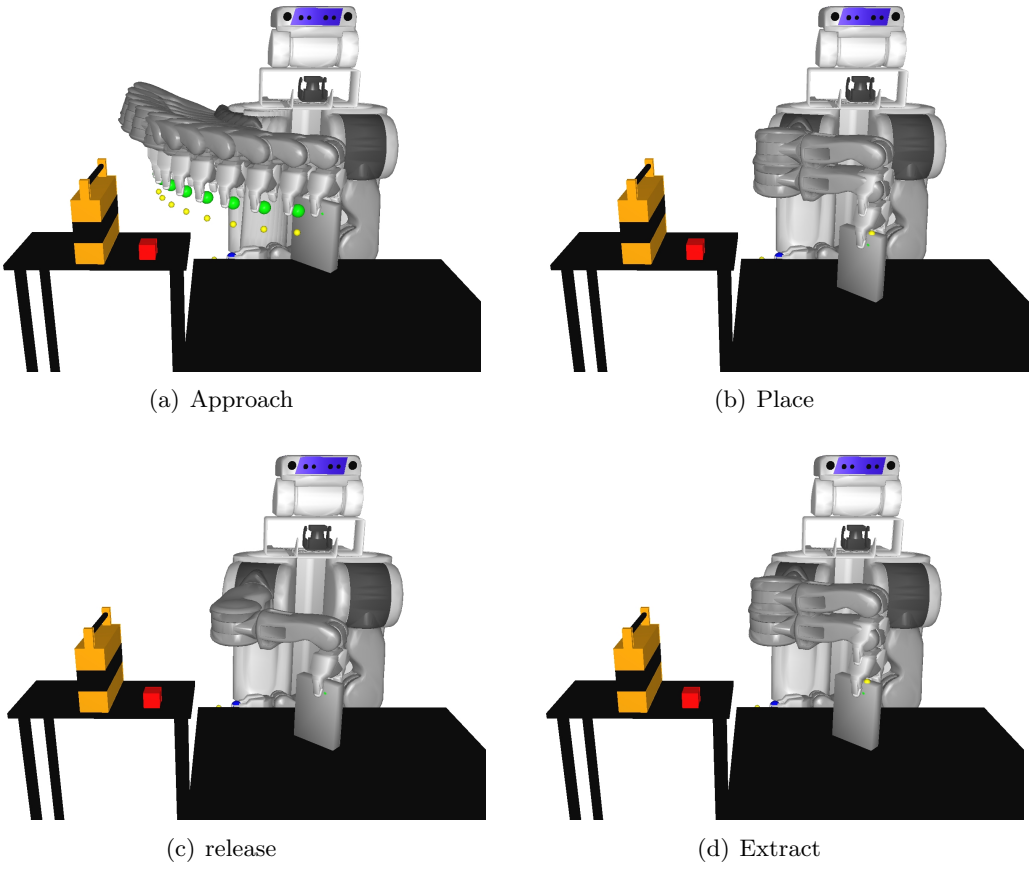
In the **Place** action, the agent needs to approach the support with the object Figure 3.10(a), **Place** it Figure 3.10(b), release it Figure 3.10(c) and then extracts his arm Figure 3.10(d). To describe the **Place** action, one more pre-condition is needed: $SReach_a(ag, so)$, support so is reachable by a, and some more final constraints:

- HA$ppr_{(x,y)-p}(ag, o, so)$: the subset of $Cspace$ where o is at a given distance[3] above the support so at coordinate (x,y) –relative to the support– with a stable configuration $p \in P_o$.
- HR$el_{(x,y)-p}(ag, o, so)$: the subset of $Cspace$ where o is on the support so at coordinate (x,y) with a stable configuration $p \in P_o$.
- EEF$ree_a(ag, o)$: the subset of $Cspace$ where WMJ of $a.ee$ is at a given distance[3] away from o.

The description of the action **Place** is then:

$$\begin{aligned}\textbf{Place}(ag, o, so) = \exists a \in A_{ag}, \exists s \in S_{so}, \exists (x, y) \in s, \exists p \in P_o,\\ (\text{HF}ull_a(ag, o) \& SReach_a(ag, so),\ \text{OF}ix_a(ag, o),\ \text{HA}ppr_{(x,y)-p}(ag, o, so))\\ \circ (\emptyset,\ \text{OF}ix_a(ag, o),\ \text{HR}el_{(x,y)-p}(ag, o, so)) \circ (\emptyset,\ \text{EEF}ix_a(ag),\ \text{EEO}_a(ag))\\ \circ (\text{HF}ree_a(ag),\ Fix_a(ag),\ \text{EEF}ree_a(ag, o))\end{aligned} \quad (3.4)$$

The inputs for the **Place** are the agent ag, the manipulable object o, and the support object so. The framework chooses which support to use in the support object (in the case of more than one support), where exactly (x, y) to **Place** the manipulable object and which stable configuration p to use.



(a) Approach (b) Place

(c) release (d) Extract

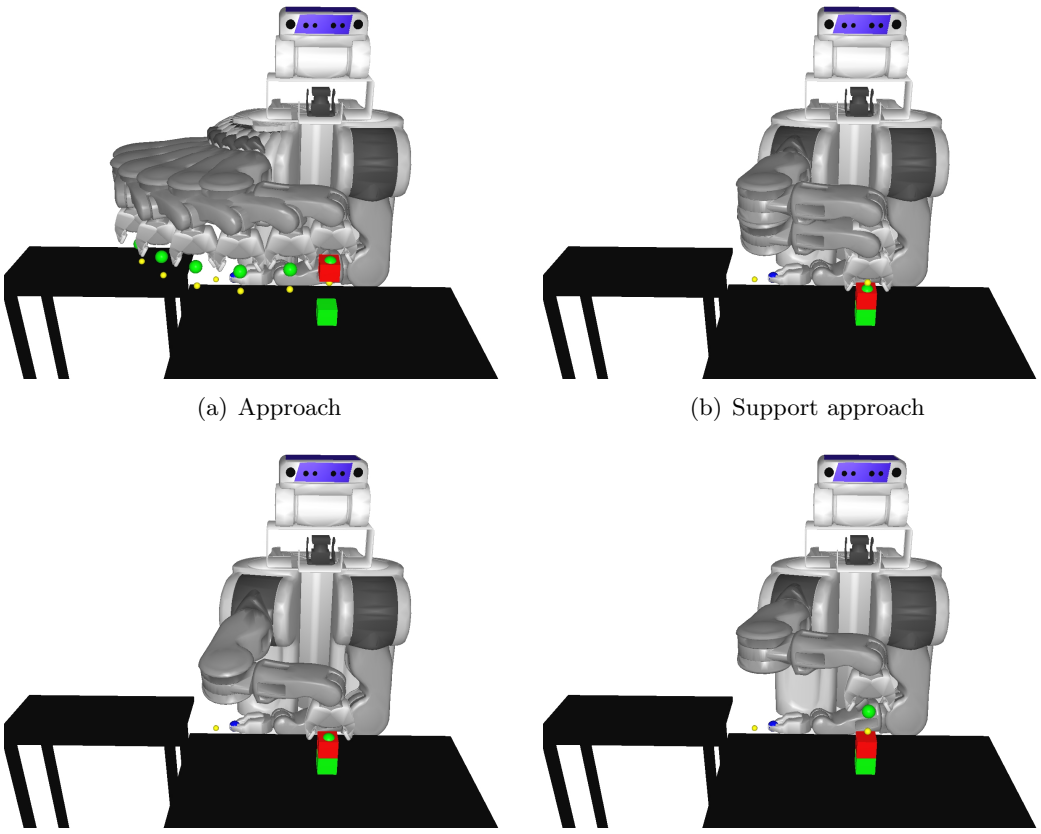
FIGURE 3.10: The different GSAS of the **Place** action, a trace of the trajectories is shown.

3.3.2.3 Stack

The **Stack** action is very close to the **Place** action, the only difference is that in the **Stack** action the exact position where to place the object is given as input (under the form of the support, with the position at its centre (c_x, c_y)):

$$\begin{aligned}\textbf{Stack}(ag, o, so) = \exists a \in A_{ag}, \exists p \in P_o,\\ (\text{HF}ull_a(ag, o)\&SReach_a(ag, so),\ \text{OF}ix_a(ag, o),\ \text{HA}ppr_{(c_x,c_y)-p}(ag, o, so))\\ \circ (\emptyset,\ \text{OF}ix_a(ag, o),\ \text{HR}el_{(c_x,c_y)-p}(ag, o, so)) \circ (\emptyset,\ \text{EEF}ix_a(ag),\ \text{EEO}_a(ag))\\ \circ (\text{HF}ree_a(ag),\ Fix_a(ag),\ \text{EEF}ree_a(ag, o))\end{aligned} \quad (3.5)$$

Figure 3.11 shows an example of a **Stack** action.



(a) Approach (b) Support approach

(c) Release (d) Extract

FIGURE 3.11: The different GSAS of the **Place** action, a trace of the trajectories is shown.

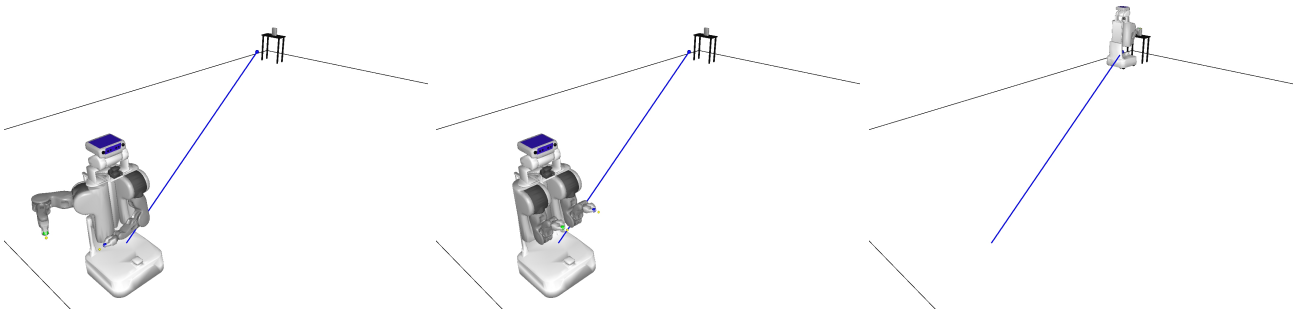
3.3.2.4 NavigateTo

In the **NavigateTo** action, the agent needs to go into a navigation configuration then navigates to the target. One specific search space is needed: $UpperBody(ag)$: the subset of $Cspace$ where all entities are fixed apart from the agent upper body[5] and the object attached to his end effectors. The specific final constraints of this action are: $NavPos(ag)$, the subset of $Cspace$ where the agent is in a navigation configuration, and $OnTarget(ag, e)$: the subset of $Cspace$ where the robot reached the target entity e. Reaching a target entity depends on the entity type, if it is an agent, the agents should be able to reach each other extended arms, if the entity is an object or a support, it should be reachable by the agent. Its description is:

$$\textbf{NavigateTo}(ag, e) = (\emptyset,\ UpperBody(ag),\ NavPos(ag)) \circ (\emptyset,\ UpFix(ag), OnTarget(ag, e)) \quad (3.6)$$

The input is a 2d zone, which can be specified by providing an entity: the zone will be the one immediately around this entity. Figure 3.12 shows an example of a **NavigateTo** action where a robot goes to a table.

[5]By upper body we mean all the DoFs not needed for the navigation



(a) Initial configuration (b) Navigation configuration (c) Target configuration

FIGURE 3.12: The different steps of the **NavigateTo** action, the initial and final world state of each GSAS is shown (the blue line is the robot navigation path).

3.3.2.5 Handover

The **Handover** as defined in the previous chapter is complex, as it involves two agents. We are going to differentiate between them as ar for the receiver and ag for the giver. Two specific final constraints for this action are: $DistTarget_{a-b}(ar, ag)$ the subset of $Cspace$ where the distance between the two agents is smaller than the sum of ar arm a length and ag arm b length, and $AgentReach_{a-b}(ar, ag)$: $a.ee$ the subset of $Cspace$ where can reach $b.ee$. The action description is:

$$\begin{aligned}\textbf{Handover}(ag, ar, o) = \exists(x_r, y_r, \theta_r) \in (\mathbb{R}, \mathbb{R}, [-\pi, \pi]), \exists(x_g, y_g, \theta_g) \in (\mathbb{R}, \mathbb{R}, [-\pi, \pi]) \exists a \in A_{ar}, \exists b \in A_{ag} \\ ((\text{HF}ree_a(ar),\ UpperBody(ar),\ NavPos(ar)) \| (\text{HF}ull_b(ag),\ UpperBody(ag),\ NavPos(ag))) \\ \circ (\emptyset,\ UpFix(ar) \cup UpFix(ag),\ DistTarget_{a-b}(ar, ag)) \\ \circ (\emptyset,\ Fix_a(ar) \cup \text{OF}ix_b(ag, o), AgentReach_{a-b}(ar, ag)) \quad (3.7)\end{aligned}$$

The inputs are the agents and the object to exchange. This description is the one used in Section 2.3 to find handover position, even if it was done outside of this framework, it is still covered by the description. The first part, $\exists(x_r, y_r, \theta_r) \in (\mathbb{R}, \mathbb{R}, [-\pi, \pi]), \exists(x_g, y_g, \theta_g) \in (\mathbb{R}, \mathbb{R}, [-\pi, \pi]) \exists a \in A_{ar}, \exists b \in A_{ag}$ means that at least one pair (giver position, receiver position) exists where the agents, after moving to a navigation position $((\text{HF}ree_a(ar),\ UpperBody(ar),\ NavPos(ar)) \| (\text{HF}ull_b(ag),\ UpperBody(ag),\ NavPos(ag)))$, can navigate to $(\emptyset,\ UpFix(ar) \cup UpFix(ag),\ DistTarget_{a-b}(ar, ag))$, and that at least one pair (giver arm, receiver arm) exists where in these positions, the agents arms can reach each others one $(\emptyset,\ Fix_a(ar) \cup \text{OF}ix_b(ag), AgentReach_{a-b}(ar, ag))$.

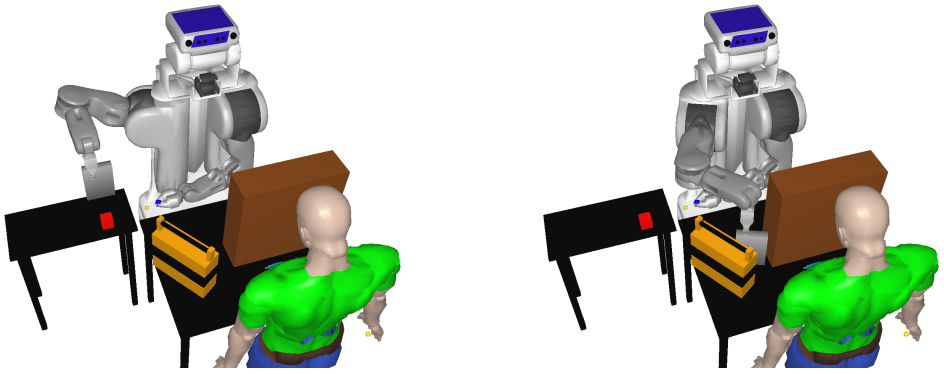
3.3.2.6 PlaceReachable

Following the formalization, we can add human-aware actions, for example, **PlaceReachable** is an action where the agent ag holding the object place the object in a place which is reachable by a target agent ar. let's consider the final constraints: $\text{AR}eachO(at, o)$ based on the fact: $\{o,\ \textbf{is reachable by},\ ar,\ true\}$, it defines the subset of $Cspace$ where the object o is reachable by agent ar. The action description is

then:

$$\begin{aligned}\textbf{PlaceReachable}(ag, ar, o, so) = \exists a \in A_{ag}, \exists s \in S_{so}, \exists (x,y) \in s, \exists p \in P_o,\\ (\mathrm{HF}ull_a(ag,o) \& SReach_a(ag,so),\ \mathrm{OF}ix_a(ag,o),\ \mathrm{HA}ppr_{(x,y)-p}(ag,o,so) \cup \mathrm{AR}eachO(ar,o))\\ \circ (\emptyset,\ \mathrm{OF}ix_a(ag,o),\ \mathrm{HR}el_{(x,y)-p}(ag,o,so)) \circ (\emptyset,\ \mathrm{EEF}ix_a(ag),\ \mathrm{EEO}_a(ag))\\ \circ (\mathrm{HF}ree_a(ag),\ Fix_a(ag),\ \mathrm{EEF}ree_a(ag,o)) \quad (3.8)\end{aligned}$$

This description is the same as the Description 3.4 with the additional $AReachO(at)$ as well as the inputs with an additional target agent at. Figure 3.13 shows an example of this action.



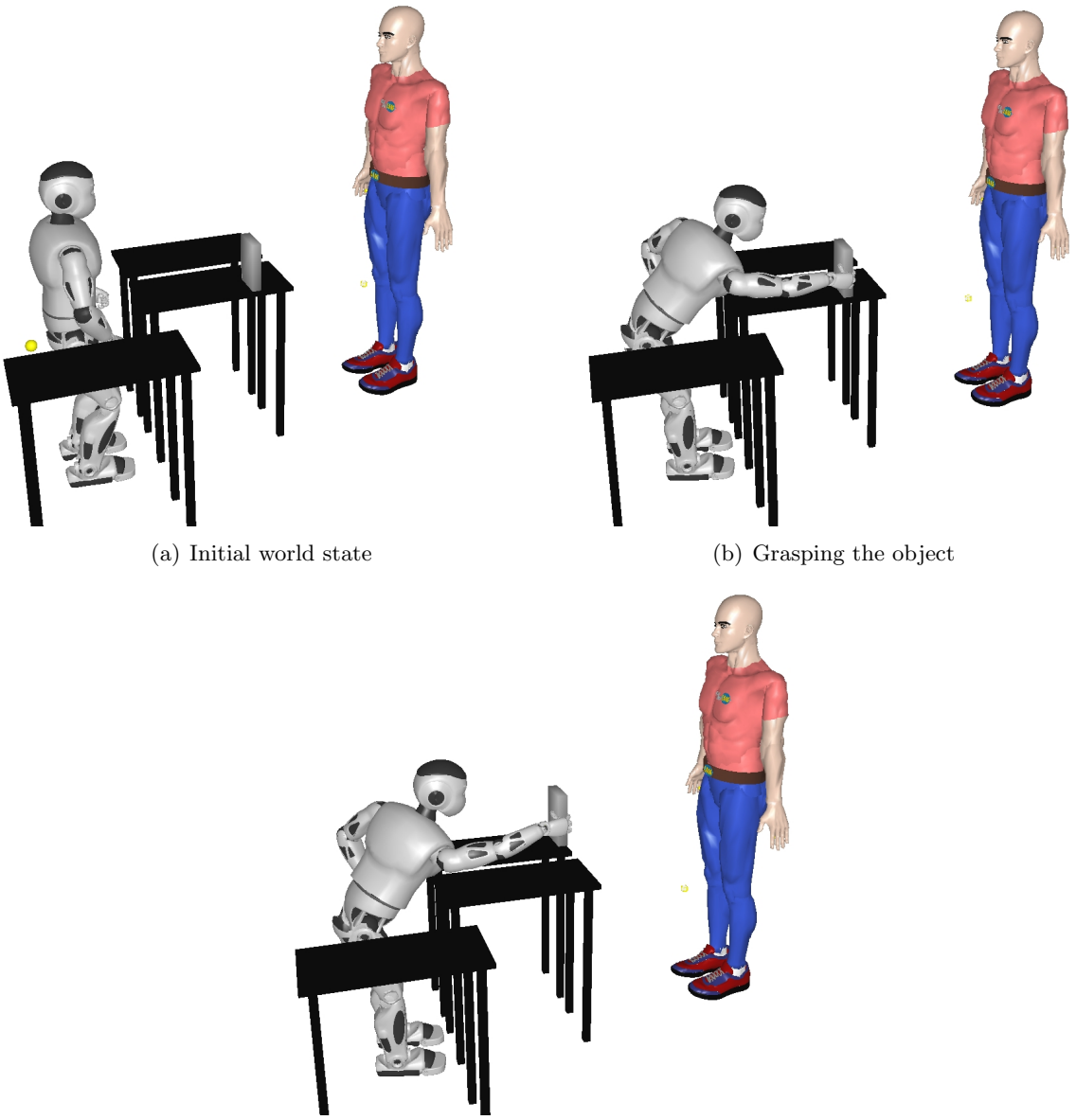
(a) Initial configuration (b) Object is reachable to the human

FIGURE 3.13: The initial and final world state is shown for a **PlaceReachable** action. Note that the object is reachable to the human in the second figure.

3.3.2.7 More possibilities

These actions are examples of what the framework offers, but do not show all its possibilities. For example, the **Pick** and **Place** actions are designed for a mobile manipulator using one arm. It is possible to extend it to multiple arms manipulation or to other kinds of robots such as humanoids robots (Figure 3.14 shows an example of a humanoid robot **ROMEO** (`http://projetromeo.com/en`) performing a **Pick**). In this particular context, one of the transformations we have made was switching from Fix_a to $UpperBody$[6] to account for the stability constraint (if a humanoid robot moves his arm only, and extends it too much, there is a risk of falling). The work concerning the humanoid robots was done in cooperation with Renaud Viry. In some cases, the inputs list can also be changed, without changing the description of the action: for example, for a **Place**, one can specify the object to place, an arm (retrieving the object thanks to the attachments) or both (checking if the attached object is the same as the one specified). The support, the stable configuration or even the exact position on the support can also be given as inputs to a **Place** action. When additional inputs are given to the actions, the framework replaces the search it would do when nothing is specified by the direct selection of the inputs.

[6]Reminder: *UpperBody* means that all entities are fixed apart from the agent DoFs which are not needed for the navigation.



(a) Initial world state

(b) Grasping the object

(c) Lifting the object

FIGURE 3.14: The different GSAS of the **Pick** action, performed by a humanoid robot. Note that all the upper body is moving when performing the action to keep the robot stability.

Some inputs can also be omitted, in which case the system needs to consider them as additional variable to look for: in the **Place**, omitting the support object will result on checking the nearest support object to the agent and use it.

3.3.3 The proposed Algorithms

This subsection presents different algorithms able to solve these actions. The general idea of these algorithms is to find the initial and final world states of the GSAS and then to compute the corresponding trajectories. In order to find these world states, the simplifications and information described in 3.3.1 are used in addition to the inverse kinematic (IK) computation. For example, when computing a **Pick**, a number of grasps are tested to find a feasible one (collision free). Then, the agent configuration is computed (IK) resulting on a final world state. For the **Place**, a number of placements on the table are

tested with various stable configurations, as shown in Figure 3.15. When one of them is collision free, the final configuration is computed based on the grasp used to attach the object to the end effector in the initial world state of the action.

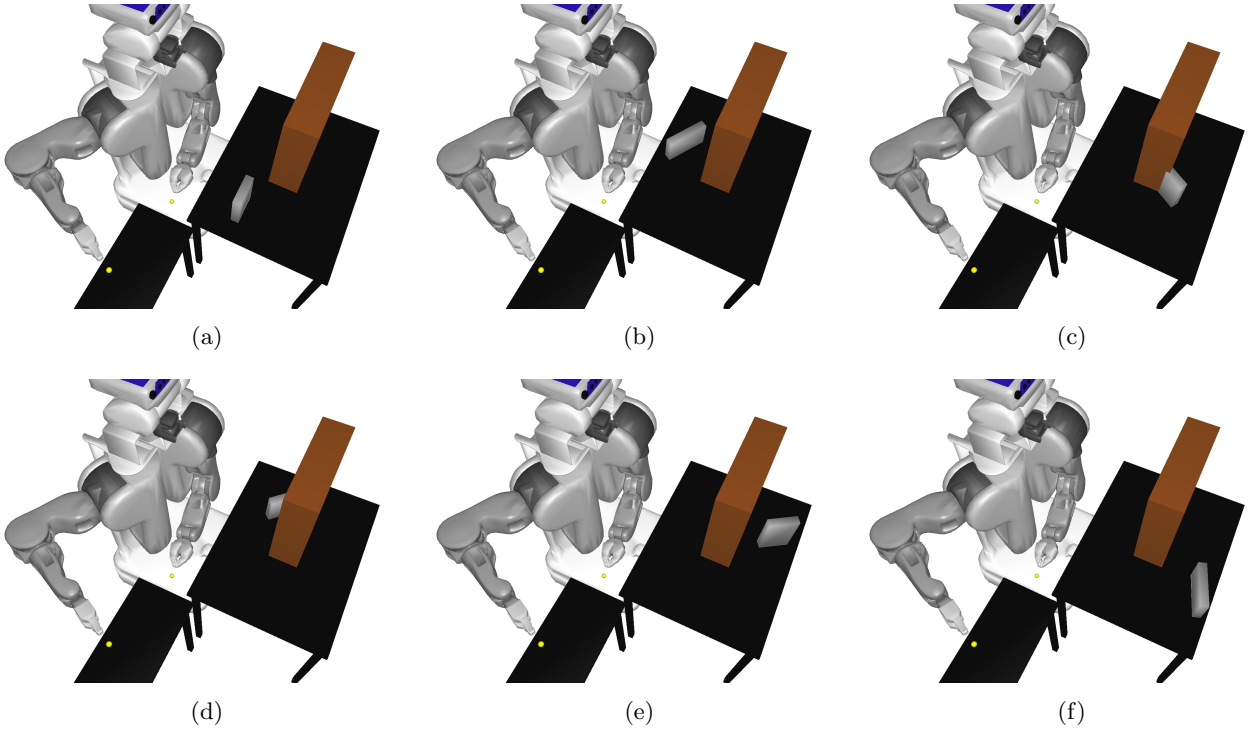


FIGURE 3.15: Different placements using various stable configuration of the object Grey book. Some of them (c and d) are in collision.

The most basic algorithm is to find a solution for each sub-action description separately, and then to combine them with the geometric causal links Subsection 3.3.3.1. The second algorithm consists on finding the final world state of every sub-action description (using inverse kinematics for example) and then computing the motion plans between the computed world states Subsection 3.3.3.2. The last algorithm consists on finding all the possible final world states and then choose between them the best one (based on human aware costs) and compute the motion plan for it: Subsection 3.3.3.3.

3.3.3.1 Separated sub-action descriptions algorithm

In this version, the Algorithm 3 processes the sub-action descriptions one by one until finding a solution for all of them. The first lines of the algorithm initialize the different variables needed later, such as $CSTD$ (Line 3) which is the current sub-action description, initialized to the first sub-action described in the action. Then, the algorithm enters a loop (from Line 8 to Line 30) where it first retrieves the performing agent[7] then, it checks if the conditions specified by $CSTD$ are respected in the current world state $currWS$ (Line 10). In the case where the conditions are not met, the algorithm sets the current variable to the one from the previous sub-action description (Algorithm 4) and if no previous sub-action description is found, it breaks out of the loop and fails to find a GAS for this action. When the conditions

[7]In the current framework, as specified above, there is only one performing agent per GSAS, selected in the description (Line 9)

Algorithm 3 Resolving an action based on the algorithm separated sub-action descriptions

1: **function** ComputeAction(aId,IN)
2: $descr \leftarrow$ GetActionDescription(aId) ▷ *retrieving the action description from* D_g
3: $CSTD \leftarrow$ GetFirstSubActionDescr*(descr)* ▷ *CSTD is the current sub-action description*
4: $currWS \leftarrow IN.ws_{init}$
5: $prevST \leftarrow \emptyset$
6: Clear*(STList)* ▷ *STList is the GAS*
7: $solLeft \leftarrow true$
8: ***while*** $CSTD \neq \emptyset$ ***do***
9: $a \leftarrow$ getAgent*(CSTD)* ▷ *This is the performing agent (needed in every GSAS)*
10: ***if*** *not* checkCond*(CSTD.conditions, currWS)* ***then***
11: *(CSTD, prevST, currWS)* $\leftarrow$ GoToPreviousST*(CSTD,descr,STList)*
12: *continue*
13: ***end if***
14: ***while*** $solLeft > 0$ *and* $traj = \emptyset$ ***do***
15: $(ws_{tmp}, solLeft) \leftarrow$ FindWS*(currWS, CSTD.searchSpaces* $\cap$ *CSTD.finalConstraints, IN)*
16: ***if*** $ws_{tmp} = \emptyset$ ***then***
17: *continue*
18: ***end if***
19: $traj \leftarrow$ ComputeTraj*(currWS,* ws_{tmp}*, CSTD.searchSpaces)*
20: ***end while***
21: ***if*** $traj = \emptyset$ ***then***
22: *(CSTD, prevST, currWS)* $\leftarrow$ GoToPreviousST*(CSTD,descr,STList)*
23: *continue*
24: ***end if***
25: $tmpST \leftarrow (a, traj, \emptyset)$
26: SetNextSubAction*(prevST, tmpST)*
27: AddToList*(STList, tmpST)*
28: $prevST \leftarrow tmpST$; $currWS \leftarrow ws_{tmp}$
29: $CSTD \leftarrow$ getNextSubActionDescr*(descr,CSTD)*
30: ***end while***
31: ***return*** *(STList,* ComputeCost*(STList),* CreateNewGasNum, CreateNewGasAltNum*)*
32: ***end function***

are met, the algorithm enters another loop (Line 14 to Line 20) where it searches the solution for this sub-action description: it tries to find the agents and objects configurations (Line 15 with $FindWS$ function which will be detailed later in this subsection) and by extension the GSAS final world state ws_{tmp}. Then, it computes a trajectory to link $currWS$ and ws_{tmp}. If this trajectory is found, the algorithm breaks out of the deepest loop, and continues by creating the new GSAS related to this trajectory (Line 25), then adding the geometric causal link between this GSAS and its predecessor in *descr* (Line 26) and finally store it in the GSAS list (Line 29). Before the end of the loop, the current sub-action description $CSTD$ is updated with the next one in *descr* (Line 29). The algorithm escapes the loop when there is no more sub-action descriptions and returns the GAS, represented by the GSAS list, its cost and the unique identifiers for the GAS and the alternative –those are computed at the end of the algorithm only if a solution was found, otherwise their value is *nan*–.

The $FindWS$ function in Line 15 of Algorithm 3 is a function computing, based on the search space and the final constraints of a sub-action description the final world state ws_{tmp} for said description. It

Algorithm 4 The procedure to apply before looping in Algorithm 3

```
1: function GoToPreviousST(CSTD,descr,STList)
2:     CSTD ← getPreviousSubActionDescr(descr,CSTD)
3:     RemoveLast(STList)
4:     prevST ← LastST(STList)
5:     currWS ← GetEndWS(prevST)
6:     return (CSTD, prevST, currWS)
7: end function
```

uses inverse kinematic coupled with the WMJ of the arms to find configurations corresponding to the description. This function, as it is action dependent usually needs to be implemented separately for each action. As said before in Subsection 3.2.5.2, the possible solutions available in the search space and the final constraints are not unique, but can be false solutions: even if a solution is found, the trajectory might not be feasible. $FindWS$ returns in addition to the final configurations it found (under the form of a world state) an integer $solLeft$ indicating the number of solution left in the space defined by the search space and the final constraints. Note that this exploration is stored and each time $FindWS$ is called for the same sub-action description, the already tested solutions are removed from this space. This is true even when an alternative is computed, the information stored in this function are retrieved from all the previous alternatives already computed. The number of possible solution in the search spaces can be infinite (continuous spaces), in which cases, we set a numerical limit for the possible number of solutions (e.g. 200 for the **Place** action).

3.3.3.2 Configurations first algorithm

The difference between this algorithm and the previous one is about when to compute the motion plan: in the previous one, it was computed for each sub-action description, in this one, the motion plan is left to the end, until all the world states are found.

These differences are shown in Algorithm 5. Its main loop (Line 3 to Line 31) consists of two main steps:

Computing the world state list (Line 7 to Line 18) this step consists on looping over the sequence of sub-action descriptions and for each one, checking the conditions (Line 8) and, if respected, computing the corresponding world state (Line 10). This world state is then used to check the pre-conditions for computing the next GSAS and to compute the next world state, and so on, until all the start and final world states of every GSAS is computed and stored in $WSList$ (Line 16)

Computing the motion plans (Line 20 to 30) As all the world states are computed and the corresponding conditions checked, the second step consists on linking them by computing the trajectories. For each start and final world state computed in the previous step and stored in $WSList$ the algorithm will compute the trajectory (Line 22) and will create the corresponding GSAS and add it to the result. As soon as one trajectory cannot be computed (Line 23), the solution cannot be found, and the algorithm loops back to the first step.

Algorithm 5 Resolving an action based on the algorithm with configuration first

```
1: function ComputeActionConfs(aId,IN)
2:     descr ← GetActionDescription(aId)
3:     while SolutionNotFound and solLeft > 0 do
4:         Clear(STList) ; Clear(WSList)
5:         CSTD ← GetFirstSubActionDescr(descr) ; currWS ← IN.ws_init
6:         AddToList(WSList, currWS)
7:         while CSTD ≠ ∅ do
8:             if checkCond(CSTD.conditions, currWS) then
9:                 (ws_tmp, solLeft) ←
10:                    FindWS(currWS, CSTD.searchSpaces ∩ CSTD.finalConstraints, IN)
11:            end if
12:            if not checkCond(CSTD.conditions, currWS) or ws_tmp = ∅ then
13:                (CSTD, prevST, currWS) ← GoToPreviousST(CSTD,descr,STList)
14:                continue
15:            end if
16:            AddToList(WSList, [currWS, ws_tmp, CSTD])
17:            CSTD ← getNextSubActionDescr(descr,CSTD) ; currWS ← ws_tmp
18:        end while
19:        prevST ← ∅
20:        for i ← 0; i < Size(WSList); i++ do
21:            a ← getAgent(WSList[i][2])
22:            traj ← ComputeTraj(WSList[i][0], WSList[i][1], WSList[i][2])
23:            if traj = ∅ then
24:                break
25:            end if
26:            tmpST ← (a, traj, ∅)
27:            SetNextSubAction(prevST, tmpST)
28:            AddToList(STList, tmpST)
29:            prevST ← tmpST
30:        end for
31:    end while
32:    return (STList, ComputeCost(STList), CreateNewGasNum, CreateNewGasAltNum)
33: end function
```

The algorithm breaks out of the main loop on two conditions: (1) a solution is found, (2) no more solutions are available in at least one $FindWS$ (10).

3.3.3.3 Integration of Human aware constraint

This third algorithm differs from the two previous ones by taking explicitly into account the human: in order to achieve this, the algorithm computes, as the previous one, the sequence of world states, then compute the trajectories. The difference lies in the following, the algorithm computes all the worlds state corresponding every available solutions in $FindWS$ and then, sorts them according to a human-aware costs, and, finally, computes the trajectories for the best one (if it fails, it computes the trajectories for the second best one and so on)

This algorithm can be divided into three main parts:

Algorithm 6 Resolving an action based on the algorithm computing costs

```
1:  function ComputeActionCosts(aId,IN)
2:      descr ← GetActionDescription(aId)
3:      while solLeft > 0 do
4:          Clear(WSList)
5:          CSTD ← GetFirstSubActionDescr(descr) ; currWS ← IN.ws_init
6:          AddToList(WSList, currWS)
7:          while CSTD ≠ ∅ and solLeft > 0 do
8:              if checkCond(CSTD.conditions, currWS) then
9:                  (ws_tmp, solLeft) ←
10:                     FindWS(currWS, CSTD.searchSpaces ∩ CSTD.finalConstraints, IN)
11:             end if
12:             if not checkCond(CSTD.conditions, currWS) or ws_tmp = ∅ then
13:                 (CSTD, prevST, currWS) ← GoToPreviousST(CSTD,descr,STList)
14:                 continue
15:             end if
16:             AddToList(WSList, [currWS, ws_tmp, CSTD])
17:             CSTD ← getNextSubActionDescr(descr,CSTD) ; currWS ← ws_tmp
18:         end while
19:         AddToList(GlobalWSList, WSList)
20:     end while
21:     Sortlist(GlobalWSList)
22:                 ▷ this sorting is done based on cost computing for configurations in world states
23:     for j ← 0; j < Size(GlobalWSList); j++ do
24:         CWSList ← GlobalWSList[j] ; Clear(STList) ; prevST ← ∅
25:         for i ← 0; i < Size(CWSList); i++ do
26:             a ← getAgent(CWSList[i][2])
27:             traj ← ComputeTraj(CWSList[i][0], CWSList[i][1], CWSList[i][2])
28:             if traj = ∅ then
29:                 break
30:             end if
31:             tmpST ← (a, traj, ∅)
32:             SetNextSubAction(prevST, tmpST)
33:             AddToList(STList, tmpST)
34:             prevST ← tmpST
35:         end for
36:         if AllTrajsAreFound then
37:             break;
38:         end if
39:     end for
40:     return (STList, ComputeCost(STList), CreateNewGasNum, CreateNewGasAltNum)
41: end function
```

Computing the world states (Line 3 to Line 20) In this part, the algorithm computes all the possible sequences of world states. The limit is the one fixed by $solLeft$ computed by $FindWS$ (Line 10).

Sorting the sequences of world states (Line 21) Once all the sequences of world states computed, the algorithm sort them according to a cost. This cost is computed based on various parameters related to the human safety and comfort depicted in the next subsection.

Computing the trajectories (Line 23 to Line 39) In this part, the algorithm tries to find the trajectories for the first sequence in the sorted list of possible sequences $GlobalWSList$, and if it fails, it tests the second best sequence, then the third and so on, until finding a solution and breaking the loop or testing all the sequences.

3.3.4 Additional implementations

In addition to the main algorithm. Some other implementation has been done:

3.3.4.1 Facts

The function GetFacts(ws) can be called upon a world state (ws) and computes the facts that hold in it. The available facts are as follows:

Is On first object is over the second object

Is In first object is inside the second object

Is Next To both objects are next to each other

Is bigger than first object bigger than the second one (**Is smaller** is also available)

Is reachable by object can be reached by the agent

Is visible by object is in the field of vision of the agent

This implementation is based on previous works on this domain, such as Warnier et al. (2012) and Sisbot et al. (2011).

3.3.4.2 Cost

The cost function used to sort the world state sequence list is relevant only when the performing agent is a robot and there are humans in the environment. It is linked as in Subsection 2.3.1.3 to three parameters: the *distance* (this part of the cost is inversely related to the smallest distance between the robot and every human in the environment) the *visibility* of the robot by the humans where we test if some part of the robot is not hidden to the humans, and the *musculoskeletal effort* (when needed) related to the euclidean distance between the initial and final configurations during a GSAS, and the potential energy in the final world state Marler et al. (2005). These three parameters enable us to compute the human-aware GSAS and by extension human-aware GAS.

3.3.4.3 Alternatives

The framework also proposes the possibility of calling the function FindAlternative($gasNum$) which retrieves, from the stored GASs, the corresponding one (with the same $gasNum$), the action aId and the inputs IN, and call again the search algorithm. As said before, the function FindWS stores the different failed and succeeded final world states it computed for each GSAS and proposes a new one each time it is called.

3.3.4.4 Additional Constraints

As the constraint can be directly added in the action definition (as presented in Subsection 3.2.5.4), they are actually solved by the algorithms as it is. They are included in the inputs IN of the action under the form of facts (as said before, facts define search spaces that can be used in an action description) in addition to their position in the action description. For example, when using a **PlaceReachable**(ag,ar,o,so) action, we can add as a constraint the fact: $\{o,$ **is visible by**, $ar,$ $true\}$ when finding the object placement. This constraint will force the algorithm to find only objects positions that are visible by the agent ar.

3.3.4.5 Geometric plan

The input world state in the search algorithms can be replaced by a reference to a previously computed GAS (the reference must contain both $gasNum$ and $gasAltNum$), in which case, ws_{init} can be retrieved from the corresponding GAS as it final world state, and a geometric causal link is created between the referenced GAS and the computed one (in this order). This is how the geometric plans are stored.

In order to compute geometric plans, we developed an algorithm able to compute them based on a list of actions and an initial world state. This algorithm is very simple: for each action in the list, it computes the GAS. If the computation succeeds, it computes the next action GAS based on the computed final world state, otherwise, it backtracks to the previous action it computed, finds an alternative for this action and proceeds to compute the failed action GAS again with the new world state obtained. If, during a backtrack, there is no more alternatives to the action, the algorithm backtracks one more step. It repeats this steps until it finds a geometric plan or until all the alternatives have been tried and have failed. Figure 3.16 shows a plan where the robot performs three successive **Pick** and **Place** on three objects. This plan was written under the form represented in Algorithm 7 as a set of actions to perform.

Algorithm 7 Resolving an action based on the algorithm computing costs

1: SetInitialWorldState
2: Pick(R, RED_CUBE)
3: Place(R, RED_CUBE)
4: Pick($R, GREY\_BOOK$)
5: Place($R, GREY\_BOOK$)
6: Pick($R, ORANGE\_BOX$)
7: Place($R, ORANGE\_BOX$)

3.3.5 Results and discussion

This subsection presents, through Table 3.1, the results obtained when running the second algorithm (Subsection 3.3.3.2) on the action **Pick**, **Place** and **PlaceReachable** as described in Subsection 3.3.2. These results have been evaluated in a scenario where a PR2 robot needs to **Pick** (or **Place**, or **PlaceReachable** to a human in the other side of the table) a green bottle, with one of its two 7 *DoFs* arms, on a table in front of him. Some initial world states are depicted in Figure 3.17 (the robot arm and the bottles initial configuration have been randomized, the figure shows only some examples of this initial

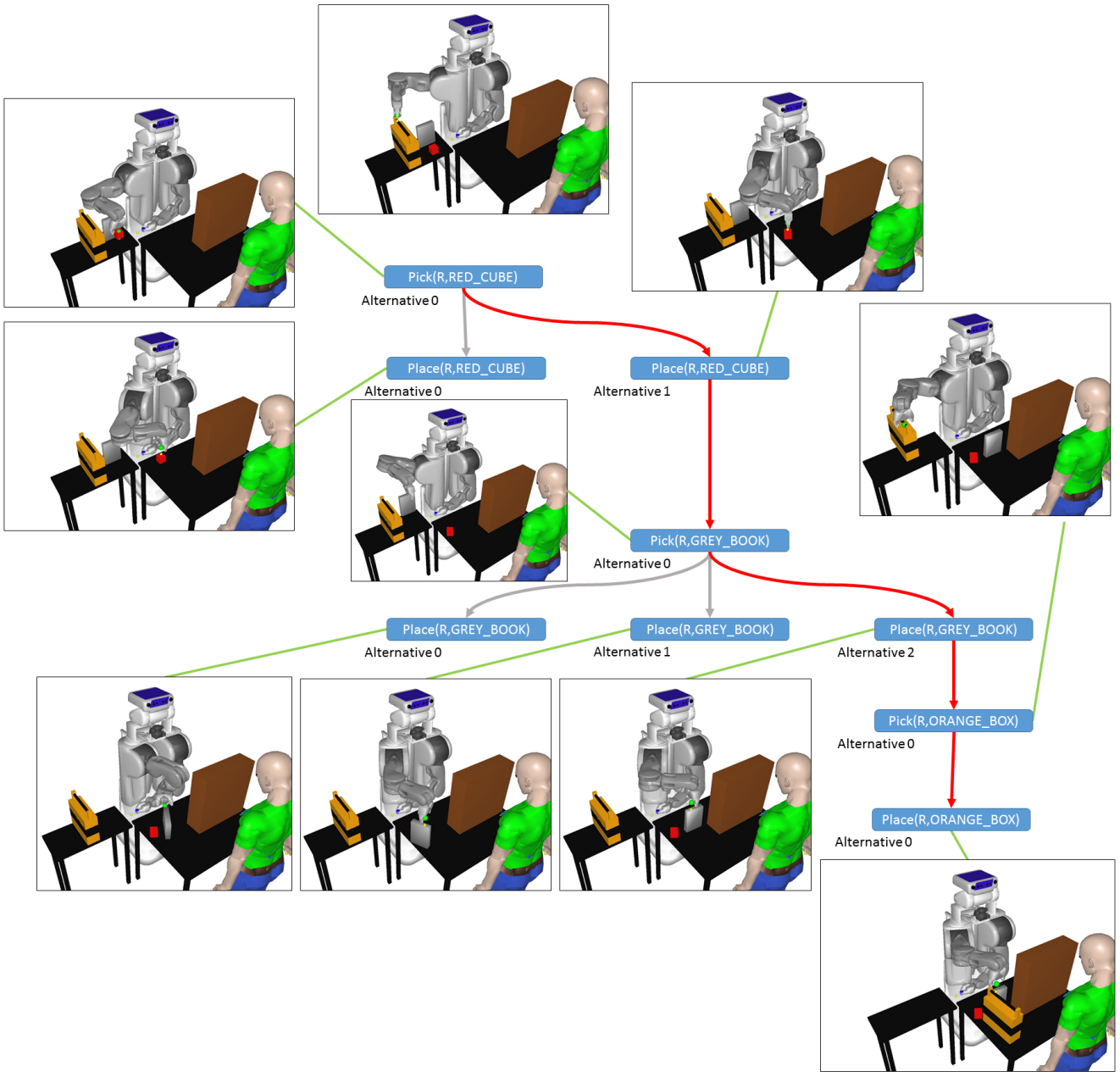
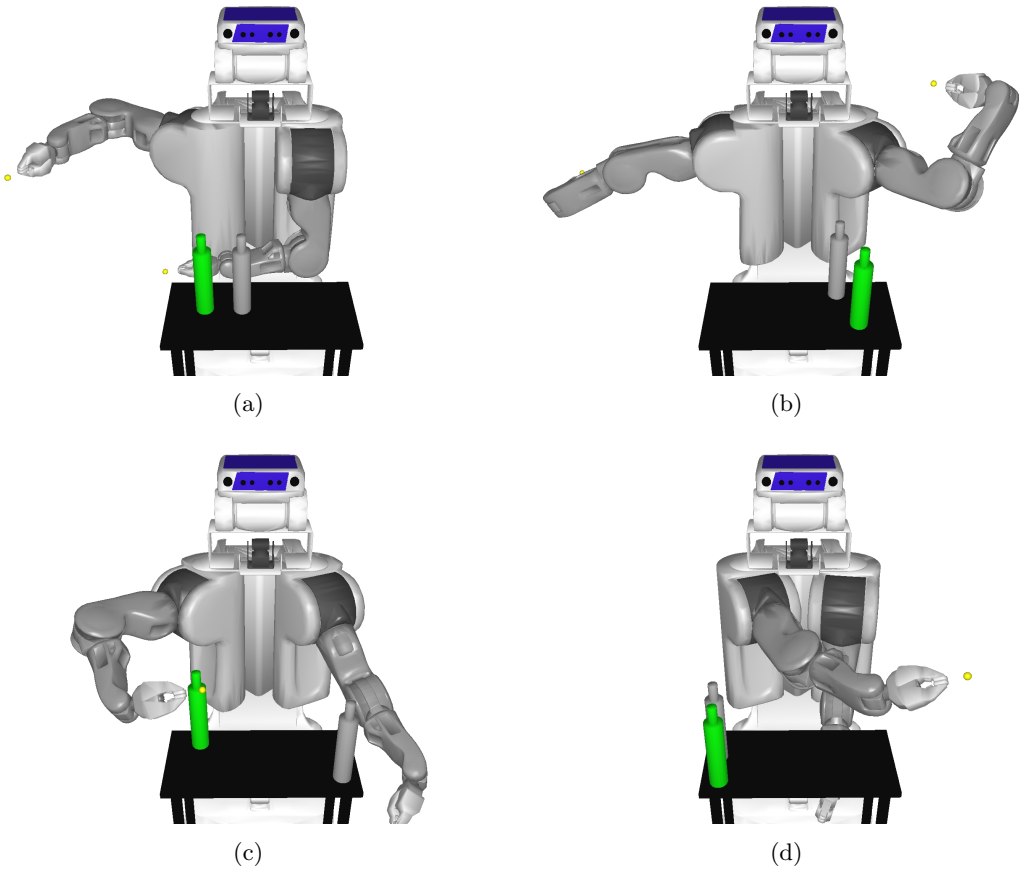


FIGURE 3.16: The different steps of a geometric plan, including the unused alternatives. In the plan, the robot performs three successive **Pick** and **Place** on three objects.

scenarios for a **Pick**). The motion planner used is a bidirectional balanced RRT LaValle (2006), and the test was run on an Intel® Xeon(R) CPU W3520 @ 2.67GHz × 4, on one core only.

The table is divided into two parts, the left one is the computation without motion plan and the right one with motion plan. This separation is meant to show the framework performance decoupled from the motion planner performances, as the one used can be exchanged with any off-the-shelf one. The first line shows each action mean computation time. The first interesting aspect is that the motion planning takes most of the computation time. Another interesting aspect is that **PlaceReachable** needs more computation time than the **Place** as it incorporates an additional constraint and takes more time to account for it. One can also notice that the number of solutions explored in the part without motion plan is significantly smaller than the one with the motion plan: the algorithm fails often to find trajectories, which is reflected in the two last parameters, the number of calls to the inverse kinematic solver is bigger

in the right side and the mean calls number to the motion planner is ≈ 2 with a variance ≈ 2. During the motion planning, most of the examples were very fast to compute (as shown by the low averages of the computation times) but in very few examples, the motion planning took a long time, making the variance and the standard deviation very high. One particular number to look for in the table is the variance of the number of solutions explored in the case of a **Place**: this high number can be explained by the number of variable the **Place** action needs to instantiate in order to find a solution.



(a) (b) (c) (d)

FIGURE 3.17: Various initial world state where the **Pick** has been evaluated

The framework can also handle multiple agents at the same time, performing different actions (in sequence) using different motion planners. Figure 3.18 shows an environment where a PR2 robot and an unmanned aerial vehicle (UAV) cooperate to bring an object to its final position: in the initial scenario, the UAV cannot **Pick** the bar as there is an object obstructing it path, the PR2 removes that object in order to let the UAV perform a **Pick**. One additional feature available in this framework is the possibility to use different motion planners and/or different type of motions depending on the tasks. Here, The PR2 uses classical linear motion primitives defined in its *Cspace* while the UAV uses kinodynamic motion primitives defined in its state space (i.e. integrating speed and acceleration) Boeuf et al. (2014).

In addition to that, it is also able to handle geometric plans Subsection 3.3.4.5, to compute actions alternatives Subsection 3.2.5.2 and facts Subsection 3.2.4, and to add these facts as constraints to any action Subsection 3.2.5.4.

This framework has been implemented on the PR2 robot using the architecture explained in Fiore et al. (2016). In this architecture, a supervision module communicates with a task planner (HATP) and

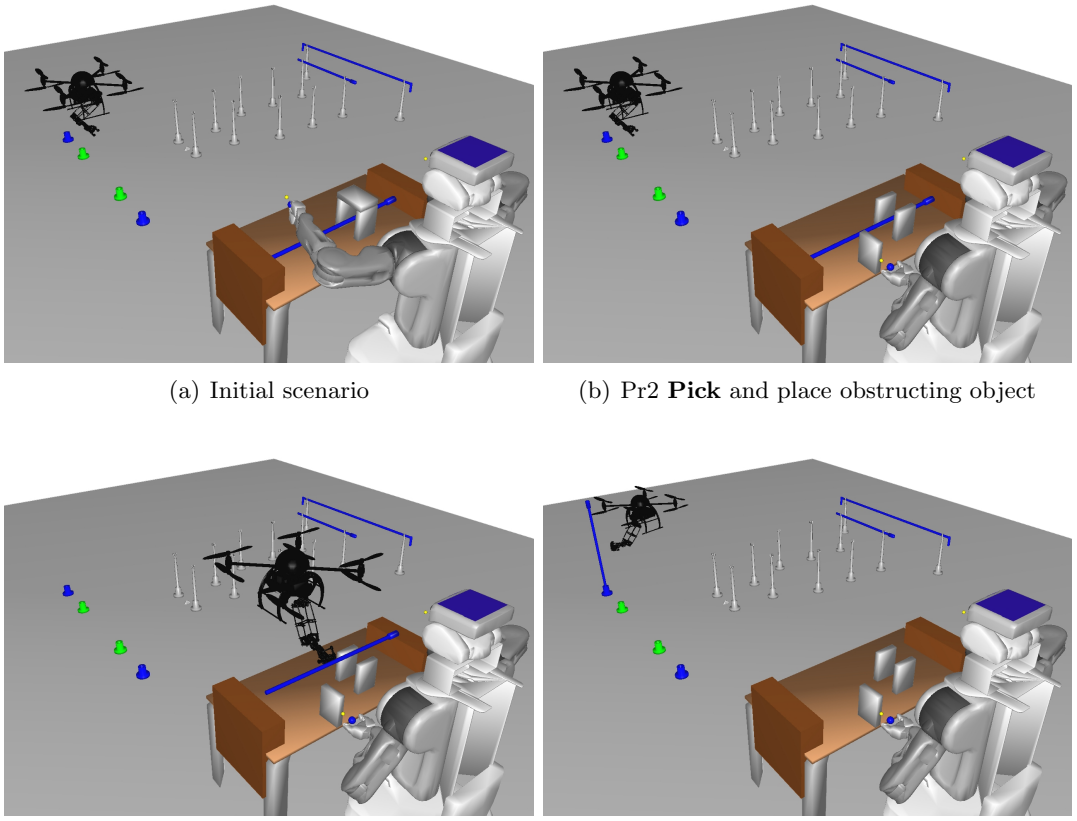
for one action	without motion plan			with motion plan		
Pick	average	variance	stand dev	average	variance	stand dev
Time	0.026	0.0001	0.0108	2.8553	17.1426	4.1403
Sol tests	2.525	3.6193	1.9024	8.2130	124.558	11.1606
Inverse kinematic	4.61	4.5379	2.1302	11.4556	128.899	11.3534
Motion plan	-	-	-	2.0532	2.1687	1.4726
Place						
Time	0.0201	0.0007	0.0270	2.7153	22.8922	4.7845
Sol tests	4.4522	19.7352	4.4424	18.5033	1166.78	34.1582
Inverse kinematic	4.9296	7.3216	2.7058	11.7219	217.101	14.7344
Motion plan	-	-	-	2.0463	2.4548	1.5667
PlaceReachable						
Time	0.0477	0.0016	0.0403	3.0798	47.1862	6.8692
Sol tests	5.5577	78.4879	8.8593	12.2692	236.735	15.3862
Inverse kinematic	5.1658	10.5303	3.2450	9.4359	57.8741	7.6075
Motion plan	-	-	-	1.8846	1.8969	1.3773

TABLE 3.1: Time means the computation time, Sol Tests means the number of solutions explored (by how much $solLeft$ decreased), Inverse kinematic means the number of inverse kinematic called, and Motion plan means the number of calls to the motion planner. These averages, variance and standard deviation (stand dev) are computed in over 150 successful action computation

obtains a plan (which is computed based on the information available in the knowledge base). This plan is then used to ask, step by step, the human-aware motion and manipulation planners module to compute the actions. In order to compute these actions, this module uses the world state provided by SPARK (the situation assessment module Sisbot et al. (2011)) and the framework we developed to compute a GAS. The trajectories computed in this GAS are then sent to the sensorimotor layer to execute them.

The robot can now **Pick**, **Place**, **PlaceReachable** and **Stack** with real objects. Figure 3.19 shows it during a session of **Pick** and **PlaceReachable** (the video is available here: `https://youtu.be/85KiC35qkPE`).

This framework enables us to solve a number of problem is still limited, for example, it cannot compute anything else then the given action (or sequence of action) which can be problematic in some cases. For instance, if the object G obstructs the access to the object o the agent needs to **Pick**, the framework will fail to find a solution as it would need first to remove or push the object G to access o. This limitation in particular is a choice: as we are going to see in the next chapter the choice of action is let to the task planner, which can take into account more parameters.



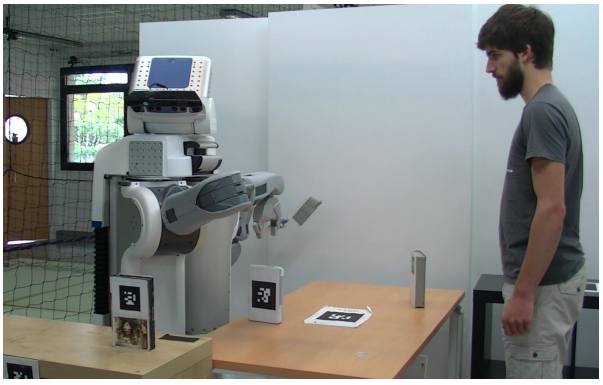
(a) Initial scenario

(b) Pr2 **Pick** and place obstructing object

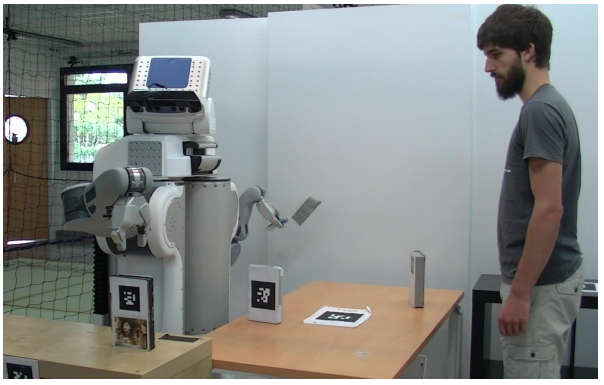
(c) UAV **Pick** the bar

(d) UAV places the bar

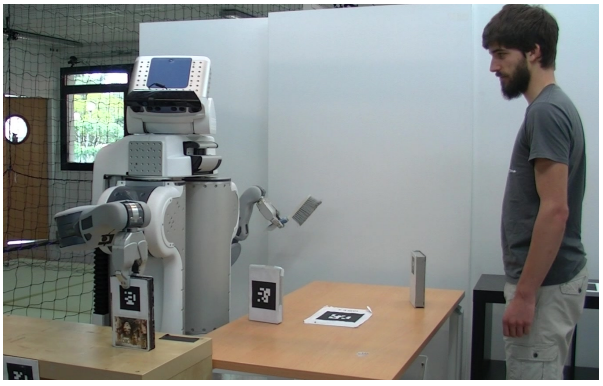
FIGURE 3.18: A geometric plan where the PR2 and the drone cooperate to bring the bar to its final location (it is planned in sequence)



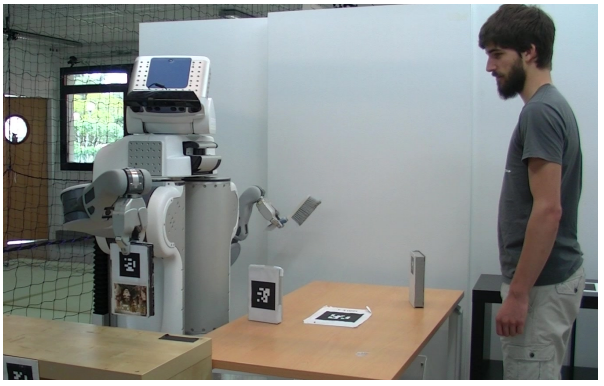
(a) Initial scenario



(b) **Pick** approach



(c) **Pick** grasp



(d) **Pick** extract

(e) **Place** approach

(f) **Place** release

(g) **Place** extract

FIGURE 3.19: An example of the framework running on the PR2 robot and executing a **Pick** and **Place** sequence.

3.4 Future work

This work can be enhanced in a number of ways, here are some of them:

Graph reuse each query to the motion planner ($ComputeTraj$ in the algorithms) is done in the framework as an RRT query (LaValle (1998)). One way to enhance the motion planning search is to reuse the RRT graph (or any other motion planning graph). Some approaches Ferguson et al. (2006), Phillips et al. (2013) use one graph and make it evolve with the time and the queries but it does not exactly match our needs as they usually replan in the same or nearly the same environments as the previous queries. In this framework, we might need to replan in the nearly (or exactly) the same scenarios, but it also might happen for two consecutive action computation to be in totally different environment. The idea of reusing graphs here is to first store each computed graph and link it to an action, and then, when computing the motion plan for a new action, try to find –based on information provided by the framework, such as the performing agent, the object manipulated, the action type and so on– the closest action to the new one, and use the stored graph(s) linked to this closest action.

Search space exploration The exploration in the search space is done randomly but can be enhanced to take the geometry into account: for example, when placing an object on a table, if an object placement fails because there is no inverse kinematic possible in that case then testing an object position close to this one will probably fail too. The idea is to explore the space in the most efficient way to cover it as fast as possible.

Combining actions based on the formalism, it is possible to concatenate actions and solve them with the same (or nearly) algorithms as the one presented in Subsection 3.3.3

Multi-robot & parallelism as presented before, the framework can handle only one robot moving at the time, although the formalism enables us to have multi-robot and parallelism, by using the geometric causal links. The framework can be extended to take this into account.

3.5 Contribution to the geometric planning and reasoning in a nutshell

This chapter contains two main contributions:

Geometric actions formalization The actions are described as a sequence of sub-action descriptions, related between each other with geometric causal links. The actions can have alternatives and can be linked between themselves to form a geometric plan.

A framework using this formalization The framework proposes different algorithms enabling the use of the previous formalization, while integrating a human-aware parameter. It shows also the results obtained by implementing one of these algorithms on a simulation and a real robot.

Chapter 4

Combined Task And Motion Planning

Contents

4.1 Introduction

In the previous chapter, a geometric planner able to plan fetch and carry actions for an autonomous robot was presented. The scope of this planner enables it to find solutions for simple problems such as to pick an object or to place it on a table. For more complex scenarios where the robot would need to perform a number of actions, which order is not known in advance, this planner is clearly not enough. On another hand, task planning methods enable a system to plan ahead for multiple actions and sequence of actions. The idea of this chapter is to combine these two planning methods into one by using both planners strengths: geometric planning strength lies in its capability of handling the continuous 3D space where humans and robots coexist, while taking into account their respective positions and preferences, the objects and the environment in general, but it uses very specialized algorithms to find solutions in specific cases. Task planning strength lies on its ability to handle large discrete domains with a great semantic variety and to find an optimal way to achieve a given goal in these domains but it lacks the specific knowledge to deal with the geometric description of the world.

The usual approach consisted on first computing a symbolic plan, and then, testing its feasibility at geometric level. This approach rises various issues, such as the ramification, the computation time or the completeness. The ramification problem occurs when the effects of an action are unknown or only partially known, which is the case when performing actions in the real world: for example, moving an object might result in a chain of actions (removing an object from a pile of objects might result on the whole pile to collapse) which was not expected. This ramification leads the geometric level computation to often fail, leading to a higher computation time (the ramification problem is more detailed in Subsection 4.3.5). It also affects the completeness, as some geometric choices might not be tested, before switching to a different symbolic plan. In the rest of this chapter, we will be referring to the geometric level as the geometric reasoner, or the geometric planner, as it reasons about the geometric space and is able to refine the symbolic plan into trajectories.

This work was held in cooperation with Lavindra De Silva and Raphaël Lallement, and was based on previous work, such as De Silva et al. (2013). Part of this work was published in De Silva et al. (2014) and the other part is published in Gharbi and Alami (2015). This chapter is structured as follows: Section 4.2 presents the actual state of the art in this field, Section 4.3 depicts a formalization of the problem alongside an algorithm to solve it, Section 4.4 shows different possible enhancements enabling a computation time speed up, Section 4.5 discusses our solutions and enhancements and proposes clues on the future possibilities, and finally, Section 4.6 summarizes the contributions of this chapter.

4.2 State of the art

Combining task and motion planning has been of great interest in a number of studies during the few last decades. One of the first works concerning this particular topic was done in aSyMov by Cambon et al. (2003) and extended later in Gravot et al. (2005), where the authors essentially propose a principled way to link the two planners thanks to a geometric level able to tackle the so-called "manipulation planning problem" Choset (1991) and that allows to explicitly take into account the topological changes occurring in the configuration space, when a robot grabs or releases an object. Asymov provided a well-founded translation of pick and place actions (and similar actions) into 'transit' and 'transfer' motion planning requests even in multi-object and multi-robot contexts.

In this section, we first identify the various names given to this problem and then we propose a categorisation of the work done in this field, using and extending the analysis presented by Erdem et al. (2016).

The problem was given various names and appellations, such as hybrid planning Guitton and Farges (2009b), CPMP Choi and Amir (2009) for combining planning and motion planning or TAMP Lozano-P??rez and Kaelbling (2014) for Task And Motion Planning and its variants: ITMP in Nedunuri et al. (2014) and Hauser and Latombe (2009) for Integrated TAMP, STAMP in Sucan and Kavraki (2012) for Simultaneous TAMP or CTAMP in Lagriffoul et al. (2014) for Combined TAMP. In this chapter, we will refer to this problem as the Symbolic Geometric Planning (SGP) problem.

Different approaches were proposed, Erdem et al. (2016) distinguish four different strategies among them: "(i) low-level checks are done for all possible cases in advance and then this information is used during plan generation, (ii) low-level checks are done exactly when they are needed during the search for a plan, (iii) first all plans are computed and then infeasible ones are filtered, and (iv) by means of replanning, after finding a plan, low-level checks identify whether the plan is infeasible or not; if it is infeasible, a new plan is computed considering the results of previous low-level checks". We propose another categorisation which keeps the same differences as these ones, but add some other categories and sub-categories:

Symbolic calls geometric reasoner In this case, the symbolic planner performs the plan search as usual, but verifies the feasibility of the plans produced at geometric level. This category groups (ii), (iii) and (iv) as sub-categories.

Geometric reasoner uses symbolic level In this case, the geometric planner knows all the possible solutions and uses the symbolic planner to determine which ones to explore and choose. It corresponds to (i).

Search in both levels simultaneously The search space is a compound space between the geometric and the symbolic spaces, the search is done in this compound space with no distinctions between the levels. This category does not exist in Erdem et al. (2016).

The next subsections, propose a state of the art categorisation following these points.

4.2.1 Symbolic calls geometric reasoner

This category can be divided into three sub-categories: compute all symbolic plans then computing the geometric plan (Subsection 4.2.1.1), find one symbolic plan then the geometric plan (Subsection 4.2.1.2), and compute the geometric plan during the symbolic plan search (Subsection 4.2.1.3).

4.2.1.1 Compute all symbolic plans then computing the geometric plan

In this sub-category the symbolic planner computes all the possible task plans, and knowing this, the geometric planner tries to find one plan among them that is geometrically feasible. Şucan and Kavraki (2011) present an approach where, provided a list of possible plans (which can be interleaved), they are able to find a feasible set of motions that fulfil the given symbolic goal. Sucan and Kavraki (2012) extend this approach by introducing uncertainties, and using a Markov Decision Process to guide the search. Lagriffoul (2013) propose a different way to solve the problem: they argue that part of the geometric reasoning may be endowed to the task planning level. They use a Hierarchical Task Network (HTN, explained in more details in Subsection 4.3.2), where they broke the geometric actions into basic primitives, to find all the possible plans, then, they use a geometric reasoner to test the geometric feasibility of the plan, using what is called geometric backtracking.

We have seen in the previous chapter that a geometric action might have multiple alternatives (Subsection 3.2.5.2). Geometric backtracking occurs when the geometric reasoner fails to find a solution for an action, and tries, without notifying the symbolic planner, different alternatives of previously succeeded actions, until it finds a feasible set of actions (including the current one) or it reaches a limit. This limit can be the maximum number of geometric alternatives for a specific action or the branching factor which is the maximum number of alternatives allowed by the symbolic planner.

4.2.1.2 Find one symbolic plan then the geometric plan

Approaches in this sub-category are the closest to the classical approach, as they first compute the whole symbolic plan, and then test it at geometric level. The difference is that here, the geometric level is taken into account directly by the symbolic planner, and they interact together to find a feasible plan. The idea is to prune out impossible symbolic plans right from the start of the planning process.

Lozano-P??rez and Kaelbling (2014) build a plan skeleton based on task planning, containing geometrical constraints and formulate the problem as a *Constraint Satisfaction Problem* then they use a general solver to test the plan geometrical feasibility. In Srivastava et al. (2013b) case, once they found a task plan, they try to plan the geometric actions, and if they fail, an error is returned to update the symbolic state, and a new task plan is created. Caldiran et al. (2009a) and Caldiran et al. (2009b) present a different approach where they use an action description language $\mathcal{C}+$ to provide a robot with high-level reasoning able to find complete symbolic plan, and, based on this plan, they extract the collision free trajectories. In case of problem –collisions– they report it to the reasoner, and a new plan is computed where they try to extract trajectories again. They provide an example of two robots moving object in a 2D grid, and propose another example in Haspalamutgil et al. (2010): the tower of Hanoi

problem. Erdem et al. (2011) keeps nearly the same framework but use in place of the action description language, a *Causal Reasoner* to find the symbolic plans, and if the geometric resolution fails, it changes the planning problem, by adding constraints to the causal reasoner in order to take the cause of failure into account. As before, they used a two robot moving object as an example and Havur et al. (2013) add another example: the tower of Hanoi.

In this sub-category, some approaches are also based on geometric backtracking. Srivastava et al. (2014) and Srivastava et al. (2013a) present an interface between a task planner and a geometric planner where, once a symbolic plan is computed, they use geometric backtracking to test it feasibility. If no collision free trajectory is found, the geometric reasoner informs the symbolic planner about the infeasible action and the reason for its failure, information used by the task planner to change the part of the plan coming after the last feasible action. Lagriffoul et al. (2012) also use geometric backtracking on a complete plan, but they introduce the notion of constraints on interval bounds to speed up the search. Once they get the symbolic plan, they use it to define constraints on the robot configurations, at each step, starting from the last step. These constraints reduce the search space of each action making the number of geometric backtracks drops. Lagriffoul et al. (2014) extend this approach by adding constraints concerning more degree of freedom at once and expose a study of the time complexity of their algorithm. Dearden and Burbridge (2013) also compute the complete symbolic plan before computing the geometry, then they map the symbolic states with geometric ones, starting from the final states, and finally they try to find trajectories between the states. If a trajectory does not exist, the geometric backtrack is triggered in order to change the symbolic geometric state mapping. This mapping is learnt through a set of training data in the form of geometric states labelled with the predicates which are true in them.

4.2.1.3 Compute the geometric plan during the symbolic plan search

This sub-category contains approaches where the geometric reasoner is called each time a feasibility test is needed, during the symbolic plan search. The idea is to not explore infeasible symbolic plans if we already test their infeasibility at geometric level.

Dornhege et al. (2009) introduce the notion of semantic attachments, in the context of SGP, which are external procedures able either to evaluate if a condition is true or false, or compute the numerical value of a state variable. The condition validation is used as action predicate, and compute if a motion plan is feasible or not. The state variable computation is used to retrieve the new world state from the geometry. Dornhege et al. (2010) present a soundness and completeness study on this approach in addition to multiple examples and relevant results using this method. Dornhege et al. (2012) introduce the possibility of using heuristics during the search by enabling the semantic attachments to only return an evaluation of their computation and propose the use of different off-the-shelf task planner able to use these heuristics such as Fast Forward Hoffmann and Nebel (2001) or Temporal Fast Downward Eyerich et al. (2012). Hertle et al. (2012) propose a new planning language: Object-oriented Planning Language, where the task description is written in an easy object like form (such as c++ or java) and which can handle semantic attachments. The latest extension added by Dornhege et al. (2013) to this work consists on caching the external procedures return values and states, in order to use them later, in case of the

same or similar request to the external procedure is done. They also use relaxed external procedures as heuristics to prune out part of the infeasible solutions before computing the complete external procedure.

Other approaches are also based on calls to external procedures, such as Ferrer-mestres et al. (2015) who worked on adapting a first order planning language named Functional STRIPS by adding requests to external function (geometric tests for feasibility) as a component of a symbolic action. Guitton and Farges (2009a) also modify the symbolic action description, but they add geometric constraints to the action precondition, which are passed to the geometric reasoner who use them to compute a new geometric state and then find a path from the previous geometric state to the new computed one. Gaschler et al. (2013a) and Gaschler et al. (2013b) also uses external call at symbolic level combined with a detailed symbolic state of the world –they are able to represent the state of a variable (known, unknown, incomplete or will be known at run time)– to compute feasible plans. Gaschler et al. (2015) extend this approach by adding specific geometric predicates to their actions, enabling a search speed enhancement. Kaelbling and Lozano-Pérez (2011) use an aggressively hierarchical planner which embed in the action description primitives to compute and execute the action. They use fluents to transform the geometric state to symbolic states and assess if the preconditions of the next actions holds or not. Kaelbling and Lozano-Perez (2011) extend this approach by adding uncertainties: the planning is done in a hierarchical belief-space: the world is not known but is observable, when performing an action, a previously unknown parameter might become known or partially known (looking inside a cupboard might end with knowing the position of a certain object or knowing that said object is not in the cupboard). Kaelbling and Lozano-Pérez (2013) extend even more the approach by adding domain models used as heuristics to guide and speed up the search in the robot's belief-space.

Wolfe et al. (2010) present an approach where they use high level action primitives as actions in a Hierarchical Task Network (HTN) planner. These actions can be refined to primitives such as navigate to somewhere, move arm to grasp or close gripper. Shivashankar et al. (2014) propose a formalism which is goal directed and based on HTN, and they link it with the geometric reasoning. They achieve this by computing, at each step of the symbolic search, a symbolic state used to find a corresponding geometric state. Then, they compute the trajectories linking these geometric states. In case of failure, a new geometric state is produced, until the branching factor is reached (maximum number of allowed geometric states corresponding to the same symbolic state) in which case, the planner backtracks to the previous action. Once a trajectory is found, they compute its cost in order to let it aside if its quality is not satisfying compared to the rest of the plan (it is not removed, the computation of the corresponding plan is just postponed).

Some researchers also propose approaches including geometric backtracking. Alili et al. (2009) proposes a combination of an HTN planner with a geometric reasoner, where the symbolic action and description embed a call to a geometric refinement of the action. Before the geometric reasoner informs the symbolic planner about the infeasibility of the action, it triggers a geometric backtracking. They also keep the symbolic state updated by computing facts after each geometric computation and handing them to the symbolic level. De Silva et al. (2013) extend this work by adding the ground literal protection: in the previous work, geometric backtracking didn't check if the new created plan respects the symbolic precondition set before each action. In this one the ground literals (which are facts passed to the symbolic

level to assess some preconditions) are cached by the system for each task and protected when an action alternative is computed. Karlsson et al. (2012) depict a solution where, by using geometric backtracking with external calls and geometric predicates (predicates computed at geometric level and used at the symbolic level), they find feasible plans for a two-arm humanoid robot. Bidot et al. (2015) extend this approach, by first proposing a formal definition of the problem and then by adding geometric constraints able to guide the geometric backtracking in order to stress out the most interesting/constrained actions.

4.2.2 Geometric reasoner uses symbolic level

This category corresponds to the approaches where a geometric search is held and uses the symbolic level to guide this search in order to reach the desired goal. Zickler and Veloso (2009) for example perform their search in the geometric state space of the agents, where they compute, for each state, symbolic information enabling the search to be guided toward the goal. Choi and Amir (2009) propose to explore the model of the world with a motion planner algorithm (such as RRT) and use the generated graph to automatically create feasible actions: if the motion generated by an edge (or a group of edges) of the graph, changes the state of an object, then it is considered as an action. Then a symbolic planner is used to find a plan using these actions. Nedunuri et al. (2014) base their work on an extended version of a manipulation graph LaValle (2006) which contains information about the robot base placement and arm placement to manipulate objects. They use a given plan outline to guide the search through the possible sequence of actions available in the graph. Garrett et al. (2014a) and Garrett et al. (2014b) also use a graph capturing the possible manipulation actions in the environment and use a Fast Forward (Hoffmann and Nebel (2001)) task planner to find the best plan based on these actions. The graph is constructed by sampling the objects positions and computing one or multiple robots inverse kinematic for each one and then linking this configuration between themselves through trajectories.

Plaku and Hager (2010) have a different approach where they sample the continuous space guided by the symbolic level, until reaching a state which satisfies the goal (this state is given to the geometric planner). In order to achieve this, they create a tree, and at each iteration of a loop, expand it by choosing the more relevant node (based on a utility function) and explore the space from there. Plaku (2012b) and Plaku (2012a) extend this approach by replacing the symbolic planner by an automata described by a Linear Temporal logic (LTL).

4.2.3 Search in both levels simultaneously

In this last category, the search for plans is done at the same time at geometric and symbolic levels. Hauser and Latombe (2009) consider that the robots can move inside a feasible space only, and can switch between "feasible spaces" through transitions: inside a "feasible space" the robot cannot change his contacts with the outside world (if he is moving an object for example) but can do it through a "transition space" (for example placing the object on a table). They create a Probabilistic Road Map (PRM) (Kavraki et al. (1996)) in each "feasible space" and aggregate them through milestones in the "transition spaces", and during the search for a solution (specified as a goal state) they are able to begin the search in a direction, stop it, and postpone it (in case it is taking too long, to explore other

directions). Hauser (2010) extend this work by creating a symbolic language able to make requests to their previous system and by doing so, obtain a larger range of possible actions. This last paper enters in the sub-category of Subsection 4.2.1.3, as it makes requests to the geometric planner during the symbolic search. Ficuciello et al. (2013) and Barry et al. (2013) use a similar method (as Hauser and Latombe (2009)) but using a RRT algorithm in place of the PRM.

Cambon et al. (2004) and Cambon et al. (2009) describe the aSyMov planner presented in the beginning of this section and which is also part of this category.

4.2.4 Synthesis, discussion, and contributions

Table 4.2 shows the different works cited in this section organised by author, with some characteristics stressed out. Interestingly, Lagriffoul et al. (2013) argue that, as it is the case some of these approaches, completely combining task and motion planning might not be efficient in every case: they are efficient to solve geometrically complex problems but their performance might be less interesting than the classical approach when the problem is geometrically simple.

Our contribution with their specificities are noted at the end of Table 4.2 and it belongs to the sub category depicted in Subsection 4.2.1.3.

1 - Symbolic calls geometric reasoner
2 - Geometric reasoner uses symbolic level
3 - Search in both levels simultaneously
4 - Find one symbolic plan then the geometric plan
5 - Compute the geometric plan during the symbolic plan search
6 - Compute all symbolic plans then computing the geometric plan
7 - Call to external procedures
8 - Compute geometric states from symbolic states
9 - Create symbolic knowledge from geometry
10 - Geometric alternatives
11 - Geometric backtracking
12 - Uses constraints
13 - Account for uncertainties
14 - Using a graph covering the entire space

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Şucan and Kavraki (2011)	X					X								
Sucan and Kavraki (2012)	X					X							X	
Nedunuri et al. (2014)		X		X										X
Lagriffoul et al. (2012)	X			X						X	X	X		
Karlsson et al. (2012)	X				X				X	X	X			
Lagriffoul (2013)	X					X								
Lagriffoul et al. (2014)	X			X						X	X	X		
Bidot et al. (2015)	X				X		X		X	X	X	X		
Kaelbling and Lozano-Pérez (2011)	X				X				X					
Kaelbling and Lozano-Perez (2011)	X				X				X				X	
Kaelbling and Lozano-Pérez (2013)	X				X				X				X	
Ficuciello et al. (2013)			X					X		X				
Barry et al. (2013)			X					X		X				
Lozano-P??rez and Kaelbling (2014)	X			X								X		
Garrett et al. (2014a)		X							X	X				X
Garrett et al. (2014b)		X							X	X				X
Srivastava et al. (2013a)	X			X				X	X	X	X			
Srivastava et al. (2013b)	X			X					X					
Srivastava et al. (2014)	X			X				X	X	X	X			
Caldiran et al. (2009a)	X			X					X					
Caldiran et al. (2009b)	X			X					X					
Haspalamutgil et al. (2010)	X			X					X					
Erdem et al. (2011)	X			X					X			X		

Havur et al. (2013)	X			X					X			X		
Gaschler et al. (2013a)	X				X		X						X	
Gaschler et al. (2013b)	X				X		X						X	
Gaschler et al. (2015)	X				X		X						X	
Dornhege et al. (2009)	X				X		X		X					
Eyerich et al. (2009)	X				X		X		X					
Dornhege et al. (2010)	X				X		X		X					
Dornhege et al. (2012)	X				X		X		X					
Dornhege et al. (2013)	X				X		X		X					
Plaku and Hager (2010)		X							X	X		X		X
Plaku (2012b)		X							X	X		X		X
Plaku (2012a)		X							X	X		X		X
Guitton and Farges (2009a)	X				X				X			X		
Zickler and Veloso (2009)		X							X	X				
Choi and Amir (2009)		X							X	X				X
Wolfe et al. (2010)	X				X		X							
Shivashankar et al. (2014)	X				X			X						
Dearden and Burbridge (2013)	X			X				X		X	X			
Ferrer-mestres et al. (2015)	X				X		X					X		
Hauser and Latombe (2009)			X					X		X				X
Hauser (2010)	X				X		X			X				
Cambon et al. (2003)			X					X	X					X
Cambon et al. (2004)			X					X	X					X
Gravot et al. (2005)			X					X	X					X
Cambon et al. (2009)			X					X	X					X
Alili et al. (2009)	X				X		X		X	X	X			
De Silva et al. (2013)	X				X		X		X	X	X			
Silva et al. (2013)	X				X		X			X				
De Silva et al. (2014)	X				X		X			X				
Gharbi and Alami (2015)	X				X		X			X		X		

TABLE 4.2: a synthetic reorganisation of the state of the art, coupled with some characteristics, where works are regrouped by authors. Our recent contributions are in the last rows of the table.

4.2.4.1 Discussion

Each one of the different approaches described in this section has some advantages and disadvantages. In this subsection, we tried to find some of them, but the list is not exhaustive and it is based on the analyses of these approaches.

The advantages of computing all the symbolic plans first and then to compute the geometric plan (Subsection 4.2.1.1) are the possibility to rule out the plan parts which will not achieve a complete plan and the possibility to choose among all the plans the "best" one. When computing the symbolic plans, the algorithms might find the beginning of a plan which has no chance to achieve a complete plan because of a not respected symbolic pre-condition. In this approach, we do not refine geometrically this plan part which might take some time. In order to choose the "best" plan, heuristics might be used (such as the shortest plan). One disadvantage of using this method is that we might loose time computing all the plans and choosing among them.

One advantage of first finding one symbolic plan and then refine it, (Subsection 4.2.1.2) is, as for the previous approach, the ability to rule out the plan parts which will not achieve a complete plan. This approach also contains geometric backtracking, which has the advantage of being easily enhanced and tuned for the domains used. The disadvantage of using this approach is the inability to change the symbolic choices once they are taken: the algorithm needs to exhaust all the geometric possibilities before changing the symbolic plan (and it can be time consuming as the spaces can be big). Concerning the geometric backtrack, the algorithm needs to take into account the pre-conditions, which might introduce some undesirable latencies.

The advantage of computing the geometric plan during the symbolic search (Subsection 4.2.1.3) is the ability to change the symbolic choices based on geometric problems. Also, this approach does not need to handle explicitly the action pre-conditions. The disadvantage of this approach is the possibility to compute some geometric actions (with their motion plans) that might not be needed because the plan part is unfeasible due to a symbolic pre-condition not holding.

The advantage of having a geometric reasoner that uses the symbolic level (Subsection 4.2.2) is the possibility to use the motion planning state of the art algorithms (which are now very efficient) to solve the problems. The disadvantage of such an approach is the tight link to the domains: it is very domain dependent.

The advantages of the last approach where the search is held at both levels at the same time (Subsection 4.2.3) are the completeness of the approach and the ability to optimize the plans depending on the needs. The disadvantages are the huge search space generated by the combination of both spaces and the difficulty to implement such approaches in a generic way.

4.3 Formalism and algorithms

The SGP problem consist in computing a valid symbolic plan while ensuring its feasibility at the geometric level. Assessing the plan validity implies to take into account the action direct and indirect effects (the indirect effects that the geometry can compute).

In order to tackle this problem, we propose a method combining an extended version of a Hierarchical Task Network (HTN) planner and the GRP framework presented in the previous chapter. In this section we first present a brief description of the HTN planner, then we present our extended version, named Hierarchical Agent-based Task Planner (HATP) and finally, we present the Symbolic and geometric action planner (SGAP) that combines both levels of planning.

4.3.1 HTN Planning

An HTN planner (as presented in Ghallab et al. (2004)) is a task planner able to transform a domain, an initial situation and a goal (provided under the form of a task[1] to achieve) into a series of tasks bringing the system from this initial situation to the requested goal.

The planning process consists in two different activities: (1) decomposing the goal task down to operator level, (2) binding the tasks parameters left free (e.g. choose actors). The planning process iteratively builds a tree by decomposing the tasks, starting with the goal task, following the rules: if the task is a method, a decomposition is explored and the other possible decompositions are added as backtracking points. If the task is an operator its preconditions are tested, then the instantiated operator is added to the current plan, otherwise the planner goes back to the last backtracking point and tries another decomposition. When an instantiated operator is added to the current plan, its effects are applied to the current state to obtain the next state and its cost is added to the current plan cost. If a decomposition of the goal allows to reach down to the operator level, then a plan is found. If one wants to keep the completeness or find the best plan, it is possible to explore all decompositions. In the case where all the decompositions are explored but no plan was found, the planning stops with a failure, the goal cannot be achieved from the initial state.

This is a very succinct explanation of the algorithm. In the next section we present HATP, which is an implementation of the HTN algorithm and in Subsection 4.3.2.6 we highlight the differences between this implementation and the classical algorithm.

4.3.2 Hierarchical Agent-based Task Planner

Hierarchical Agent-based Task Planner or HATP is an implementation of the HTN algorithm which integrate some enhancements, as presented in Lallement et al. (2014). HATP is based on SHOP Nau et al. (1999) and is designed to be used by roboticist: the domain representation is user-friendly and the agents (humans and robots) are considered as "first order" entities in the language. Also, HATP uses a total order representation: all the actions in the current (partial) plan are ordered enabling it to

[1] as presented later, a task can be either a method or an operator.

compute, at any given time, the complete context of the world. HATP is based on a number of basic notions, some of them contained in the following list and the rest presented later in this subsection.

Predicates: Boolean-valued function which capture the symbolic state of a parameter in the world, such as object X *is reachable by* agent A. It is written under the form: X.isReachable = A.

Context: A context is a set of predicates that capture the whole state of the world at a specific moment. It is under the closed-world assumption (if the predicates does not appear in the list, it is supposed to be false).

Entity An element from the environment, for instance, a robot, a table or a book.

Entity description: Contains the entity *id*, and the predicates that can be applied to them. For example a manipulable object X accepts the predicate: *isReachable*, *isOn*, *isIn* and so on.

Operators: [2] An operator is a parametrized executable primitive. It is represented by a 2-uplet $\langle pre, \text{eff} \rangle$ where *pre* is the list of preconditions and eff the list of effects. Both of them are a set of predicates, the preconditions are the predicates that should hold in the context where the operators needs to be applied, and applying an operator means instantiating it and adding its effects to the context it was applied to. It can take parameters such as an entity, or a set of entities as inputs. A cost function can be linked to an operator, enabling the planner to assess its quality.

Methods: A method can also be applied to a context, but cannot be directly executed, it needs to be "decomposed" into other methods and/or operators. Decomposing a method means trying to apply its components following the order it specifies.

Tasks: A task is a denomination that refers to either an operator or a method.

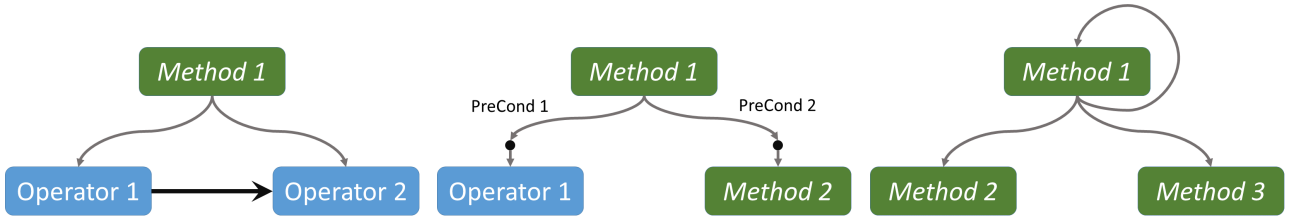
A method can be decomposed into other tasks (methods and operators) combined through one of three different links, depicted in Figure 4.1. The first link Figure 4.1(a), is where all the tasks composing the method needs to be applied, in the specified order (in the figure the order is given by the thick arrow) we call this case the *causal link*. The second link, Figure 4.1(b), is the *Exclusive disjunction*, which mean that one and only one task of the decomposition can to be applied. The last link, Figure 4.1(c), is *Asynchronous*, where all the tasks needs to be applied but no connexion exists between them.

It is possible to define a number of operators, here is some of them which are going to be used in the rest of this chapter:

Pick(A,O)[3]: A is an agent –omitted when obvious– and O an object (those holds the same meaning for all the operators). The preconditions are $A.hasInHand = NULL$ and $O.isReachable = A$. The effect is $A.hasInHand = O$.

[2] In order to avoid confusions, operator will be used for the symbolic actions (in an HTN planner, both action and operator can usually be used), while the geometric actions will keep the name actions.

[3] In order to differentiate between task level symbols and geometric level symbols, task level symbols will be written in *italic* in the rest of this thesis



(a) *Causal link* (b) *Exclusive disjunction* (c) *Asynchronous* and recursive

FIGURE 4.1: A method decomposition can be only operators (a), only methods (c) or a mix between them (b) and it can also be recursive (c). In (a) *Operator 2* cannot be applied unless *Operator 1* is applied successfully. In (b) to decompose *Method 1*, either we apply *Operator 1* if *PreCond 1* holds (precondition for this decomposition) or we apply *Method 2* if *PreCond 2* holds, but not both at the same time even if the two preconditions holds at the same time. In (c) the decomposition is asynchronous: all the methods in the decomposition are applied, but are not ordered.

Place(A,O,S): S is the support. The preconditions are $S.isReachable = A$, $A.hasInHand = O$. The effects are $A.hasInHand = NULL$ and $O.isOn = S$.

PlaceR(A,O,S,AT): the same as *place(A,O,S)* with the additional effect $O.isReachable = AT$, the target agent (*placeR* holds for "Place Reachable").

Navigate(A,E): where E is an entity, there are no preconditions, and the effect is $A.isNextTo = E$.

PaintGreen(A,O): The precondition is $A.isNextTo = O$, and the effect is $O.isColoured = Green$.

Based on those elements, we are now able to define the domain and the problem.

4.3.2.1 HATP Domain

A HATP domain $\mathcal{D}$ can be defined by the 3-uplet $\langle \mathcal{M}, \mathcal{O}p, \mathcal{E} \rangle$, where:

- $\mathcal{M}$ is all the available methods in the domain with their decomposition,
- $\mathcal{O}p$ is all the available operators with their representation (preconditions and effects), and
- $\mathcal{E}$ is all the available entities and their description.

4.3.2.2 HATP Problem

A HATP problem is defined by $\langle \mathcal{D}, c_0, m(p) \rangle$ where:

- $\mathcal{D}$ is the domain as defined above,
- c_0 is the initial context, where the first operator(s) is going to be applied, and
- m is the task to apply on c_0 in order to obtain the solution plan and p is its parameters.

4.3.2.3 Solution plan

The expected output for a HATP problem is a sequenced list of symbolic actions solutions (SAS) called a symbolic plan. A SAS is defined by $\langle o, stNexts \rangle$ where $op \in \mathcal{O}p$ is an operator and $stNexts$ is the list of SASs which operators can be applied only when st operator is successfully applied. As seen before, $stNexts$ represent the causal links.

HATP can handle multiple-agent by creating a stream for each one (a stream is a sequence of operator joined together by causal links) and interleaving the streams by connecting them through causal links. Figure 4.2 shows a plan example where agent 1 and agent 2 have two separate streams synchronized through causal links.

FIGURE 4.2: An example plan generated by HATP: it contains two streams, for agent 1 and agent 2, connected through causal links 2 agent same task

HATP is able to work under two different modes: first plan, or best plan. In the first plan mode, as soon as a plan is found, it is returned. In the best plan mode, all the plans are computed and the plan with the best quality is returned. One interesting feature concerning HATP is its ability to abort the computation of a partial plan as soon as it detects that its quality is lower than the best plan already computed.

4.3.2.4 HATP Algorithm

Algorithm 8 shows HATP implementation of the HTN algorithm which is based on a depth first search[4]. It goes through a main loop (Line 3 to Line 28) and go out of it when one of the three stopping conditions is reached:

- the algorithm was asked to find only one plan, and one plan was found,
- a maximum time limit is reached, or
- all the possibilities has been tried and failed.

Note that the $StopAtFirstPlan$ and the maximum time are also inputs of the algorithm.

Next, the algorithm tests the cost of the current plan (even if not complete) and compare it to the cost of the best plan found in Line 4 (if there is no plan found yet, this test is skipped). If the cost of the current plan is higher than the best plan cost, the current plan is abandoned, and the algorithm restarts from the previous backtracking point (BP) Line 5 (this process, called backtracking, is explained later in this subsection).

[4]The HATP algorithms are not included in the contributions of this thesis, but are needed to understand the combination between the symbolic and the geometric layers which is part of the contributions, and is presented later in this chapter.

The next step of the algorithm is to retrieve the applicable methods and operators in the current world state: the T variable store all the not explored operators and methods and a loop (from Line 9 to Line 13) checks if they are applicable or not in the current context. If they are applicable, they are added to the applicable task list *App*. Otherwise, they stay in T until they become applicable, as the current context change. For an operator to be applicable, it needs its predecessors (following the causal links) to be already applied and its preconditions to be valid in the current context. For a method, having all its predecessors applied is enough to add it to the list. One last check, is the task locking: when a task is locked, it cannot be applied to any context, unless it gets unlocked: this system is used to tackle the asynchronous tasks problem, as explained later in this subsection.

The last part of the algorithm (Line 14 to Line 27) uses the previous lists to choose and apply tasks, depending on different variables:

No applicable task in *App* and no task in T, Line 14 This means that a plan was found then it backtracks to the last cached BP.

No applicable task in *App*, tasks exist in T and all tasks in T are locked, Line 17 In this case the algorithm create BPs for each one of the locked task: in each BP, only one task is unlocked, the rest stay locked.

No applicable task in *App*, tasks exist in T and some task in T are not locked, Line 20 This means that, within the rest of the actions still not explored, no one is applicable in the current context, forcing the algorithm to backtrack to the previous BP.

There is one and only one applicable task in *App*, Line 22 In this case, the task is directly applied, through the Algorithm 10.

There is more than one applicable task in *App*, Line 25 This case arise when faced with the asynchronous decomposition, the algorithm creates as many BPs as there are applicable tasks, where only one task is not locked. When the unlocked task is applied, the algorithm goes back to the second case of this enumeration (all tasks locked).

A backtracking point (BP) is composed by the current states of the main variables in the algorithm: the list of yet to be explored tasks T (and if they are locked or not), the current context c_{curr} and the current plan $plan_{curr}$. The creation of such a point, as done in Line 19 and Line 26 is depicted in Algorithm 9. When a backtrack is triggered, it retrieves the last BP saved (and removes it from the saved list of backtracking points, this is a stack: last in, first out) and instantiate it as the current state of the algorithm. If no BP is left in the list, all the decompositions have been tested and no more plan will be found.

The case of the asynchronous tasks is tricky: in order to test all the possible task orders, the algorithm uses the locking process. Locking an action means that the action is not applicable yet, later it will be unlocked to allow the search to continue. The process consists on creating as many BPs as there are tasks. In each BP all the tasks are locked but one, which is the first task to be tried. If the task can be applied, then all the current tasks (in T) are locked which correspond to the second case of the list

above. In this case, we create as many BPs as there are tasks left, with one unlocked task in each one of the BPs (as we just did). This process goes on until all there is only one task left. At each step of this process, we unlock only one task, making the algorithm tries all the possible orders.

Algorithm 8 HATP implementation of the Classical HTN algorithm

```
1: function SolveHTN(D, c_0, m(p), StopAtFirstPlan, MaxTime)
2:     T ← m(p) ; c_curr ← c_0
3:     while (¬FirstPlanFound or ¬StopAtFirstPlan) and ¬Reach(MaxTime) and T ≠ ∅ do
4:         if GetCost(plan_curr) > GetBestPlanCost then
5:             (T, plan_curr, c_curr) ← BacktrackToLast(backtrackList)
6:             continue
7:         end if
8:         App ← ∅
9:         for t | t ∈ T, IsUnlocked(t) do
10:            if ((ValidPredecessor(t) and IsOperator(t) and ValidPreconditions(c_curr, t)))
                       or (ValidPredecessor(t) and IsMethod(t))) then
11:                App ← t
12:            end if
13:        end for
14:        if App = ∅ and T = ∅ then
15:            P ← plan_curr
16:            (T, plan_curr, c_curr) ← BacktrackToLast(backtrackList)
17:        else if App = ∅ and T ≠ ∅ and ∀t ∈ T, IsLocked(t) then
18:            V | V ⊂ T, ∀t ∈ V, IsLocked(t)
19:            CreateBacktrackPoints(V, T, plan_curr, c_curr, backtrackList)
20:        else if App = ∅ and T ≠ ∅ and ∃t ∈ T, IsUnlocked(t) then
21:            (T, plan_curr, c_curr) ← BacktrackToLast(backtrackList)
22:        else if |App| = 1 then                                  ▷ size of App is 1
23:            a | a ∈ App
24:            Apply(a, backtrackList, T, plan_curr, c_curr)
25:        else if |App| > 1 then
26:            CreateBacktrackPoints(App, T, plan_curr, c_curr, backtrackList)
27:        end if
28:    end while
29:    return P
30: end function
```

Algorithm 9 depicts how to create backtracking points out of a subset V of the yet to explore task list T. For each task in V it creates a backtracking point where every other task in V is locked.

Algorithm 9 The function to create the backtracking points

```
1: function CreateBacktrackPoints(V, T, plan_curr, c_curr, backtrackList)
2:     for a | a ∈ V do
3:         UnLock(a)
4:         for tmp | tmp ∈ V, tmp ≠ a do
5:             Lock(tmp)
6:         end for
7:         backtrackPoint ← (T, plan_curr, c_curr)
8:         backtrackList ← backtrackPoint
9:     end for
10: end function
```

Algorithm 10 shows the way a task is applied. Whatever the kind of the task, as it is going to be applied, it is removed from T. When the task is an operator (Line 24) it effects are added to the current context to create a new one and the operator is added at the end of the current plan. When adding an operator to the plan, it causal links are also updated, from the domain, but also using logic: if a tested predicate of this operator has been changed by the effect of another operator, the algorithm links them through a causal link. If the task is a method (Line 3), first we check the applicable decompositions by testing their preconditions (in case of an exclusive disjunction). Once we retrieved the list of all applicable decomposition *validD* three cases arise: no applicable decomposition, (Line 10), in which case, the algorithm triggers a backtrack, only one decomposition is applicable (Line 13), the algorithm adds its corresponding tasks to T and the last case is when multiple decompositions are possible (Line 16). This last case arise only when the decomposition is an exclusive disjunction, and more than one decomposition has valid predicate in the current context. In this case, for each valid decomposition a BP is created, and one among them is chosen to continue the algorithm.

Algorithm 10 Implementation of the apply function

```
1:  function Apply(a, backtrackList, T, plan_curr, c_curr)
2:      T ← T \ a                                              ▷ remove a from T
3:      if IsMethod(a) then
4:          D ← GetAlldecompositions(a)
5:          for d ∈ D do
6:              if ValidPreconditions(c_curr, d) then
7:                  ValidD ← d
8:              end if
9:          end for
10:         if ValidD = ∅ then
11:             (T, plan_curr, c_curr) ← BacktrackToLast(backtrackList)
12:             return
13:         else if |ValidD| = 1 then
14:             d | d ∈ ValidD
15:             T ← GetAllTasks(d)
16:         else
17:             for d | d ∈ ValidD do
18:                 T_tmp ← T ∪ GetsAllTasks(d)
19:                 backtrackPoint ← (T_tmp, plan_curr, c_curr)
20:                 backtrackList ← backtrackPoint
21:             end for
22:             (T, plan_curr, c_curr) ← BacktrackToLast(backtrackList)
23:         end if
24:     else                                                   ▷ a is an operator
25:         c_curr ← ApplyOperatorEffects(c_curr, a)
26:         plan_curr ← a                ▷ adding the operator and its causal links to the plan.
27:     end if
28: end function
```

These algorithms are not included in the contributions of this thesis, but are needed to understand the combination between the symbolic and the geometric layers which is part of the contributions.

```
operator PlaceR(Agent A, Object O, Support S, Agent AT){
    preconditions {
        A.hasInHand == O;
    };
    effects {
        A.hasInHand = NULL;
        O.IsOn = S;
        O.IsReachable = AT;
    };
    cost{costFct(A,O,S,AT)};
    duration{durationFn(3, 5)};
}

method MoveObj(Agent A, Object O, Support From, Support S, Agent AT) {
    {
        preconditions {
            A.type == "ROBOT";
            AT.type == "HUMAN";
            O.isOn == From;
        };
        subtasks {
            1: Pick(A, O);
            2: PlaceR(A, O, S, AT) after 1;
        };
    }
}
```

Listing 4.1: HATP code example

4.3.2.5 HATP example

Listing 4.1 shows an extract of a HATP domain, illustrating the operator *PlaceR* which makes the agent A place the object O on the support S reachable by the agent AT. It has as a precondition: the object should be in the robot hand, and the effects are: the object is not in the agent hand anymore, it is on the support S and is reachable by AT. It is the same description as the one presented in the beginning of this section. The example also contains the cost of the operator computed by an external procedure (*costFct*) which take all the operator parameters as inputs. It also contains the duration of the solution (here from 3 to 5 seconds).

The second part of the example shows the method *MoveR* which can be decomposed into two operators *Pick* and *PlaceR* in this order. Its preconditions are that A is a robot, AT a human and the object O is on the support $From$.

This example shows the simplicity of creating domains with HATP, one of its main features as presented in de Silva et al. (2015).

4.3.2.6 HTN-HATP differences

The principal differences between HATP and the most known HTN planners (such as SHOP2 Batista (2011)) are:

User-friendly language: as seen in the previous section, the description language is easy to learn and use. It is based on a close world assumption which ease the domain design.

Control over variable binding in classical HTN, the choices (for variable binding) are made randomly, in HATP these choices can be made following rules (or directly set by the domain expert).

Totally ordered: in HATP, all the actions are ordered by the causal links, as opposed to the HTN algorithm where the actions are partially ordered.

Agents based: HATP considers the agents as "first order" entities, for which actions are computed. It computes for each agent, a stream of actions, linked between themselves with causal links.

Real robot use: HATP was implemented in the robot and used with a complete architecture to plan and execute its plans. Even if slower than SHOP2, HATP still enables real time use.

Cost based: HATP aborts plans with a cost that exceeds the current best plan.

C++ structures: HATP is coded in C++ which enables an easy integration with other C++ modules, as seen in the next section.

4.3.3 Symbolic Geometric Action Planner

The Symbolic Geometric Action Planner or SGAP is the framework we devolved to tackle the SGP problem. In this framework, we use for the symbolic layer HATP and for the geometric layer, we use GRP (presented in the previous chapter).

This framework can use any kind of forward task planner, but using an HTN planner brings some benefits: as different levels of actions are available in the GRP, having a hierarchical domain enables the programmer to choose which level of operators he needs/wants to use. For example, if an operator *PickThenPlace* is available in addition to the operators *Pick* and *Place*, using the first one might speed up the search, while using the decomposed version might enable the system to choose another operator after *Pick* (such as *Give* or *Throw* depending on the context). Moreover, an HTN planner enables its programmer to add constraints to the lower level operator, for example, he can use the operator *PlaceR* but if it is not available, he can use the operator *Place* with a reachability constraint.

In our particular case, we chose to use HATP because of its ability to manage multiple agents plans (let us remember that this framework was developed in the context of human-robot interaction), its simple domain language and also its ability to use external C++ calls. These calls can be of different kinds, such as cost computation, geometric tests, and so on. In the rest of this chapter, we will discuss a number of these external calls.

The approach we are going to explain in more details in this section is based on the following: HATP begins the search in the given symbolic domain, and when an operator needs to be applied, if the operator has a geometric counterpart (such as *Pick* or *Place*) an external call is made to the GRP with the aId of the geometric action corresponding to the current operator in order to test its feasibility in the current world state. This call is named **Projection** or **Geometric refinement** and is about finding the geometric action solution (GAS) of said action. When the GRP computes this GAS, meaning that

the action is feasible, the current world state is updated with the new information then the relevant facts are computed and sent back to HATP.

When GRP sends back these facts, they are transformed into predicates and used to update the symbolic context of HATP. We call these predicates **Shared predicates** as they are computed in the geometry but used in the symbolic search as the usual predicates (to test the tasks pre-conditions). As shown in the previous chapter, facts are computed by GRP (Subsection 3.2.4) under the form of $\{X,$ **is reachable by**, $A,$ $true\}$. When HATP receives these facts, a mapping enables it to transform them from this form to the one used in the algorithm: $X.isReachable = A$. These **shared predicates** are used to tackle a number of problems such as the ramification, as explained in Subsection 4.3.5.

As shown in the previous chapter, the GRP framework is able to find multiple alternatives for the same action, starting from the same initial world state. GRP is also able to compute, in any world state, shared predicates. Using these properties, we combined HATP and GTP into the SGAP framework, giving it the ability to assess actions feasibility at geometric level, to request actions alternatives when needed, and to integrate the shared predicate into the planning process.

In the next subsection, we are going to present the differences between SGAP and HATP, for every part of the planner.

4.3.3.1 The basic notions

For the formalization, some additional information was added to the basic elements:

Predicates: The predicates can have two sources: purely symbolic predicates, and *shared predicates*.

Context: In addition to all the predicates it contains, each context is linked to a geometric world state (Subsection 3.2.2) from where the shared predicates can be computed.

Entity description: It is the same as for the HATP algorithm, with the constraint that the entities *id* should be the same at symbolic and geometric level (Subsection 3.2.1).

Operators: The operator description is transformed to $\langle pre, act, \text{eff}\rangle$ where *pre* and eff are the same as before, and *act* is the action identifier, in the GRP framework. *act* can be empty, in which case the action is purely symbolic and does not need a geometric counterpart. Once this operator is "projected" into the geometric level, the action is recognized through a number $act.gasNum \in \mathbb{N}$ at the GRP framework level.

Methods: The methods and their possible decompositions are the same as for the HATP algorithm.

Tasks: The tasks are the same as for the HATP algorithm either Operators or Methods.

The previously defined operators can now be redefined within the SGAP framework, but as the preconditions and the effects does not change (although some of them would be computed directly from the geometric level as shared predicates) the main link to be added is the geometric action *aIds* as it was defined in Subsection 3.3.2:

Pick(A,O): geometric action: **Pick**.

Place(A,O,S): geometric action: **Place**.

PlaceR(A,O,S,AT): geometric action: **PlaceReachable**.

Navigate(A,E): geometric action: **NavigateTo**.

PaintGreen(A,O): geometric action: $\emptyset$ (purely symbolic action).

The other parts of the HATP definition also go under the following transformations.

4.3.3.2 SGAP Domain

A SGAP domain $\mathcal{D}_{sgp}$ can be defined by the 5-uplet $\langle \mathcal{M}, \mathcal{O}p, D_g, \mathcal{E}, E \rangle$ where:

- $\mathcal{M}$ is all the available methods in the domain with their decomposition, as before,
- $\mathcal{O}p$ is all the available operators with their representation (preconditions, action aIds and effects),
- D_g is the domain that contains all the available geometric actions with their aIds and their descriptions,
- $\mathcal{E}$ contains the available entities with their ids and symbolic description, and
- E contains the available entities with their ids and geometric information, the ids are the same as for $\mathcal{E}$.

4.3.3.3 SGAP Problem

A SGAP Problem is defined by $\langle \mathcal{D}_{sgp}, c_0, ws_{init}, m(p) \rangle$ where:

- $\mathcal{D}_{sgp}$ is the domain,
- c_0 the initial context,
- ws_{init} the initial world state, and
- $m(p)$ the method or operator to apply to this initial context and world state.

Note that c_0 initially contains only symbolic predicates, the initial shared predicates are computed from ws_{init}.

4.3.3.4 Solution plan

The solution computed by SGAP is a sequenced list of Action Solution (AS) called a plan. An Action Solution is defined by $\langle o, gas, tNexts \rangle$ where o is the operator, gas the corresponding Geometric Action Solution (GAS) and $tNexts$ the causal links. An Action solution is a combination of a SAS and a GAS! In parallel to the plan, the GRP framework build at the same time a geometric plan linked to this plan as depicted in Figure 4.3.

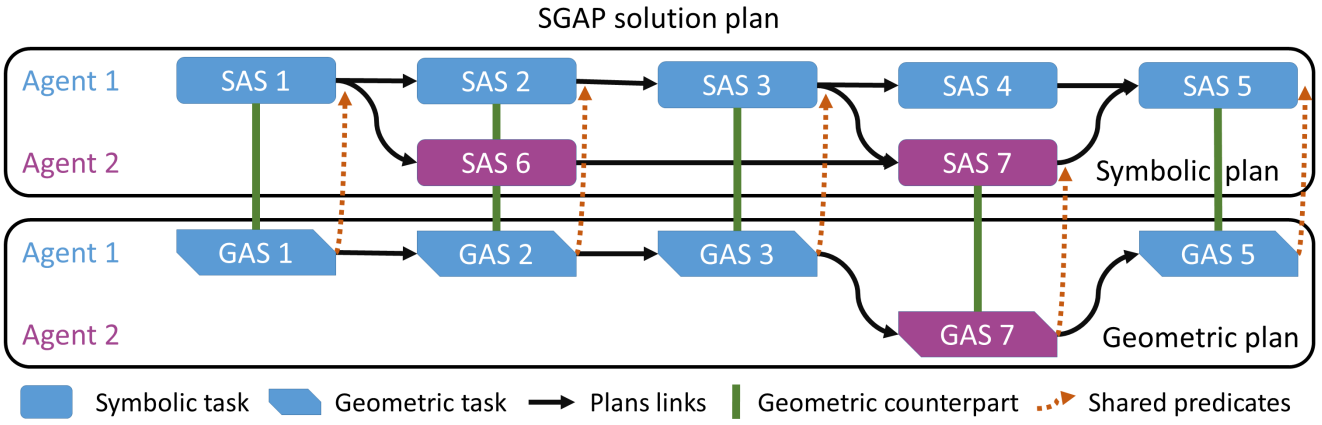


FIGURE 4.3: An example plan generated by SGAP: it contains the symbolic part of the plan, composed in two streams and linked to the geometric part (thick lines) which contains a geometric plan. Note that AS 4 and AS 6 are purely symbolic and have no geometric counterpart. Once a GAS is computed, the geometric planner sends to the symbolic one the shared predicate computed in the final world state of the GAS.

4.3.3.5 Algorithm

The algorithm we used to combine symbolic and geometric planning is very similar to the ones depicted in the algorithms 8, 9, and 10. The main differences can be spotted in three different places of the algorithm, the first one happens just after the initialization. The algorithms need to update the initial context with the shared predicates from the initial world state. The second one is at Line 10 of Algorithm 8: in the part concerning the operator, one additional function is tested: PROJECTACTION$(t, plan_{curr})$ (described in Algorithm 11). The last one concerns the operator part of the apply function (Algorithm 10) which is depicted in Algorithm 12 (The part concerning the application of a method is the same as in Algorithm 10, it is omitted in Algorithm 12)

Algorithm 11 is called to test if an action can be projected or not at geometric level. It first tries to compute the corresponding GAS *gas* (Line 2 to Line 11) then, if it succeeds, links *gas* to the current operator t and returns $True$, otherwise it returns $False$. In order to compute *gas*, there are two possibilities: the operator has never been projected before in this context and world state[5], in this case, the algorithm retrieves the inputs of the geometric action from the operator parameters, retrieves the current world state by getting the predecessor of the current operator in the current plan (Line 5) and finally calls one of the functions described in Section 3.3.3 (here COMPUTEACTIONCONFS()) by giving it the action aId specified in the operator t. The second case happens when t has already been projected in a previous decomposition (Line 8 to Line 11), in which case, the algorithm retrieves the GAS number $gasNum$ from the previously computed *gas* stored in t and computes a geometric alternative.

Algorithm 12 is split into two parts, the first one, concerning the methods, is the same as the first part of Algorithm 10 and is omitted in this one. The second part, concerning how to apply an operator, has two more steps added when the operator has a geometric counterpart (otherwise these steps are ignored). The first step (Line 26 to Line 32) is the one handling the possibility of trying different geometric alternatives: it first specifies that the current operator has been projected and creates as many

[5]They can change when a backtrack is triggered and the new decomposition is similar to this one. As a new BP has been loaded, the parameter informing if it was projected or not (function ISNOTPROJECTED) is also reloaded

Algorithm 11 Implementation of the project action function

```
1: function ProjectAction(t, plan_curr)
2:     if IsNotProjected then
3:         predessessor ← GetPredessessor(t,plan_curr)
4:         IN ← GetParameters(t)
5:         IN ← GetWorldState(predessessor)
6:         aId ← GetActionId(t)
7:         gas ← ComputeActionConfs(aId,IN)
8:     else
9:         gasNum ← getGasNum(t)
10:        gas ← FindAlternative(gasNum)
11:    end if
12:    if gas ≠ Null then
13:        SetGas(t,gas)
14:        return True
15:    end if
16:    return False
17: end function
```

new PBs as their branching factor[6] allows and in each BP a decomposition with the same operator is added. If the algorithm backtracks to this BP, as the operator has already been projected, an alternative will be requested (Algorithm 11). The second added step is about retrieving the geometric part of the current context c_{curr} (Line 36 and Line 37): as said before, part of the context is retrieved from the corresponding world state under the form of shared predicates.

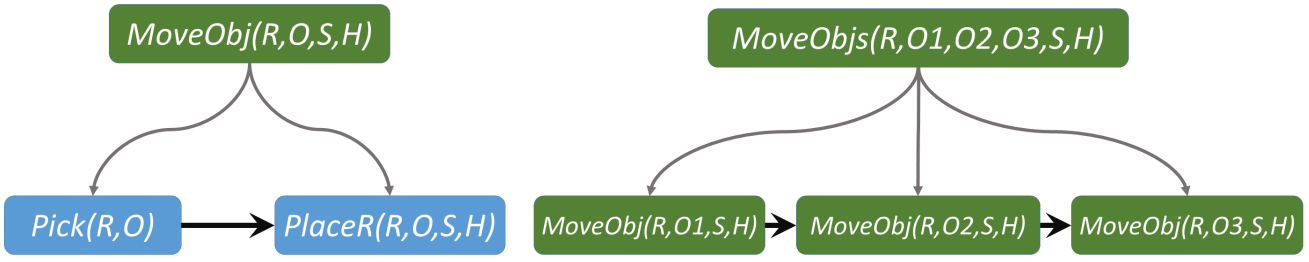
Algorithm 12 Implementation of the apply function concerning the SGP algorithm

```
1: function Apply(t, backtrackList, T, plan_curr, c_curr)
2:     T ← T \ t                                   ▷ remove t from T
3:     if IsMethod(t) then
           ...                                     ▷ Omitted, the same as Algorithm 10
24:    else                                        ▷ t is an operator
25:        if HasActionId(t) then
26:            SetProjected(t)
27:            b ← GetBranchingFactor(t)
28:            for i | i ∈ ℕ, i ∈ [0, b] do
29:                T_tmp ← T ← t
30:                backtrackPoint ← (T_tmp, plan_curr, c_curr)
31:                backtrackList ← backtrackPoint
32:            end for
33:        end if
34:        c_curr ← ApplyOperatorEffects(c_curr, t)
35:        if HasActionId(t) then
36:            ws ← GetEndingWorldState(t)
37:            c_curr ← GetFacts(ws)               ▷ adding the shared predicates to the state
38:        end if
39:        plan_curr ← t
40:    end if
41: end function
```

[6]The number of possible geometric alternative allowed by the symbolic level

4.3.4 Example

Figure 4.5 shows an example where the robot needs to place the books on the table in front of it so that all of them are reachable at the same time by the human. The symbolic domain $\mathcal{D}$ for this problem is depicted in Figure 4.4(a) and Figure 4.4(b). The involved entities are the three books (*O1*, *O2*, *O3*), the two tables (stock and target) and the agents: the human and the robot. The geometric domain contains the description of **Pick** and **PlaceReachable**. The initial context contains one predicate: $R.hasInHand = NULL$, and the initial world state (from where the shared predicates for the initial context are computed) is depicted in Figure 4.5(b). The solution plan our SGP algorithm computes is given in Figure 4.5(a) in addition to the different geometric alternatives it went through before failing: some world states does not enable the robot to place the last book, some of them are depicted in Figure 4.5(d), Figure 4.5(e) and Figure 4.5(f). One possible solution our system found is shown in Figure 4.5(c). The difficulty in this example lies in the size of the target table, if one of the first books takes too much space on it, the last one will not fit, making the algorithm backtracks until the faulty book changes position and all three of them can fit on the table.



(a) (b)

FIGURE 4.4: The domain for an example where the robot needs to place three books in front of the human.

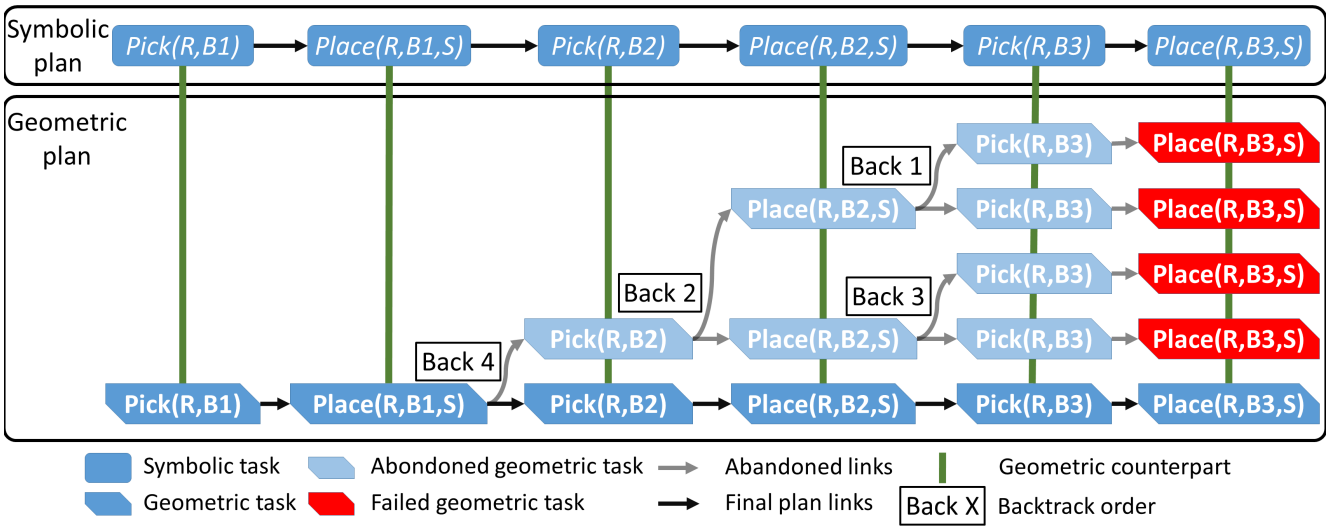
4.3.4.1 Domain definition

The domain definition for SGAP is done with nearly the same code as for HATP, as shown in Listing 4.2. It contains the operator *PlaceR* and the method *MoveObj*. As seen in the code, the method is exactly the same as for HATP, but there are two differences between the HATP operator *PlaceR* and this one:

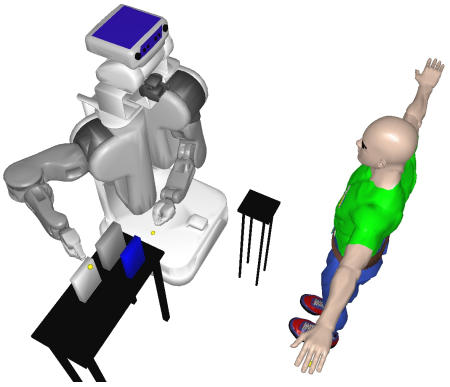
The projection after testing the precondition, the projection of the action at geometric level is tested by calling the function *projects* with the *aId* of the geometric action and the corresponding inputs (here the inputs are the agent performing the task, the object, the support and the agent target).

The cost computation in HATP, the cost is given by the domain expert, in SGAP, the cost is computed by the GRP and set to the Action Solution by the line $cost\{GetGRPCost()\}$;

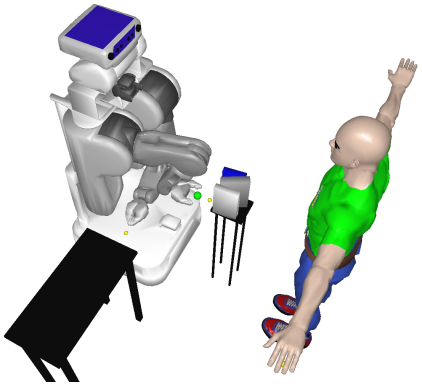
In the *projects* function, if the action is successfully projected, the shared predicates are computed and added to the context, with the effects defined in the domain.



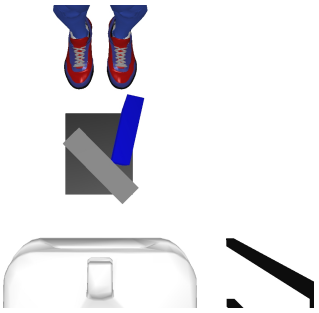
(a) The Plan and the different alternatives computed in the geometric level



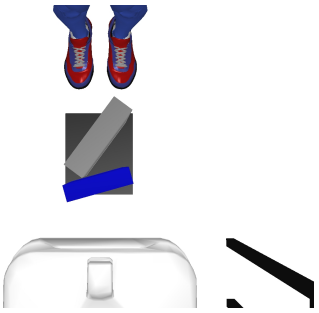
(b) Initial world state



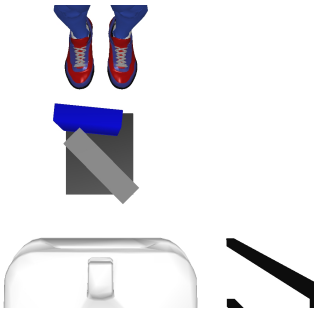
(c) Final world state



(d) Failed attempt



(e) Failed attempt



(f) Failed attempt

FIGURE 4.5: An example where the robot needs to place three books in front of the human. (a) Shows the resulting plan alongside with the failed geometric alternatives that correspond to (e), (f), and (g). (c) And (d) shows respectively the initial and final computed world state. (a) shows also the order the backtracks happened: after failing to compute the first *Place(R,B3,S)*, the algorithm backtracks to **Back 1**, and tries an alternative of *Pick(R,B3)* before failing again to place the object. Then it backtracks to **Back 2** where it computes an alternative to *Place(R,B2,S)*. The process continues, and fails two more times before finding a solution (backtracking first to **Back 3** then to **Back 4**.

```
operator PlaceR(Agent A, Object O, Support S, Agent AT){
    preconditions {
        A.hasInHand == O;
    };
    projects {placeR(A, O, S, AT)};
    effects {
        A.hasInHand = NULL;
        O.IsOn = S;
        O.IsReachable = AT;
    };
    cost{GetGRPCost()};
    duration{durationFn(1, 1)};
}

method MoveObj(Agent A, Object O, Support From, Support S, Agent AT) {
    {
        preconditions {
            A.type == "ROBOT";
            AT.type == "HUMAN";
            O.isOn == From;
        };
        subtasks {
            1: Pick(A, O);
            2: PlaceR(A, O, S, AT) after 1;
        };
    }
}
```

Listing 4.2: SGAP code example

4.3.4.2 Results

We run the algorithm on this example[7] and Table 4.3 represents the results obtained over 30 runs for each branching factor. The plan length is 6 actions, consisting on 3 successive *Pick* and *Place*. The success rate is nearly perfect starting from a branching factor of 3 but the Computation time also grows accordingly. Note that the success rate of the algorithm when the branching factor is 5 drops. Failing with this many possible alternative is possible as the search space is not complete: for completeness, the branching factor should be infinite.

Branching factor	1	2	3	4	5	6
Computation time (s)	5	19	19	27	31	31
Success rate (%)	10	80	100	100	96.6	100
Nb alternatives	0	9.4	14.5	24.7	32.7	36.6
Nb actions computed	5.6	31.6	41	62.8	78.7	85.8

Table 4.3: For a plan length of 6 actions, the system is able to compute with a success rate approaching the 100% a solution for the example in Figure 4.5, starting a branching factor of 3. These values are averaged on 30 runs.

[7] The runs on this section were all made on a computer with an i7-3720QM CPU @ 2.60GHz processors an a memory of 8Go

4.3.5 The ramification problem

The ramification problem is the problem of characterizing the indirect effects of an action (more details are available in McIlraith (2000)). In other words, it means computing the consequence of an action in addition to its direct effects, which are the effects described in the action model.

Usually, in task planning, the problems are simplified to handle the direct effects only, and the ramification problem is not addressed. One way to compute indirect effects is to use Truth Maintenance systems Doyle (1979) which use inference and assumption to compute them. Nowadays, in robotics, these inferences and assumptions are made by the Ontologies systems, such as Tenorth and Beetz (2009), but it is not used to tackle the ramification problem.

When addressing the problem of symbolic geometric planning, it appears that it is possible to compute at geometric level a number of properties that correspond to facts (the shared predicates), and, therefore compute in a more valid way the actions consequences. This is even more important when humans are present, as the action consequences (shared predicate and cost) can allow the planner to find better or preferred plans.

Figure 4.6 shows an example of this problem: the robot needs to place three objects on the table in front of it in order for the human to be able to reach the three of them at the same time (The same as the previous example, with different objects and environment). In Figure 4.6-C the robot places the third object reachable, but the first object is no longer reachable (this example is further detailed in Subsection 4.4.2).

In order to (partially) tackle this problem, we use the shared predicates: after a geometric action is planned, we compute those predicates for the new context based on the final world state found at geometric level. If some predicates prevent a further action preconditions to apply, a backtrack is triggered and an alternative geometric solution is requested. This process goes on until a valid plan is found, the branching factor (for now this number is given by the domain expert) is reached or no other geometric solution is available.

The problem is only partially tackled due to the discrete set of shared predicates the system is able to compute: if a shared predicate does not exist, the problem will not be tackled.

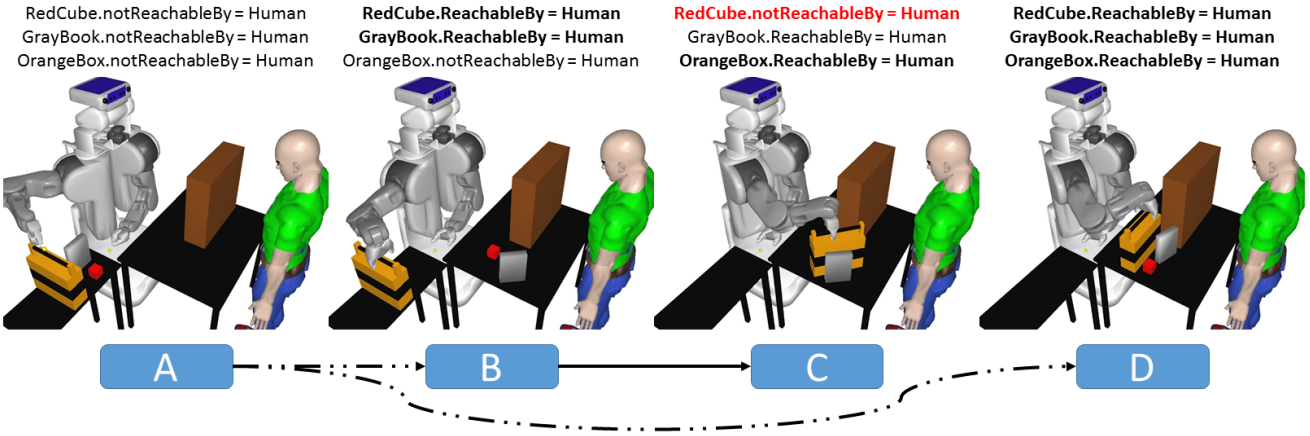


FIGURE 4.6: In this example, the robot can reach the three objects (the red cube, the grey book and the orange box) and needs to place them on the table in front of it, reachable by the human. A is an initial geometric situation, B is a step of the planning process where the robot already placed the red cube and the grey book reachable to the human. In C, the robot places the orange box reachable to the human, and by doing so, obstructs the human reach to the red cube. Finally, D shows the result of a plan found after backtracking on number of actions. This illustrates the ramification problem.

4.4 Enhancing the search efficiency

The examples presented in the previous sections show that SGAP can compute interleaved symbolic and geometric plans in the context of human-robot interaction. Although, these plans are short (10-15 actions) with a small number of objects, in a not too constrained environment, but the use of well-informed motion planning and geometric reasoning allow us to deal with not so trivial problems.

However, when confronted with more constrained challenges (with greater number of objects or longer plans), a combinatorial explosion occurs, making the planning process very long. In order to enhance the search efficiency, we propose a number of features able to better inform both planners with relevant information and heuristics to guide the search even if we might lose completeness.

4.4.1 Geometric requests

A geometric request is made by the symbolic planner to the geometric one in order to test a property in the geometric world. This request is usually very fast (less than 50ms) and is used as a precondition of a task. The only function that changes in the algorithm is $ValidPreconditions()$ (in Algorithm 8 and Algorithm 10) which takes as additional parameter the current world state, and, when faced to a geometric requests, compute it on that world state.

The geometric parameter tested can be of various types such as testing if there is enough space for the robot to stand near the human in a constrained area, or if there is enough space to use a hammer on a particular object in a cluttered space. These requests are generally domain specific, which work well with HTN algorithms and are used as heuristics to guide the search toward the most promising plans. In order to test the pertinence of these requests, we have developed a "*virtualPlace(O,S)*" which tests if there is enough space on the support S to place the object O with no collision with any other object. One more addition to this test is the virtual objects: these objects are tools used to test collisions only within the *virtualPlace* request, otherwise, the geometry ignores them.

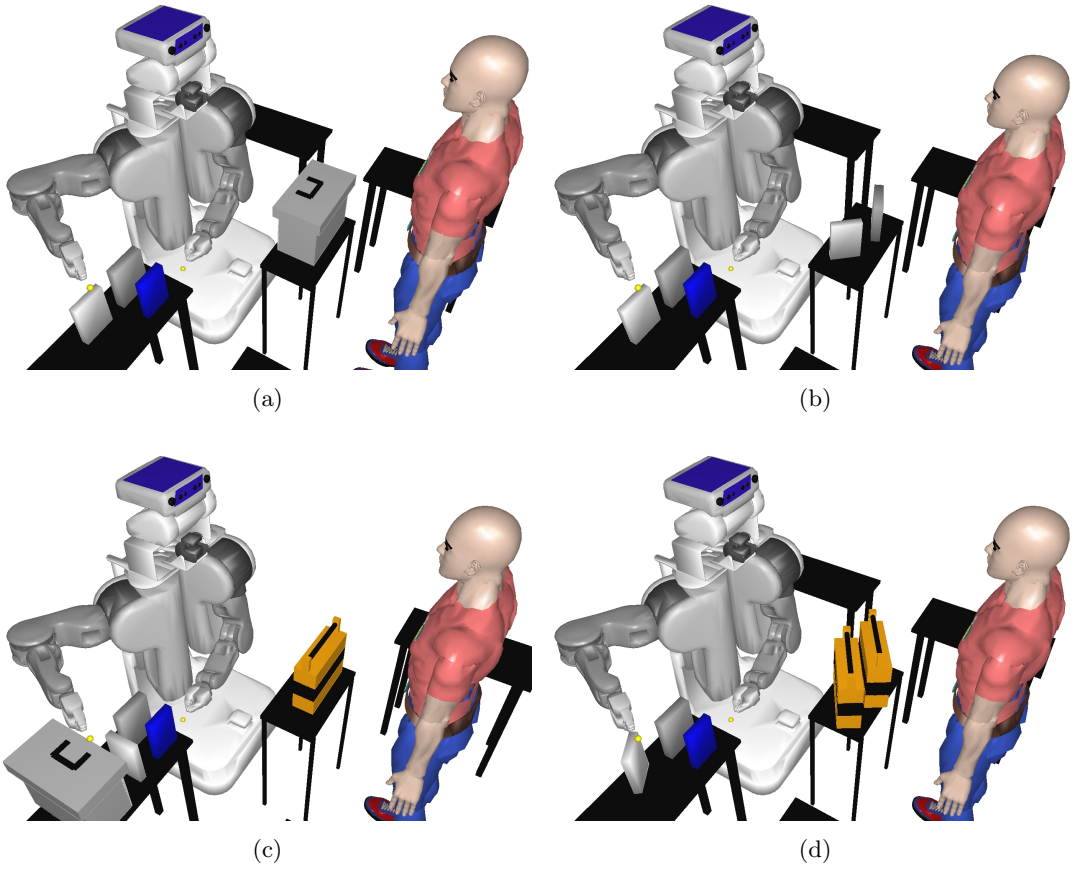
In order to use these virtual objects, we formulate an assumption:

Assumption. *If the virtual object V can contain object O1, O2 and O3, and it can be placed on the table T then the objects can also be placed on the table. It can be considered as a heuristic.*

Note that if the virtual object cannot be placed on a table, it does not mean that the objects cannot. The virtual objects are given to the system as independent entities with the same properties as the other ones, in addition to the virtual part. In this implementation they are tuned by hand for every environment, but it is possible to design algorithms to compute them on-line depending on the number and geometry of the objects they should contain.

The *virtualPlace* request is used as follows: it tests if a valid placement (collision free) for a virtual object, which can contain smaller objects, on a support exists. If it does exist, the small objects can be placed on this support.

Figure 4.7 shows an environment with different world states that illustrate when this request might be useful, and Figure 4.8 shows the same world states with the virtual object that does not fit on the



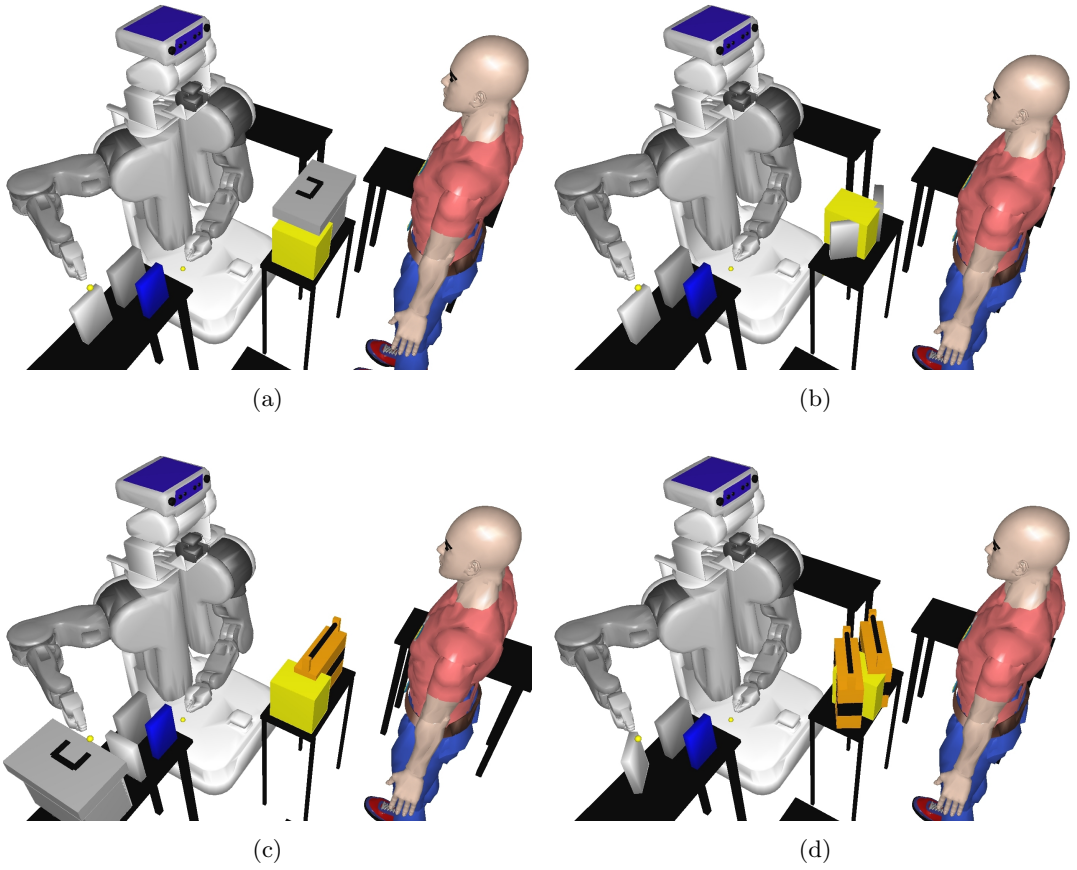
(a) (b) (c) (d)

FIGURE 4.7: Examples of scenarios where the geometric requests enable an enhancement in speed.

table. Note that in the state shown in Figure 4.7(b) and Figure 4.8(b) the virtual object does not fit on the table, however, there is enough room to place the small object. In this particular case the heuristic fails and the solution plan is not the best one as shown later in the results.

The domain used to illustrate this enhancement, is depicted in Figure 4.9. The main method given to the SGP problem is *TestAndMove* in Figure 4.9(d). This method has two possible decompositions: in the first one it tries directly to place the objects on the target table (Figure 4.9(a) and Figure 4.9(b)), and in the second one it first tries to remove obstacles from the target table, by placing them on another surface, before placing the objects on the target table. Removing the obstacles can be performed by either the robot or the human, depending on the feasibility of the task. For example in Figure 4.7(a) even if the robot has enough space to place the object in the table at his right, it cannot grasp the object, in Figure 4.7(d) it can *Pick* the objects but does not have enough space to *Place* it anywhere. The choice of which decomposition to apply is done by testing the geometric request *virtualPlace(VirtualO)* on the starting world state, with *VirtualO* a virtual object that can contain the three books *O1, O2* and *O3*. In this example, the robot and the human can only manipulate the object, they cannot navigate.

Figure 4.10 shows a solution plan found for the example in Figure 4.7(a) where, first the human clean the table by moving out the obstacle, *Ob*, then the robot place the three books. In order to assess the interest of this enhancement, we built a similar domain where the decomposition of *TestAndMove* which first tries to place the three object on the table, and only if it fails tries to empty the table. This complementary domain enables us to determine the speed up when using the *virtualPlace* request.



(a) (b) (c) (d)

FIGURE 4.8: The same states as in Figure 4.7 with a placed virtual object. The virtual object is drawn in yellow and does not fit in any of the tables.

Table 4.4 shows the results of this experiment. The left side of the table shows clearly the interest of having this heuristic when the table is cluttered: the computation time is nearly divided by 10. When the table is empty, as the request is not time consuming, both domain have similar results. The second half of the table shows an interesting behaviour: in Figure 4.7(b) there is enough space to place the three books, but not enough to place the virtual object (Figure 4.8(b)). This property misguide the heuristic, which indicate that an object should be removed before placing the three books causing the plan to be longer and the computation time to be bigger than the case where it would directly place the books, as it will directly succeed.

4.4.2 High level actions and Constraints

As seen in Subsection 3.3.5, the most computationally expensive step is motion planning. The idea of these enhancements is to avoid the calls to the motion planner as much as possible. The motion planner calls occur when an action needs to be projected, or an alternative to an already projected action is needed. Requesting an action alternative means that a backtrack has been triggered. One way to reduce the number of motion planner calls is then to reduce the number of backtracks. In order to achieve this, we propose to "protect" some predicates –or, more precisely, shared predicates– that the domain expert knows might be broken by some future actions.

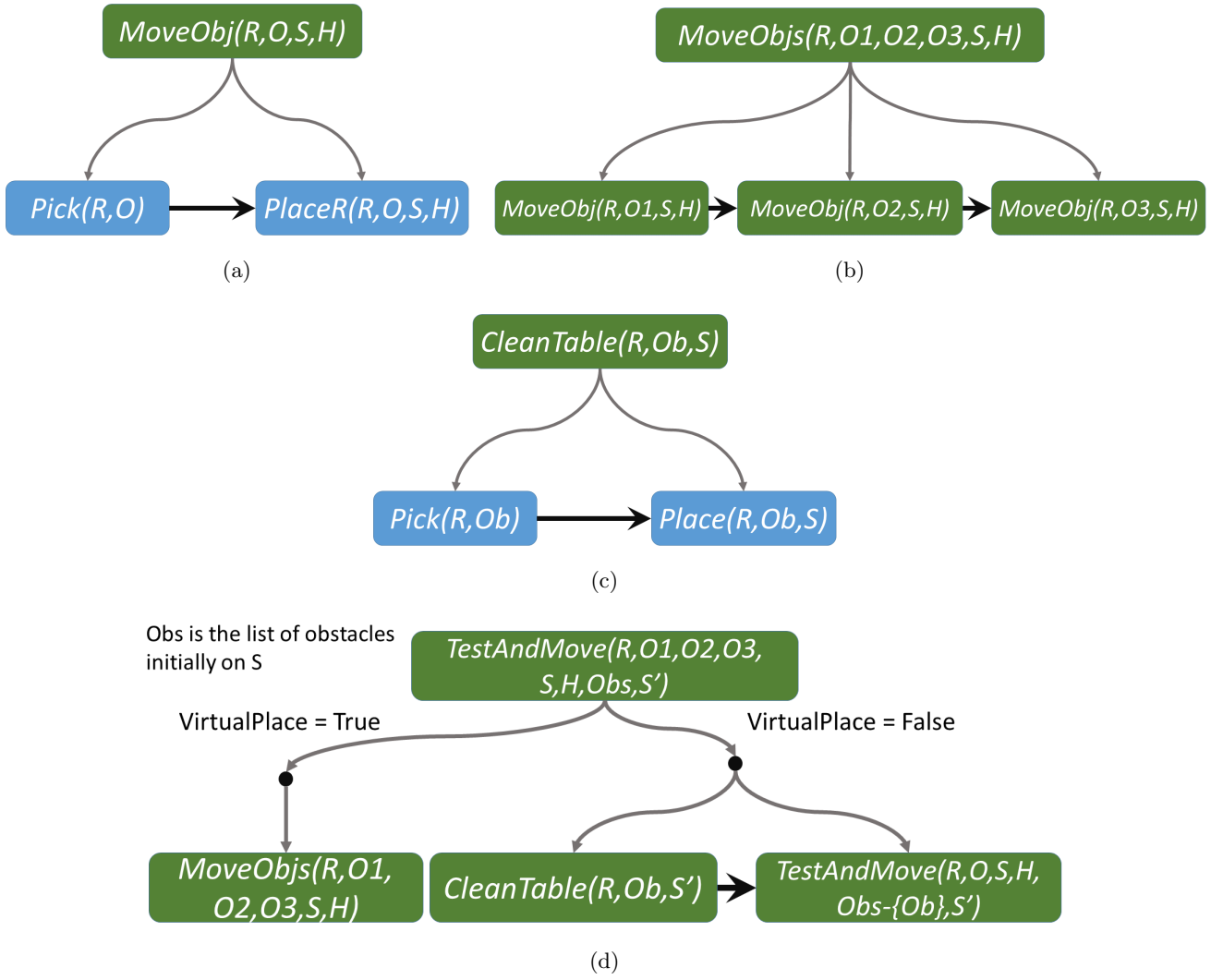


FIGURE 4.9: The symbolic domain used to assess the enhancement geometric requests

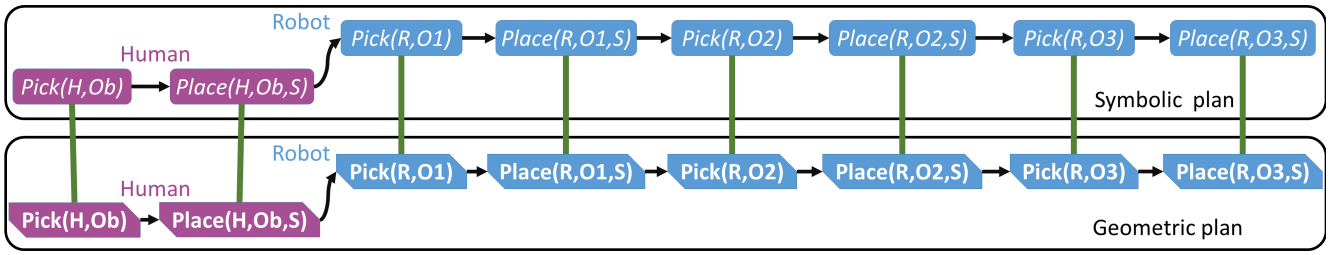


FIGURE 4.10: The resulting plan of the algorithm when called for the example depicted in Subsection 4.4.1

Following the framework description of Section 3.3 two ways exists (which are equivalent) to communicate this specific information within our implementation. The first way is to design geometric actions with a higher level of abstraction that embed restricted search spaces for specific operator, we can consider the **PlaceReachable** action as an example, as it is of a higher abstraction than the **Place** action. When an object needs to be placed on a table in order to let another agent reach it, using directly the **PlaceReachable** action will reduce the number of backtracks as the approach with the place action would be to apply the corresponding operator and then test the shared predicate $O.isReachable = H$

example	Figure 4.7(a) example		Figure 4.7(b) example	
type	without	with	without	with
plan length	8	8	6	8
Computation time	191.2	21.7	16.9	20.2
→ standard deviation	8.6	1.45	1.64	1.34
Nb alternatives	62.9	0.3	0.2	0.2
Nb actions computed	162.6	12.6	6.4	8.6

TABLE 4.4: The results are averaged on 30 runs on both the examples in Figure 4.7(a) and Figure 4.7(b) with a branching factor of 3 and the *first plan found* mode. The advantage of using the geometric requests (with) is clear when the table is cluttered (Figure 4.7(a)). In Figure 4.7(b) the geometric request fails as there is not enough space to place the virtual object on the table even if there is enough space to place the three books, which can be seen in the results. The examples in Figure 4.7(c) and Figure 4.7(d) give very similar results to Figure 4.7(a) results.

in the next preconditions check. The second way to inform the geometric reasoner is by adding constraints to the action linked to a specific operator. In order to achieve this, we need to transform the operator definition to: $\langle pre, act, const, \text{eff}\rangle$ where *const* is a list of constraints under the form defined in Subsection 3.3.4.4, and adding the following line between Line 4 and Line 5 of Algorithm 11:

$$\text{IN} \leftarrow \text{GETCONSTRAINTS}(t)$$

If no constraints are specified ($const = \emptyset$), the above line does not add anything to the inputs.

In order to illustrate and assess this enhancement, we used the environment depicted in Figure 4.11(d), and designed a domain, described in Figure 4.11(a), Figure 4.11(b) and Figure 4.11(c). In this domain, there are three new operators:

PlaceRC(R, O, S, Ap, AlreadyPlace): precondition, action and effects are the same as for *PlaceR(R,O,S,Ap)* and it has one additional constraint, placed in the final constraint of the first sub-action description in the place action: $\forall Oi \in AlreadyPlace\{Oi,$ **is reachable by**, $Ap,\ true\}$. AlreadyPlace is the list of object that are already placed on the destination table.

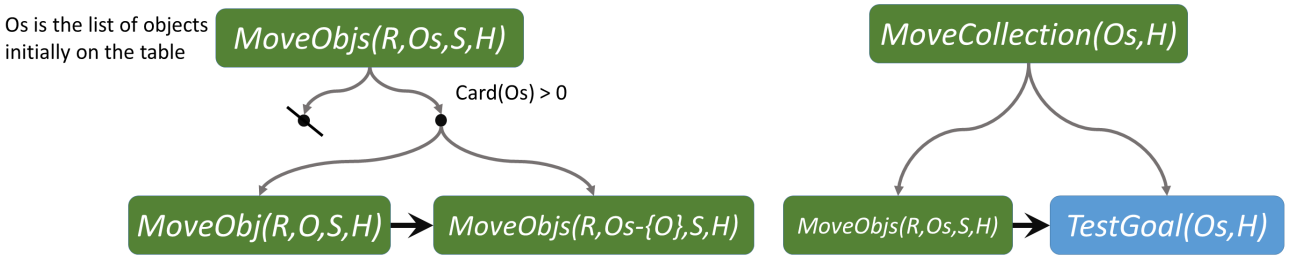
TestReach(O,Ap): has only a precondition: $O.isReachable = Ap$ (no action, constraint nor effect)

TestGoal: has only a precondition: $\forall O \in AlreadyPlaced\ O.isReachable = Ap$ where *AlreadyPlaced* is the group of object that has already been placed reachable to the human

In the figures, the "operator" PlaceX appears. It is not really an operator as it is replaced by one of the three operators *Place*, *PlaceR* or *PlaceRC*: by doing so, we create three different domains, one where no enhancement is used, one where a higher level action is used (*PlaceR*) and finally one with a high level action and a constraint specified (*PlaceRC*). The method *MoveObjs* is recursive, and, when decomposed, tries to apply the method *MoveObj(O)* with one of the object available on the storing table. When no more objects are on this table, the method goes out of the recursive behaviour. In this domain, the robot needs to place the three objects next to him (the red cube, the grey book, and the orange box, in that order) reachable for the human at the same time. The difficulty here is that when placing the orange box, it can hide and make unreachable one or both objects already placed on the table as seen

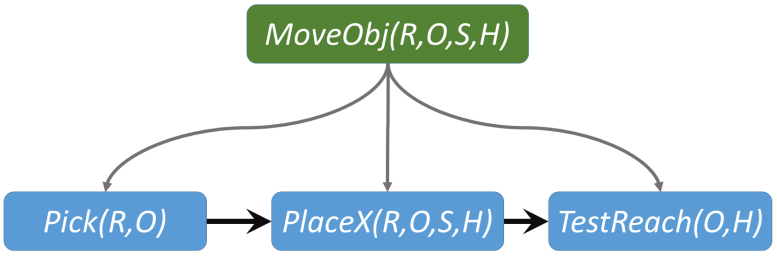
in Figure 4.6 (placing the grey book can also make the red cube unreachable). Figure 4.11(e) shows one possible solution found by this planner.

Table 4.5 shows the results for the three domains. As expected, the results when using the *PlaceRC* operator are the best, dividing the computation mean time by two. The number of computed actions projected and alternatives requested drops accordingly.

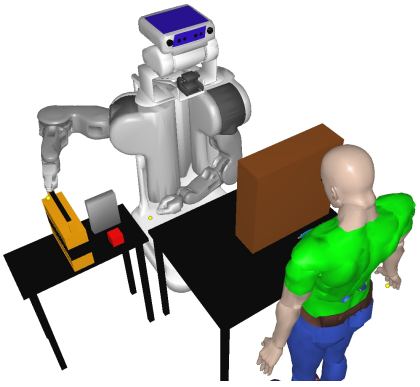


(a) Part of the symbolic domain

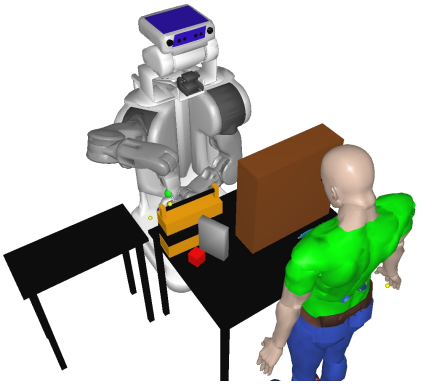
(b) Part of the symbolic domain



(c) Part of the symbolic domain



(d) Initial world state



(e) Final world state

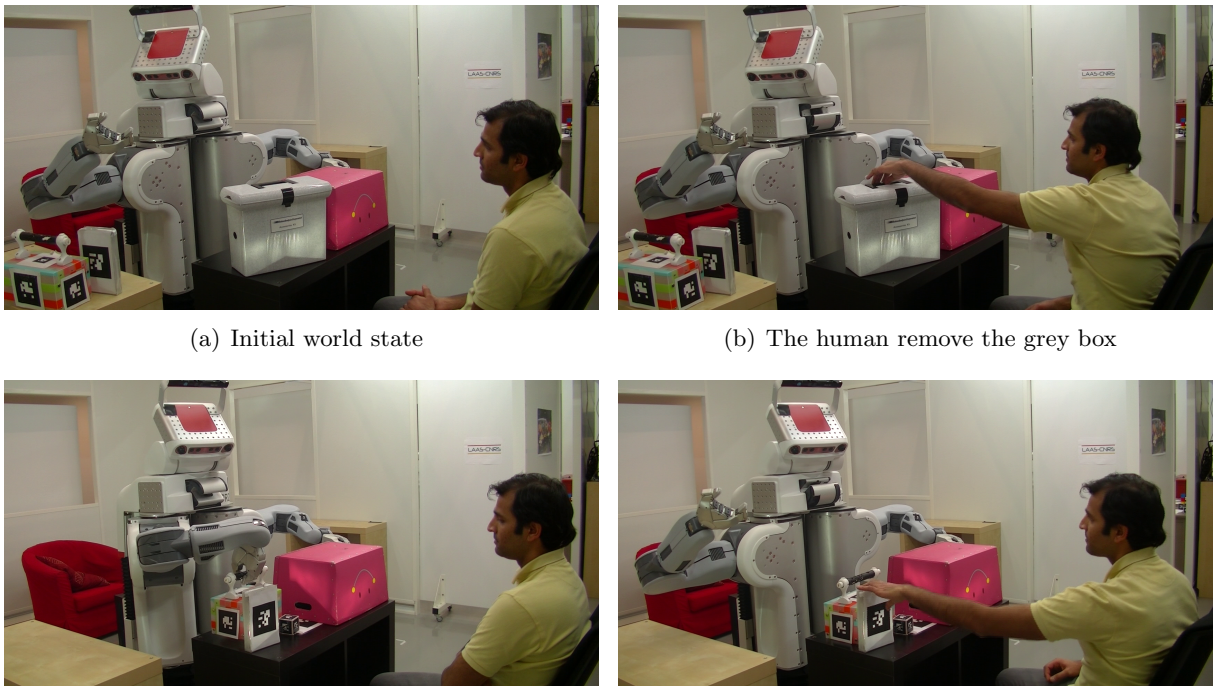
FIGURE 4.11: The domain, and the initial and final world states of an environment where using the enhancement related to high level action and constraints is useful.

type	*Place*	*PlaceR*	*PlaceRC*
Mean time	44.3	27	16.2
Mean time standard deviation	6.14	3.56	1.8
Nb actions computed	67.5	11.6	8.6
Nb alternatives	59.6	3.8	1.6

TABLE 4.5: For a plan which length is 6 actions, the use of high level actions in addition to specific constraints enables a great computation time improvement. These values are averaged on 30 runs and the branching factor was 5.

4.4.2.1 Real robot implementation

The SGP algorithm and its enhancements were implemented on the PR2 robot. Figure 4.12 shows a scenario where the robot needed to place three objects on the table in front of the human, and the human, once he saw all the objects, needed to choose one of them and take it. In order to achieve this, the three objects needed to be reachable at the same time. The table was cluttered with two boxes that the robot is not able to move, the human needs to participate in the tasks in order to achieve the goal. The corresponding video combined with different simulation cases is available here: https://youtu.be/KUF4Gdhc2Do



(a) Initial world state

(b) The human remove the grey box

(c) The three objects are accessible

(d) The human takes one object

FIGURE 4.12: The implementation of the SGP algorithm on a PR2 robot. The task is to place the three objects in front of the human, in order to let him choose one of them. The table is cluttered and need first to be emptied.

4.4.3 Cost driven search

The algorithm presented in Subsection 4.3.2 enables HATP to prune out plans when the cost of their first part is greater than the best plan already found (Line 4 to Line 7 of Algorithm 8). The cost used in this algorithm is provided by the domain expert as input of the problem, the idea of this enhancement is to compute the cost automatically at geometric level. The GRP framework compute this cost at the same time as computing the GAS and return it alongside, it is then stored in the Action Solution (AS) until the function GETCOST($plan_{curr}$) is used (it can be a sum or a maximum of all the tasks the current plan contains).

Computing the cost at geometric level, where social rules can be taken into account, enables the system to explicitly take into account the human preferences. In order to illustrate this, we implemented two scenarios depicted in the followings where we run the SGP algorithm with the option of finding all

the possible plans (and returning the best one). In these scenarios the interesting operator is *Navigate* as the cost computation in its linked action, navigateTo, is based on the work of Sisbot et al. (2007b) as it maximizes the cost when the robot navigates out of the human's field of vision (behind the human for example) or too close to him.

4.4.3.1 The "Book scenario"

In this scenario, a human asks the robot to bring him a book, but two copies of this book are available in the environment. In order to choose the best book to bring to the human, the robot uses the costs computed by the geometric reasoner. Figure 4.13(a) and Figure 4.13(b) show the domain for this environment, the higher method *BringObj* needs to choose either the book $O1$ or $O2$ to bring to the green human. Figure 4.13(c) and Figure 4.13(d) depict two world states where this domain has been used. In the first one, the robot fetches the closest book, where the navigation distance is the smallest. In the second one the algorithm chooses the other book as, by taking the same path as the previous example, the robot would pass close and behind the blue human which increases the cost (without human = 7.9, with human = 15.5).

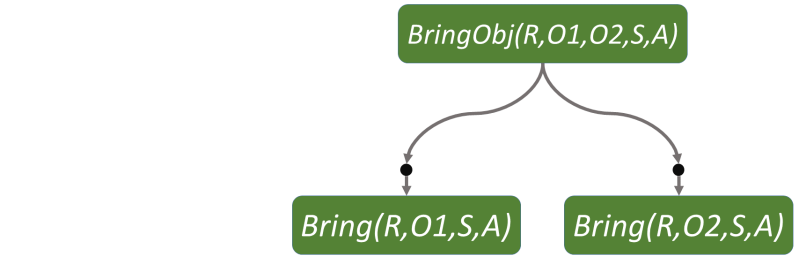
With no information from the geometric level, the symbolic level would make a random choice on which decomposition to apply. Adding these costs computation enable the symbolic planner to make informed choices during its search for the best plan.

4.4.3.2 the "Paint scenario"

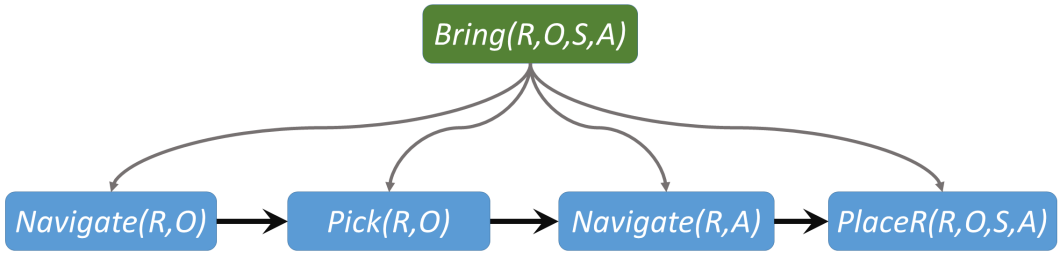
This second scenario is more complex than the first one as it involves more actions and agents: Figure 4.14 and Figure 4.15 show the symbolic domain used, where the top method is *BringAll*, and the robot needs to bring to the client two green cubes. In the environment, there is one already green cube, and two red cubes that needs to be painted (Figure 4.16 shows the starting world state). The blue agent can paint the objects in green if needed. The client is the green human (A) and the red human is a co-worker occupied in another task.

The main difficulty in this example is to choose which object to bring to the client: the green cube is easily accessible and does not need to be painted, the first red cube (top right) is also easily accessible but makes the robot navigate behind the red human and finally the last cube (bottom right) is hard to access, the robot needs to first remove the box obstructing his path and then he becomes able to take the object. This last possibility (removing the box) is depicted by the method *PickObj* (Figure 4.15(b)) where the robot check if the object is reachable. If it is not reachable, it tries to move any reachable object (*O"*) which in this case is the orange box.

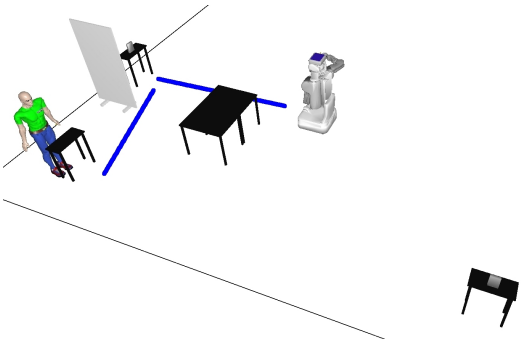
The plan produced by our algorithm chooses to go fetch first the green cube (O1), then the red cube (O3) at the bottom of the environment: even if more tasks are needed to get this object (moving the orange box *OB*), it does not disturb the red human (by passing and manipulating behind him). These examples and some others are shown and explained in the video available here: https://youtu.be/mxDRQEGqQK4



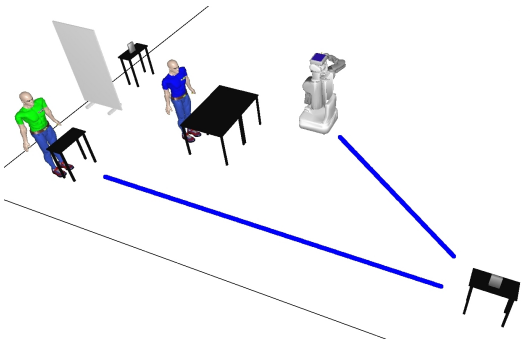
(a) Part of the symbolic domain



(b) Part of the symbolic domain



(c) Environment without human



(d) Environment with human

FIGURE 4.13: The domain, and two different world states depicting the interest of using cost computation at geometric level.

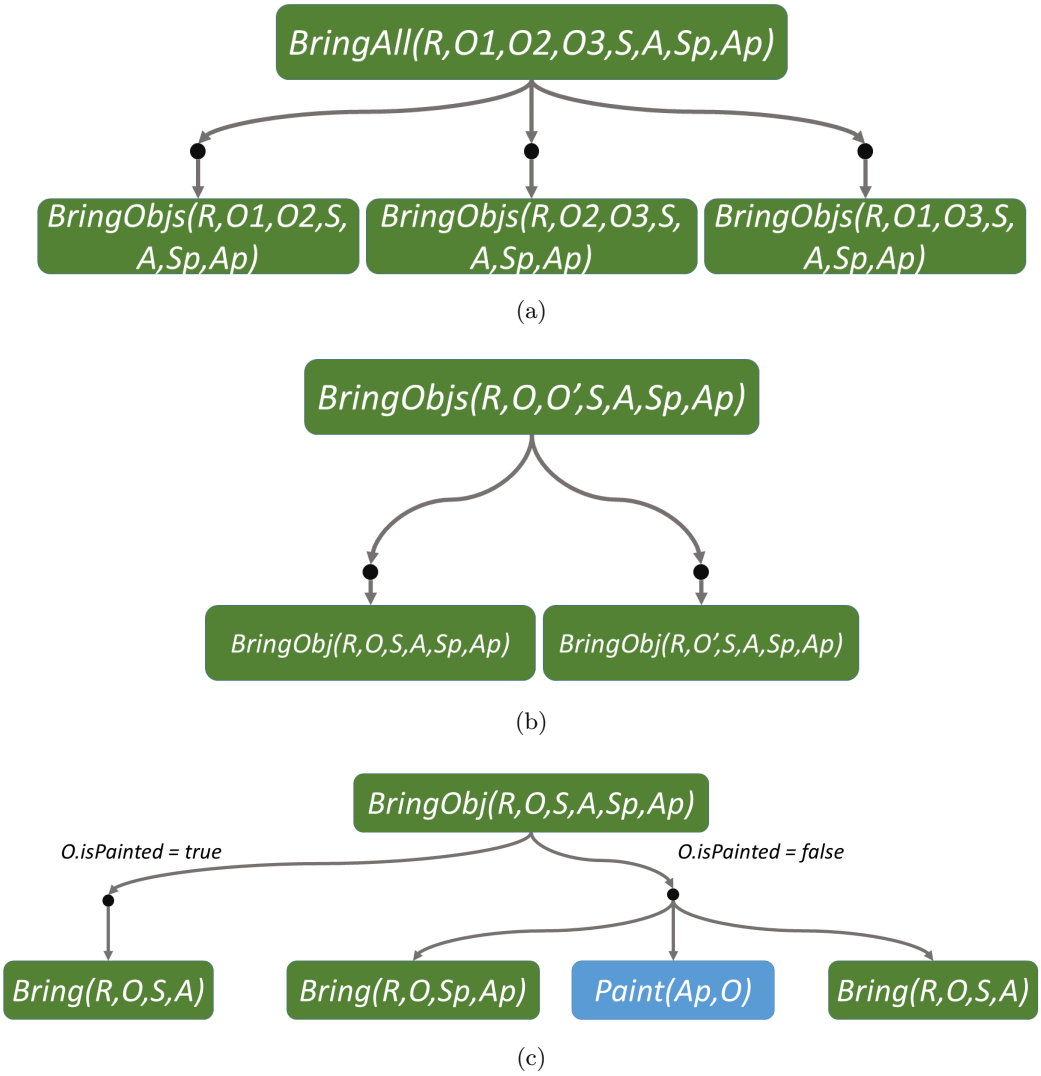


FIGURE 4.14: In this domain A is the agent requesting two green objects, and Ap an agent who can paint in green the objects. In this domain, the top function *BringAll* can choose between bringing two of the three available objects (a). Bringing an object consists on checking if it is green or not, if it is, brings it directly to the client, if not, brings it to Ap, in order to let him paint it and then brings it to the client. Note that if an object is not reachable after navigating next to it, the domain asks the robot to move any other object around it until the target object is reachable. O" is an object of the environment for which the predicate: $O''.isNextTo = O$ is true. The rest of the domain is shown in Figure 4.15

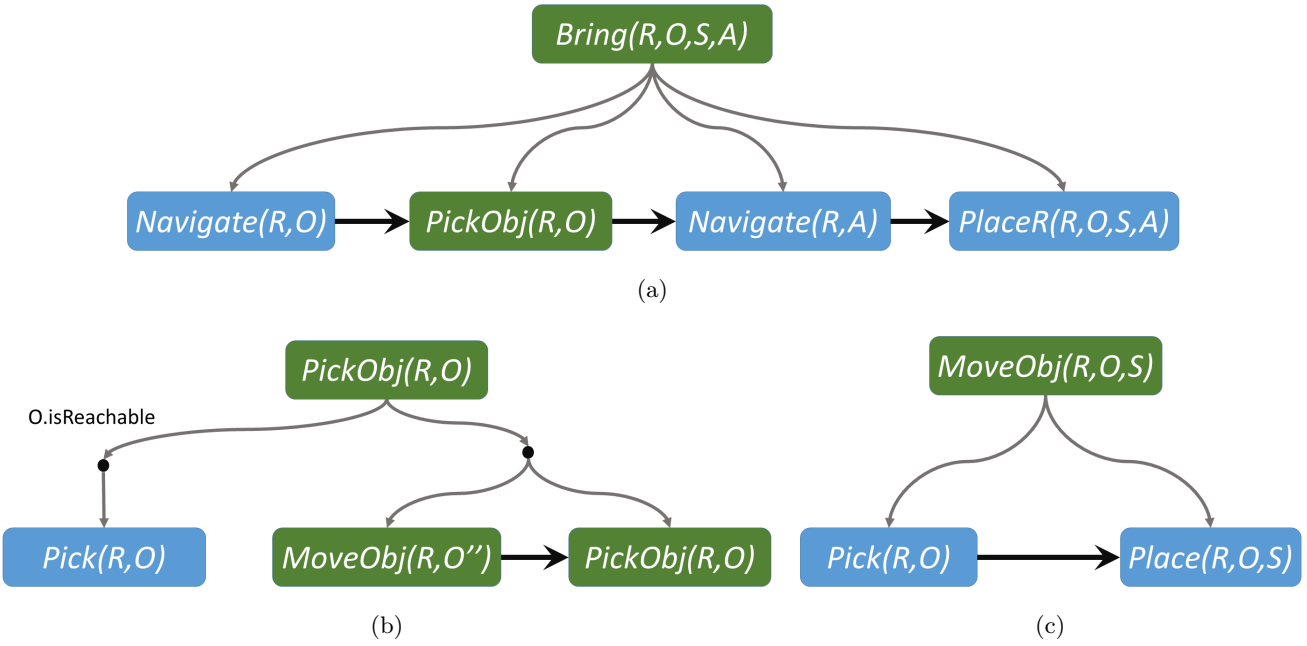


FIGURE 4.15: The rest of the domain of Figure 4.14

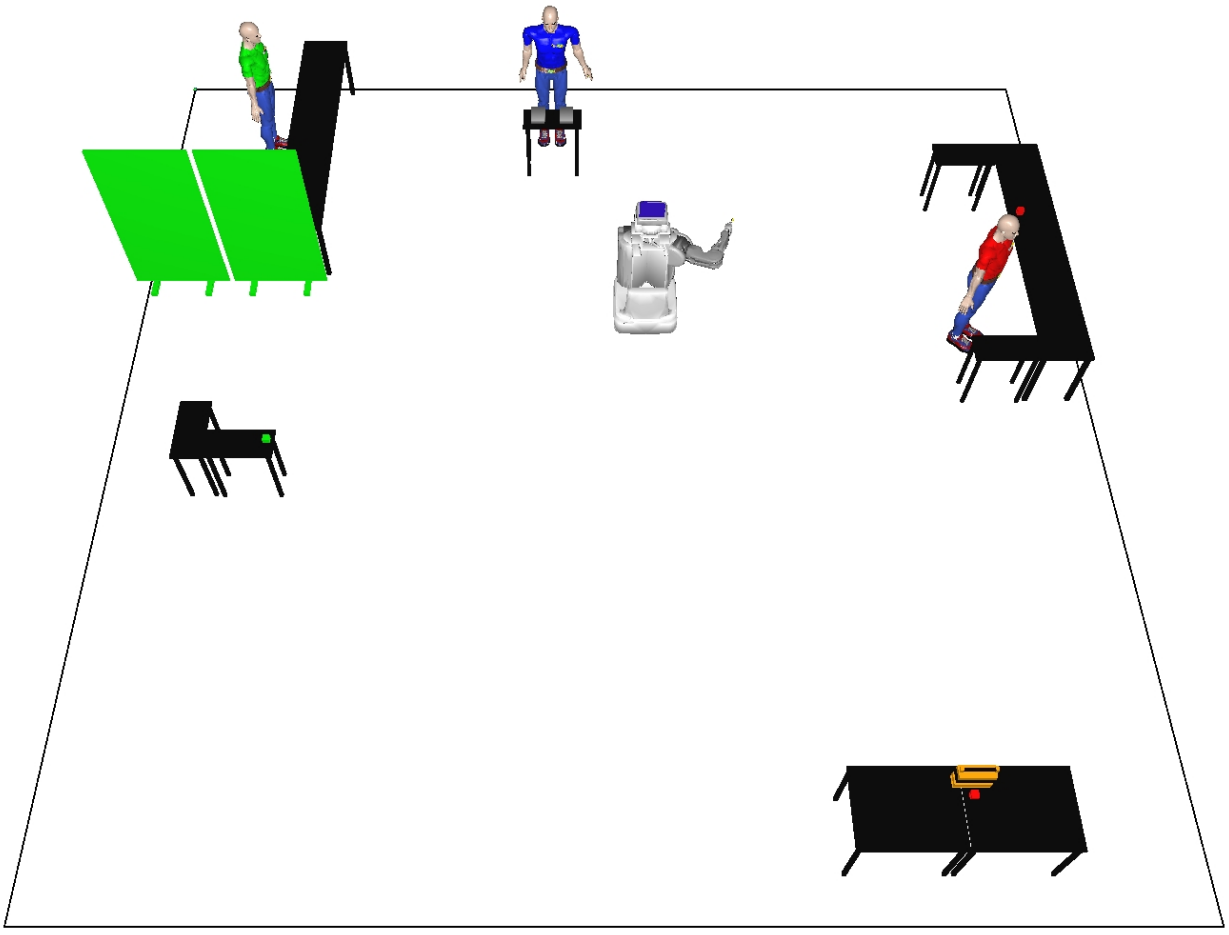
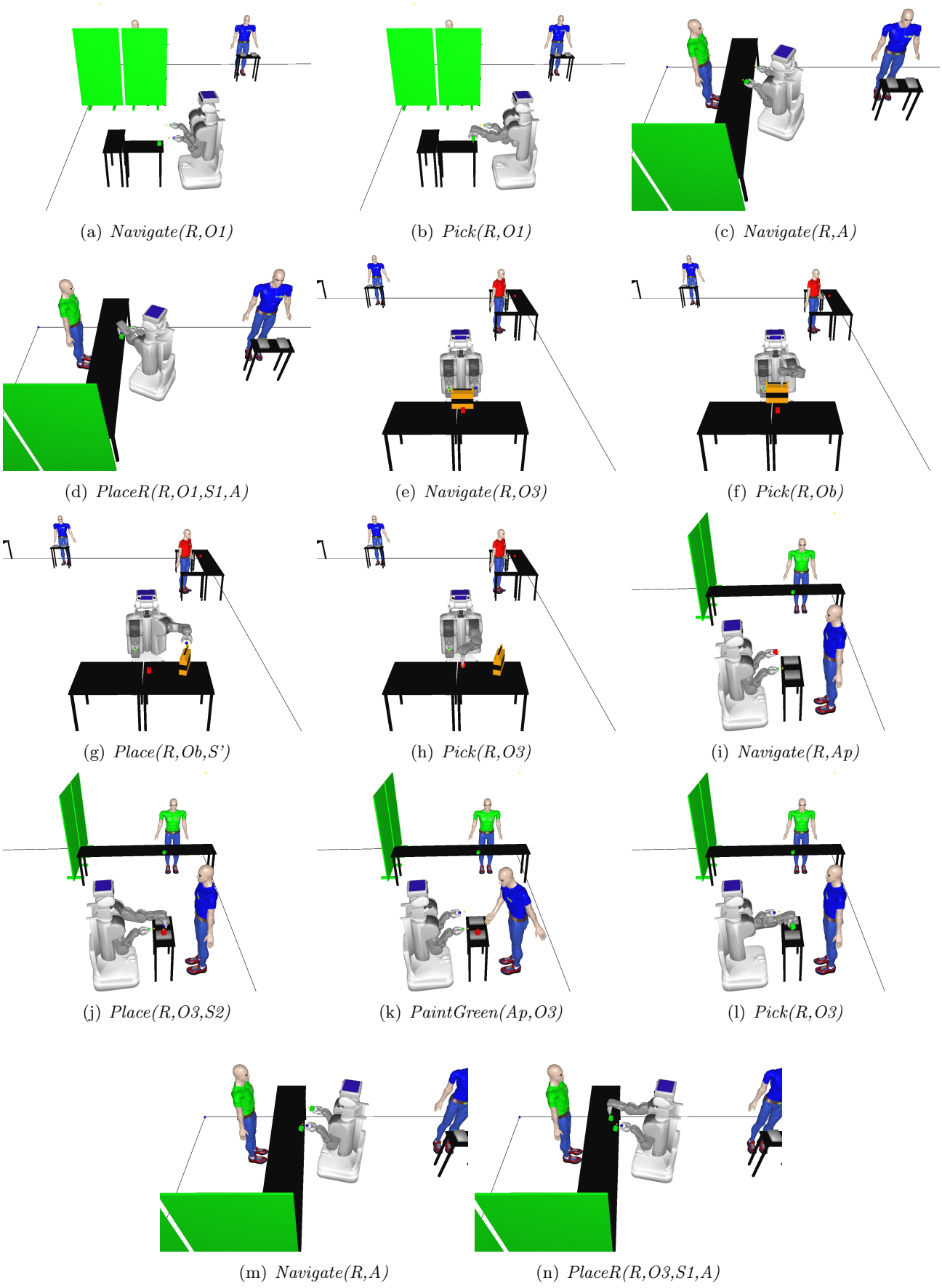


FIGURE 4.16: An environment where the green human (A) asked the robot to bring him two green cubes. The blue human (Ap) is able to paint red cubes. There are three cubes in the environment, a green one (middle left) and two red ones (top and bottom right).



(a) *Navigate(R,O1)* (b) *Pick(R,O1)* (c) *Navigate(R,A)*

(d) *PlaceR(R,O1,S1,A)* (e) *Navigate(R,O3)* (f) *Pick(R,Ob)*

(g) *Place(R,Ob,S')* (h) *Pick(R,O3)* (i) *Navigate(R,Ap)*

(j) *Place(R,O3,S2)* (k) *PaintGreen(Ap,O3)* (l) *Pick(R,O3)*

(m) *Navigate(R,A)* (n) *PlaceR(R,O3,S1,A)*

FIGURE 4.17: The different plan steps for the "paint scenario"

4.5 Future work

As seen in the state of the art, this work falls into the category of a "Symbolic planner calling the geometric reasoner", and more specifically in the sub-category of "in search calls". We proposed three main enhancements of this algorithm linked to the features provided by the GRP framework presented in Chapter 3. Even if these enhancements enable a faster computation in some domains, the main problem remains, the exponential growth of the backtracking number when the branching factor is big. In order to tackle this problem we propose some possible line of works:

Choosing the backtracking point For now, when the backtrack is triggered, the last saved backtrack point (BP) is loaded and the search continues from there. The idea is to change this behaviour by introducing a weight on the BPs, and prioritizing the more promising ones. A criterion to determine those interesting BPs can be provided by the geometry: for the BP created from a geometric action projection, the size and shape of this action search space might be a good indicator, a small convex search space may not give as many opportunities as a large one.

The branching factor For now, it is set by the SGAP domain expert, but can be also provided by the geometry or computed on-line. It can be computed based on the search space sizes, or the current needs of the algorithm, by extending some of them if no solution was found.

Postponing the motion planning Lagriffoul et al. (2013) argue that systematically computing the geometric part alongside the symbolic part may not be always efficient. A possible approach may be to partially link the planners by enabling the geometric level to compute partially the actions (just the world states, without the trajectories) and calling the motion plan at the end. The GRP framework already enables computing actions without motion plans, the challenge is to choose when to call the motion plan or not, and the behaviour in case a postponed motion plan call fails.

4.6 Contributions to the symbolic geometric planning in a nutshell

In this chapter, we propose two main contributions:

A symbolic geometric planning algorithm named Symbolic Geometric Action Planner (SGAP) which combines HATP with the GRP framework from the previous chapter, by linking symbolic operators to geometric actions and computing the shared predicates from the resulting world states. This enables the planner to tackle challenges such as the ramification problem.

Enhancement of search in SGAP We also proposed some specific enhancements based on a tighter communication between the layers. The first enhancement consisted on providing the symbolic planner with possible request to the geometric reasoner enabling it to check the potential feasibility of an action (Subsection 4.4.1). The second one was about providing the geometry with more information to avoid future backtracks (Subsection 4.4.2). The last one consisted on making the geometric level give to the symbolic planner the exact cost of an action based on social rules taking explicitly the human into account to choose the best possible plan (Subsection 4.4.3).

Table 4.6 summarizes the differences between the HATP algorithm and SGAP.

Type	HATP	SGAP
Predicates	Entirely given by the domain	Partially computed from the geometry
Operators	$\langle pre, \text{eff} \rangle$	$\langle pre, act, const, \text{eff} \rangle$
Methods	Similar	Similar
Entity description	Symbolic	Same identifier as symbolic
Domain	$\langle \mathcal{M}, \mathcal{O}p, \mathcal{E} \rangle$	$\langle \mathcal{M}, \mathcal{O}p, D_g, \mathcal{E}, E \rangle$
Problem	$\langle \mathcal{D}, c_0, m(p) \rangle$	$\langle \mathcal{D}_{sgp}, c_0, ws_{init}, m(p) \rangle$
Action Solution	$\langle o, stNexts \rangle$	$\langle o, gas, tNexts \rangle$
Cost	Entirely given by the domain	Computed by the geometry
Constraints	None	Possible to add
Ramification problem	Not handled	Partially handled
Applying operators	Adds the effects to the current context	Adds the effects to the current context and computes the shared predicates from the world state
Precondition check	Checks preconditions in context	Checks preconditions in context and test actions feasibility in world states
Projecting actions	None	Tests the feasibility and create backtracking points for future alternatives

TABLE 4.6: The differences between the HATP and SGAP algorithms

Chapter 5

Conclusion

Planning in the vicinity of humans rises a number of challenges and one of them concerns the geometric planning and reasoning problems. These problems relate to the link between the high level reasoning, usually represented by the task planner but also by the supervision system, and the low level motion planning, which computes actual trajectories that the robot can execute.

The idea behind the work of this thesis is to incorporate some symbolic knowledge into the geometric reasoning in order to give both symbolic and geometric levels more leeway in their interactions. The first example depicted in this dissertation is about a specific action, that requires symbolic knowledge at geometric level, the handover (Chapter 2). The knowledge acquired while designing this task helped to build a framework generalizing the geometric reasoning and planning while providing different actions besides the handover (Chapter 3). The last part concerns how this framework has been interleaved with the higher level task planning (Chapter 4). These contributions are depicted in the followings:

Sharing the effort with the human for a handover, Section 2.3 This part presents an algorithm that computes a handover configuration (the position and arm placement of both the giver and the receiver during a handover) using a grid based approach, where the position of the receiver is sampled and the position of the giver inferred from it. The algorithm samples a large variety of possible handover configurations and chooses the best one based on a human aware cost including the human comfort (such as posture and displacement), the distance between the giver and the receiver, the visibility of the giver by the receiver, and the *mobility* parameter, which is an expression of the task urgency. A user study was also held to determine the interest of this last parameter.

Multi-agent handover, Section 2.4 As one handover didn't seem enough in some occasions, an algorithm able to compute a solution where multiple agents are involved into a sequence of handovers was designed. The algorithm is based on a lazy weighted A*, searching a path in a graph where each node represents an agent holding the object and the edges represent the possible transitions: either a navigation action, or a handover action. After a solution is found a post process is triggered in order to optimize the schedule and avoid all possible collisions.

The handover gaze cues, Section 2.5 We propose a user study where the gaze cues during the object exchange are considered in details. In the user study, the subjects were asked to assess the naturalness of videos while equipped with an eye tracker enabling us to track their eye pattern during the action. In the videos, the giver (which was, for half of the subjects, a human and for the other half, a robot) placed an object in front of the subject while following one of the patterns: looking only at the **O**bject (**O**), looking only at the **R**eceiver (**R**), looking first at the **O**bject, then at the **R**eceiver (**OR**), looking first at the **R**eceiver, then at the **O**bject (**RO**), looking first at the **O**bject, then at the **R**eceiver, and finally back to the **O**bject (**ORO**), looking first at the **R**eceiver, then at the **O**bject, and finally back to the **R**eceiver (**ROR**). Two patterns emerge from both the subjective and objective measurements: **OR** and **ROR**.

Geometric actions formalization, Section 3.2 In this section, a proper formalization of an action, as defined at the geometric level is given. An action can be characterized as a sequence (or a parallelized sequence) of sub-actions which can be described by preconditions, search spaces and restraints. An action needs a world state (a snapshot of the current state of every entity) to be

defined. In order to compute an action, its preconditions need to be true in this world state, and a trajectory must be found in its search spaces between this world state and a final world state computed based on its restraints. The result is a geometric task which is a sequence (or a parallelized sequence) of trajectories coupled with geometric causal links (ensuring the precedence of each trajectory).

A framework using this formalization, Section 3.3 The formalization described in the previous contribution was used to design a framework able to compute a number of basic actions such as pick, place, placeReachable, navigateTo. Three algorithms are proposed, where the first one goes over all the sub-actions one by one and tries to find a solution for each one. The second algorithm, the one implemented, finds all the transition world states between all the sub-actions of an action and then computes the trajectories between them. The third algorithm computes all the possible sequences of world states and, based on a human aware cost, computes the trajectories for the best feasible one.

A symbolic geometric planning algorithm, Section 4.3 An algorithm which combines a Hierarchical Task Network planner with the previously defined framework is depicted. The planner defines its basic operators as preconditions and effects. In order to apply an operator, the preconditions are tested in the current context and the effects are added to it in order to obtain the new context where the next operator can be applied. In order to achieve the combination, we added an action to the operator description, which is evaluated at the same time as the preconditions. To evaluate it, an external call to the geometric framework is done. Once an operator's action is computed, the resulting world state is used to retrieve the shared predicates (predicates computed at geometric level and used by the symbolic level) which are added to the resulting context alongside the operator's effects. These additions enable a combination between the two levels and enable us to tackle partially the ramification problem.

Enhancement of the SGP algorithm, Section 4.4 We also proposed some specific enhancements based on a tighter communication between the symbolic and the geometric layers. The first enhancement consisted on providing the symbolic planner with possible requests to the geometric reasoner enabling it to validate the feasibility of an action/operator. The second one was about providing the geometry with more information to avoid future backtracks by using higher level actions such as placeReachable rather than place and using constraints to limit the search space and restraint of specific action's operator. The last one consisted on making the geometric level give to the symbolic planner the exact cost of an action based on social rules taking explicitly the human into account to choose the best possible plan among all the feasible ones.

Bibliography

Admoni, H., Dragan, A., Srinivasa, S. S., and Scassellati, B. (2014). Deliberate delays during robot-to-human handovers improve compliance with gaze communication. In *Proc. 2014 ACM/IEEE Int. Conf. Human-robot Interact. - HRI '14*. 15

Alami, R., Chatila, R., Fleury, S., Ghallab, M., and Ingrand, F. (1998). An Architecture for Autonomy. *Int. J. Rob. Res.* 65

Alami, R., Gharbi, M., and Vadant, B. (2014). On human-aware task and motion planning abilities for a teammate robot. In *Rss*. 5

Aleotti, J., Micelli, V., and Caselli, S. (2012). Comfortable robot to human object hand-over. *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* 10

Aleotti, J., Micelli, V., and Caselli, S. (2014). An Affordance Sensitive System for Robot to Human Object Handover. *Int. J. Soc. Robot.* 10

Alili, S., Pandey, A. K., Sisbot, E. A., and Alami, R. (2009). Interleaving Symbolic and Geometric Reasoning for a Robotic Assistant. *Assoc. Adv. Artif. Intell.* 105, 109

Barry, J., Kaelbling, L. P., and Lozano-Perez, T. (2013). A hierarchical approach to manipulation with diverse actions. In *Proc. - IEEE Int. Conf. Robot. Autom.* 107, 108

Basili, P., Huber, M., Brandt, T., Hirche, S., and Glasauer, S. (2009). Investigating Human-Human Approach and Hand-Over. *Hum. Centered Robot Syst.* 13

Batista, D. C. T. (2011). SHOP2 : An HTN Planning System. *System*. 118

Becchio, C., Sartori, L., and Castiello, U. (2010). Toward You: The Social Side of Actions. *Curr. Dir. Psychol. Sci.* 52

Berenson, D., Kuffner, J., and Choset, H. (2008). An optimization approach to planning for mobile manipulation. *2008 IEEE Int. Conf. Robot. Autom.* 10

Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: a review. *Proc. 2000 ICRA. Millenn. Conf. IEEE Int. Conf. Robot. Autom. Symp. Proc. (Cat. No.00CH37065)*. 10

Bidot, J., Karlsson, L., Lagriffoul, F., and Saffiotti, A. (2015). Geometric backtracking for combined task and motion planning in robotic systems. *Artif. Intell.* 106, 108

Boeuf, A., Cort??s, J., Alami, R., and Sim??on, T. (2014). Planning agile motions for quadrotors in constrained environments. In *IEEE Int. Conf. Intell. Robot. Syst.* 94

Boucher, J. D., Pattacini, U., Lelong, A., Bailly, G., Elisei, F., Fagel, S., Dominey, P. F., and Ventre-Dominey, J. (2012). I reach faster when i see you look: Gaze effects in human-human and human-robot face-to-face cooperation. *Front. Neurorobot.* 14, 52, 53, 56, 60

Bradley, M. B., Miccoli, L. M., Escrig, M. a., and Lang, P. J. (2008). The pupil as a measure of emotional arousal and automatic activation. *Psychophysiology.* 60

Broquère, X., Sidobre, D., and Herrera-Aguilar, I. (2008). Soft motion trajectory planner for service manipulator robot. *2008 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS.* 14

Cabibihan, J. J., Wu, K. W., and Ramalingam, A. (2013). Tactile sensing in an object passing task. In *Proc. 2013 IEEE Conf. Cybern. Intell. Syst. CIS 2013.* 14

Cakmak, M., Srinivasa, S. S., Lee, M. K., Forlizzi, J., and Kiesler, S. (2011a). Human preferences for robot-human hand-over configurations. *IEEE Int. Conf. Intell. Robot. Syst.* 13

Cakmak, M., Srinivasa, S. S., Lee, M. K., Kiesler, S., and Forlizzi, J. (2011b). Using spatial and temporal contrast for fluent robot-human hand-overs. *Proc. 6th Int. Conf. Human-robot Interact. - HRI '11.* 14

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., and Patoglu, V. (2009a). Bridging the gap between high-level reasoning and low-level control. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics).* 103, 108

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., and Patoglu, V. (2009b). From Discrete Task Plans to Continuous Trajectories. In *ICAPS Work. Bridg. Gap Between Task Motion Plan.* 103, 108

Cambon, S., Alami, R., and Gravot, F. (2009). A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *Int. J. Rob. Res.* 107, 109

Cambon, S., Gravot, F., and Alami, R. (2003). Overview of aSyMov: Integrating motion, manipulation and task planning. In *ICAPS Dr. Consort.* 102, 109

Cambon, S., Gravot, F., and Alami, R. (2004). A Robot Task Planner that Merges Symbolic and Geometric Reasoning. In *Eur. Conf. Artif. Intell.* 107, 109

Chan, W. P., Kakiuchi, Y., Okada, K., and Inaba, M. (2014). Determining proper grasp configurations for handovers through observation of object movement patterns and inter-object interactions during usage. *IEEE Int. Conf. Intell. Robot. Syst.* 11

Chan, W. P., Kumagai, I., Nozawa, S., Kakiuchi, Y., Okada, K., and Inaba, M. (2013a). Creating socially acceptable robots: Leaning grasp configurations for object handovers from demonstrations. *Proc. IEEE Work. Adv. Robot. its Soc. Impacts, ARSO.* 11

Chan, W. P., Parker, C. a. C., Van der Loos, H. F. M., and Croft, E. a. (2013b). A human-inspired object handover controller. *Int. J. Rob. Res.* 14

Choi, J. and Amir, E. (2009). Combining planning and motion planning. In *Proc. - IEEE Int. Conf. Robot. Autom.* Ieee. 102, 106, 109

Choset, H. (1991). *Robot motion planning.* Kluwer Academic, Norwell, MA, USA. 17, 24, 102

Choset, H., Burgard, W., Hutchinson, S., Kantor, G., Kavraki, L. E., Lynch, K., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementation.* MIT press. 17

Cohen, B., Phillips, M., and Likhachev, M. (2014). Planning Single-arm Manipulations with N-Arm Robots. In *Rss.* 38, 43, 164

Coltin, B. and Veloso, M. (2014). Online pickup and delivery planning with Transfers for mobile robots. *Proc. - IEEE Int. Conf. Robot. Autom.* 38

Cosgun, A., Hermans, T., Emeli, V., and Stilman, M. (2011). Push planning for object placement on cluttered table surfaces. In *IEEE Int. Conf. Intell. Robot. Syst.* 65

De Silva, L., Gharbi, M., Pandey, A. K., and Alami, R. (2014). A new approach to combined symbolic-geometric backtracking in the context of human-robot interaction. *Proc. - IEEE Int. Conf. Robot. Autom.* 5, 101, 109

de Silva, L., Lallement, R., and Alami, R. (2015). The HATP Hierarchical Planner : Formalisation and an Initial Study of its Usability and Practicality. In *Int. Conf. Intell. Robot. Syst.* 118

De Silva, L., Pandey, A. K., and Alami, R. (2013). An interface for interleaved symbolic-geometric planning and backtracking. *IEEE Int. Conf. Intell. Robot. Syst.* 5, 101, 105, 109

Dearden, R. and Burbridge, C. (2013). An Approach for Efficient Planning of Robotic Manipulation Tasks. In *Twenty-Third Int. Conf. Autom. Plan. Sched.* 104, 109

Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artif. Intell.* 46

Dehais, F., Sisbot, E. A., Alami, R., and Causse, M. (2011). Physiological and subjective evaluation of a human-robot object hand-over task. *Appl. Ergon.* 14

Dornhege, C., Eyerich, P., Keller, T., Brenner, M., and Nebel, B. (2010). Integrating Task and Motion Planning Using Semantic Attachments. *Work. Bridg. Gap Between Task Motion Plan.* 104, 109

Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., and Nebel, B. (2012). Semantic attachments for domain-independent planning systems. *Springer Tracts Adv. Robot.* 38, 104, 109

Dornhege, C., Gissler, M., Teschner, M., and Nebel, B. (2009). Integrating symbolic and geometric planning for mobile manipulation. *2009 IEEE Int. Work. Safety, Secur. Rescue Robot. SSRR 2009.* 104, 109

Dornhege, C., Hertle, A., and Nebel, B. (2013). Lazy Evaluation and Subsumption Caching for Search-Based Integrated Task and Motion Planning. In *IROS Work. AI-based Robot.* 104, 109

Doyle, J. (1979). A truth maintenance system. *Artif. Intell.* 127

Dragan, A. D., Bauman, S., Forlizzi, J., and Srinivasa, S. S. (2015). Effects of Robot Motion on Human-Robot Collaboration. In *Proc. Tenth Annu. ACM/IEEE Int. Conf. Human-Robot Interact. - HRI '15*. 14

Dvorak, F., Bartak, R., Bit-Monnot, A., Ingrand, F., and Ghallab, M. (2014). Planning and Acting with Temporal and Hierarchical Decomposition Models. In *Proc. - Int. Conf. Tools with Artif. Intell. ICTAI*. 62

Edsinger, A. and Kemp, C. C. (2007). Human-robot interaction for cooperative manipulation: Handing objects to one another. In *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* Ieee. 13

Endo, S., Pegman, G., Burgin, M., Toumi, T., and Wing, A. M. (2012). Haptics in between-person object transfer. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*. 14

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., and Uras, T. (2011). Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. *Proc. - IEEE Int. Conf. Robot. Autom.* 104, 108

Erdem, E., Patoglu, V., and Schüller, P. (2016). A Systematic Analysis of Levels of Integration between Low-Level Reasoning and Task Planning. In *AI Commun.* 102

Erden, M. S., Leblebicio??lu, K., and Halici, U. (2004). Multi-Agent System-Based Fuzzy Controller Design with Genetic Tuning for a Mobile Manipulator Robot in the Hand Over Task. *J. Intell. Robot. Syst. Theory Appl.* 14

Eyerich, P., Keller, T., and Nebel, B. (2009). Combining Action and Motion Planning via Semantic Attachments. In *ICAPS Work. Comb. Action Motion Plan.* 109

Eyerich, P., Mattmüller, R., and Röger, G. (2012). Using the context-enhanced additive heuristic for temporal and numeric planning. *Springer Tracts Adv. Robot.* 104

Fedrizzi, a., Mosenlechner, L., Stulp, F., and Beetz, M. (2009). Transformational planning for mobile manipulation based on action-related places. In *2009 Int. Conf. Adv. Robot.* 65

Ferguson, D., Kalra, N., and Stentz, A. (2006). Replanning with RRTs. In *Proc. - IEEE Int. Conf. Robot. Autom.* 98

Ferrer-mestres, J., Frances, G., and Geffner, H. (2015). Planning with State Constraints and its Application to Combined Task and Motion Planning. -. 105, 109

Ficuciello, F., Palli, G., Melchiorri, C., and Siciliano, B. (2013). Experimental Robotics. *ISER, June*. 107, 108

Finkel, R. A. and Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta Inform.* 62

Fiore, M., Clodic, A., and Alami, R. (2016). On planning and task achievement modalities for human-robot collaboration. In *Springer Tracts Adv. Robot.* 16, 94

Fraisse, P. and Simeon, T. (2012). *Towards Socially Intelligent Robots in Human Centered Environment Beetz*. PhD thesis, INSA. 66

Furlanetto, T., Cavallo, A., Manera, V., Tversky, B., and Becchio, C. (2013). Through your eyes: incongruence of gaze and action increases spontaneous perspective taking. *Front. Hum. Neurosci.* 53

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2014a). Heuristic Search for Task and Motion Planning. In *2nd ICAPS Work. Plan. Robot. PlanRob*. 106, 108

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2014b). Heuristic Search for Task and Motion Planning. *2nd ICAPS Work. Plan. Robot. PlanRob*. 106, 108

Gaschler, A., Kessler, I., Petrick, R. P. A., and Knoll, A. (2015). Extending the Knowledge of Volumes approach to robot task planning with efficient geometric predicates. In *Robot. Autom. (ICRA), 2015 IEEE Int. Conf.* 105, 109

Gaschler, A., Petrick, R. P. A., Giuliani, M., Rickert, M., and Knoll, A. (2013a). KVP: A knowledge of volumes approach to robot task planning. In *IEEE Int. Conf. Intell. Robot. Syst.* 105, 109

Gaschler, A., Petrick, R. P. a., Kr, T., Khatib, O., and Knoll, A. (2013b). Robot Task and Motion Planning with Sets of Convex Polyhedra. *Robot. Sci. Syst. Work. Comb. Robot Motion Plan. AI Plan. Pract. Appl.* 105, 109

Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Elsevier. 4, 10, 111, 174

Gharbi, M. and Alami, R. (2015). Combining Symbolic and Geometric Planning to synthesize human-aware plans: toward more efficient combined search. In *Int. Conf. Intell. Robot. Syst. (IROS). IEEE/RSJ*. 6, 101, 109

Gharbi, M., Lemaignan, S., Mainprice, J., and Alami, R. (2013). Natural interaction for object hand-over. In *ACM/IEEE Int. Conf. Human-Robot Interact.* Ieee. 5, 32

Gharbi, M., Paubel, P.-V., Clodic, A., Carreras, O., Alami, R., and Cellier, J.-M. (2015). Toward a better understanding of the communication cues involved in a human-robot object transfer . In *Symp. Robot Hum. Interact. Commun.* 6, 52

Gibson, J. J. (1977). The Theory of Affordances. *Perceiving, Acting, and Knowing*. 65

Goodrich, M. a. and Schultz, A. C. (2007). Human-Robot Interaction: A Survey. *Found. Trends® Human-Computer Interact.* 3

Gravot, F., Cambon, S., and Alami, R. (2005). aSyMov: A Planner that Deals with Intricate Symbolic and Geometric Problems. *Robot. Res.* 102, 109

Guitton, J. and Farges, J. (2009a). Taking into account geometric constraints for task-oriented motion planning. *BTAMP'09 (ICAPS Work.* 105, 109

Guitton, J. and Farges, J. (2009b). Towards a Hybridization of Task and Motion Planning for Robotic Architectures. In *IJCAI-09 Work.* 102

Hall, E. (1966). *The Hidden Dimension.* Doubleday, Garden City. 11, 13, 21, 22, 161

Hart, J. W., Sheikholeslami, S., Croft, E. A., Chan, W. P., and Pan, M. K. X. J. (2015). Predictions of Human Task Performance and Handover Trajectories for Human-Robot Interaction [Extended Abstract]. *HRI Work. Timing Human-Robot Teaming.* 13

Haspalamutgil, K., Palaz, C., and Uras, T. (2010). A tight integration of task planning and motion planning in an execution monitoring framework. In *AAAI Work. Bridg. Gap Between Task Motion Plan.* 103, 108

Hauser, K. (2010). Task Planning with Continuous Actions and Nondeterministic Motion Planning Queries. In *AAAI Work. Bridg. Gap Between Task Motion Plan.* 107, 109

Hauser, K. and Latombe, J.-C. (2009). Integrating task and prm motion planning: Dealing with many infeasible motion planning queries. In *Int. Conf. Autom. Plan. Sched.* 102, 106, 107, 109

Havur, G., Haspalamutgil, K., Palaz, C., Erdem, E., and Patoglu, V. (2013). A case study on the Tower of Hanoi challenge: Representation, reasoning and execution. In *Proc. - IEEE Int. Conf. Robot. Autom.* 104, 109

He, W. and Sidobre, D. (2015). Synchronization of grasp-release by online force classification for interactive object exchange. *J. Human-Robot Interact.* 14

Hertle, A., Dornhege, C., Keller, T., and Nebel, B. (2012). Planning with semantic attachments: An object-oriented view. *Front. Artif. Intell. Appl.* 104

Hoffmann, J. and Nebel, B. (2001). the Ff Planning System. *J. Artif. Intell.* 104, 106

Huber, M., Knoll, A., Brandt, T., and Glasauer, S. (2009). Handing over a cube: Spatial features of physical joint-action. In *Ann. N. Y. Acad. Sci.* 13

Huber, M., Rickert, M., Knoll, A., Brandt, T., and Glasauer, S. (2008). Human-robot interaction in handing-over tasks. *Proc. 17th IEEE Int. Symp. Robot Hum. Interact. Commun. RO-MAN.* 14, 15

Imai, M., Kanda, T., Ono, T., Ishiguro, H., and Mase, K. (2002). Robot mediated round table: Analysis of the effect of robot's gaze. *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* 55

Jindai, M., Ota, S., and Ikemoto, Y. (2015). Hand-over Motion Model Based on Timing between Voice Utterances and Release Motions of Humans. In -. 14

Kaelbling, L. P. and Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *Proc. - IEEE Int. Conf. Robot. Autom.* Ieee. 38, 105, 108

Kaelbling, L. P. and Lozano-Perez, T. (2011). Hierarchical task and motion planning in the now. *Proc. - IEEE Int. Conf. Robot. Autom.* 105, 108

Kaelbling, L. P. and Lozano-Pérez, T. (2013). Integrated task and motion planning in belief space. *Int. J. Rob. Res.* 105, 108

Kahneman, D. and Beatty, J. (1966). Pupil diameter and load on memory. *Science.* 60

Kajikawa, S., Saito, N., and Okano, H. (2002). Receiver robot's motion for handing-over with a human. *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* 14

Karami, A. (2014). *Modeles Decisionnels d'Interaction Romme-Robot.* PhD thesis, University of Caen. 16

Karlsson, L., Bidot, J., Lagriffoul, F., Saffiotti, A., Hillenbrand, U., and Schmidt, F. (2012). Combining Task and Path Planning for a Humanoid Two-arm Robotic System. In *TAMPRA 2012 Proc. Work. Comb. Task Motion Plan. Real-World Appl.* 38, 106, 108

Kavraki, L. E., ??vestka, P., Latombe, J. C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. 106

Kiesler, S., Powers, A., Fussell, S. R., and Torrey, C. (2008). Anthropomorphic Interactions with a Robot and Robot–like Agent. *Soc. Cogn.* 54

Kim, J., Park, J., Hwang, Y., and Lee, M. (2004). Advanced Grasp Planning for Handover Operation Between Human and Robot: Three Handover Methods in Esteem Etiquettes Using Dual Arms and Hands of Home-Service Robot. In *2nd Int. Conf. Auton. Robot. Agents.* 10

Kirchner, N., Alempijevic, a., and Dissanayake, G. (2011). Nonverbal robot-group interaction using an imitated gaze cue. In *Human-Robot Interact. (HRI), 2011 6th ACM/IEEE Int. Conf.* 12

Koay, K. L., Sisbot, E. A., Syrdal, D. S., Walters, M. L., Dautenhahn, K., and Alami, R. (2007). Exploratory Study of a Robot Approaching a Person in the Context of Handing Over an Object. *AAAI spring Symp. Multidiscip. Collab. Soc. Assist. Robot.* 11, 21, 22

Koene, A., Remazeilles, A., Prada, M., Garzo, A., Puerto, M., Endo, S., and Wing, A. M. (2014). Relative importance of spatial and temporal precision for user satisfaction in human-robot object handover interactions. In *Third Int. Symp. New Front. Human-Robot Interact.*, London. 13, 14

Kruse, T., Pandey, A. K., Alami, R., and Kirsch, A. (2013). Human-aware robot navigation: A survey. *Rob. Auton. Syst.* 11

Lagriffoul, F. (2013). Delegating Geometric Reasoning to the Task Planner. *ICAPS Work. Plan. Robot.* 103, 108

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., and Karlsson, L. (2014). Efficiently combining task and motion planning using geometric constraints. *Int. J. Rob. Res.* 102, 104, 108

Lagriffoul, F., Dimitrov, D., Saffiotti, A., and Karlsson, L. (2012). Constraint propagation on interval bounds for dealing with geometric backtracking. In *IEEE Int. Conf. Intell. Robot. Syst.* 104, 108

Lagriffoul, F., Karlsson, L., Bidot, J., and Saffiotti, A. (2013). Combining Task and Motion Planning is Not Always a Good Idea. In *RSS Work. Comb. Robot Motion Plan. AI Plan. Pract. Appl.* 38, 107, 141

Lallement, R., De Silva, L., and Alami, R. (2014). HATP: An HTN Planner for Robotics. In *ICAPS Work. Plan. Robot.* 111

LaValle, S. (1998). Rapidly-Exploring Random Trees: A New Tool for Path Planning. -. 98

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge Univ Pr. 4, 17, 40, 65, 67, 93, 106

Lee, M. K., Forlizzi, J., Kiesler, S., Cakmak, M., and Srinivasa, S. (2011). Predictability or adaptivity?: designing robot handoffs modeled from trained dogs and people. In *Human-robot Interact.* 12

Lemaignan, S., Gharbi, M., Mainprice, J., Herrb, M., and Alami, R. (2012a). Roboscopie. In *Proc. seventh Annu. ACM/IEEE Int. Conf. Human-Robot Interact. - HRI '12.* 5

Lemaignan, S., Ros, R., Mösenlechner, L., Alami, R., and Beetz, M. (2010). ORO, a knowledge management platform for cognitive architectures in robotics. In *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc.* 32

Lemaignan, S., Ros, R., Sisbot, E. A., Alami, R., and Beetz, M. (2012b). Grounding the Interaction: Anchoring Situated Discourse in Everyday Human-Robot Interaction. *Int. J. Soc. Robot.* 32

Likhachev, M., Gordon, G., and Thrun, S. (2004). ARA•: Anytime A• with provable bounds on sub-optimality. *Adv. Neural Inf. Process. Syst.* 43

Lozano-Perez, T., Jones, J., Mazer, E., O'Donnell, P., Grimson, W., Tournassoud, P., and Lanusse, a. (1987). Handey: A robot system that recognizes, plans, and manipulates. *Proceedings. 1987 IEEE Int. Conf. Robot. Autom.* 65

Lozano-P??rez, T. and Kaelbling, L. P. (2014). A constraint-based method for solving sequential manipulation planning problems. In *IEEE Int. Conf. Intell. Robot. Syst.* 102, 103, 108

Mainprice, J., Gharbi, M., Simeon, T., and Alami, R. (2012). Sharing effort in planning human-robot handover tasks. *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* 5, 17

Mainprice, J., Sisbot, E., Siméon, T., and Alami, R. (2010). Planning safe and legible hand-over motions for human-robot interaction. *IARP Work. Tech. ....* 14

Mainprice, J., Sisbot, E. A., Jaillet, L., Cort??s, J., Alami, R., and Sim??on, T. (2011). Planning human-aware motions using a sampling-based costmap planner. *Proc. - IEEE Int. Conf. Robot. Autom.* 21, 177

Marin-Urias, L. F., Akin Sisbot, E., and Alami, R. (2008). Geometric tools for perspective taking for human-robot interaction. In *7th Mex. Int. Conf. Artif. Intell. - Proc. Spec. Sess. MICAI 2008.* 65

Marler, R. T., Rahmatalla, S., Shanahan, M., and Abdel-malek, K. (2005). A New Discomfort Function for Optimization-Based Posture Prediction. Technical report, SAE Technical Paper. x, 22, 23, 91

Mason, A. H. and MacKenzie, C. L. (2005). Grip forces when passing an object to a partner. *Exp. Brain Res.* 14

McIlraith, S. A. (2000). Integrating actions and state constraints: a closed-form solution to the ramification problem (sometimes). *Artif. Intell.* 127

Micelli, V., Strabala, K., and Srinivasa, S. (2011). Perception and control challenges for effective human–robot handoffs. *RSS 2011 RGB-D Work.* 13, 14

Miller, A. T., Knoop, S., Christensen, H. I., and Allen, P. K. (2003). Automatic grasp planning using shape primitives. *Robot. Autom. 2003. Proceedings. ICRA'03. IEEE Int. Conf.* 76

Moon, A., Troniak, D. M., Gleeson, B., Pan, M. K. X. J., Zeng, M., Blumer, B. A., MacLean, K., and Croft, E. A. (2014). Meet me where i'm gazing: how shared attention gaze affects human-robot handover timing. In *Human-Robot Interact.* 14, 52, 53, 56, 60

Mutlu, B. (2009). *Designing gaze behavior for humanlike robots.* PhD thesis, Ford Motor Company. 52, 53

Nau, D., Cao, Y., Lotem, A., and Muftoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. In *IJCAI Int. Jt. Conf. Artif. Intell.* 111

Nedunuri, S., Prabhu, S., Moll, M., Chaudhuri, S., and Kavraki, L. E. (2014). SMT-based synthesis of integrated task and motion plans from plan outlines. In *Proc. - IEEE Int. Conf. Robot. Autom.* 102, 106, 108

Palinko, O., Sciutti, A., Rea, F., and Sandini, G. (2014). Weight-Aware Robot Motion Planning for Lift-to-Pass Action. In *Proc. Second Int. Conf. Human-agent Interact.* 14

Pandey, A. K., Saut, J. P., Sidobre, D., and Alami, R. (2012). Towards planning Human-Robot Interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement. *Proc. IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatronics.* 10

Phillips, M., Dornbush, A., Chitta, S., and Likhachev, M. (2013). Anytime incremental planning with E-Graphs. In *Proc. - IEEE Int. Conf. Robot. Autom.* 98

Plaku, E. (2012a). Planning in discrete and continuous spaces: From LTL tasks to robot motions. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics).* 106, 109

Plaku, E. (2012b). Planning Robot Motions to Satisfy Linear Temporal Logic, Geometric, and Differential Constraints. In *ICAPS Work. Comb. Task Motion Plan. Real-World Appl.*, Sao Paolo, Brazil. 69, 106, 109

Plaku, E. and Hager, G. D. (2010). Sampling-based motion and symbolic action planning with geometric and differential constraints. *Proc. - IEEE Int. Conf. Robot. Autom.* 106, 109

Prada, M., Remazeilles, A., Koene, A., and Endo, S. (2014). Implementation and experimental validation of Dynamic Movement Primitives for object handover. *IEEE Int. Conf. Intell. Robot. Syst.* 13, 14

Sahin, E., Cakmak, M., Dogar, M. R., Ugur, E., and Ucoluk, G. (2007). To Afford or Not to Afford: A New Formalization of Affordances Toward Affordance-Based Robot Control. *Adapt. Behav.* 65, 71

Savelsbergh, M. W. P. and Sol, M. (1995). The General Pickup and Delivery Problem. *Transp. Sci.* 38

Sebanz, N., Bekkering, H., and Knoblich, G. (2006). Joint action: Bodies and minds moving together. *Trends Cogn. Sci.* 9

Shi, C., Shiomi, M., Smith, C., Kanda, T., and Ishiguro, H. (2013). A Model of Distributional Handing Interaction for a Mobile Robot. In *Robot. Sci. Syst.* 11

Shibata, S., Tanaka, K., and Shimizu, A. (1995). Experimental analysis of handing over. *Robot Hum. Commun. 1995. RO-MAN'95 TOKYO, Proceedings., 4th IEEE Int. Work.* 14

Shivashankar, V., Kaipa, K. N., Nau, D. S., and Gupta, S. K. (2014). Towards Integrating Hierarchical Goal Networks and Motion Planners to Support Planning for Human-Robot Teams. In *Int. Conf. Intell. Robot. Syst.* 105, 109

Silva, L. D., Pandey, A. K., Gharbi, M., and Alami, R. (2013). Towards Combining HTN Planning and Geometric Task Planning. In *RSS-Workshop*. 109

Simeon, T. (2004). Manipulation Planning with Probabilistic Roadmaps. *Int. J. Rob. Res.* 65

Siméon, T., Laumond, J.-P., and Lamiraux, F. (2001). Move3D: A generic platform for path planning. In *IEEE Int. Symp. Assem. Task Plan. (ISATP)*. 29, 79

Sisbot, E. A., Clodic, A., Alami, R., and Ransan, M. (2008). Supervision and motion planning for a mobile manipulator interacting with humans. In *Proc. 3rd Int. Conf. Hum. Robot Interact. - HRI '08*. 16

Sisbot, E. A., Marin, L. F., and Alami, R. (2007a). Spatial reasoning for human robot interaction. *2007 IEEE/RSJ Int. Conf. Intell. Robot. Syst.* 11, 13

Sisbot, E. A., Marin-Urias, K. F., Alami, R., and Siméon, T. (2007b). A human aware mobile robot motion planner. *IEEE Trans. Robot.* 21, 72, 136, 177

Sisbot, E. A., Marin-Urias, L. F., Broqu??re, X., Sidobre, D., and Alami, R. (2010). Synthesizing robot motions adapted to human presence: A planning and control framework for safe and socially acceptable robot motions. *Int. J. Soc. Robot.* 11

Sisbot, E. A., Ros, R., and Alami, R. (2011). Situation assessment for human-robot interactive object manipulation. In *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* 91, 95

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2013a). A Modular Approach to Task and Motion Planning with an Extensible Planner-Independent Interface Layer (Full Version). In -. 104, 108

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. - IEEE Int. Conf. Robot. Autom.* 104, 108

Srivastava, S., Riano, L., Russell, S., and Abbeel, P. (2013b). Using Classical Planners for Tasks with Continuous Operators in Robotics. In *ICAPS Work. Plan. Robot.* 103, 108

Staudte, M. and Crocker, M. W. (2009). Visual Attention in Spoken Human-Robot Interaction. *Proc. 4th ACM/IEEE Int. Conf. Human-Robot Interact. (HRI 09)*. 52

Strabala, K., Lee, M. K., Dragan, A., Forlizzi, J., Srinavasa, S. S., Cakmak, M., and Micelli, V. (2012a). Towards Seamless Human-Robot Handovers. *J. Human-Robot Interact.* 15, 52, 60

Strabala, K., Lee, M. K., Dragan, A., Forlizzi, J., and Srinivasa, S. S. (2012b). Learning the communication of intent prior to physical collaboration. *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* 12

Sucan, I. A. and Kavraki, L. E. (2012). Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs. In *Proc. - IEEE Int. Conf. Robot. Autom.* 102, 103, 108

Tenorth, M. and Beetz, M. (2009). KNOWROB - Knowledge processing for autonomous personal robots. *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*. 127

Vadant, B., Gharbi, M., Alami, R., and Sim, T. (2014). Being safe around the robot. In *Work. Algorithmic Hum. Robot Interact. Int. Conf. Human-Robot Interact.* 6

Vesper, C., Butterfill, S., Knoblich, G., and Sebanz, N. (2010). A minimal architecture for joint action. *Neural Networks*. 52, 53

Waldhart, J., Gharbi, M., and Alami, R. (2015). Planning handovers involving humans and robots in constrained environment. In *Int. Conf. Intell. Robot. Syst. (IROS). IEEE/RSJ*. 6, 38

Walters, M. L., Dautenhahn, K., Woods, S. N., and Koay, K. L. (2007). Robotic etiquette. In *Proceeding ACM/IEEE Int. Conf. Human-robot Interact. - HRI '07*. 11

Warnier, M., Guitton, J., Lemaignan, S., and Alami, R. (2012). When the robot puts itself in your shoes. Managing and exploiting human and robot beliefs. *Proc. - IEEE Int. Work. Robot Hum. Interact. Commun.* 91

Willow Garage (2008). https://www.willowgarage.com/. 3, 32, 34, 38

Wolfe, J., Marthi, B., and Russell, S. (2010). Combined task and motion planning for mobile manipulation. In *Int. Conf. Autom. Plan. Sched.* 105, 109

Woods, S., Walters, M., Koay, K., and Dautenhahn, K. (2006). Comparing human robot interaction scenarios using live and video based methods: towards a novel methodological approach. *9th IEEE Int. Work. Adv. Motion Control. 2006*. 54

Yamane, K., Revfi, M., and Asfour, T. (2013). Synthesizing object receiving motions of humanoid robots with human motion database. In *Proc. - IEEE Int. Conf. Robot. Autom.* 13

Zacharias, F., Borst, C., and Hirzinger, G. (2006). Bridging the Gap between Task Planning and Path Planning. In *2006 IEEE/RSJ Int. Conf. Intell. Robot. Syst.* Ieee. 75

Zheng, M., Moon, A. J., Gleeson, B., Troniak, D. M., Pan, M. K. X. J., Blumer, B. A., Meng, M. Q. H., and Croft, E. A. (2014). Human behavioural responses to robot head gaze during robot-to-human handovers. In *2014 IEEE Int. Conf. Robot. Biomimetics, IEEE ROBIO 2014*. 14

Zickler, S. and Veloso, M. (2009). Efficient physics-based planning: sampling search via non-deterministic tactics and skills. In *8th Int. Conf. Auton. Agents Multiagent Syst.* 106, 109

Şucan, I. A. and Kavraki, L. E. (2011). Mobile manipulation: Encoding motion planning options using task motion multigraphs. In *Proc. - IEEE Int. Conf. Robot. Autom.* 103, 108

Appendix A

French Extended Abstract

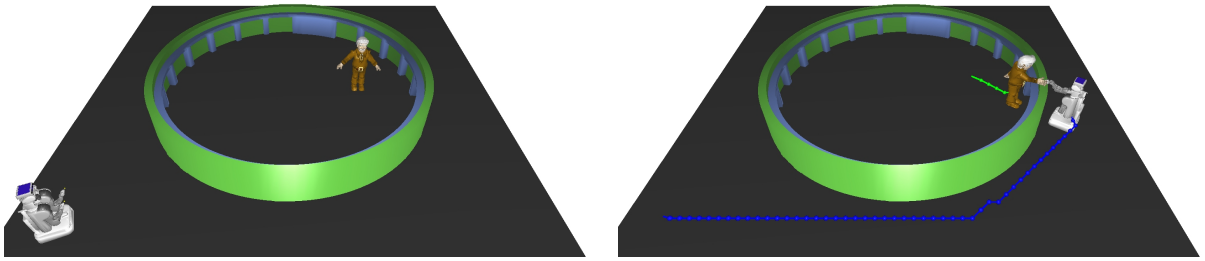
Cette thèse porte sur le raisonnement et la planification géométrique dans le contexte de l'interaction homme robot. Dans ce cadre, nous avons d'abord exploré la tâche particulière de transfert d'objet entre un robot et un humain, puis, nous avons développé un framework qui permet de planifier des actions et de raisonner au sujet des informations géométriques disponibles. Finalement, nous avons combiné ce framework avec un planificateur de tâche permettant ainsi de traiter des problèmes complexes et intéressants.

A.1 Transfert d'objet

L'action du transfert d'objet, comme son nom l'indique, est l'action où un agent (humain ou robot) donne un objet à un autre agent. Ma contribution dans ce domaine peut être divisée en trois parties. La première partie concerne l'échange d'objets entre un robot et un humain : sachant que l'endroit de l'échange n'est pas défini à l'avance, le robot doit proposer une solution proactive où il planifie le mouvement de l'humain pour s'assurer de la faisabilité de l'échange. La deuxième contribution étend la première à des problèmes incluant plusieurs robots et/ou plusieurs humains. La troisième contribution concerne 2 études utilisateurs qui nous ont permis de mieux comprendre certains comportements durant le transfert d'objet.

A.1.1 Partage d'effort durant le transfert d'objet

La figure A.1 montre un exemple où l'humain ne peut pas être atteint directement par le robot; celui-ci choisi donc une solution "intelligente" en se rapprochant au plus près de l'humain avant de lui tendre l'objet. Nous avons poussé ce raisonnement plus loin en prenant en compte les envies/besoins de l'humain afin de choisir la meilleure solution. La figure A.2 montre deux situations : dans la première, la personne préfère aller chercher l'objet au bar même si le robot pourrait venir le lui apporter à sa table comme montré dans la seconde situation.

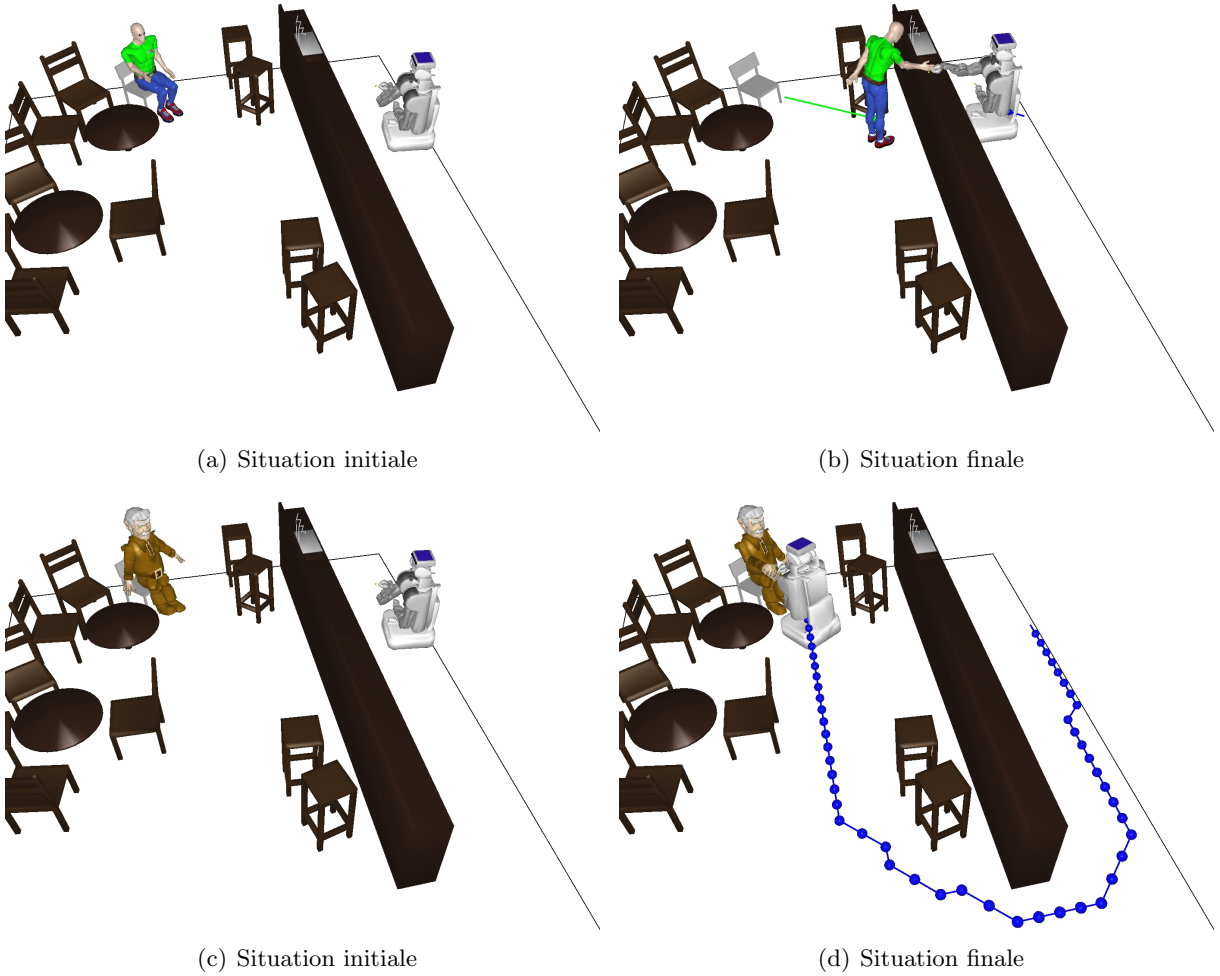


(a) Situation initiale (b) Situation finale

FIGURE A.1: Le robot ne peut pas atteindre l'humain directement, mais il lui propose une solution acceptable pour effectuer le transfert d'objet.

Afin de trouver cet emplacement où les agents pourront effectuer le transfert d'objet, nous nous basons sur deux critères : la faisabilité et la qualité.

La faisabilité : afin que le transfert d'objet soit faisable, les deux agents doivent être dans une position stable lors de l'échange et pouvoir accéder à l'objet en même temps; la position doit être sans



(a) Situation initiale (b) Situation finale

(c) Situation initiale (d) Situation finale

FIGURE A.2: Une personne pressée de récupérer sa boisson, se sentira plus à l'aise d'aller chercher sa boisson au bar, mais une personne un peu moins mobile, ou un peu moins impatiente, préférera attendre que le serveur (le robot) ramène la boisson à sa table.

collision et, finalement, les deux agents doivent pouvoir attendre la position d'échange à partir de leur positions initiales respectives.

La qualité : afin d'évaluer la qualité d'un transfert d'objet, nous allons nous baser sur les critères suivants :

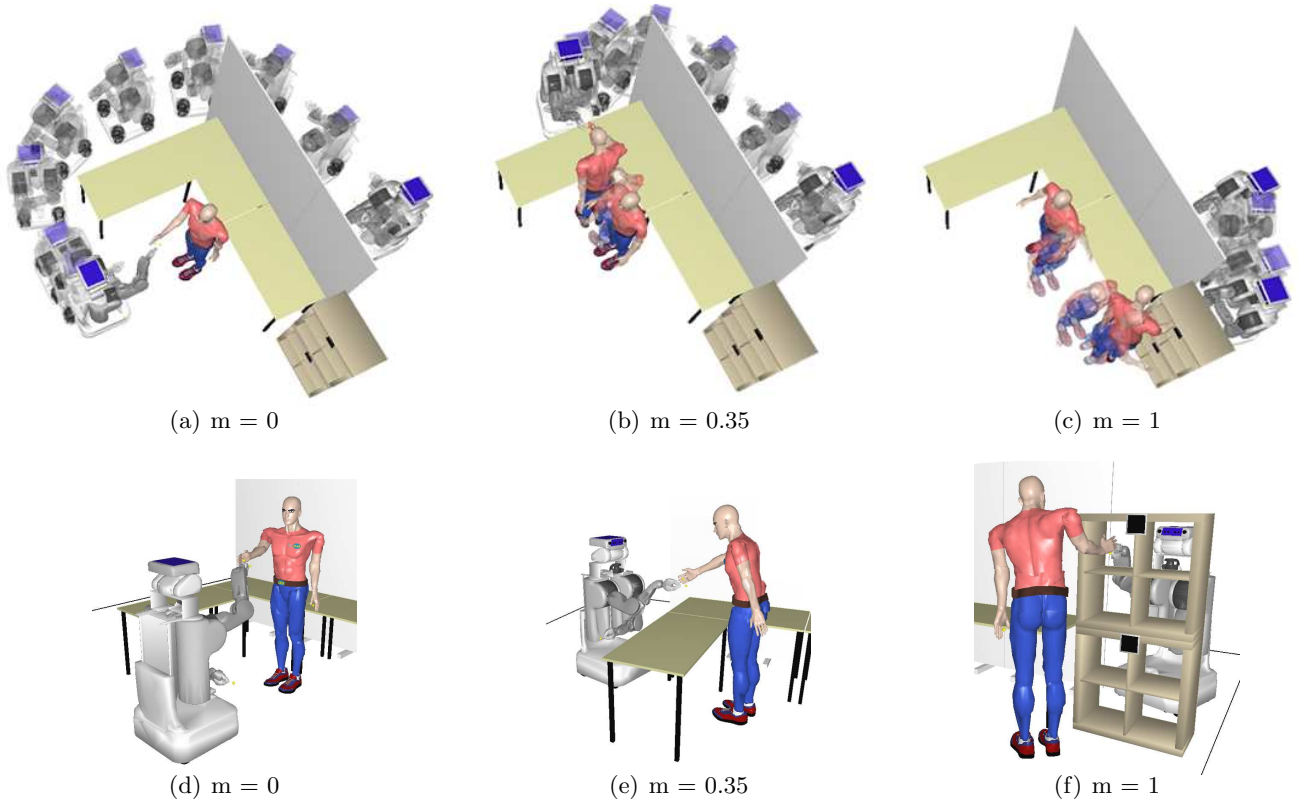
- le théorème de proximité Hall (1966)
- la visibilité du donneur par le receveur
- le confort de la position de transfert basé sur un coût musculoskeletal
- l'effort de déplacement fourni par l'humain

En plus de ces deux critères, nous voulons réaliser la tâche aussi vite que possible, et pour faire cela, la méthode supposée la plus rapide est de partager la navigation entre les deux agents. Ceci dit, ce partage est antagoniste à l'idée de minimiser l'effort de déplacement que l'humain doit fournir. Dans le but d'équilibrer ces deux critères, nous utilisons un paramètre nommé "mobilité". La mobilité est élevée pour l'exemple A.2(a), A.2(b) et elle est basse pour l'exemple A.2(c), A.2(d).

L'algorithme utilisé pour trouver une solution à ce problème correspond à une boucle qui se déroule en 4 étapes: d'abord, définir aléatoirement une position pour le receveur; ensuite, en se basant sur un ensemble de positions relatives des agents, déjà en position de transfert d'objet, nous pouvons déduire la position du donneur. Ensuite, nous calculons la trajectoire des deux agents et, finalement, nous calculons un coût en prenant en compte la qualité du transfert et la préférence de l'humain concernant la mobilité.

Afin d'améliorer les performances de cet algorithme, nous avons changé le tirage aléatoire de la position du receveur afin de la biaiser vers les positions les plus prometteuses. Pour cela, nous avons utilisé une partie des critères de qualité (lié uniquement à la navigation) pour évaluer les zones les plus intéressantes de l'espace, et y diriger nos recherches.

La figure A.3 montre le même scénario avec différentes valeurs pour la mobilité, et la figure A.4 montre les résultats obtenus.



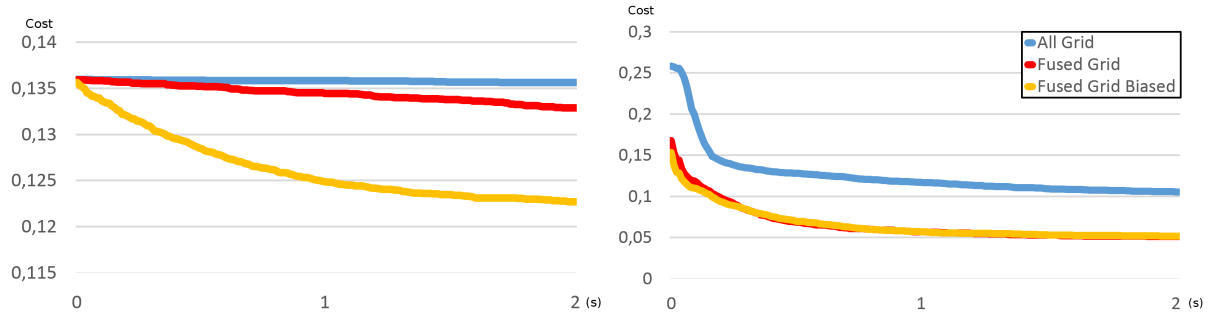
(a) m = 0 (b) m = 0.35 (c) m = 1

(d) m = 0 (e) m = 0.35 (f) m = 1

FIGURE A.3: Trois valeurs de la mobilité utilisées pour générer trois différentes stratégies de transfert d'objet. Les trois images du haut montrent les trajectoires, alors que les trois du bas montrent la position final.

A.1.2 Transfert d'objet entre divers agents

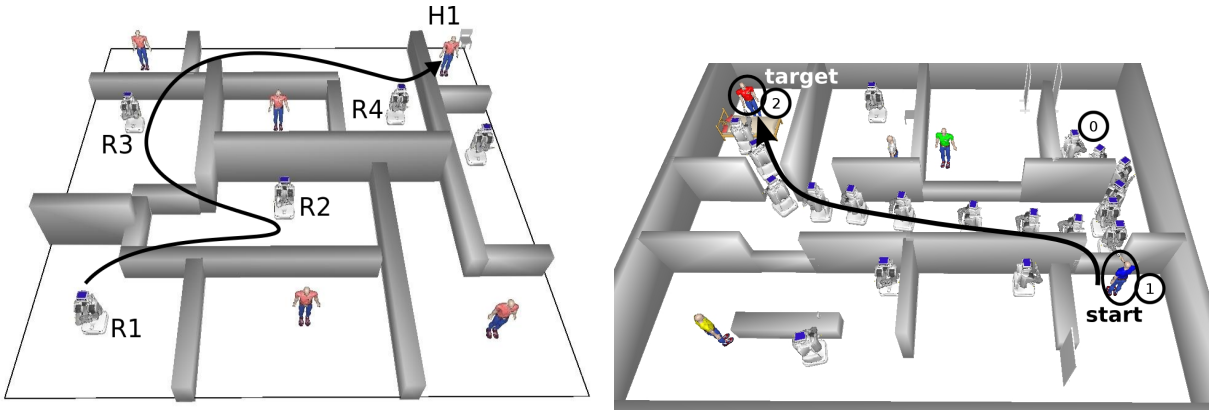
Dans le but d'étendre le travail sur le transfert d'objet, nous avons intégré la possibilité de transfert d'objet entre plusieurs agents: a partir d'un agent source, on atteint l'agent but en passant par un nombre indéfini d'agents, tout en prenant en compte leur confort et leurs occupations du moment ainsi que tous les déplacements nécessaires.



(a) mobilité=0.35 (b) mobilité=1

FIGURE A.4: Résultats concernant les temps de recherche moyennés sur 300 tests de l'algorithme. La mobilité est à 0.35 à gauche et 1 à droite.

La figure A.5 montre deux exemples où ce type d'algorithme est utile pour trouver une solution. D'autre approches peuvent trouver des solutions similaires (notamment une approche combinant la planification de mouvement et la planification de tâche) mais l'algorithme que nous proposons résout le problème de manière optimale (selon les critères choisis), et dans un délai permettant son utilisation en ligne.



(a) (b)

FIGURE A.5: (a) Un exemple où, afin d'amener l'objet de R1 à H1, l'agent R2, R3 et R4 performe une série de transferts d'objets. Dans cet exemple chaque agent est dans une zone séparée. (b) Un exemple dans un environnement où les agents sont séparés dans deux zones où ils peuvent se déplacer, et à partir desquels ils peuvent s'échanger des objets à travers une fenêtre ou au-dessus du comptoir. L'humain en bleu est en possession de l'objet désiré par l'humain en rouge. La solution trouvée par l'algorithme consiste à demander au robot 0 de naviguer jusqu'à l'humain en bleu pour attraper l'objet qu'il lui tend au-dessus du comptoir, et ensuite le ramener à l'humain en rouge.

A.1.2.1 L'approche

L'approche utilisée pour trouver une solution pour ce genre de problème se base sur une exploration de graphes. Deux graphes sont considérés; le premier, le graphe d'agent, sert à guider le deuxième que nous appellerons graphe d'état.

Le graphe d'agent permet d'avoir une idée générale et approximative des transferts d'objets possibles entre les agents. Dans un premier temps, une vérification basée seulement sur les distances permet de mettre de côté une partie des binômes candidats pour les transferts d'objet (si les zones où les agents

se trouvent trop éloigné les un des autres, l'échange est considéré infaisable). Dans un second temps, ce graphe est mis à jour régulièrement avec les informations du graphe d'état où un calcul plus fin permet de savoir si un échange d'objet entre 2 agents est possible ou non.

Le graphe d'état est constitué :

- de nœuds qui correspondent à une position associée à un agent: en d'autre terme, chaque nœud représente un agent qui tient l'objet à une position donnée. Notez qu'il est donc possible que différents agents puissent tenir l'objet à une même position (pas en même temps bien sûr).
- d'arrêtes qui peuvent être de deux types: soit un simple déplacement de l'agent avec l'objet, soit un transfert d'objet.

Afin de trouver une solution, le graphe est exploré à l'aide d'une variante paresseuse et pondérée du A* Cohen et al. (2014). L'heuristique utilisée dans cette variante, comprend bien sûr une estimation de la distance au but, mais aussi les différents transferts d'objet possibles (grâce au graphe d'agent).

A.1.2.2 Le post processing

Après avoir trouvé une solution basée sur le graphe d'état, un certain nombre d'informations nécessite encore d'être calculé:

Synchronisation Une étape de synchronisation (basées sur des règles simples) permet de trouver l'enchainement d'actions le plus efficace afin d'atteindre le but.

Trajectoire de retour Bien que les trajectoires soient calculées pour tous les agents qui tiennent l'objet, nous considérons que ceux-ci doivent revenir à leur position de départ. Ces trajectoires de retour sont donc calculées et ajoutées dans le plan.

Collision entre agents Durant le calcul du plan à l'aide du graphe d'état, les collisions entre agents sont ignorées. Durant cette phase, l'algorithme vérifie qu'il n'y a effectivement pas de collision, et si il en trouve, il essaie de trouver d'autres trajectoires pour les agents créant la collision afin de dégager le passage.

A.1.2.3 Résultats

Nous avons testé l'algorithme dans plusieurs cas d'exemples, représentés dans les figures A.5(b), A.6, A.7 et A.8. Cela a permit de trouver le temps de calcul moyen (20,8s) ainsi que de vérifier que ce temps est très dépendant de l'environnement et de la qualité de l'heuristique:

	premier exemple	labyrinthe	grande salle	robots réels
mean time (s)	11.2	17.9	40.7	13.4

TABLE A.1: le temps de calcul moyen pour chaque environnement

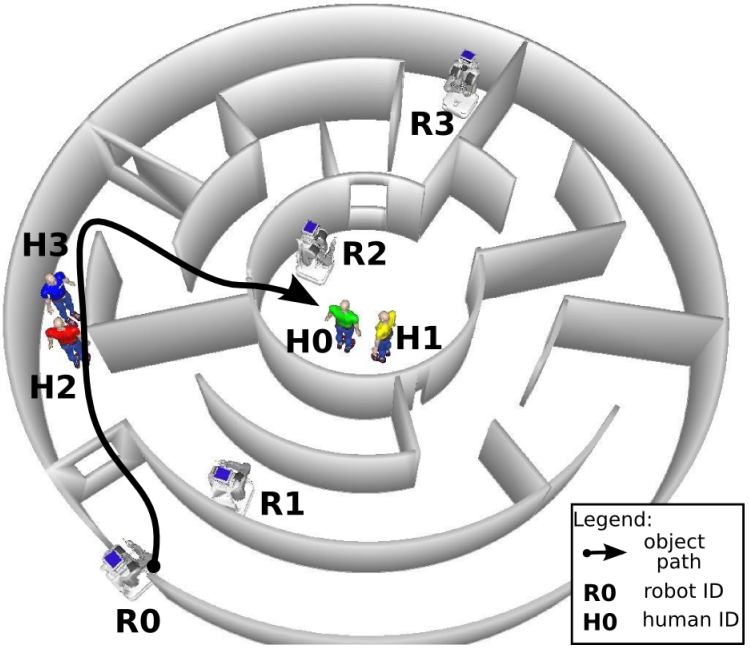
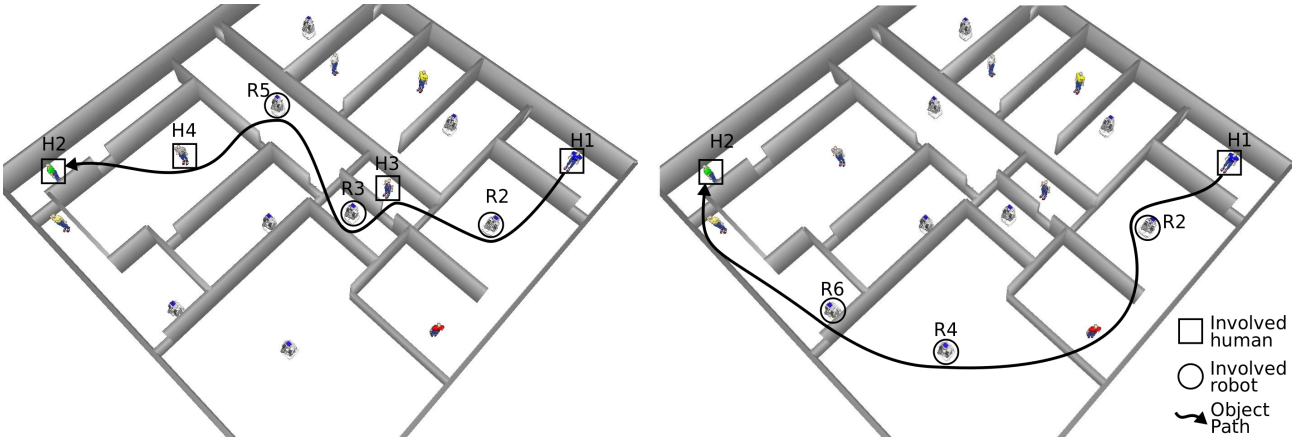


FIGURE A.6: Le labyrinthe



(a) (b)

FIGURE A.7: La grande salle

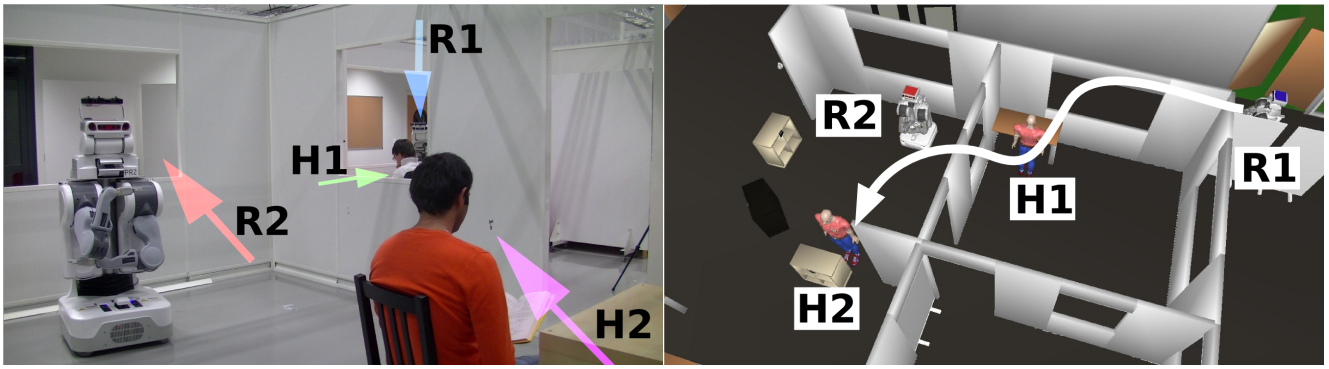


FIGURE A.8: L'implémentation sur les robots réels.

A.1.3 L'étude utilisateur concernant le regard durant le transfert d'objet

La dernière contribution concernant le transfert d'objet a été menée en coopération avec des psychologues et des ergonomistes afin d'étudier les modalités du regard durant un transfert d'objet.

Effectivement, durant ce transfert, un certain nombre d'informations transite entre le donneur et le receveur, et certaines d'entre elles sont portées par le regard. L'idée de cette étude est d'essayer d'identifier des patterns que le regard suit durant un échange d'objet.

Le reste de cette sous-section est divisé en deux parties, la première concernant la méthodologie choisie et la deuxième présentant les résultats obtenus.

A.1.3.1 Méthodologie

Afin de trouver et d'évaluer les patterns du regard permettant une interaction homme robot plus naturelle, nous avons demandé à un ensemble d'utilisateurs naïfs (Age 22–38, 21 hommes et 12 femmes) de regarder des vidéos où un agent (humain ou robot) leur tend un objet. La figure A.9 montre deux captures de vidéos qui ont servi durant l'expérience.

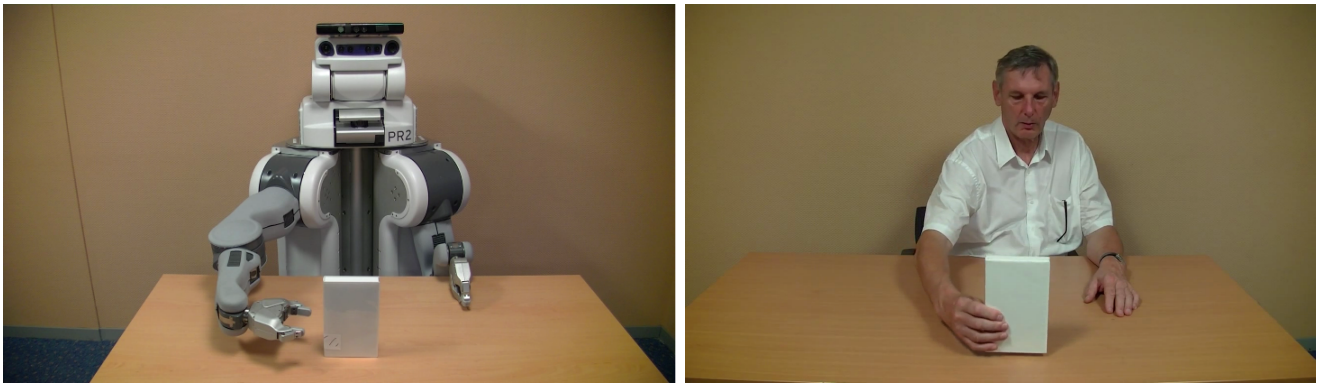


FIGURE A.9: Capture d'écran de deux vidéos utilisées durant l'expérience.

Les vidéos montrent soit l'humain soit le robot effectuant un geste pour tendre l'objet à la personne qui regarde, tout en regardant à différents endroits selon le pattern choisi. Nous avons retenu 6 patterns :

- **O** Le donneur suit l'**O**bjet tout du long.
- **R** Le donneur regarde uniquement le **R**eceveur
- **RO** Le donneur commence par regarder le **R**eceveur puis regarde l'**O**bjet.
- **ORO** Le donneur regarde d'abord l'**O**bjet puis il regarde le **R**eceveur et finalement son regard revient sur l'**O**bjet
- **OR** Le donneur regarde l'**O**bjet puis il regarde le **R**eceveur.
- **ROR** Le donneur regarde le **R**eceveur, puis il regarde l'**O**bjet et finalement il regarde de nouveau le **R**eceveur.

Les sujets ont regardé chacune des vidéos 2 fois et après la visualisation de chaque vidéo ils devaient remplir un questionnaire leur demandant d'évaluer la naturalité du mouvement. Les vidéos étaient montrées au participant dans un ordre aléatoire et une vidéo d'entrainement leur était montrée avant les tests.

Durant les évaluations, les sujets étaient équipés d'un eye tracker permettant de savoir où ils regardaient à tout moment.

A.1.3.2 Résultats

L'analyse des données subjectives (figure A.10) montre que les patterns **OR** et **ROR** sont significativement mis en avant par les sujets.

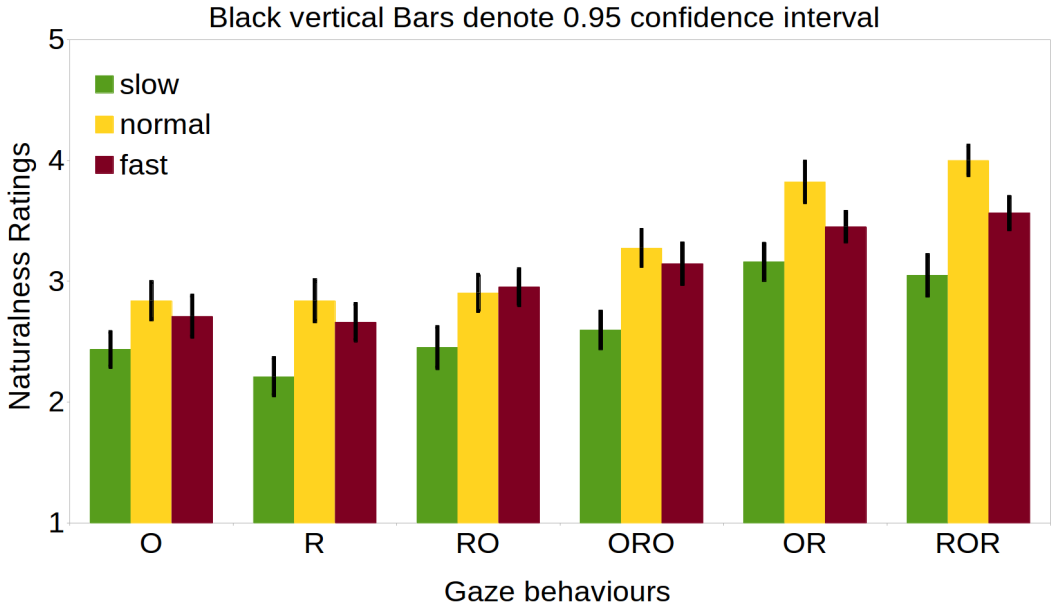


FIGURE A.10: Evaluation de la naturalité par rapport aux patterns et à la vitesse du mouvement

D'autre part, au niveau des résultats occulometriques, nous pouvons remarquer (figure A.11) que dans le cas d'un échange entre humain, le regard du receveur se porte principalement sur le visage du donneur alors que pour un échange robot-humain, le regard de l'humain receveur sera moins déterminé. Nous supposons que plus le mouvement semble naturel plus les données entre l'humain et le robot seront identiques. Nous remarquons donc que pour les pattern **OR** et **ROR** chez le robot les données sont plus proches de celles relevées pour l'humain que pour le reste des patterns. A noter que le pattern **ORO** fait exception: il est très proche des données relevées pour l'humain; nous supposons que ce comportement chez le receveur est dû à une attente d'une confirmation visuelle du robot après le dernier regard vers l'objet, et ne correspond pas exactement à ce qui est recherché.

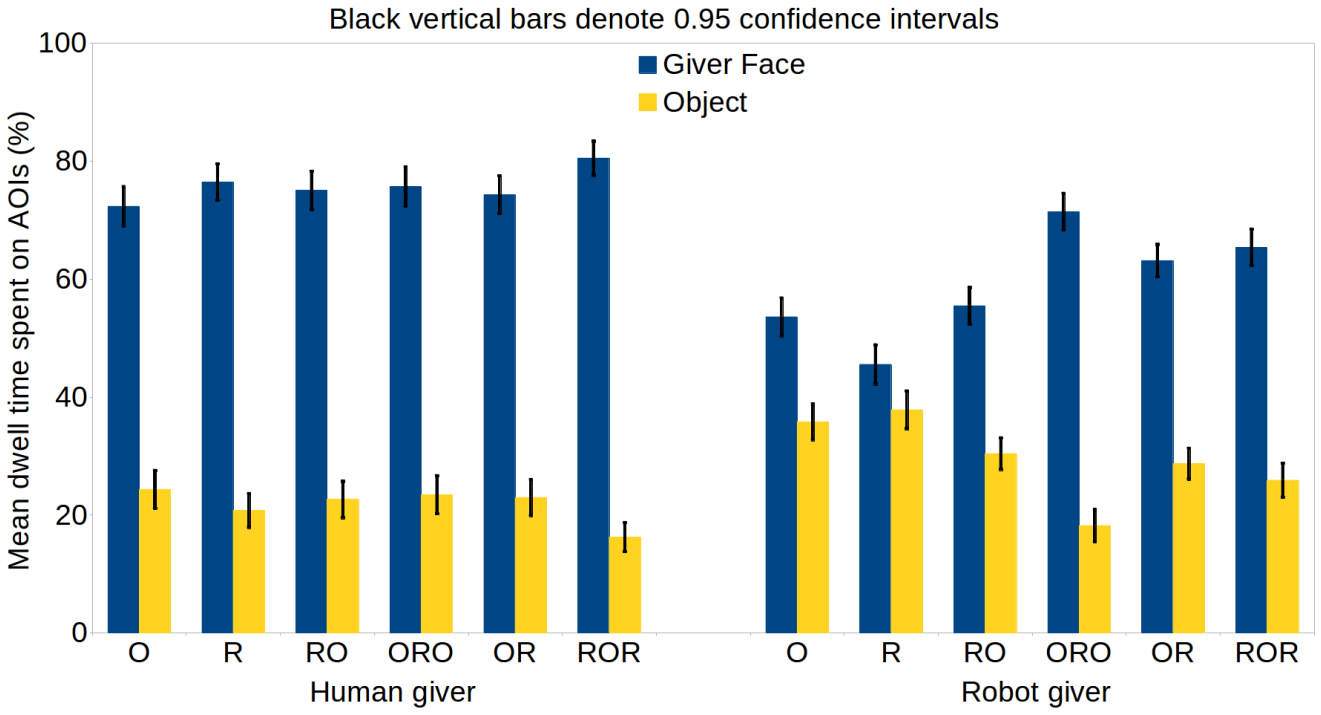


FIGURE A.11: La distribution visuelle de l'attention entre les centres d'intérêt par rapport aux différents patterns et au type du donneur.

A.2 Raisonnement et planification géométrique

Dans le cadre de la planification et du raisonnement dans l'entourage d'un humain, nous avons développé un framework. Sur la base de données symboliques simples telle que robot R1 attrape l'objet O2, ce framework est capable de calculer, non seulement les trajectoires qui permettront d'atteindre ce but, mais aussi de trouver les configurations finales de tous les agents et objets présents dans la scène et ceci en prenant en compte, au besoin, les différents placements, saisies et rotations.

Pour atteindre ce but, chaque action a été définie de manière autonome, avec des entrées et un comportement spécifique. Par exemple, pour planifier une action où le robot attrape un objet, les entrées sont l'identifiant du robot, et l'identifiant de l'objet. Comme le lecteur peut s'en douter à présent, il n'y a pas besoin de définition numérique du but à atteindre (pas de configuration définie à atteindre) ce qui sort du domaine de la planification de mouvement, et se trouve au niveau de la planification géométrique en général.

Étant donné le caractère continu (même si il est discrétisé) du domaine dans lequel la planification géométrique a lieu, la configuration finale est rarement unique, créant ce que nous appelons des alternatives: pour une seule action spécifiée, un certain nombre (dépendant de l'espace de recherche) de configurations but peut être trouvé et utilisé.

Afin de trouver des solutions aux différentes actions disponibles (Pick, Place, PlaceReachable, Stack, navigateTo, Drop) deux algorithmes ont été développés. Le premier peut être décrit ainsi:

- Trouver l'espace de recherche dont la tâche a besoin. Par exemple, pour un Pick, l'espace sera constitué des différentes saisies disponibles, pour un Place, il s'agirait de la surface de pose de l'objet en question.
- Dans une boucle avec une condition d'arrêt au nombre d'essais:
  - Tirer au hasard un point dans l'espace de recherche (pour un pick, un grasp est tiré)
  - Calculer, utilisant les techniques de cinématique inverse, les configurations utilisant ce point (pour un pick, la position du bras est calculée)
  - La chemin liant la configuration initiale et la configuration ainsi calculée est planifiée.

Si une étape de la boucle échoue (la première car toutes les possibilités ont été explorées, la deuxième car il n'existe pas de position respectant les contraintes et la troisième car il n'existe pas de chemin sans collision) l'algorithme retourne au début de la boucle jusqu'à trouver une solution ou atteindre un nombre maximum de tests.

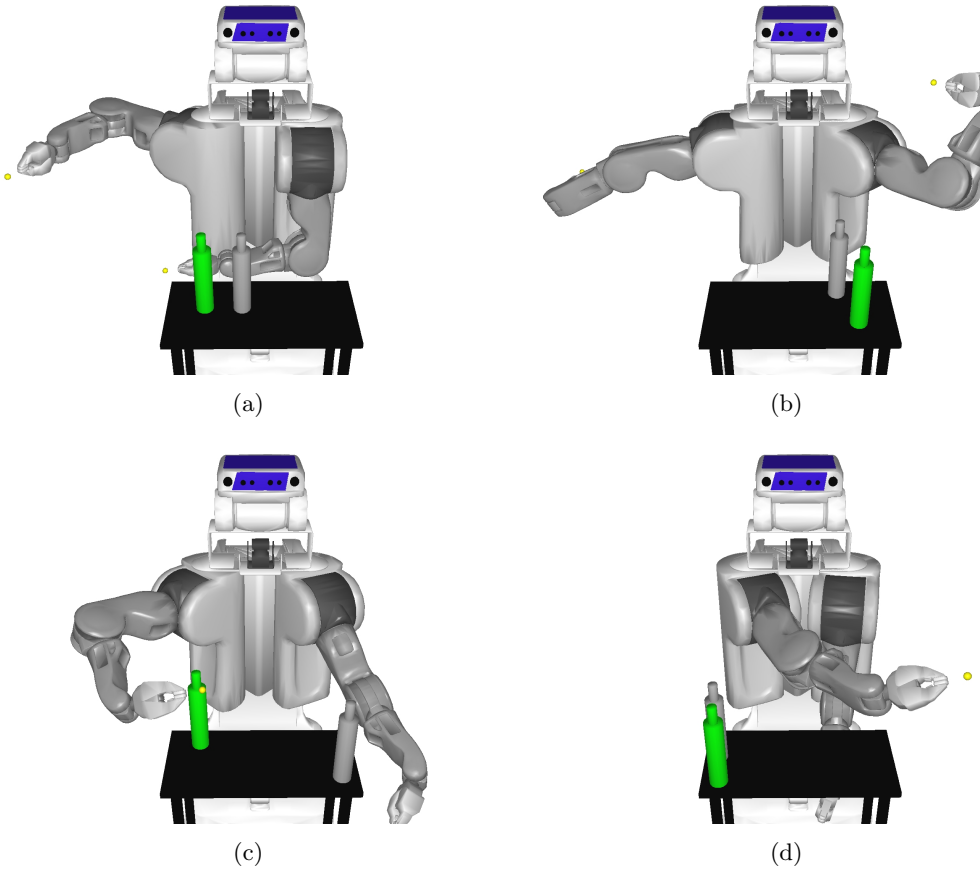
Le deuxième algorithme proposé fait intervenir une composante lié à l'homme: afin de le prendre explicitement en compte, la première étape de l'algorithme est remplacée par un choix déterministe du point présentant la meilleure possibilité. En utilisant une approche basée sur des coûts, représentant des règles sociales (tel que respecter une distance de confort/sécurité de l'homme, la visibilité du robot par l'humain et ainsi de suite), il est possible de trier les points, d'un point de vue acceptable pour l'humain, du meilleur au moins bon. Ainsi, le choix des points devient déterministe.

Le reste de cette section sera dédié aux résultats d'une part, et aux différents rôles et fonctionnalités d'autre part de ce framework.

A.2.1 Résultats et discussion

Afin de tester le planificateur, trois actions différentes ont été testées: le Pick, le Place et le Place Reachable. Pour cela, une configuration est tirée au hasard (comme représenté dans la figure A.12) et à partir de cette position, on demande au planificateur de calculer l'action (150 requêtes pour chaque action). Le Tableau A.2 montre les résultats obtenus, divisés en deux parties: à gauche sans planification de mouvement et à droite avec. La raison de cette division est de montrer la vitesse du planificateur géométrique de manière indépendante de la planification de mouvement.

Une des premières remarques est que la planification de mouvement prend presque tout le temps. D'autre part, l'action **PlaceReachable** prend plus de temps que le place due au calcul additionnelle obligatoire pour assurer l'atteignabilité de l'objet par l'autre agent (placeReachable tente de placer l'objet de telle façon à ce qu'un autre agent puisse l'atteindre). On peut aussi noter que le nombre de solutions explorées dans la partie sans planification de mouvement est significativement inférieur à celui avec la planification de mouvement: l'algorithme échoue souvent à trouver une trajectoire. Chose qui est aussi visible dans les deux derniers paramètres, le nombre d'appels à la cinématique inverse est plus grand du côté droit du tableau et le nombre d'appels moyen au planificateur de mouvement est ≈ 2 avec une variance ≈ 2.



(a) (b) (c) (d)

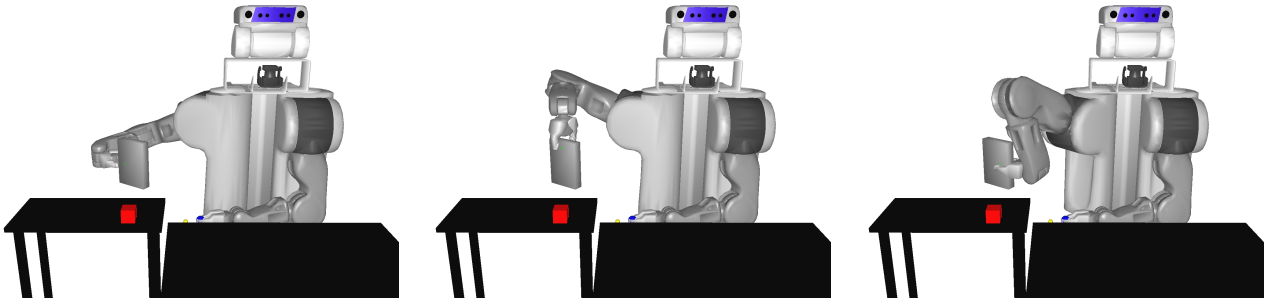
FIGURE A.12: Différents états initiaux où l'action **Pick** a été testée

pour une action	sans planification			avec planification		
Pick	moyenne	variance	écart type	moyenne	variance	écart type
Temps	0.026	0.0001	0.0108	2.8553	17.1426	4.1403
Nb Sol testé	2.525	3.6193	1.9024	8.2130	124.558	11.1606
Cinématique inverse	4.61	4.5379	2.1302	11.4556	128.899	11.3534
Planification de mouvement	-	-	-	2.0532	2.1687	1.4726
Place						
Temps	0.0201	0.0007	0.0270	2.7153	22.8922	4.7845
Nb Sol testé	4.4522	19.7352	4.4424	18.5033	1166.78	34.1582
Cinématique inverse	4.9296	7.3216	2.7058	11.7219	217.101	14.7344
Planification de mouvement	-	-	-	2.0463	2.4548	1.5667
Place Reachable						
Temps	0.0477	0.0016	0.0403	3.0798	47.1862	6.8692
Nb Sol testé	5.5577	78.4879	8.8593	12.2692	236.735	15.3862
Cinématique inverse	5.1658	10.5303	3.2450	9.4359	57.8741	7.6075
Planification de mouvement	-	-	-	1.8846	1.8969	1.3773

TABLE A.2: Temps signifie le temps de calcul, Nb Sol testé signifie le nombre de solution testées. Cinématique inverse et planification de mouvement réfère au nombre d'appels respectifs aux algorithmes correspondant. Ces chiffres sont calculés sur 150 actions réussies.

A.2.2 Alternatives

Comme signalé auparavant, le planificateur géométrique est capable de calculer non seulement une solution pour l'action, mais aussi, au besoin, différentes alternatives pour cette même action: le choix étant laissé au planificateur géométrique, il doit être aussi capable de les changer quand le besoin s'en fait sentir. La figure A.13 montre différentes alternatives de l'action **Pick**

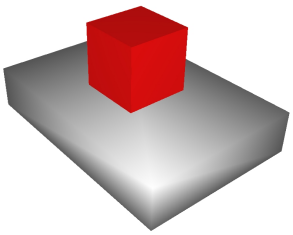


(a) Première alternative (b) Deuxième alternative (c) Troisième alternative

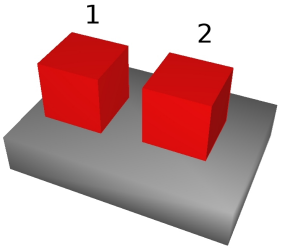
FIGURE A.13: Différentes alternatives pour l'action **Pick**

A.2.3 Faits

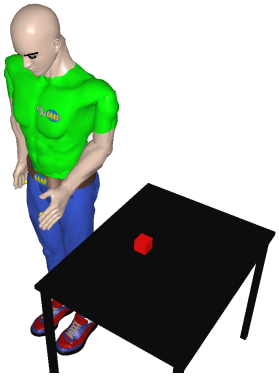
Le planificateur et raisonneur géométrique est aussi capable de calculer différents faits tel que un objet est sur un autre objet, ou un objet est dans un autre objet. Ces capacités de raisonnement englobent aussi certaines capacités des agents tels qu'un agent peut atteindre tel objet ou peut voir tel autre objet. La figure A.14 montre différents types de faits.



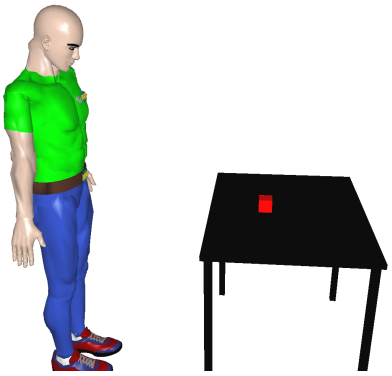
(a) {*CubeRouge*, **est sur**, *LivreGris*, *true*} le polygone qui forme le bas du *CubeRouge* est dans le polygone qui forme le haut du *LivreGris*



(b) {*CubeRouge*1, **à coté de** *RedCube*2, *true*} la distance entre *CubeRouge*1 et *CubeRouge*2 est plus petite qu'un certain seuil



(c) {*CubeRouge*, **est atteignable par**, *Humain*, *true*} La cinématique inverse de l'*Humain* lui permet d'atteindre le *CubeRouge*



(d) {*CubeRouge*, **est visible par**, *Humain*, *true*} Le *CubeRouge* Est dans le champs de vision de l'*Humain*

FIGURE A.14: Différents type de fait.

A.2.4 Plan géométrique

Le planificateur géométrique est aussi capable de créer et de maintenir un plan complet, sous forme d'arbre, où chaque branche est un plan complet, et les différentes feuilles de même niveau et de même racine sont les différentes alternatives d'une action. La figure A.15 montre un exemple de plan géométrique.

Afin de calculer ce plan, un algorithme simple de backtrack géométrique a été mis en place: il prend en entrée une suite d'action et essaye de les exécuter dans l'ordre; s'il échoue, il revient sur l'action d'avant et essaye encore avec une alternative de cette action (Pour des raisons pratiques le nombre d'alternatives est limité à quelques unes afin de limiter les temps de calculs).

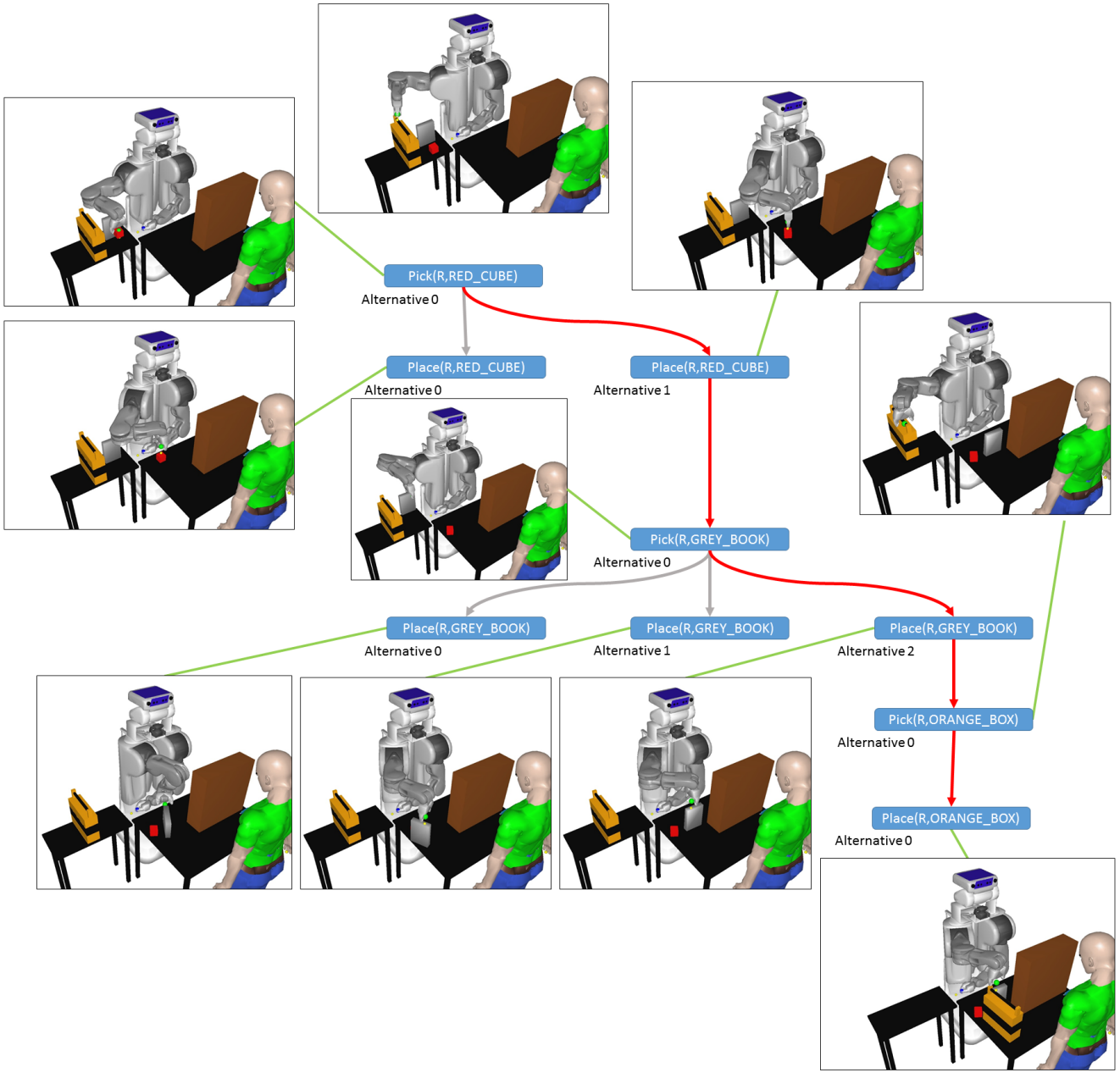


FIGURE A.15: Les différentes étapes d'un plan géométrique incluant les alternatives non utilisées. Dans ce plan, le robot réussi à planifier **trois Pick** puis **Place** successifs de trois différents objets.

A.3 Combinaison de la planification symbolique et géométrique

Dans le but de fournir au robot un horizon de planification plus étendu, nous avons combiné la planification symbolique, dont les points forts sont justement de traiter les problèmes à long terme de manière discrète, avec la planification géométrique qui, elle, s'occupe plutôt de la partie continue de la tâche mais à horizon beaucoup plus court.

Afin de combiner ces deux niveaux de planification nous avons utilisé le planificateur et raisonneur géométrique décrit dans la section précédente, ainsi que le planificateur de tâche HATP pour "Hierarchical Agent-based Task Planner", qui est une implémentation de l'algorithme HTN (Hierarchical Task Network) Ghallab et al. (2004) mais qui prend aussi en compte de manière explicite les humains.

HATP ne fait pas partie des contributions de cette thèse, cependant, un petit rappel de ses fonctionnalités et de ses algorithmes sont nécessaires pour la bonne compréhension de cette section. Après une description rapide de ce logiciel, la sous-section A.3.2 présentera l'approche que nous proposons, suivi (dans la sous section A.3.3) d'une explication concernant la ramification géométrique et la manière dont nous résolvons ce problèmes. Finalement, la sous-section A.3.4 présente les contributions de cette partie au contexte d'interaction homme-robot.

A.3.1 HATP

Un domaine de HATP est constitué de méthode et d'opérateur, sachant qu'une méthode est une combinaison de méthodes et/ou d'opérateurs, qui eux sont les briques de base constituant le plan final. Le but fourni au planificateur l'est sous forme d'une méthode (ou un opérateur), que le planificateur doit appliquer. Afin d'appliquer une tâche[1], le planificateur doit tester si ces préconditions sont vérifiées dans le contexte actuel: si elles sont vérifiées, la méthode est appliquée sinon, le plan contenant cette tâche n'est pas faisable.

Appliquer une tâche dépend de divers paramètres:

La tâche est un opérateur l'opérateur est ajouté au plan courant, et le contexte est mis à jour grâce aux effets du dit opérateur.

La tâche est une méthode la méthode est décomposée, et dépendant du type de décomposition, différentes procédures sont nécessaires:

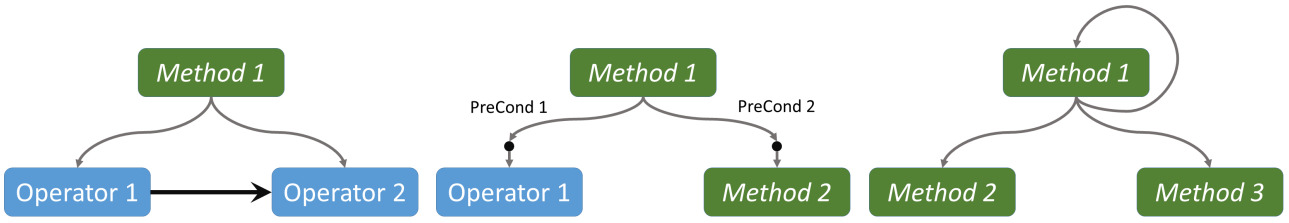
Séquence Les tâches obtenues doivent être appliquées dans l'ordre défini par la décomposition. La figure A.16(a) en montre un exemple.

Ou exclusif Une seule des tâches peut être appliquée : les autres sont sauvegardées en tant que points de backtrack et seront visitées quand le choix courant donnera un plan ou qu'il échoue. Un point de backtrack contient toutes les informations nécessaires (plan courant, contexte courant, méthode en cours d'application/appliquée,...) à reprendre la planification à partir du même endroit. La figure A.16(b) en montre un exemple.

[1] Une tâche peut être soit une méthode soit un opérateur.

Parallélisme Les tâches n'ont pas de lien causal entre elles: l'ordre n'est pas important. La figure A.16(c) en montre un exemple.

Récursion Une méthode peut être récursive: sa décomposition peut se contenir elle-même. La figure A.16(c) en montre un exemple.



(a) *Séquence* (b) *Ou exclusif* (c) *Parallélisme et récursion* and recursive

FIGURE A.16: Les différentes décompositions d'une méthode.

Quand une tâche n'est pas applicable, le planificateur revient au dernier point de backtrack sauvegardé, et reprend la planification de cet endroit. Ainsi toutes les possibilités peuvent être explorées.

A.3.2 La combinaison des deux planificateurs

Afin de combiner les deux planificateurs nous avons modifié le comportement de certaines fonctionnalités de l'algorithme de HATP. La plus importante de ces modifications, se trouve dans **l'application des opérateurs**: quand les préconditions de l'opérateur sont vérifiées dans le contexte courant, au lieu de l'appliquer directement, une requête est envoyée au planificateur géométrique qui teste la faisabilité de l'action dans l'état courant du monde, qu'il maintien aussi, mais au niveau géométrique. Le planificateur géométrique, comme vu dans la section précédente, ne calcule pas seulement des trajectoires, mais aussi les grasps, les placements, les rotations et en général les configurations de tous les agents et objets présents dans le monde. Les requêtes sont dessiné en bleu dans la figure A.17.

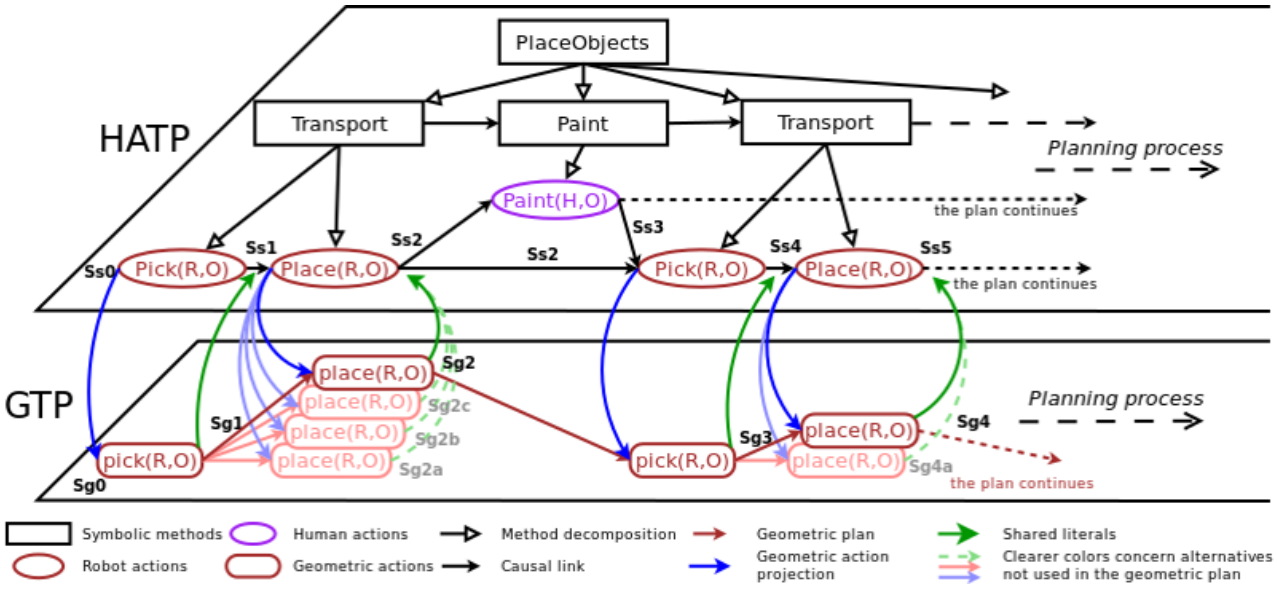


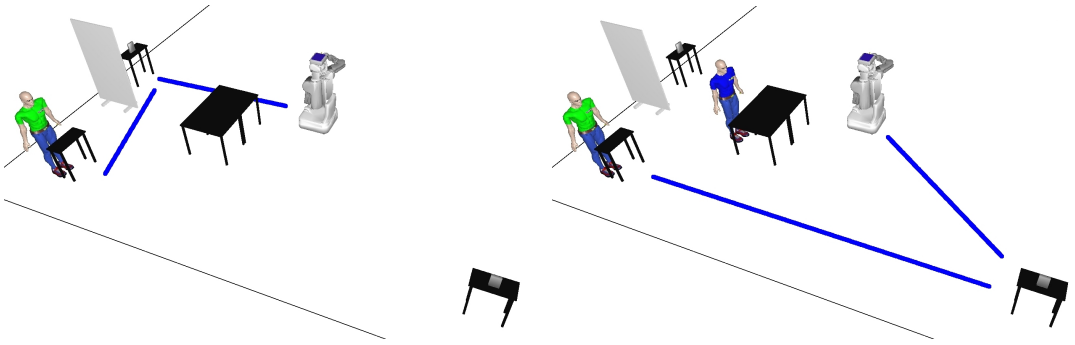
FIGURE A.17: Un exemple de fonctionnement combiné entre HATP et le planificateur géométrique.

Une deuxième altération de l'algorithme de base de HATP, consiste à rajouter ce que nous appelons des instances géométriques: quand une action avec une contrepartie géométrique doit être appliquée, un nombre défini par avance (pour des raisons de temps de calcul) d'instances géométriques est créé sous forme de points de backtrack. A chaque fois que l'algorithme revient sur un de ces points de backtrack, il n'essaye pas une nouvelle tâche, mais essaye d'appliquer le même opérateur que précédemment mais avec un choix géométrique différent, proposé par le planificateur géométrique sous forme d'alternative. La figure A.17 montre des exemples où des alternatives étaient nécessaires pour trouver une solution au problème.

A.3.3 Le problème de la ramification géométrique

La dernière modification importante de l'algorithme de HATP est la prise en compte des effets indirects des actions géométriques: le problème de la ramification géométrique est ainsi partiellement traité. Afin d'arriver à ce résultat, après que l'algorithme de HATP ait testé la faisabilité d'une action géométrique, il applique les effets de l'opérateur (comme précédemment) mais en plus de cela, rajoute les faits que le planificateur est capable de planifier. Comme ce dernier maintien un état du monde géométrique, il est capable de calculer après chaque action divers faits, et les remonter au niveau du planificateur symbolique. La figure A.17 montre en vert les mises à jour renvoyées au planificateur symbolique.

Le problème n'est que partiellement traité car les effets indirects qui ne sont pas traduits en fait par le planificateur géométrique ne sont pas pris en compte par le système.



(a) Environement sans humain (b) Environement avec humain

FIGURE A.18: Dans cet exemple, le robot doit rapporter à l'humain en vert un des deux manuels disponibles (les deux sont similaires, la seule différence est leur position de départ). On peut voir en bleu la trajectoire (et donc le choix) empruntée par le robot.

A.3.4 Une approche prenant en compte l'humain

Afin de prendre en compte l'humain explicitement, un certain nombre de fonctionnalités ont été rajoutées aux deux planificateurs. Concernant HATP, certaines règles sociales ont été rajoutées afin d'identifier parmi tous les plans disponibles lequel choisir. Effectivement, HATP calcule tous les plans possibles, et, se basant sur la somme des coûts de tous les opérateurs du plan, choisit le meilleur. De plus HATP utilise des règles telle que la minimisation de la complexité des plans, des temps d'attente et ainsi de suite. Ces fonctionnalités ne font pas partie des contributions de cette thèse.

Afin d'intégrer le paramètre humain dans le système nous avons tout d'abord intégré dans la planificateur géométrique des planificateurs de mouvement prenant en compte l'humain tel que Sisbot et al. (2007b) et Mainprice et al. (2011) qui utilise entre autres des notions de distance et de visibilité pour garder le robot dans des configurations qui soient acceptables pour l'homme. Ensuite, nous avons mis à jour les coûts des opérateurs avec des contreparties géométriques à l'aide de ces coûts prenant en compte l'humain. Ceci a permis, en outre, de choisir, non seulement de meilleures trajectoires (d'un point de vue d'acceptabilité pour l'homme) mais aussi de meilleurs plans quand cela est possible.

La figure A.18 montre une situation où le robot doit apporter à l'humain en vert un des deux manuels disponibles (la seule différence est leurs positions initiales). Quand l'humain en bleu n'est pas là, le robot choisi le manuel le plus proche est le rapporte à l'homme en vert. Dans le cas où l'humain en bleu se trouve près de la table, même si la première solution reste faisable, l'allée retour jusqu'à l'autre manuel reste préférable, pour ne pas passer trop proche de lui (ou même derrière).

A.4 Conclusion

Nous pouvons résumer les contributions principales de cette thèse en trois points:

Le transfert d'objet d'un robot à un humain ou vice versa, étendu au transfert d'objet entre plusieurs agents avec une étude détaillée du comportement du regard durant le transfert.

Le planificateur et raisonneur géométrique qui permet de calculer les trajectoires, grasps, positions et rotations de tous les agents et objets impliqués dans une tâche en prenant en entrées seulement des symboles.

La combinaison des planificateur symbolique et géométrique en utilisant un planificateur symbolique existant (HATP) et en le liant au planificateur géométrique, nous arrivons à obtenir des résultats satisfaisant.