Proceedings of the 1st Workshop on
Planning and Robotics

PlanRob 2013

Rome, Italy - June 10, 2013

Edited By:

Alberto Finzi, Felix Ingrand, Andrea Orlandini

Organizing Commitee

Alberto Finzi
"Federico II" University, Naples, Italy

Felix Ingrand
LAAS-CNRS, Toulouse, France

AndreA Orlandini
ISTC-CNR, Rome, Italy

Program committee

Rachid Alami, LAAS-CNRS, France
Amedeo Cesta, ISTC-CNR, Italy
Tara Estlin, NASA JPL, USA
Alberto Finzi, Federico II University, Italy
Maria Fox, King's College, UK
Malik Ghallab, LAAS-CNRS, France
Joachim Hertzberg, University of Osnabrueck, Germany
Felix Ingrand, LAAS-CNRS, France
Leslie Kaebling, MIT, USA
Sven Koenig, University of Southern California, USA
Maria Dolores Moreno, Universidad de Alcala, Spain
Karen Myers, SRI, USA
Andrea Orlandini, ISTC-CNR, Italy
Thierry Peynot, University of Sydney, Australia
Fiora Pirri, Sapienza University, Italy
Frederic Py, MBARI, USA
Alessandro Saffiotti, Orebro University, Sweden
Reid Simmons, Carnegie Mellon University, USA
David Smith, NASA Ames, USA
Siddharth Srivastava, UC Berkeley, USA
Florent Teichteil-Königsbuch, ONERA, France
Manuela Veloso, Carnegie Mellon University, USA

Additional Reviewer

Daniele Magazzeni, King's College, UK

Foreword

Robotics is one of the most appealing and natural applicative areas for the Planning and Scheduling (P&S) research activity, however, this potential interest seems not reflected in an equally important research production for the Robotics and Planning communities. On the other hand, the fast development of field and social robotics applications poses planning as a central issue in the robotic research with several real-world challenges for the planning community (e.g. continuous planning and execution with real-time constraints, deliberative and dynamic planning integrated with motion planning and reactive control, human-aware planning and execution, formal methods for plan-based autonomy, etc.).

In this perspective, the goal of this workshop is twofold. From one side, it aims at providing a fresh impulse for the ICAPS community to recast its interests towards robotics problems and applications. On the other side, it would attract representatives from the Robotics community to discuss their challenges related to planning for autonomous robots as well as their expectations from the P&S community. More in general, the PlanRob workshop would constitute a stable, long-term establishment of a forum on relevant topics concerned with the interactions between Robotics and P&S communities. The workshop would present a stimulating environment where researchers could discuss about the opportunities and challenges for P&S when applied in Robotics.

The first edition of the workshop has attracted a positive response from the research community as 16 papers has been accepted for oral presentation covering many relevant topics, such as high-level task planning, CSP/Timeline-based approach, task and motion planning, multi-robot coverage/exploration, traveling problems and reasoning with uncertainty. This seems to us as a very good result for the PlanRob Workshop and, overall, it shows a good feedback from the ICAPS community (but not only) on the PlanRob topics.

Finally, two notable researchers have accepted our invitation to complete an already rich and interesting program providing an invited keynote:

- **Malik Ghallab**, LAAS-CNR, France, "*Acting is the Purpose of Planning: the Actor's view of Deliberation*"
- **Stuart Russell**, UC Berckley, USA, "*Life: Play and Win in 20 Trillion Moves*"

Alberto Finzi, Felix Ingrand and AndreA Orlandini

The PlanRob 2013 Chairs

Table of Contents

Affordance-Based Reasoning in Robot Task Planning

Iman Awaad and **Gerhard K. Kraetzschmar**
Bonn-Rhein-Sieg University
and B-IT Center
Grantham-Allee 20
53757 Sankt Augustin, Germany

Joachim Hertzberg
Osnabrück University
and DFKI RIC Osnabrück Branch
Albrechtstrasse 28
49076 Osnabrück, Germany

Abstract

Humans are able to come up with plans to achieve their goals, and to adapt these plans to changes in their environment, finding fixes, alternatives and taking advantages of opportunities without much deliberation. For example, they may use a tea kettle instead of a watering can to water the plants, or a mug instead of a glass to serve water. Despite decades of research, artificial agents are not as robust or as flexible. In this work, we introduce three reasoning phases that use affordances to enable such robustness and flexibility in robot task planning. The first phase generates a focused planning problem. The second phase expands the domain where necessary while the third and final reasoning phase uses affordances during plan execution and monitoring. This is accomplished by combining Hierarchical Task Network planning, description logics, and a robust execution/monitoring system.

Introduction

A paradigm shift has been taking place among researchers in various fields of AI, such as perception and manipulation in robotics. In recent years, an increasing number of approaches are *task-oriented*. For example, object grasping is no longer solely dependent on the physical properties of the object to be grasped and those of the manipulator, but now takes into account the purpose of the grasping taking place. For example, grasping an object to pour from it may require a different grasp than transporting it.

In the planning field, the task-based perspective predates the current change in other robotics fields. The Hierarchical Task Network (HTN) approach (Erol, Hendler, and Nau 1994a) has enabled researchers to reduce the search space by allowing them to encode the 'best way' of carrying out tasks; thereby improving the quality of the plans along the way. Complexity is at a minimum when there are no choices to be made – when there is exactly one way to decompose a non-primitive task into primitive tasks. This may, however, limit the ability to generate a plan: if we have no applicable way to decompose a non-primitive task at a given state. We would like to make use of another behavior that humans often exhibit: finding alternative ways to accomplish a task. Quite often, not so much by changing the plan per se (the *how*), but by making the right substitutions (the *with what*). For example, by using a mug instead of a glass for drinking or by using a tea kettle instead of a watering can for watering plants. The effect is equivalent to adding a new method to decompose the task that uses the substituted object, and depending on the result at execution, annotating it with a preference index. When using lifting, it would enable additional objects as options to ground the methods and operators. In mixed-initiative approaches, this could be seen as a resource assignment (in our case, one that the agent should arrive at autonomously). But, how would we determine which objects to substitute?

We propose a modified HTN planning algorithm that reuses the procedural knowledge of the methods and finds object substitutes when necessary and appropriate, thereby mimicking the resourcefulness of human planners and actors. We first answer the question of how the agent recognizes when it should make a substitution. Applying justification structures, borrowed from explanation-based approaches for annotating the derivation process of a plan (Veloso 1994; Fernandez and Veloso 2006) accomplishes this and enables us to understand why a planner made a particular decision and why it may have failed to generate a plan. In cases where planning fails due to a missing object, the algorithm uses a reasoning process that employs the concept of *affordances* to expand the domain so that effective alternative choices can be made. Such a choice may be the most appropriate substitution, or the cheapest based on spatial proximity, or some weighted combination of both of these.

Affordances describe "opportunities for action" (Gibson 1979). This notion of affordances is retained in this work, although Gibson's action/perception coupling is not dealt with directly. Gibson's original definition has been refined by many researchers, but a generally agreed upon interpretation narrows the list of action choices to those that an actor is aware of. Using the refined definition, affordances are neither solely a property of the object, nor of the actor, but of their relationship. We adopt Norman's definition (Norman 2002), and the subsequent extensions of this definition by others, such as (Gaver 1991) and (Hartson 2003)) of *perceived affordances* which allude to "how an object may be interacted with based on *the actors's goals, plans, values, beliefs and past experience*" (Norman 2002). We propose to include affordances within the domain model and to represent this in Description Logics (DL) so that we may use the reasoning powers of existing tools to enable the robust and

flexible behavior described above.

To this end, we attempt to answer the following specific questions: How does the agent recognize when it should make a substitution? How does it acquire the *functional affordance* (Hartson 2003) of an object? How is this functional affordance represented? When should it attempt to make a precondition true (for example, wash the dirty glass) as opposed to making a substitution (use a mug instead of the glass)? When should an agent act on the *affordance cues* (Fritz et al. 2006) and when should it stick with the execution of a generated plan? The solution we propose incorporates ideas from each of these domains:

- Planning in highly dynamic domains, where we need to consider actions which may or may not be applicable due to changes in the state of the world. In our case, alternative actions and alternative objects also need to be considered in order to enable successful plan generation and execution. Here, the choice is based on affordances, spatial proximity, and preferences.
- Explanation-based approaches, where information dealing with the decisions made by the planner are recorded to help diagnose the planning process. Some information, e.g. on the absence of objects with which variables can be grounded, can provide us with the necessary cue to expand the domain to include other objects. Unsatisfied preconditions and the affordance that a method or operator was meant to enable would provide us with the information needed to make appropriate substitutions.
- Case-based planning, where previous plans, or subplans, are used instead of new ones being generated. In any domain with tasks that are often repeated, such as making coffee every morning, it makes sense to avoid deriving a plan each time. In our case, it is desirable to keep track of both the plans that turned out to be good solutions as well as the ones that were not (and possibly why).
- Closely related is planning by analogy, where a similarity in problem descriptions identifies and enables a previous plan's structure to be adapted to the new case. Together with case-based planning and explanation-based planning, this approach provides the means to adapt existing plans according to given guidelines.
- Opportunistic planning and reasoning approaches, which use opportunities to achieve goals that at a previous time were unachievable. This approach seems well-suited to planning with affordances, as these affordances could serve as the cues that trigger opportunistic behavior.
- In mixed-initiative planning, a human user steers a plan generation process by modifying, rearranging or adding goals; assigning resources or even by simply approving the plan. The latter is desirable in order to ensure, that any resulting substitutions are indeed acceptable to the user, and safe to perform. Our approach also adheres to priorities set by the user when making substitutions, resulting in varying levels of *obedience* versus *flexibility*.

The goal of this work is to demonstrate the utility of using the powerful concept of affordances in the planning process to reduce complexity and increase flexibility – two tasks that may appear to be impossible to achieve simultaneously.

Application Scenarios

The scenarios presented below demonstrate how an affordance-based agent within the area of domestic service robotics would benefit from enhanced performance and increased robustness.

Object substitution Common household tasks involve fetching objects, for example, a glass of water. If during execution a glass cannot be found, the agent would fail to achieve its goal. A planner using affordance-based reasoning, however, would use its knowledge about the affordances of a glass and substitute another item, such as a mug for it. The agent may use the functional affordances and/or physical properties of the objects to arrive at viable substitutions.

Object substitution as tool usage Taking Norman's definition, the use of an actor's goals to pick up affordance cues can result in interesting uses of everyday objects. For example, the use of a magazine instead of a coaster for placing a bottle on a table is an affordance of a magazine that a human might take advantage of in order to achieve his/her goal of placing a cup on something other than the table. This use case illustrates such emergent behavior. Knowing the functional affordances of the original object and using the conceptual similarity of the substituted object would enable this use case.

Object substitution or use as performance enhancement Another common task involves fetching a cup of coffee. If one uses affordances for planning, an agent could reason that a more appropriate object would be a mug for the coffee as opposed to the cup. The agent might use the functional affordances and/or physical properties of the objects to arrive at the most appropriate object.

Action substitution The existence of an affordance depends as much on the morphology of the actor as it does on a particular set of features of the object. An object may be graspable for one embodiment but not for another. Even with the necessary morphology to allow grasping, a fault (be it temporary or not) with the system may prohibit the use of the manipulator. The object could also be too heavy to pick up. A human faced with a similar situation might attempt to push an object to its destination (assuming that its current and final positions allow this). Here, it is the similarity of the effects of the original action with its possible substitutes that has the greatest impact. The clustering of agent behavior instances (e.g. pick and place, push, pull) which have the same effect on the objects they act upon to form *abstract affordances* (in this example: move) could enable this.

Approach

From the use cases presented in the scenarios above, it is clear that modeling functional affordances is required. These functional affordances can be seen as a subcategory of what Norman called "perceived affordances" which are based on

an agent's experience and goals, and should not be confused with affordances which are perceived from the environment directly by an agent's senses.

Functional affordances are extremely important for a number of reasons. First, they enable intelligent behavior by using objects for carrying out tasks that they were *meant* to. Second, the concept of affordances could very easily lead to an explosion in computational complexity as the action possibilities of objects are numerous: a chair can be sat on, but also thrown, stood on, etc. By using functional affordances, the action space can be successfully reduced when the substitution of objects is called for. In addition, the functional affordances also enable us to specify more general tasks. For example, when asked to "serve a drink" any object 'for drinking' could be served without the need to explicitly specify a particular drink. This is more important than it seems at first glance since much of our interaction with each other involves a great deal of underspecification. Finally, they are especially important given the difficulty researchers usually face in representing such *a priori* or experience-based knowledge. Functional affordances can be compactly represented in DL and can be reasoned about efficiently. Objects may have more than one functional affordance. A bottle for example, may have a primary functional affordance of storing liquids and a secondary functional affordance of drinking from.

In addition to functional affordances, *affordance cues* (Fritz et al. 2006), which may be picked up from the environment through the perception process, are also needed. They would serve as triggers of action behaviors. In addition, a means to measure the similarity between objects would enable the synergistic use of these affordances for the object substitution, and object substitution as tool usage scenarios. For action substitution, the affordances are mainly related to the effects of the actions and a similarity measure between these would then robustly enable the scenario.

In the following sections, we demonstrate how our affordance-based approach minimizes modeling, enables flexibility and versatility, makes the representations more compact, and reduces search at planning time.

Generating the Planning Problem

The task of generating a problem description for planners is key. Part of this description is the domain (including methods, operators, and objects). Modeling the domain is difficult and time consuming. The choices of what to include and what to exclude pose a dilemma for domain developers. While it is desirable to limit the size of these models to keep problems tractable, this may also result in failures to generate plans. The increasing complexity due to the size of the search space (caused by both the number of operators and the sheer number of objects in real world domains) remains a challenge - so much so that much of the benchmarking problems that are often used for planners can still be considered "toy problems" (Mastrogiovanni et al. 2010) .

Domain knowledge has long been used to help constrain the size of the planning problems. Hartanto takes this further by coupling DL reasoning with task planning. In (Hartanto 2009), he represents the domain in DL by transforming the HTN syntax to DL and vice versa. This enables the inference of a *constrained* planning domain by selecting only relevant elements, for example, only considering rooms whose doors are open in a navigation domain. The modeling of the domain in DL is thus a crucial element in handling computational complexity. By linking affordances to tasks and representing them in DL the use of domain knowledge is increased and the robustness of the system is improved.

As mentioned before, complexity increases when a choice is required; when there is more than one way to accomplish a task or to achieve a goal, or, put differently, when we have more flexibility. The cost of generating plans may differ by orders of magnitude. The HTN planning approach improves the situation by providing an expert's way of carrying out the task, and thus improving the quality of the plans. Moreover, in an environment shared by humans and artificial agents, this approach is beneficial as it is more human-readable and a good agent should be able to communicate their plan at all times (Bradshaw, Feltovich, and Johnson 2011). In addition, it enables the human user to specify the way he/she wishes to have a task accomplished in an intuitive way.

Let us take the task of watering plants as an example. The domain modeler would specify methods which provide both procedural knowledge (how the task is to be accomplished) and domain knowledge (specifying that a watering can in particular should be used) resulting in a task network such as the one shown in Figure 1. If there is no watering can in our domain, or, despite all of our methods and operators no decomposition is found to accomplish the task, the plan generation process will fail. For example, when the watering can exists but is inaccessible and we have no means by which to make it accessible. The following section details this process of expanding the domain to enable substitutions.

Expanding the Domain

In (Magnenat, Chappelier, and Mondada 2012), the authors use the HTN domain to constrain the search space and then learn the probabilities of success in order to enable more robust plans. The lifting process is done over categories of objects rather than instances, thus reducing the complexity. The proposed use of this lifting over categories of objects and/or their functional affordances in our approach provides us with the chance to increase the number of possible objects to be ground and at the same time remain focused on achieving the task by choosing a more general category or a functional affordance.

The question is, how does the robot decide which substitutions are admissible and how do we represent this knowledge? We argue that the most appropriate substitutions are the ones that are meant to be used for the same task. For example, glasses and mugs are both used to drink from. This is knowledge that humans learn and that may be found in the dictionary for example, or possibly through projects that aim to make common sense knowledge available to artificial agents (such as OpenCyc (Cycorp httpwwwopencycorg), RoboEarth (Hubel et al. 2010), ConceptNet and WordNet). These knowledge sources may be used to provide Norman's "values, beliefs" and even "past experience" (Norman 2002). Objects that are used for the same task (i.e. share

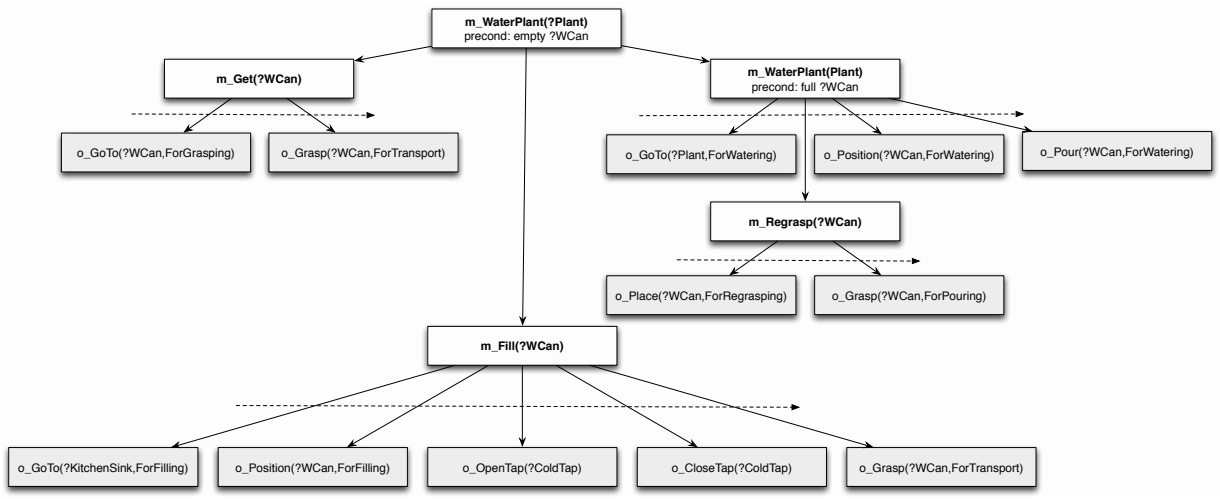


Figure 1: Task network for the WaterPlant method

the same 'functional affordance') would provide the most 'appropriate' substitutions. In addition, they can of course also come from the humans co-inhabiting the environment (for example, they might ask the robot to only clean the bathrooms with the blue cleaning cloths (restricting it to the subcategory), or to only serve them tea in their favorite cup (a single instance)). This answers the question of where the functional affordances of objects come from.

Some objects, such as watering cans, are used for a very specific task. In this example, for watering plants. The only other object which is used for the same task would be a 'hose', and this is only for watering plants outdoors. In this case, both share the same functional affordance of watering plants, but whereas it may be desirable to substitute the watering can for the hose, the opposite is not true, and so a substitution using only functional affordances may fail. Here, the agent would need to look for objects which are conceptually similar to the watering can. The similarity measures which are often used may not yield the results we have in mind (we may not care about the color of an object, but rather the presence of a handle for example).

Using Conceptual Spaces to Measure Similarity

For describing similarity, we propose the use of *Conceptual Spaces* (Gärdenfors 2004). They provide a multi-dimensional feature space where each axis is represents a quality dimension, for example brightness, intensity, and hue. Points in a conceptual space represent objects, while regions represent concepts.

Let's take the example given in (Gärdenfors 2004) (see Figure 2): the three quality dimensions in our example above can together be used to describe the 'color' domain. A region on the red axis could be described as having the property 'red'. A point in this region could represent the concept 'apple' in conjunction with other domains such as 'taste' or 'shape'. We could even relate the property 'red', when talking about apples, to the taste 'sweet'.

As conceptual spaces are built up by the various quality dimensions (some or all of which may be sensed by the agent, depending on its sensing capabilities), the idea is to see if we can determine a relation between these quality dimensions and given tasks. For example, for the task of lifting an object, the most important quality dimension would be its weight — its color would be irrelevant. These relations could then be used as weighting factors to determine how well an object would substitute for another in achieving a given task (similarity would be measured as the weighted Euclidean distance).

Conceptual spaces can also be used to represent shape, such as handles, or spouts. The detection of these quality dimensions obviously requires more processing by the perception components than e.g. the simple detection of hue. This would serve as a more robust and 'focused' similarity measure for achieving a given task. In the case of finding a substitute for the watering can, and given that for such a task the capacity to hold water is perhaps the most important affordance, followed by the presence of a handle and a spout, conceptual spaces could find that the most similar item for this task would be the tea kettle.

If we are to make use of the reasoning power of DL, it would be beneficial to represent Conceptual Spaces in DL. The use of the *Conceptual Spaces Markup Language* (Adams and Raubal 2009) standard and its suitability for reasoning about it will be investigated.

Using functional affordances and conceptual similarity, an artificial agent can start by attempting to satisfy the constraints which are at the bottom of Figure 3, i.e. only using a unique instance – if this was specified in the goal – or an instance of a given object. If it fails to ground the suitable object, the domain would be expanded: it would attempt to find objects which satisfy the constraints which are in the level above until it reaches the final level (minimal constraints, maximum flexibility). Objects with the same functional affordance as that of the originally called-for object are pre-

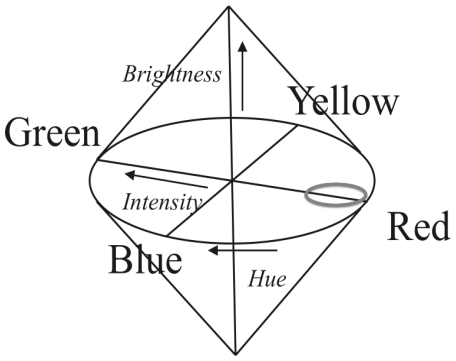


Figure 2: The color spindle formed by the quality dimensions brightness, intensity and hue (based on the diagram in (Gärdenfors and Warglien 2012) p. 4).

ferred. This seems to be inline with our own preferences. The first level above that of using an instance of a given object is to use any object with the same functional affordance and high conceptual similarity. The next higher level would remove the constraint that the substitute should be conceptually similar, relying only on a shared functional affordance. Should the agent not find such objects and given the old adage that "form follows function" (the form of objects is based on their function), conceptual similarity is then used to identify those objects which do not share the same functional affordance and yet are conceptually similar. The top level attempts to infer the function-relevant attributes and identify objects matching these properties.

The importance of spatial proximity can be altered to make it either easier to move from one level of constraints to another by increasing its importance (prefer objects which are close, even if they are in a less constrained category of objects), or more difficult to move up by decreasing its importance. This steering of the domain expansion process can be used both at planning time and at execution time as described in the following section. This answers the questions of when the system should attempt to make a pre-condition true (for example, wash the dirty glass) as opposed to making a substitution (use a mug instead of the glass), when it should act on the affordance cues and when it should stick with the execution of a generated plan.

If the agent uses a tea pot to water plants, it still needs to fill it, and the procedural knowledge of how to do this for a tea pot may differ from that of filling a watering can. This brings us to how actions are dealt with.

We propose to map the execution of the actions in conceptual spaces as in (Gärdenfors and Warglien 2012). '*Behaviors*' are clustered by their effect on the objects they act upon to obtain various *abstract affordances*, one per cluster; for example 'fill'. To reduce complexity during planning, a method for each of these clusters can then be created. The planning process then generates a plan that uses these methods. These could then be matched with the closest matching 'behavior' instance.

For example, the behaviors 'pick and place', 'push', and 'pull' all have a similar effect on the objects they act on; and can be clustered together to form an abstract affordance that we can call 'move'. New affordances can appear as a result of a behavior, for example, an object becomes stackable as a result of being turned where the turning behavior may result from carrying out any number of other behaviors: (pick and place, push, pull), and stacking by executing the'pick and place' behavior (Lörken and Hertzberg 2008).

Plan Execution and Monitoring

Having successfully provided a compact planning problem to the planner and generated a plan, its execution and careful monitoring is necessary. During this phase, a number of issues need to be addressed. We would like the system to robustly handle unexpected situations and to take advantage of opportunities.

Unexpected situations could occur due to partial observability of the environment or as a result of an environment's dynamic nature. For example, a door which was previously known to be open may be closed at the time of execution, or the watering can which was known to be in a given location can no longer be found. In this case, the system behaves much as it might during the plan generation phase, with a slight difference.

Just as humans prefer to take advantage of objects within their immediate spatial surroundings in such situations, the agent might do the same. In the example of the coaster and the cup presented in the 'application scenarios' section above, humans would no doubt consider the use of objects which are already on the table in the absence of the coasters (such as magazines). Similarly, through the use of affordances, we hope to accomplish the same.

In order to truly take advantage of opportunities within the environment, which by definition are unexpected opportunities, we need to combine both the execution of plans which have been generated through the deliberation process and reactive behaviors which may be triggered by affordance cues.

We propose a simple blackboard architecture where affordance cues (in the form of conceptual space quality dimensions) are being posted as the agent moves through its environment as part of executing a plan. These might be of varying complexity (from simple color hues which would cost

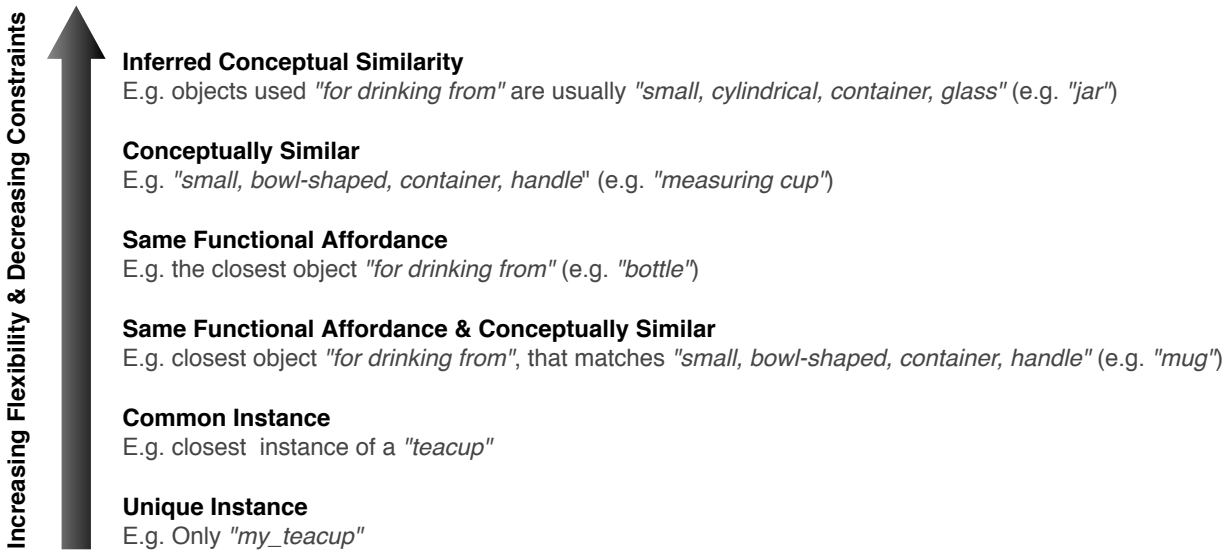


Figure 3: Decreasing constraints to increase flexibility in substituting objects

very little in terms of perceptual processing to more complex concepts such as shape which might have been picked up as part of the plan's execution) and would be kept in the system for a given duration. Upon plan failure, the cues which are in close proximity can be used to identify viable candidates for substitutions.

Of course, the same behavior can be used to guide plan execution even when things are going as planned, and to take advantage of opportunities before failures occur. For example, cues that are associated with a drink bottle may have been picked up on the way to the location specified in a plan. This 'short cut' could be taken advantage of, again depending on the flexibility that the human user has allowed. A cupboard full of glasses would guide the agent to grasp any of them, if there are no additional constraints like using a specific glass. In the case of plan failure, an agent might take the more 'resourceful route' of making a substitution or attempt to use the same object by finding other instances, or of using objects with the same functional affordance.

In fact, it makes sense to take the same task-based perspective mentioned in the introduction and to apply it to perception: the combination of active perception at execution time and task-oriented perception would enable the agent to actively search for features which are relevant to the task at hand, as opposed to passively picking up any and all cues mentioned above. (Arkin 1998) has shown that the time complexity for such a search is far better when compared to a data-driven search.

Discussion

Over a decade ago, (Wilkins and desJardins 2001) stated that

> An ideal system would be able to behave like humans do in these sorts of [complex, dynamic] environments; in particular, it would have to exhibit creativity, devising new actions that can solve a problem or shorten a plan; use analogy to transfer solutions from other problems; effectively interact with humans to use their knowledge in decisions; and behave intelligently in the face of conflicting or incomplete information.

This is precisely what we have aimed to accomplish. One could argue that the scenarios described here can be solved without a distinct theory of affordances by, for example, introducing a more general model of constraints on objects in the procedures themselves. This would have a number of disadvantages however. The domain modeler would need to decide on the necessary constraints for achieving a task and model them. Both are non-trivial tasks. If the constraints are too general, then the number of objects which may be used may increase drastically, resulting not only in an increase in complexity, but also in the use of objects which are suboptimal for the task when compared with *the* object which should be used to start with. On the other hand, if they are too specific, then we are left with our initial problem of not finding any object to carry out our task with.

For example, we could specify that any object used for watering a plant should have a handle, a long spout and hold a given amount of water (perhaps within a given range). The procedural knowledge could then be formulated in terms of these constraints. Such a specification could be too limiting, while a more general set of constraints (e.g. any container) might allow too many objects to be substituted, some of which may be quite inefficient (e.g. a mug could be used to water plants). One could of course annotate all possibilities with preferences, but this would again involve more modeling and increase the compactness of the representation.

Our approach can also be seen as being based on a set of constraints. In our work, these are based on the synergistic use of functional affordances and conceptual similarity. Contrary to the approach described above, which would include these constraints with the procedural knowledge, the

domain modelers in our approach only design the methods and operators as they would normally do, without explicitly modeling these constraints. Instead, only the functional affordances are modeled. The representation is much more compact and the system enables a versatility in steering the possible solutions that would simply not be possible using the approach described above. These constraints are iteratively decreased in a structured way to provide the flexibility with which to choose substitutions.

Related Work

The noun *planning* is often appended to a variety of adjectives that describe the underlying mechanism or the type of domain an approach addresses. For example, we talk about continuous planning in open-ended domains, planning under uncertainty in highly dynamic and/or partially observable domains, mixed-initiative planning when there is a human somewhere in the loop, cooperative planning when we talk about humans and robots working together to achieve a task and multiagent planning when the planning is carried out for execution by multiple agents. Then, we have a host of other terms to describe what we do with plans once we have them: plan management, case-based planning, explanation-based planning, analogical derivation, goal transformation and so on. In one way or another, the system we describe here is related to the majority of these approaches.

In (Off and Zhang 2012), the authors address the problem of planning with incomplete information by modifying the HTN algorithm to consider all possibly relevant and "possibly-aquirable extensions of a domain model" (as opposed to using a conditional planning approach). Our system adopts an open world assumption (through the use of Description Logics which by default assumes incomplete information), so unless our knowledge base contains a statement (or can infer one) to the effect that something is true or that it is false, our query would return 'don't know'. The tight integration between the planning and execution components is also shared.

Like the work presented here, (Cox and Zhang 2007) address the case where planners fail due to insufficient resources or because of changes in the environment, although, their work focuses on improving the performance of novice users in mixed-initiative approaches. The authors provide a user interface where a human user can transform the goals directly, thereby steering the planning process. Interestingly, they identify two hierarchies (Cox and Veloso 1998) along which the goals may be steered. The first is a type hierarchy for instances and the second a predicate abstraction hierarchy; both of which can be seen as essentially providing abstract information for objects and actions, akin to our functional affordances and conceptual similarity for objects and our abstract affordances for actions. In our approach too, the human user plays a vital role in the overall system. First by providing the priority weighting for the process (which varies the obedience/flexibility of the system), and second by providing feedback on whether a substitution is acceptable or not (and possibly the reason why it is not), thus enabling the system to improve its performance over time.

(Hayes-Roth and Perrault 1979; Birnbaum 1986; Simina and Kolodner 1995; Mohan 2008) are examples of work on opportunistic planning approaches which generally deal with recognizing when a previously suspended goal can be pursued. This is slightly different than our use of the term. By opportunistic, we refer to acting on cues that deal with current (and near-future) goals to enable the 'shortcuts' we describe above.

The case-based approach (Cox, Muñoz-Avila, and Bergmann 2005; Aha 2002) does describe what we attempt to achieve here by maintaining a library of plans and annotating them with their outcome (successful or not). Adapting existing plans by using explanation-based approaches (together with analogy-based plan derivation) is also of interest. It could be that the 'critic' (Erol, Hendler, and Nau 1994b) functionality built into HTN planners is sufficient to create the necessary queries for the knowledge base to enable the expansion of the domain. This is still being investigated.

The representation of domain knowledge (as in (Tenorth and Beetz 2009) and (Tenorth et al. 2012)) is of great interest. In the latter, the researchers have built upon the knowledge base which was created in the former, and created a formal language to model the environment, actions, objects and the robots themselves (including capabilities) with the aim to enable knowledge exchange between agents. This is of course extremely appealing and it will be interesting to see if our system can interface with such a system and use the knowledge available (and possibly contribute to it).

The use of the affordances concept in robotics is by no means new. Early attempts to represent affordances for computation purposes in artificial agents were made by (Steedman 2002a) as well as (Chemero and Turvey 2007), and were quickly followed by many others. A great deal of research has recently been invested in the use of affordances for manipulation tasks (see (Ridge et al. 2009; Kraft et al. 2009; Moldovan et al. 2011)), and as such, the means by which affordances can be perceived and learned is also being addressed (see (Fritz et al. 2006; Stoytchev 2005; Hermans, Rehg, and Bobick 2011; Ridge et al. 2009; Stark et al. 2008; Montesano et al. 2007; Varadarajan and Vincze 2011)). Researchers in the field of human-robot-interaction (HRI) have also used affordances for their processes (for example (Moratz and Tenbrink 2008; Mason and Lopes 2011)).

Some researchers have focused on planning with affordances in mind. Of note here is the work by (Lörken and Hertzberg 2008), (Ugur, Sahin, and Oztop 2011), (St. Amant 1999) and (Steedman 2002b), while task execution in particular was addressed by (Lörken 2006). (Raubal and Moratz 2008) as well as (Lörken and Hertzberg 2008) went on to integrate affordances into traditional, layered cognitive architectures.

The work presented in (Janowicz and Raubal 2007) and (Raubal and Moratz 2008), while addressing a different focus, is perhaps the closest to ours in that they also use functional affordance and conceptual spaces to measure similarity. Their work is based on an adapted version of the HIPE theory of action and so they have included additional types of affordances based on both physical and socio-institutional

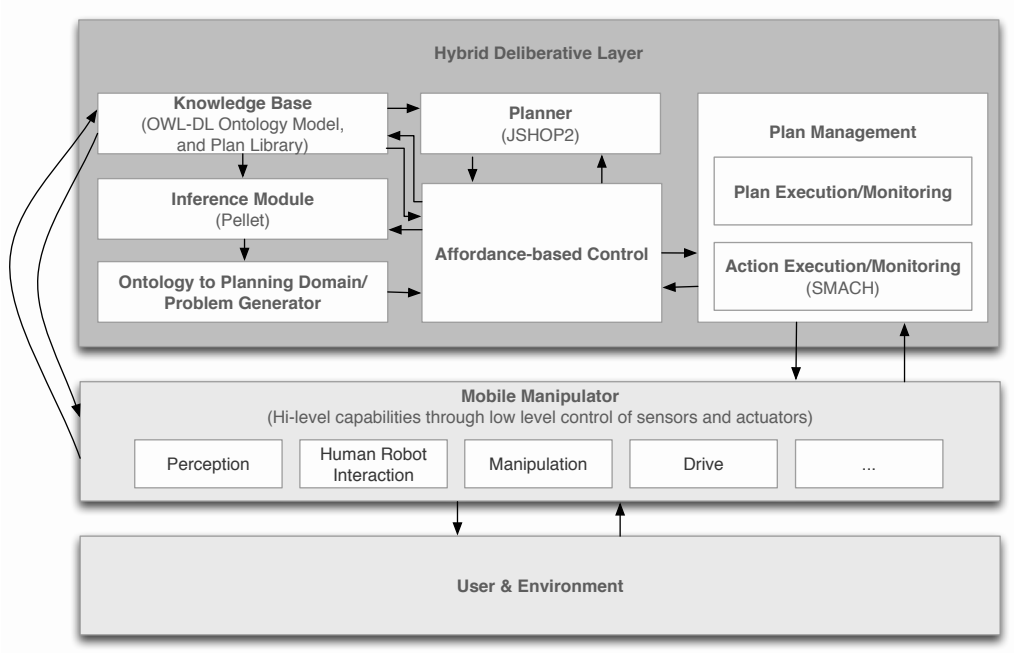


Figure 4: Software architecture of the system extending the hybrid deliberative layer (Hartanto 2009) to use affordance-based reasoning in a domestic environment

constraints.

A survey on the use of DL in planning approaches is provided in (Gil 2005).

Current Status and Planned Developments

The tools and libraries which will be used in the system have been identified, tested and chosen. The software architecture of the system (see Figure 4) has been developed and it has been integrated into the current framework of our b-it-bots RoboCup@Home team's domestic service robot (a Care-O-bot 3 named Jenny). We are currently working on the coordination and optimization of the knowledge representations used by the various components of the system.

Two of the monolithic SMACH scenario scripts (state machine-like execution scripts) for the robot have been converted into modular states that can be called dynamically as a result of the planning process. The procedural knowledge encoded in such scripts was used to model the planning domain for two RoboCup scenarios. A component, the *task dispatcher*, which iterates through a generated plan and calls the appropriate SMACH states has been implemented. Individual monitoring actions have been designed and included in the SMACH scripts to enable the robot to monitor the execution of its actions, thereby providing additional robustness. The concepts have been proven and the foundations for the planning, execution, and monitoring systems have been built.

The design and implementation of the first use case for affordances (object substitution) is underway. This requires the modeling of the functional affordances in DL. The possibility of autonomously acquiring these functional affordances from online sources is currently being investigated. The object substitution use case will be used to validate the designed model of the functional affordances. Extending JSHOP2 (Ilghami and Nau 2003) to use lifting over categories and the justification structures is also a current task. Designing the plan library (including preferences, etc.) is the next major task. Once these steps are successfully completed, we will look into the integration of the other affordance use cases.

Acknowledgements

The author I. Awaad would like to acknowledge the financial support provided by a PhD project scholarship of the Department of Computer Science of Bonn-Rhein-Sieg University. The work of J. Hertzberg reported here is supported by the RACE project, grant agreement no. 287752, funded by the EC Seventh Framework Programme theme FP7-ICT-2011-7. The authors would like to thank Elizaveta Shpieva and Daniel Höller for their help in implementing some of the ideas presented here. The authors also thank the reviewers for their valuable feedback which has helped to improve this manuscript.

References

Adams, B., and Raubal, M. 2009. Conceptual Space Markup Language (CSML): Towards the Cognitive Semantic Web. In *2009 IEEE International Conference on Semantic Computing*, volume 0, 253–260. Los Alamitos, CA, USA: IEEE.

Aha, D. E. 2002. Mixed-initiative case-based reasoning: Papers from the ECCBR'02 workshop. Technical report, Robert Gordon University, Aberdeen, Scotland.

Arkin, R. C. 1998. *Behavior-Based Robotics*. Intelligent Robots and Autonomous Agents. Cambridge, MA, USA: MIT-Press.

Birnbaum, L. A. 1986. *Integrated processing in planning and understanding*. Ph.D. Dissertation, Yale University, New Haven, CT, USA. AAI8728109.

Bradshaw, J. M.; Feltovich, P. J.; and Johnson, M. 2011. *The handbook of human-machine interaction: a human-centered design approach*. Farnham, Surrey, England; Burlington, VT: Ashgate. chapter 13, 283–300.

Chemero, A., and Turvey, M. T. 2007. Gibsonian affordances for roboticists. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems* 15(4):473–480.

Cox, M. T., and Veloso, M. M. 1998. Goal transformations in continuous planning. In *1998 AAAI Fall Symposium on Distributed Continual Planning*, 23–3. Menlo Park, CA: AAAI Press.

Cox, M. T., and Zhang, C. 2007. Mixed-initiative goal manipulation. *AI Magazine* 28(2):62–74.

Cox, M. T.; Muñoz-Avila, H.; and Bergmann, R. 2005. Case-based planning. *Knowl. Eng. Rev.* 20(3):283–287.

Cycorp. http://www.opencyc.org/. Opencyc. Online at http://www.opencyc.org/.

Erol, K.; Hendler, J.; and Nau, D. S. 1994a. HTN planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1123–1128. AAAI Press.

Erol, K.; Hendler, J.; and Nau, D. S. 1994b. Semantics for hierarchical task-network planning. Technical report, University of Maryland.

Fernandez, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

Fritz; Paletta; Breithaupt; and Rome. 2006. Learning predictive features in affordance based robotic perception systems. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 3642–3647.

Gärdenfors, P., and Warglien, M. 2012. Using Conceptual Spaces to Model Actions and Events. *Journal of Semantics*.

Gärdenfors, P. 2004. How to Make the Semantic Web More Semantic. In *Proceedings of the Third International Conference (FOIS 2004)*, 17–34.

Gaver, W. W. 1991. Technology affordances. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology*, CHI '91, 79–84. New York, NY, USA: ACM.

Gibson, J. J. 1979. *The ecological approach to visual perception*. Houghton Mifflin (Boston).

Gil, Y. 2005. Description logics and planning. *AI Magazine* 26(2):73–84.

Hartanto, R. 2009. *Fusing DL Reasoning with HTN Planning as a Deliberative Layer in Mobile Robots*. Ph.D. Dissertation, University of Osnabrück.

Hartson, H. R. 2003. Cognitive, physical, sensory, and functional affordances in interaction design. *Behaviour & IT* 22(5):315–338.

Hayes-Roth, B., and Perrault, F. H.-R. 1979. A cognitive model of planning. *Cognitive Science* 3(4):275–310.

Hermans, T.; Rehg, J. M.; and Bobick, A. 2011. Affordance prediction via learned object attributes. In *IEEE International Conference on Robotics and Automation (ICRA): Workshop on Semantic Perception, Mapping, and Exploration*.

Hubel, N.; Mohanarajah, G.; van de Molengraft, R.; Waibel, M.; and D'Andrea, R. 2010. RoboEarth Project. Online at http://www.RoboEarth.org.

Ilghami, O., and Nau, D. S. 2003. A General Approach to Synthesize Problem-Specific Planners. Technical Report CS-TR-4597, UMIACS-TR-2004-40, University of Maryland.

Janowicz, K., and Raubal, M. 2007. Affordance-based similarity measurement for entity types. In *COSIT'07: Proceedings of the 8th international conference on Spatial information theory*, 133–151. Berlin, Heidelberg: Springer-Verlag.

Kraft, D.; Detry, R.; Pugeault, N.; Baseski, E.; Piater, J. H.; and Krüger, N. 2009. Learning objects and grasp affordances through autonomous exploration. In Fritz, M.; Schiele, B.; and Piater, J. H., eds., *Computer Vision Systems, 7th International Conference on Computer Vision Systems, ICVS 2009, Liège, Belgium, October 13-15, 2009, Proceedings*, volume 5815 of *Lecture Notes in Computer Science*, 235–244. Springer.

Lörken, C., and Hertzberg, J. 2008. Grounding planning operators by affordances. In *Proc. Intl. Conf. Cognitive Systems*, 79–84. (CogSys 2008).

Lörken, C. 2006. Introducing affordances into robot task execution. Master's thesis, Institute of Cognitive Science, University of Osnabrück.

Magnenat, S.; Chappelier, J.-C.; and Mondada, F. 2012. Integration of Online Learning into HTN Planning for Robotic Tasks. In *Proceedings of the AAAI Spring Symposium 2012: Designing Intelligent Robots, Reintegrating AI*.

Mason, M., and Lopes, M. C. 2011. Robot self-initiative and personalization by learning through repeated interactions. In *Proceedings of the 6th international conference on Human-robot interaction*, HRI '11, 433–440. New York, NY, USA: ACM.

Mastrogiovanni, F.; Scalmato, A.; Sgorbissa, A.; and Zaccaria, R. 2010. Affordance-based planning for assisting humans in daily activities. In *Proceedings of the 2010 Sixth International Conference on Intelligent Environments*, IE '10, 19–24. Washington, DC, USA: IEEE Computer Society.

Mohan, V. 2008. *Cognitive Robots From Affordance to Action and Back*. Ph.D. Dissertation, Italian Institute of Technology University of Genoa Doctoral School on Humanoid Technologies, Genova, Italy.

Moldovan, B.; Otterlo, M. V.; Lopez, P. M.; Santos-Victor, J.; and Raedt, L. D. 2011. Statistical relational learning of object affordances for robotic manipulation. In *ILP*.

Montesano, L.; Lopes, M.; Bernardino, A.; and Santos-victor, J. 2007. Modeling affordances using bayesian networks. In *IEEE/RSJ - International Conference on Intelligent Robots and Systems (IROS'07)*.

Moratz, R., and Tenbrink, T. 2008. Affordance-based human-robot interaction. In *Proceedings of the 2006 international conference on Towards affordance-based robot control*, 63–76. Berlin, Heidelberg: Springer-Verlag.

Norman, D. 2002. *The psychology of everyday things*. Basic Books (New York).

Off, D., and Zhang, J. 2012. Continual HTN planning and acting in open-ended domains - considering knowledge acquisition opportunities. In *ICAART*, 16–25.

Raubal, M., and Moratz, R. 2008. A functional model for affordance-based agents. In *Proceedings of the 2006 international conference on Towards affordance-based robot control*, 91–105. Berlin, Heidelberg: Springer-Verlag.

Ridge, B.; Skocaj, D.; ; and Leonardis, A. 2009. Unsupervised learning of basic object affordances from object properties. In Ion, A., and Kropatsch, W. G., eds., *Computer Vision Winter Workshop*.

Simina, M. D., and Kolodner, J. L. 1995. Opportunistic reasoning: A design perspective. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, 78–83.

St. Amant, R. 1999. Planning and user interface affordances. In *IUI '99: Proceedings of the 4th international conference on Intelligent user interfaces*, 135–142. New York, NY, USA: ACM.

Stark, M.; Lies, P.; Zillich, M.; Wyatt, J.; and Schiele, B. 2008. Functional object class detection based on learned affordance cues. In *6th International Conference on Computer Vision Systems (ICVS)*, volume 5008, 435–444. Santorini, Greece: Springer Berlin / Heidelberg. Oral presentation.

Steedman, M. 2002a. Formalizing affordance. In *Proceedings of the 24th Annual Meeting of*. Washington D.C.: Lawrence Erlbaum.

Steedman, M. 2002b. Plans, affordances, and combinatory grammar. *Linguistics and Philosophy* 25.

Stoytchev, A. 2005. Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of AAAI Symposium on Developmental Robotics*, 17–22. AAAI.

Tenorth, M., and Beetz, M. 2009. Knowrob - knowledge processing for autonomous personal robots. In *IROS*, 4261–4266.

Tenorth, M.; Perzylo, A.; Lafrenz, R.; and Beetz, M. 2012. The RoboEarth language: Representing and exchanging knowledge about actions, objects, and environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 1284–1289.

Ugur, E.; Sahin, E.; and Oztop, E. 2011. Unsupervised learning of object affordances for planning in a mobile manipulation platform. In *ICRA*, 4312–4317. IEEE.

Varadarajan, K., and Vincze, M. 2011. Knowledge representation and inference for grasp affordances. In Crowley, J.; Draper, B.; and Thonnat, M., eds., *Computer Vision Systems*, volume 6962 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 173–182.

Veloso, M. M. 1994. Flexible strategy learning: Analogical replay of problem solving episodes. In *Proceedings of AAAI-94, the Twelfth National Conference on Artificial Intelligence*, 595–600. Seattle, WA: AAAI Press.

Wilkins, D. E., and desJardins, M. 2001. A call for knowledge-based planning. *AI Magazine* 22(1):99–115.

A Deliberation Layer for Instantiating Robot Execution Plans from Abstract Task Descriptions

Daniel Di Marco and **Paul Levi**
Department of Image Understanding
Universität Stuttgart, Germany
dimarco@ipvs.uni-stuttgart.de

Rob Janssen and **René van de Molengraft**
Department of Mechanical Engineering
Eindhoven University of Technology, The Netherlands

Alexander Perzylo
Department of Informatics
Technische Universität München, Germany

Abstract

We present an application of Hierarchical Task Network (HTN) planning to create robot execution plans, that are adapted to the environment and the robot hardware from abstract task descriptions. Our main intention is to show that different robotic platforms can make use of the same high level symbolic task description.

As an off-the-shelf planning component, the SHOP2 HTN planner is adopted. All the domain knowledge is encoded in the Web Ontology Language (OWL) and stored in a world wide accessible database, which allows multiple systems to reuse and improve upon this knowledge. For task execution, the execution plan is generated using the CRAM plan language (CPL).

We demonstrate the functionality of the system in executing a pick-and-place task in a simulated environment with two different service robots, the TU/e Amigo robot prototype and the Fraunhofer IPA Care-O-Bot 3. The experiment shows that although the robots differ in hardware capabilities, the use of HTN planning adds information that is crucial for successful task execution and enables both systems to successfully execute the instructed task.

Introduction

Autonomous task execution in unstructured environments is an critical problem for service robotics. A lot of background knowledge is required to solve this problem, not only about the task to execute, but also on the robot itself and its environment. As there exists a wide range of different types of service robots today, finding a way of exchanging this knowledge is an interesting problem to solve.

In (Di Marco et al. 2012), we proposed a task execution system for abstract task descriptions. In the following work, we describe an extension on the system described there. To recap briefly, the previously proposed system is built upon several open-source software packages and translates abstract task descriptions, represented in a high level and stored on a global database into executable robot plans for different robot platforms. To handle significant differences in robot hardware gracefully, the task descriptions are annotated with capability requirements and matched with a robot's specific hardware. This way, only task descriptions that are executable on a given robot platform at plan construction time, are considered.

These abstract task descriptions are encoded as a sequence of hardware-agnostic actions and are represented as concepts from a common ontology with semantic annotations (Tenorth et al. 2012). The vision behind this approach is that knowledge of how a robot can execute a specific task, that is encoded on a high enough level can be used by other robots (i.e. robots with different sensing or manipulation hardware). A problem of this representation when considered from the perspective of task execution is that the high-level concepts need to be grounded in actual actions the robot can execute. In the previous system, this was expected to be done on the robot hardware layer. For example, instances of the OWL concept "Translation-LocationChange" are used to describe an agent's intentional movement in the environment. When elements of this type are encountered in an abstract task description, they are translated into calls to the robot's respective base movement implementation.

Another consequence of this form of representation is that certain low-level details required for task execution are abstracted away. Therefore executing the task descriptions necessitates some form of reasoning. Consider for example the task of picking up an object by a service robot with two arms. The information on which manipulator to use is not stored in the task description, because it should remain possible to execute the task on a robot with an arbitrary number of arms, but which arm to use has to be inferred for each specific situation again.

Finally, although there exists the possibility to provide multiple task descriptions for a specific task, the system is limited in the selection of the appropriate one by filtering them according to the required robot capabilities. For instance, while there might be task descriptions for passing an open door and a closed door, the system has no means of inferring on its own which task is appropriate for a given situation.

Therefore our proposal is to try to improve on this static grounding by using a state aware AI planning approach for robot plan composition. In analogy with the described hier-

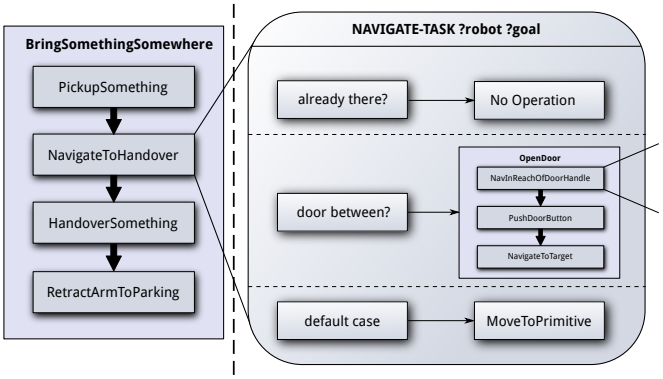


Figure 1: Augmenting abstract task description elements (left of the dashed line) with conditional decompositions (right)

archical representation of action descriptions, it is reasonable to try exploiting existing Hierarchical Task Network (HTN) planners for this objective. We use the SHOP2 planner (Nau et al. 2003) as an off-the-shelf planning component for this purpose. To encode the planning domain knowledge in a semantically expressive and widely used representation, the OWL web ontology language is employed. Fig. 1 shows an example for a task description along with the new annotations.

Related work

The problem of adding information for task execution of underspecified task descriptions has been tackled by several researchers before. The authors of (Beetz et al. 2011) describe the execution of a task for preparing pancakes, which is described in natural language retrieved from an Internet page. They extract an approximate task description using natural language text processing and match the respective action steps and the objects used to an ontology. By reasoning on the extracted structure along with semantic descriptions, a rough execution plan is generated. During plan execution, the information missing in the plan is inferred using different reasoning methods, using the CRAM framework described in (Beetz, Mösenlechner, and Tenorth 2010). This related work is especially interesting to this paper, as we rely on some of the tools provided by Beetz et al.. However, our goals differ: our interest is targeted on ways to instantiate and execute reactive robot plans that are adapted to the environment and on different hardware platforms, from abstract representations encoded in a machine-readable way.

Another interesting approach to the problem is to run a task execution in a simulated environment first. This allows a realistic projection of the possible outcomes and side-effects of task execution in a real environment, which is useful for improving the execution plan in advance. It also helps avoiding failures in task planning that occur through imperfect symbolic modeling of the robot's actions, like placing objects in a physically unstable way. The work described in (Mösenlechner and Beetz 2009) aims at optimizing execution plans using a rigid body physics simulation to project the behavior of a robot interacting with its environment. They apply transformational planning to improve the execution plans performance and robustness. A considerable drawback of this approach in practice is that while physics simulations provide good prospects on how a robot plan will perform in a specific environment, they also assume a very detailed description of the actors, and high computational effort to be accurate.

A different approach that is using HTN planning methods as a layer above task execution is presented in (Hartanto 2011). This work describes a hybrid system that combines OWL description logic reasoning techniques with HTN planning in order to have the system automatically omit superfluous information and keep the planning problem as small as possible. However, although they applied their work on a real robot during the RoboCup@Home challenges, they do not explicitly address the possibility to construct similar plans for systems with different hardware capabilities. Further, one goal of our work is the extension of our previously published task execution system, which requires the task descriptions to be formulated in the OWL variant *OWL Full* as opposed to *OWL DL*, which is used in the cited work.

In general terms of integrating knowledge represented in OWL with the HTN planning approach, (Sirin 2006) provides some interesting insights. In this work, a HTN planning method that uses OWL-DL for its planning domain representation is described. Its intended application is the automated composition of web services as opposed to creating robot plans. In their earlier work (Sirin et al. 2004) the authors describe a translation algorithm to create SHOP2 planning problems for web service composition using knowledge encoded in the vocabulary of the OWL-S process ontology.

Another highly interesting approach in the context of our work is presented in (Kaelbling and Lozano-Pérez 2011). The paper proposes a hierarchical planning approach combining symbolic and geometric planning as well as planning and plan execution. Their planner decides early on one possible decomposition and selects suitable decompositions for the sub-actions during execution time and is thus able to significantly decrease the search space. Our system uses the planning process to infer necessary actions from static information, like the robot hardware and environment description, and would thus not profit a direct application of this approach. However, we consider this approach to be a very promising direction for us to go in the future, when we adapt our system for more dynamic environments.

The work published in (Joshi et al. 2012) describes a system integrating stochastic planning with a reactive robot architecture. Due to long planning times, the planner is run off-line. It creates abstract policies that can be applied to operate in a highly reactive way in different environments. In contrast, our system creates one instantiation of a reactive plan, but is faster in common cases, due to the simpler, non-probabilistic HTN planning method employed.

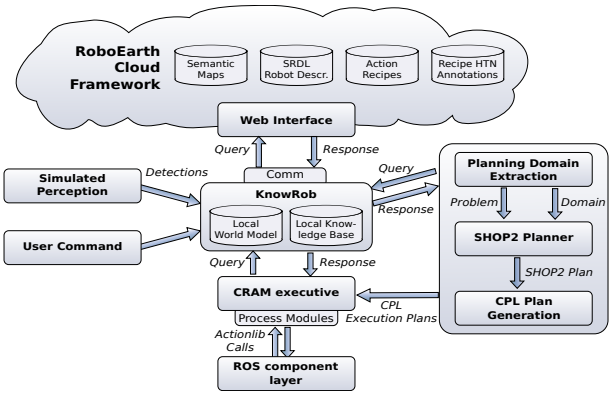


Figure 2: System overview

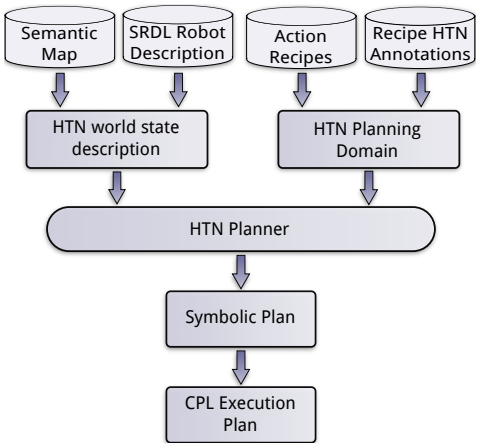


Figure 3: Plan generation process

Contributions

The core idea presented in this work is to make use of HTN planning to help instantiating task execution plans from abstract task descriptions and tailor them to a given environment and robot. For the actual task execution, we build upon the work on reactive plan execution provided by (Beetz, Mösenlechner, and Tenorth 2010).

System Overview

The overall system architecture is shown in Fig. 2. The planning domain knowledge is formulated in OWL and stored on the RoboEarth platform (Waibel et al. 2011), a database globally accessible via the world wide web.

An useful property of the system is that it separates the knowledge used. For instance, it makes use of four different sources of information, which are all stored on the database in OWL-based formats:

- Semantic maps encode a description of the environment.
- The robot hardware for different platforms is specified in terms of the Semantic Robot Description Language (SRDL) as proposed by (Kunze, Roehm, and Beetz 2011).
- Action recipes are abstract task descriptions, as mentioned in the previous section.
- Recipe HTN annotations are descriptions of task decompositions, i.e. preconditions for specific decompositions, and effects for basic operators. The right side of Fig. 1 provides an example.

The OWL descriptions are downloaded from the database and parsed by the KnowRob knowledge processing engine (Tenorth and Beetz 2009). KnowRob is based on a Prolog interpreter and can answer queries in Prolog syntax. It is used to read the information from OWL files and to do symbolic reasoning on the knowledge stored within. It can be easily connected to an object detection algorithm and a world model for object tracking, as described in (Di Marco et al. 2012) and (Elfring et al. 2012).

The module implementing the ideas presented in this paper communicates with the knowledge processing engine via Prolog queries. Fig. 3 shows its basic plan creation process. The semantic map for the respective environment is used together with the SRDL robot description to create the initial world state. It extracts a planning domain and problem in SHOP2 planner syntax as described in the following sections and tries to find at least one feasible plan. If there are multiple plans, the shortest plan (where the length is measured in terms of symbolic actions) is selected. The resulting plan is converted into an executable plan described in the CRAM plan language (Beetz, Mösenlechner, and Tenorth 2010), which finally gets executed on the robot using the ROS (Robot Operating System) framework[1].

Abstract Task Knowledge Representation

The RoboEarth language (Tenorth et al. 2012) is designed to describe task specifications for service robots from a high level view (i.e. without considering hardware or environment details which are not of interest for the task at hand). In this context, recipes are composed of a set of parametrized action primitives or other recipes. The structure is similar to Hierarchical Task Networks in the sense that sub-tasks might be decomposed recursively into other recipes. Recipes are represented as OWL classes that have sub-actions and parametrizations:

```
Class: PuttingSomethingSomewhere
  SubClassOf:
    Movement-TranslationEvent
    TransportationEvent
    subAction some PickingUpAnObject
    subAction some CarryingWhileMoving
    subAction some PuttingDownAnObject
    orderingConstraints value ActionOrdering1
    orderingConstraints value ActionOrdering2
...
Individual: ActionOrdering1
  Types:
    PartialOrdering-Strict
  Facts:
    occursBeforeInOrdering PickingUpAnObject
    occursAfterInOrdering CarryingWhileMoving
```

[1] http://www.ros.org

We consider basic actions that are implemented in the robot's hardware abstraction layer in terms of structured reactive controllers and thus are directly executable by the robot to be called "skills". Note that therefore the difference on which OWL classes represent skills depends on the robot platform used.

One advantage of having task descriptions represented this way is that they can be used by a wide range of robot platforms, provided that the basic action concepts referenced are grounded in executable actions for the given robot. While the task specifications are annotated with the requirements a robot needs to fulfill in order to be able to execute the task, it is not explicitly stated which actions have to be implemented by a robot platform as primitive skills. As a consequence, a robot might provide all required low-level primitive skills or it could replace a set of those skills with a single, more complex implementation. Making this decision is up to the developer of the robot's skill. This approach adds flexibility and eases the adaption of robot platforms to the system.

In this work, our intention is to continue the use of recipes from the previous system (Di Marco et al. 2012) and to adopt them as task decompositions for tasks in the HTN sense[2]. A simplified visualization is shown in Fig. 1. The abstract task description (depicted here without parameters) is on the left of the dashed line. It is basically a sequence of OWL concepts that refer to robot actions. The concepts can represent either basic actions like base navigation or grasping, or they can refer to other task descriptions. In this way, they can have different decompositions. However, the question of *when* these decompositions can be applied is not represented. This is what the task description annotations provide.

We created a custom ontology to represent a large subset of the functionality defined by the SHOP2 planning domain description language (see (Nau et al. 2003)) as OWL concepts. Currently supported are variables, predicates, axioms, operators and methods. The HTN method definitions link to the corresponding action recipe OWL identifier via an OWL property.

The representation stays close to the SHOP2 planning domain syntax. The basic building blocks are instances of the PlannerPredicate class, which represent logical atoms in the planning domain syntax, e.g. `(robot-at ?robot ?place)`. Logical expressions (i.e. Or-, And- or Not-Expressions) conjoin PlannerPredicate instances via the `hasOperand` property.

Neither OWL nor the RoboEarth language have a concept for variables that can have different values. In the task descriptions, objects that are to be manipulated are described as instances of OWL classes. They can be annotated with properties to help identify them further.

The sub-task parametrizations in the action recipes are implemented using OWL object properties which link to instances of objects in the assertional box. Thus, we require an explicit binding of the object properties in the recipes to each of the variables referenced in the preconditions and effects in operators or HTN-tasks. These are implemented as instances of the VariableMapping class, linking object properties to variable names in operator, method, or axiom descriptions.

```
Individual: NavigateVarMapping
  Types:
    plan:VariableMapping
  Facts:
    plan:mappedFrom knowrob:toLocation
    plan:mappedTo targetLocVar1
```

As robot hardware and thus robot capabilities can differ significantly, we expect that not every robot can use the same decompositions for the hierarchical task network. It therefore must be possible to define operators and HTN-tasks in multiple ways, depending on the robot platform used. The system allows for this by decoupling the ontology describing the operators and decompositions of HTN-tasks from the action recipe definitions. To ensure a common vocabulary, an ontology describing core concepts based on the KnowRob ontology is used. Operators and HTN-Tasks are mapped onto concepts from the common ontology.

Environment and Hardware Information

To generate a useful execution plan for a specific environment, information on the environment is necessary. E.g. the types or the expected initial positions of objects to interact with, a semantic map as defined in (Tenorth et al. 2012) is used.

Our simulation example in the next section considers a task of navigation. More specifically, the task is to infer that a command to navigate between rooms might also mean to traverse a door. For this purpose, we extended the semantic map to incorporate a simple kind of topological map by describing regions in space that are adjacent and are thus connected to each other. For example the semantic map contains the information that the region in front of the first cabinet is adjacent to the region where the door button is located. Note that this information could as well be generated automatically.

In order to generate plans for manipulation actions, the system also needs to take the robot's hardware setup into account, e.g. a description of the available manipulators and their initial configurations. The system thus requires a semantic description of the robot platform, which is available in the previously described system (Tenorth et al. 2012). The robot's physical and cognitive capabilities are being represented using the Semantic Robot Description Language (SRDL) (Kunze, Roehm, and Beetz 2011) and stored on the web database. In order to get rid of the tedious work of manually editing the SRDL description, we developed a conversion tool that automatically converts robot descriptions created with the help of the Uniform Robot Description Framework (URDF) into SRDL. In addition to the mere kinematic structure present in the URDF description, the SRDL document subsumes the structural parts under component groups, e.g. arms (this is done automatically by reading in configuration files for the manipulation planner, which defines planning groups for each arm). Furthermore, the capabilities of

[2]To help distinguishing between HTN-style tasks and the more generic word "task", we will call the former "HTN-task" in the remainder of this paper.

the robot can be explicitly advertised. This knowledge is used to check whether a robot provides the prerequisites for executing a given task with its specific requirements for sensors, actuators or software algorithms.

For our experiment described in section *Simulation Experiment* we generate SRDL descriptions for the Amigo (Lunenburg et al. 2012) and the Care-O-Bot 3-4 (Parlitz et al. 2008) robots. Fig. 4 depicts an example for the Amigo robot. The capabilities needed to run the experiment are *GraspingCapability*, *gripper_action*, *move_arm* and *move_base*, which notify the system that the robot is able to control a gripper, move its arm and its base and to grasp something. The OWL individual *AmigoLeftArm* describes the links and joints, which form the left arm of the robot, by defining the base link and the tip links of the kinematic chain of the arm.

Executable Robot Plan Instantiation

The task plan generated by SHOP2 is a sequence of operator calls parametrized by the symbols described in the initial world state. It is not immediately possible to execute this kind of plan on a robot. We use the CRAM plan language (CPL) to specify the generated robot execution plans. The plans consist of calls to reactive execution plans provided in a manually crafted plan library specific to the respective robot platform, also written using CPL. CPL builds on the Common Lisp programming language and provides several interesting features to facilitate the problem of writing reactive robot execution plans.

CPL allows the definition of process modules that allows grouping different robot functionality (e.g. for navigating the robot base) and provide a common interface to different underlying hardware drivers. In our system, we manually aligned the calls of process modules with the defined operators.

Also, CPL supplies the concept of designators. These are symbolic descriptions that specify more detailed information about actions, describing e.g. objects and locations. As was proposed by (Beetz, Mösenlechner, and Tenorth 2010), we use them to encode information that is to be resolved during plan execution time. Our system generates a designator for each object and location for each symbol that was created for describing the initial world state while omitting symbols that are not used in the actual, generated SHOP2 plan. E.g. the statement `(in-center-of coke1-pose coke1)` in the initial world state gets converted into the location designator

```
(coke1-pose
    (location `((in-center-of ,coke1))))
```

using the knowledge defined in the semantic map that "coke1" is an object and "coke1-pose" is a location. The designator definition is added before the plan definition, as can be seen in Fig. 8. Robot parts to be used in the task, like manipulators or actuated sensors, are also added as object designators. They are used for e.g. specifying which arm to use. Designators representing robot hardware are resolved in the process module for the corresponding platform.

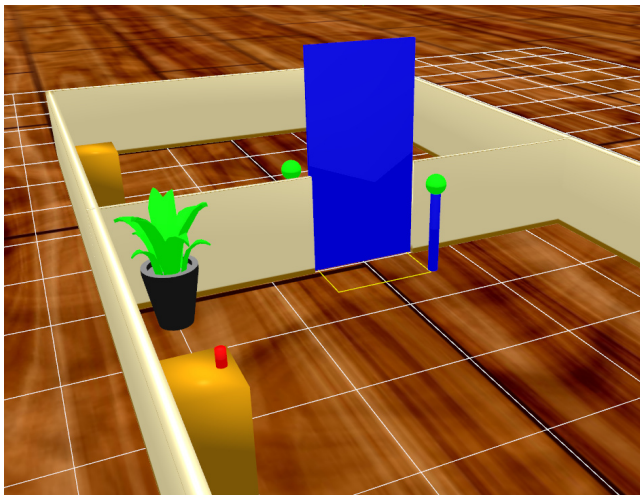


Figure 5: Simulated world used in the experiment.

As designators are basically Common Lisp variables that can depend on each other, we need to make sure that they are defined in the right order. We extract the corresponding dependency graph and apply a topological sorting algorithm to ensure a proper definition order. Object designators that describe robot parts or general concepts are currently resolved in the process modules. E.g. `object-state-closed` is used for stating that the gripper should be closed and `cobarm` refers to the KUKA manipulator of the Care-O-Bot robot. Object designators get resolved by querying the knowledge processing system. In the current implementation, objects and their poses are resolved by their type only. Note that this can lead to problems in environments with several instances of the same object type. However, the system can be extended to use a globally unique identifier for objects provided an object tracking system capable of identifying objects consistently.

The integration of perception is a highly important and challenging problem in generating robust executable robot plans. For this work, we simulated a simple passive perception which steadily publishes object detection results into the KnowRob system, as long as the object is approximately in line of sight of the robot. Thus, the robot execution plan is limited to actuation commands.

Simulation Experiment

To demonstrate our system a simplified simulation environment has been created, in which both the TU/e Amigo and the Fraunhofer Care-O-Bot will perform the task of transporting a drink from one area to another, see Fig. 5. The simulation experiment is run in the Gazebo simulator[3].

The main goal of the experiment is to demonstrate that although the two systems differ in hardware topology (i.e. two arms for the Amigo robot versus one arm for the Care-O-Bot), they are both capable of performing the same task by allowing different task decompositions. The main challenge

[3] `http://gazebosim.org`

```
Class: Amigo
  SubClassOf:
    knowrob:Robot,
    (srdl2-cap:hasCapability some srdl2-cap:GraspingCapability)
     and (srdl2-cap:hasCapability some srdl2-cap:gripper_action)
     and (srdl2-cap:hasCapability some srdl2-cap:move_arm)
     and (srdl2-cap:hasCapability some srdl2-cap:move_base)
...
Individual: AmigoArmLeft
  Annotations:
    srdl2-comp:endLinkOfComposition amigo:amigo_finger1_left,
    srdl2-comp:endLinkOfComposition amigo:amigo_finger2_left,
    srdl2-comp:baseLinkOfComposition amigo:amigo_shoulder_yaw_joint_left
  Types:
    srdl2-comp:ComponentComposition,
    PhilippsArm
```

Figure 4: Excerpt of the semantic description of the Amigo robot used in the experiment

both robots will have to overcome is to open the door that separates the two areas. This can be achieved by touching one of the buttons next to it. The door will then stay open for 45 seconds.

To start the process, a human operator has to specify the task to be executed, its parameters (i.e. the object to operate on), the robot and the environment in terms of OWL identifiers. Note that the latter two could in theory inferred automatically.

Amigo

The "opening the door" action is implemented as a HTN task and part of one possible decomposition of the "navigate" HTN task (so it could be more accurately named "pass a closed door"). Other decompositions for the latter are "no operation" (NOP) (in case the robot's current pose matches the goal pose), moving the base to the goal via an corresponding operator if the current base pose and the target pose are adjacent in the topology of the environment and a recursion step for chaining multiple navigation steps.

For the task of transporting a drink the task for Amigo decomposes into

- navigating to an approach pose in front of the drink,
- picking up the drink with one arm,
- navigating to the door,
- operating the door button with another arm,
- navigating to the drop-off location,
- dropping off the drink.

These steps are visually depicted in Fig. 6. The system provided two similar plans, that only differed in the manipulators used. E.g. in the first plan the robot used the right manipulator to pick up the target object and the left to operate the door button, while in the second plan the order was reversed. In cases like this, where there is no apparent advantage of multiple plans, the system arbitrarily selects the first one.

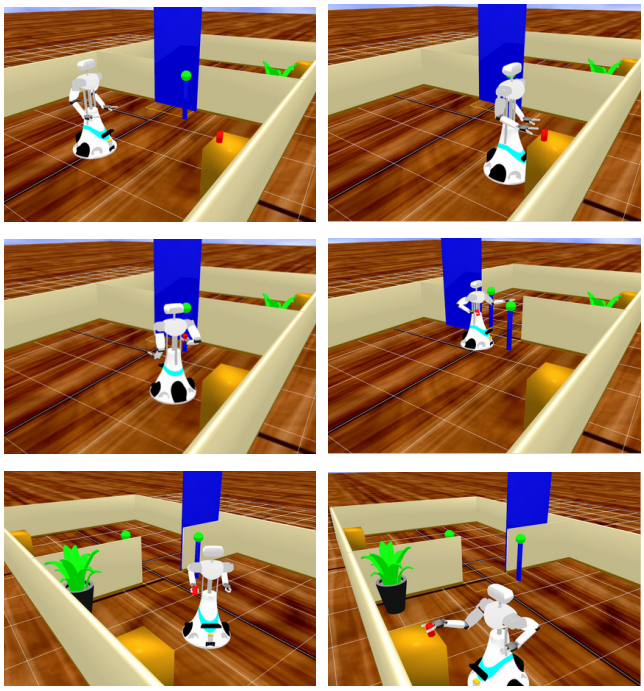


Figure 6: Plan execution steps performed by Amigo.

Care-O-Bot 3-4

The Care-O-Bot differs from the Amigo robot in that it has only one arm. To solve the scenario described above, we chose the solution of having the Care-O-Bot use its movable tray to temporarily place the drink upon, while it is manipulating the door button with its manipulator. It is noteworthy that the button could in theory be operated with the object in the gripper. However, our intention is to simulate opening a real door, so the experiment setting assumes that the arm has to be free. This is implemented in the planning domain by two additional HTN tasks. The first is called `FreeArmForGrasping`, which decomposes into NOP if

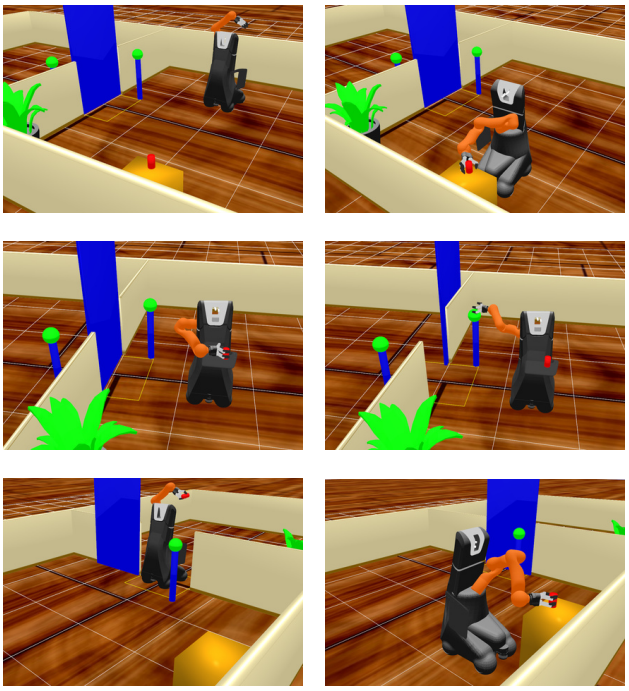


Figure 7: Plan execution steps performed by Care-O-Bot.

at least one arm is free (i.e. not attached to any object), and into an operator to put an object from the gripper on the tray if a tray is available and no arm is free. The other HTN task is called `PrepareNavigation`, which clears the carrying tray and puts the arms in parking position if necessary. PrepareNavigation is defined recursively, as it can be necessary to execute more than one of the mentioned actions.

A visual overview of the steps involved for the Care-O-Bot is shown in Fig. 7.

Conclusion

We presented a system to create robot execution plans for heterogeneous robot platforms by HTN planning on knowledge encoded in OWL. The system makes use and extends former work that was geared towards encoding task descriptions in an abstract, hardware-agnostic way. We showed its functionality by having two distinct robots execute the same task description and coping with their difference in hardware setup.

An useful feature that was inherited and extended from the former, static task execution component, is that different kinds of knowledge are kept separate. For instance, knowledge about the robot hardware is kept separate from knowledge on HTN task decompositions or the environment. This allows for easy replacement of parts and greater applicability in new environments or for new robots. Also, symbols and concepts from the underlying OWL knowledge representation are aligned in the whole chain from planning to execution; and this can be used for further reasoning during execution time.

On the other hand, the additional overhead necessary for fully describing the planning domain in OWL is significant compared to the previous approach of only representing high level actions; the given example domain encoded in RDF/XML required for instance around 4000 lines of XML.

```
(def-cram-function generated-plan nil
  (with-designators
      ((bed1
        (object
         '((name bed1)
           (type bed--piece-of-furniture))))
       (cobtray
        (object
         '((name cobtray)
           (type component-composition))))
       (object-state-closed
        (object '((name object-state-closed))))
       (door1
        (object
         '((name door1) (type door))))
       (bed1-reachable-space
        (location
         `((in-reach-of ,bed1)
           (connected-to ,door1))))
       ...
       (object-state-open
        (object '((name object-state-open)))))
    (achieve-operator
     `(!move-to-operator ,cob3-4-robot1
         ,icetea1-reachable-space))
    (achieve-operator
     `(!change-gripper-state-operator ,cobarm
         ,object-state-open))
    (achieve-operator
     `(!move-arm-to-operator ,cobarm
         ,icetea1-pose))
    ...))
```

Figure 8: Excerpt from the generated plan for the Care-O-Bot robot

In the future, we will try to apply the approach in more difficult environments and on more diverse robots. Another interesting further direction would be to use or even learn additional information in the environment, like how long the door will stay open after the button has been triggered. In the scenario described the robot has no knowledge about how long the door will stay open after the button has been pressed.

Finally, we realize that the separation of plan generation and execution reduces the robustness of the system. Future work will focus on integrating the plan generation more deeply into plan execution. To achieve that, we also will need to adapt the representation of the planning domain to be less dependent on the SHOP2 planner.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no 248942 RoboEarth.

References

Beetz, M.; Klank, U.; Kresse, I.; Maldonado, A.; Mösenlechner, L.; Pangercic, D.; Rühr, T.; and Tenorth, M. 2011. Robotic Roommates Making Pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots*.

Beetz, M.; Mösenlechner, L.; and Tenorth, M. 2010. CRAM - a cognitive robot abstract machine for everyday manipulation in human environments. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 1012–1017. IEEE.

Di Marco, D.; Tenorth, M.; Häussermann, K.; Zweigle, O.; and Levi, P. 2012. Roboearth action recipe execution. In *12th International Conference on Intelligent Autonomous Systems*.

Elfring, J.; van den Dries, S.; Molengraft, M.; and Steinbuch, M. 2012. Semantic World Modeling Using Probabilistic Multiple Hypothesis Anchoring. *Robotics and Autonomous Systems*. accepted / in press.

Hartanto, R. 2011. *A hybrid deliberative layer for robotic agents: fusing DL reasoning with HTN planning in autonomous robots*, volume 6798. Springer.

Joshi, S.; Schermerhorn, P.; Khardon, R.; and Scheutz, M. 2012. Abstract planning for reactive robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 4379–4384. IEEE.

Kaelbling, L., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1470–1477. IEEE.

Kunze, L.; Roehm, T.; and Beetz, M. 2011. Towards semantic robot description languages. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 5589–5595. IEEE.

Lunenburg, J.; van den Dries, S.; Elfring, J.; Janssen, R.; Sandee, J.; and van de Molengraft, M. 2012. Tech United Eindhoven Team Description 2012. In *RoboCup Team Description Papers 2012*.

Mösenlechner, L., and Beetz, M. 2009. Using physics- and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior. In *19th International Conference on Automated Planning and Scheduling (ICAPS'09)*.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

Parlitz, C.; Hägele, M.; Klein, P.; Seifert, J.; and Dautenhahn, K. 2008. Care-obot 3 - rationale for human-robot interaction design. In *Proceedings of 39th International Symposium on Robotics (ISR), Seoul, Korea*.

Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN planning for web service composition using SHOP2. *Web Semantics: Science, Services and Agents on the World Wide Web* 1(4):377–396.

Sirin, E. 2006. *Combining description logic reasoning with AI planning for composition of web services*. Ph.D. Dissertation, University of Maryland.

Tenorth, M., and Beetz, M. 2009. Knowrob—knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 4261–4266. IEEE.

Tenorth, M.; Perzylo, A.; Lafrenz, R.; and Beetz, M. 2012. The RoboEarth language: Representing and Exchanging Knowledge about Actions, Objects, and Environments. In *Robotics and Automatic (ICRA), 2012, IEEE International Conference on*.

Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Häussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; et al. 2011. Roboearth. *Robotics & Automation Magazine, IEEE* 18(2):69–82.

Open World Planning for Robots via Hindsight Optimization

Scott Kiesel[1] and **Ethan Burns**[1] and **Wheeler Ruml**[1] and **J. Benton**[2] and **Frank Kreimendahl**[1]

[1]Department of Computer Science
University of New Hampshire
skiesel, eaburns, ruml, fri2 at cs.unh.edu

[2]Smart Information Flow Technologies (SIFT), LLC
Minneapolis, MN USA
jbenton@sift.net

Abstract

Classical planning makes the closed world assumption in which all relevant aspects of the world are known at planning time. While this assumption holds in some domains, in many practical robotics domains the existence of relevant objects or the states of relevant fluents are initially unknown and must be actively discovered. Previous proposals for open-world planning either employ complex and expensive knowledge representations or depend on ad hoc assumptions. In this paper, we show how hindsight optimization provides a simple and general approach to planning in open and partially observable worlds. Hindsight optimization samples multiple possible worlds that are consistent with the agent's current knowledge, generates a plan in each respective world, and then selects the action that maximizes expected reward over these samples. While this approach is approximate, we demonstrate both in simulation and on a physical robot that this simple technique performs well and is more scalable than previous methods on standard benchmarks.

Introduction

Imagine a rescue robot entering a partially-destroyed building to search for survivors of an earthquake. The agent does not know the initial layout of the building, what new obstructions may exist, the locations of potential victims, or even how many victims there are. In open-world planning problems like this, the agent is not given a complete description of the initial state of the world, but it can perform sensing actions to determine the existence of relevant objects and the values of important fluents. To be useful, the planner must be fast enough to not materially delay the actions of the robot. It must be able to plan to discover and take into account newly sensed information, and ideally it would be expressive enough to handle soft goals, durative actions, temporal constraints, and actions with uncertain outcomes.

In this paper, we propose a simple on-line planning approach that handles the requirements of open-world domains. We call this new approach *Optimization in Hindsight with Open Worlds* (OH-wOW). Rather than using traditional techniques that compute a policy or contingent plan in advance, we estimate on-line at each step which action is best in light of our current knowledge of the world. The OH-wOW approach is domain agnostic and does not commit to a particular representation for open-world knowledge or goals. Instead, it can leverage any closed-world planner appropriate for the underlying domain. Our central assumption is that the agent possesses some knowledge, likely probabilistic, about the domain. In our view, performing well in an open-world depends on having expectations about that world, e.g., building dimensions are typically tens or hundreds of meters rather than centimeters or kilometers, or that people are usually found in certain densities per square meter, or are more often found in certain areas, such as offices. This type of default or prior information can be overridden by direct experience, but ought to play a role in planning until it is discovered to be inaccurate. We use these expectations to generate possible states of the world consistent with the agent's current knowledge, use a closed-world planner to estimate the future reward achievable in those worlds after taking each currently-applicable action, and then select the action with the highest expected reward.

After describing OH-wOW in detail, we contrast it with previous work. We then report on the method's empirical performance, both in simulated domains and when deployed on a physical mobile robot fully integrated with the Robot Operating System (ROS), Simultaneous Localization And Mapping (SLAM) and standard navigation. Our experience indicates that the method is surprisingly general and practical, achieving results as good as those of previous systems but with lower planning times and fewer ad hoc assumptions. This work showcases the power of Monte Carlo techniques and adds open-world planning to the list of non-classical planning settings in which simple planners can be leveraged to provide state-of-the-art performance.

A Hindsight Optimization Approach

Optimization in hindsight was originally developed for scheduling and networking problems (Chong, Givan, and Chang 2000; Mercier and van Hentenryck 2007; Wu, Chong, and Givan 2002) and has recently been applied to probabilistic planning (Yoon et al. 2008; 2010). In these previous settings, sampling is used to resolve uncertainty in the outcome of actions. In our context of open-world planning, each sample forms a concrete hypothesis about the world—which objects might exist and which fluents might hold. While these will likely be revealed to the agent as it performs actions that have, a priori, uncertain outcomes, the sampling process for open-world planning is more involved

than choosing an outcome in a PPDDL (Younes and Littman 2004) action or RDDL (Sanner 2011) description. For example, a rescue robot will generate possible world states with conceivable floor plans for the building, each with sets of victims distributed in various plausible locations. Each of these sampled worlds may potentially determine the outcome of multiple sensing actions during the course of the corresponding planning episode. Demonstrating the practicality of this approach is the central contribution of this paper.

While these samples of possible worlds are intentionally not exhaustive, they are intended to provide useful relative judgements on the expected value of actions. In order to estimate the value of an action, we apply that action in each of the sampled possible worlds, find closed-world plans from the resulting states, and average over the resulting plan costs. The action with the lowest average plan cost over the sampled worlds is chosen to be executed.

More formally, we define the value of being in a state s_1 as the minimum expected cost over plans that extend from s_1. That is, the minimum cost over all possible future action sequences of the total cost over all expected future states:

$$V^*(s_1) = \min_{A=\langle a_1,\ldots,a_{|A|}\rangle} \; \underset{\langle s_2,\ldots,s_{|A|}\rangle}{E} \left[\sum_{i=1}^{|A|} C(s_i, a_i) \right]$$

where $C(s, a)$ represents the cost of performing action a in state s. In open-world planning, these future states incorporate the sensed knowledge of the agent and the expectation is over the distribution of sensing outcomes. The agent will expect different outcomes based on its beliefs about the world. Given our expectations about sensing outcomes, we would like to find the action sequence $A = \langle a_1, ..., a_{|A|}\rangle$ that minimizes the expected sum of action costs. To compute V^* exactly, we would need to compute the expectation for each of exponentially many plans.

In optimization in hindsight, we approximate the value function by exchanging expectation and minimization, so that we are taking the expected value of minimum-cost plans instead of the minimum over expected-cost plans:

$$\hat{V}(s_1) = \underset{\langle s_2,s_3,\ldots\rangle}{E} \left[\min_{A=\langle a_1,\ldots,a_{|A|}\rangle} \sum_{i=1}^{|A|} C(s_i, a_i) \right]$$

This approximation of $V^*(s)$ uses fixed sensing outcomes in each minimization. As in other applications of optimization in hindsight, the stochastic elements have been reduced to known outcomes by sampling. For each possible outcome in the expectation, the problem is to minimize cost given a known world, i.e., standard, closed-world, cost-minimizing, deterministic planning. In OH-wOW, fixed sensing outcomes are generated using concrete hypotheses about the state of the world. For each fully-known, deterministic world hypothesis, the agent can compute the result of different sensing outcomes when solving the minimization in the equation for V^*. For example, the result of querying a vision system to look for an injured person depends on whether or not there is an injured person in the sensed portion of the world—this is fully-known for each hypothesis.

OH-wOW($s = \langle agent, world\rangle, N$)
1. for i from 1 to N do
2. $\quad w_i \leftarrow sample\_world(world)$
3. foreach action a applicable in s
4. $\quad s' \leftarrow a(s)$
5. $\quad c \leftarrow (\sum_{i=1}^{N} solve(s', w_i))/N$
6. $\quad Q(s, a) \leftarrow C(s, a) + c$
7. Return $argmin_a\, Q(s, a)$

Figure 1: The OH-wOW algorithm.

The agent is aware of what features are truly known and which are merely hypothesized, as a result the deterministic problem can require sensing actions before the agent interacts with hypothesized portions of the world. In this way, the system will still be required to plan to sense. A dummy precondition is added to all actions that involve a hypothesized variable. This precondition enforces that the value of that variable is sensed before actions requiring the value are executed. This ensures that the resulting plan executes sensing actions appropriately. More concretely, if the agent hypothesizes that there is an injured person in a room, then the deterministic planner will require a sensing action before that person can be reported. When a sensing action is carried out in the physical world, its result may differ from the hypothesis. This new information will be reflected in the samples taken at the next planning step.

We define the Q-value to be the cumulative expected cost of taking an action a_1 in state s_1:

$$Q(s_1, a_1) = C(s_1, a_1) + \underset{\langle s_2,s_3,\ldots\rangle}{E} \left[\min_{A=\langle a_2,\ldots,a_{|A|+1}\rangle} \sum_{i=2}^{|A|+1} C(s_i, a_i) \right]$$

From this, we estimate the best action choice in s_1 as $\min_a Q(s_1, a)$. Using this technique, we are said to be performing optimization with the benefit of 'hindsight' knowledge about how future uncertainty will be resolved.

The pseudocode in Figure 1 summarizes the algorithm. At each time step, the algorithm is used to find the next action to execute from the current state s, which includes information about both the agent's current configuration and its current knowledge about the world. First, we generate a set of N possible worlds that are consistent with the agent's current knowledge (lines 1–2). Next, for each currently applicable action a, we consider the resulting state $s' = a(s)$ (line 4). Then, each possible world w_i is initialized with the state s', generating a fully-known closed-world deterministic planning problem. Recall that, to incorporate sensing, the determinized problem requires the agent to sense before interacting with hypothesized features of a sampled world. Solving this problem provides an optimistic estimate of the cost from s'. The mean cost across the set of samples (line 5) along with the cost of the action $C(s, a)$ is used as the Q-value for each applicable action a in the original state s (line 6). Finally, we return the action with the minimum Q-value (line 7), the agent takes the action, possibly observing new facts and objects in the world, yielding a new current state, and the cycles begins anew.

Related Work

Open world planning is a broad problem that has been attacked from many angles. One issue is how to represent knowledge and goals related to open-ended sets; Etzioni and Weld (1994) and Babaian and Schmolze (2006) have addressed this. We do not address this issue in this paper, except to point out that the underlying planners used in our approach are closed-world and do not require a particularly expressive (and expensive) representation language. We do require that the agent tracks what is currently known about the world and that the world generator respects this knowledge when sampling possible worlds.

In conformant planning (Cimatti, Roveri, and Bertoli 2004, inter alia), one requires plans that are guaranteed to work without sensing. For most robotics domains, this is overly restrictive and renders problems unsolvable. Contingent planning (Meuleau and Smith 2003, inter alia) allows for sensing, but computes a plan before beginning execution. In addition to handling open-worlds, we aim to scale to domains in which the number of contingencies may be very large (e.g., the number of possible floor plans), making synthesis of branching plans prohibitively expensive.

In the POMDP literature, computing actions on-line is recognized to provide increased scalability (Ross et al. 2008). However, many POMDP algorithms attempt to compute future belief states of the agent, which can be expensive and cumbersome. Optimization in hindsight represents an extreme approach, disregarding future belief uncertainty and assuming that the agent can achieve the cost accrued by the plans for the fully-observed sampled worlds. Our work is perhaps most closely related to work on sampling techniques for POMDPs, where a particle filter approximates the belief space during sampling (Silver and Veness 2010). Open world planning goes beyond traditional factored POMDP representations (Boutilier, Dean, and Hanks 2011) because the structure of the world state requires representing a logically infinite domain of discourse; the universe of objects that exist and the possible relationships between them remain unknown to the agent (Doshi 2009).

There has been sustained interest from roboticists in open-world planning. One way of handling open-world planning in practice is to force the robot to move in one direction simply to explore without a concept of cost or reward. Such simple ad hoc approaches cannot exploit the agent's expectations about goals (e.g., people are likely in offices) or take sensed information into account (e.g., a hallway implies new rooms to explore). Talamadupula et al. (2010) present an approach where the planner assumes objects exist in order to instantiate goals and motivate a search and rescue robot to collect reward by discovering and reporting victims. As new information arrives about the environment, the planner replans. This can be seen as a degenerate form of our hindsight approach, where the robot operates on a single optimistic "sample". While it is simpler, it cannot generalize to domains where uncertainty is a major component.

Joshi et al. (2012) use offline symbolic dynamic programming with known goals but unknown numbers or locations of objects, which does allow for reusable policies on any instance of the domain. However, in their experiments the number of possible objects was severely limited to retain feasible computation times (they require 4 hours for their 3 room example), which makes the resulting policies suboptimal. They also do not handle temporal constraints, action costs, or goal rewards.

Evaluation

We evaluate OH-WOW by applying it in two domains: the classic omelette benchmark for planning under uncertainty, and urban search-and-rescue, which we investigate both in simulation and using a physical robot.

Omelettes

In the omelette benchmark, introduced by Levesque (1996), the agent is attempting to make a three-egg omelette with ingredients of unknown freshness. The agent has four available actions. The agent can *break* an egg into a bowl, *pour* the contents of a bowl into another bowl or the trash, *wash* a bowl, or *sniff* whether the eggs in a bowl are good. All actions are deterministic except for the sniff action. The goal is to have exactly three good eggs in a specific bowl with no trace of bad eggs. To make the domain more challenging, we extended it to have both regular white eggs, which are bad with a probability of 0.5, and local brown eggs, which are bad with a probability of 0.1. The agent is able to observe the color of the next available egg without requiring a sensing action.

We compared OH-WOW to a perfectly omniscient oracle and also to a hand-coded controller. The controller puts eggs into the goal bowl, sniffing after each addition and cleaning out bad eggs until it finds a good one. Then it does the same routine using an extra bowl, pouring good eggs into the goal bowl from the extra bowl until the goal is reached. We generated three sets of 100 random instances, each set with a different probability of the next egg being brown. OH-WOW used a domain-dependent deterministic planner based on uniform-cost search.

Figure 2 (left) shows the distribution of the resulting plan costs using box and whisker plots. Each box surrounds the middle 50% of the data, with a horizontal line indicating the median and whiskers indicating the range (values beyond $1.5\times$ the inter-quartile range are shown as circles). The gray vertical stripes inside each box show 95% confidence intervals on the mean. The plot shows the increase in cost over the optimal solution found by the oracle, of the hindsight planner with 32 and 256 samples, and the hand-coded controller (ctlr). The boxes are grouped by the probability of an egg being brown (0.0, 0.5, and 1.0). We can see that, when all eggs were white, the hindsight planner with 256 samples had a median cost that was less than the hand-coded controller (significant with $p < 0.05$ via the Wilcoxon signed-rank test). As the probability of a brown egg increased, the hindsight planner performed better, nearly dominating the controller when all eggs were brown. This is likely because the hindsight planner could recognize that brown eggs tend to be good, and put multiple into a bowl before bothering to smell, saving redundant sniff actions.

The average total planning time on a 3.1 GHz Core2 PC for OH-WOW to reach the goal using 256 samples on a

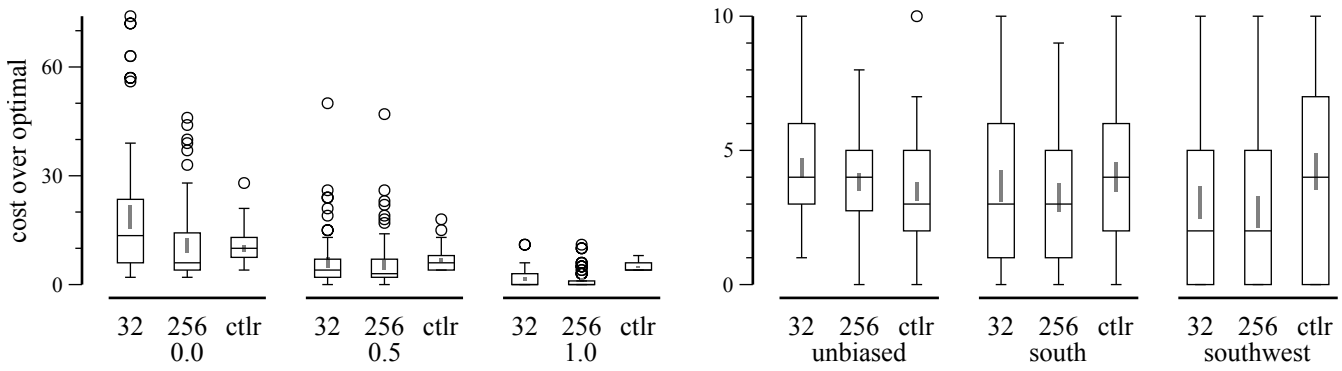


Figure 2: Plan cost in the three-egg omelette domain (left), and the search and rescue domain (right).

problem without brown eggs was 12.9 seconds (standard deviation 8.0 seconds). Each plan was an average of 24.9 actions long (standard deviation 13.7 actions) and each action in the plan took an average of 0.52 seconds to select (standard deviation 0.31 seconds) before executing it. This compares favorably with the 185 seconds of offline planning reported for approximate RTDP (CPU unspecified) by Bonet and Geffner (2001). Levesque (2005) also generates full plans offline to solve the three-egg omelette in 1.4 seconds but requires 1,681 seconds if the omelette is scaled to four eggs. When using four eggs, OH-wOW's costs relative to optimal were similar to the three-egg case, and total computation time averaged only 76.7 seconds (standard deviation 43.6). Each plan was an average of 49 actions long (standard deviation 27.3 actions) and each action in the plan took an average of 1.57 seconds to select (standard deviation 0.99 seconds) before executing. The plans found by Levesque's planner also contain strictly more actions than our hand-coded controller (which in turn finds more costly plans than OH-wOW on the median), as Levesque's solution always uses the auxiliary bowl for staging and requires an additional pour action to move the first good egg into the goal bowl.

Search and Rescue

Now, we return to the motivating example of search and rescue robotics. The robot's objective is to maximize the number of injured people it reports while still returning to its starting location by a given hard deadline.

To generate possible worlds for OH-wOW, we need to generate building layouts consistent with the robot's current map and hypothesize the possible locations of injured people. We represent building layouts as rough topological maps. We assume that undiscovered nodes will lie on a uniform four-connected grid, and that a known node can be extended if it has an adjacent grid cell that can be reached without going through an obstacle or crossing an existing edge in the map. We iteratively choose an extendable node, generate a valid neighbor and connect them. We use a bias toward extending the most recently added node, and toward generating the neighbor that forms a straight line from the

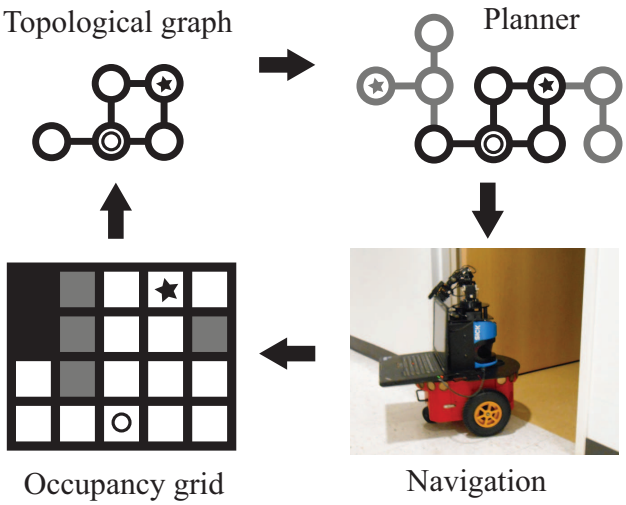


Figure 3: Architecture diagram.

chosen node's parent. This was sufficient to yield plausible building layouts with hallways. Victims are generated independently with fixed probability per hypothesized node. The upper right panel of Figure 3 shows a very small example map with hypothesized extensions shown in gray.

The base planner used by OH-wOW precomputes all-pairs shortest-paths among nodes containing people and the start location. It then uses depth-first search, considering at each step to visit each unreported person or return home. The available actions depend on the remaining time. For efficiency, we avoid considering time as a separate state variable by incorporating it into the cost function (Phillips and Likhachev 2011).

Simulation To test the planner in simulation, we created 100 random worlds with 100 nodes each. We considered three victim distributions: *unbiased*, uniform probability of 0.1 per node; *south*, nodes south of the start location contains a person with probability 0.2 and nodes north of the start location 0; and *southwest*, southwest of the start 0.4

	victims found			
deadline	0	1	2	3
1 minute	4	6	0	0
5 minutes	0	7	3	0
10 minutes	0	3	4	3

Figure 4: Number of injured victims found and reported over 10 runs using a physical robot.

and 0 elsewhere. These distributions are representative of helpful domain knowledge that can be leveraged when generating possible worlds. Skewing the probability of a victim's existence to one side of the building could be used to to represent the knowledge of a closed wing of the building or a scheduled company-wide event. We limit the total number of victims to 10. The cost of a plan is the number of unreported people remaining when the agent returns home and performs a dummy *finish* action. We compared OH-wOW to two different algorithms. The first is an oracle that knows the exact configuration of the building and location of all victims. The second is a hand-coded controller that performed a depth-first exploration of the building, reporting people that it encountered and returning to the start location when it had no more time to explore.

To gauge the complexity of these instances, we must consider the number of possible configurations of maps and victims. Considering only $n \times n$ grids, there are $2(n-1)n$ possible places for edges; our generator is limited to trees, so it must pick $n^2 - 1$. For $n = 10$, this is $\binom{180}{99} \approx 10^{52}$ maps. For each possible map, we must choose locations for victims; for 10 victims, there are $\binom{100}{10} \approx 10^{13}$ possible configurations on the map. Maintaining a belief over so many possible worlds would be challenging. Thankfully, it also seems unnecessary if we merely wish to estimate the expected value of actions.

Figure 2 (right) shows results, grouped by the victim distributions. For the unbiased case, the hand-coded controller gave the best performance, but OH-wOW was quite competitive. With a biased distribution, OH-wOW was superior as it was easily able to leverage prior knowledge about possible worlds. The average maximum per-action planning time for OH-wOW with 256 samples was 2.7 seconds (standard deviation 0.85 seconds). In order to compare with Joshi et al. (2012), we also ran smaller instances with at most three victims. The average maximum per-action time for 256 samples was 0.18 seconds (standard deviation 0.035 seconds), which is negligible compared to typical mobile robot latencies.

Physical Robot We also integrated OH-wOW with the Robot Operating System (ROS, `www.ros.org`) on a 3.7 GHz quad-core i7 laptop on-board a Pioneer 3dx equipped with a SICK LIDAR shown in Figure 3. We use the ROS Gmapping SLAM stack to generate a fine-grained occupancy grid, from which we extract a topological map with edge lengths of 1 meter to provide to OH-wOW. Figure 5 shows a final topological map overlayed on the corresponding SLAM map created during an experiment run of the search and rescue application. In the topological map, nodes are marked as either black, green, or pink. Pink nodes indicate an area of the building where a victim was found and reported. Green nodes are points in the map that can be extended when creating possible building layouts. The black nodes indicate that the layout of the building can not be extended from this area.

The ROS Navigation stack is used to execute movement actions, which are specified as the topological node to visit next. These topological nodes are then mapped to a two dimensional point in the map built by SLAM before issuing the move action to the robot. In some cases, the rough topological graph places a node very near to an obstacle and the planner can not find a safe way to achieve the requested action. We supplemented the navigation component in these instances by issuing a set of perturbed points around the initial point before returning failure to the planner. This set was simply four points, one in each cardinal direction, one half of a discretization away.

We performed experiments in a hallway of approximately 20 meters with between 2 and 5 open doors to offices and 3 victims. We simulated detection of a victim using the range capability of the laser rangefinder. When the laser is able to collect data and populate a portion of the map corresponding to certain pre-selected locations (that were unknown to the planner), we pass that detection information along to the planner. In order to report a victim the robot must navigate to the containing topological node.

We used three different deadlines, one minute, five minutes and ten minutes. As shown in Table 4, the performance of the robot improves as it is given more time to search for victims. In all experiments the robot returned within the hard deadline we provided. At first, only given a short deadline of one minute, the robot is able to find one out of the three victims in six of the ten trials before returning home. When the deadline is increased to five minutes, the robot takes advantage of this and performs more exploration and is able to find two out of the three victims in two trials and one victim in the remaining eight. When this deadline is further increased to ten minutes, the robot is able to find all three victims in two trials, two victims in four trials, and one victim in the remaining four trials.

These results demonstrate that generating possible worlds consistent with experience is feasible in practice, even as the robot's knowledge is being updated during exploration. It also shows that under realistic conditions, OH-wOW correctly trades off soft goals under temporal constraints, but without the ad hoc goal handling of Talamadupula et al. (2010) or the hours of preprocessing required by Joshi et al. (2012).

Discussion

OH-wOW requires a generative model of plausible worlds. We assume such expectations can be developed either manually or through experience. When the world contradicts the agent's expectations, this can be interpreted as surprise, which might naturally lead to increased learning. The fundamental vulnerability of sampling-based planners is when unlikely worlds play a large role in determining action

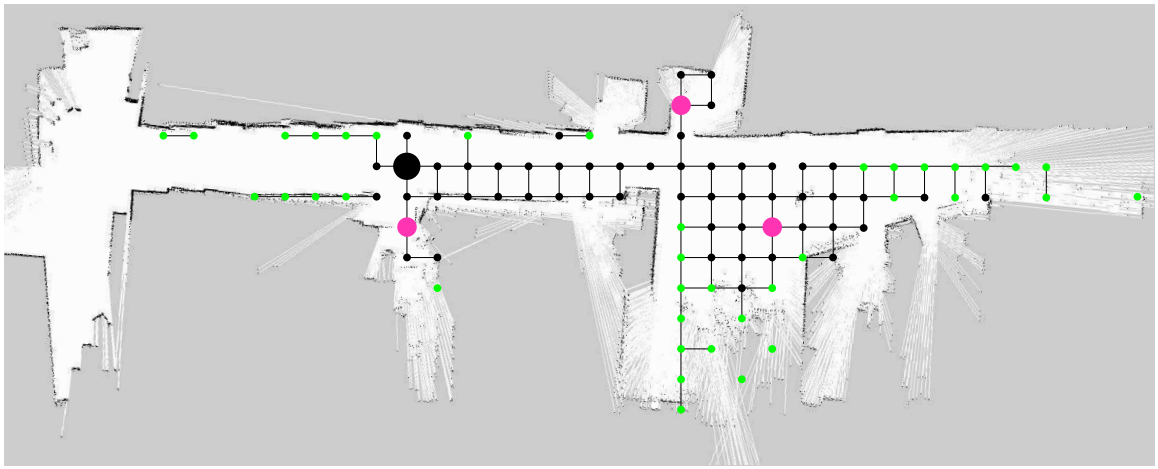


Figure 5: Example SLAM and topological map.

value; importance sampling may help here. For example, in the "Bombs in Toilets" domain (McDermott 1987; Smith and Weld 1998), OH-wOW may never sample a world in which a certain undunked package contains the bomb. The probability of this, however, is small ($7 \cdot 10^{-11}$ for 6 packages and 128 samples). In any case, optimal behavior is unattainable if one insists on fast response times in dynamic domains.

While faster than many POMDP algorithms, OH-wOW is much slower than a classical planner, as it must solve one classical planning problem for each sampled world. In our implementation, during each step, all planning problems were solved serially. These problems are entirely independent though and could trivially be solved in parallel to take advantage of multiple processor cores. OH-wOW is more general than standard off-line techniques as it can be used on-line, as shown in the experiments, and also off-line by simulating the domain to construct a branching plan. It is possible to improve the performance of OH-wOW by applying some of the enhancements of Yoon et al. (2010). One such technique is called *probabilistically helpful actions*. To find probabilistically helpful actions, the planner evaluates all samples from the current state of the agent instead of the one step lookahead states. Actions that lead to optimal plans starting from the current state are considered to be helpful while the others are not. The samples are solved as normal from the one step lookahead states, but the only actions that are considered are the ones that were deemed helpful. Another improvement presented by Yoon et al. (2010) is to save samples and plan prefixes that remain consistent with the outcome of a selected and then executed action. In domains with large amounts of determinism, this enhancement can greatly reduce the amount of planning required by saving work across deterministic transitions.

In this paper, we assume that the world remains static as we explore it and that non-sensing actions are deterministic. OH-wOW however, is very general and immediately applies to dynamic worlds, stochastic actions, and on-line goal arrival; this remains an exciting area for future work.

Conclusion

Open world planning is essential for many real-world agents. We have shown how optimization in hindsight yields a simple and general approach to open-world planning with temporal constraints, decision-theoretic reasoning, and soft goals. While the technique is approximate, it is easy to implement and our results suggest that it can be successful in practice.

Acknowledgments

This work was supported in part by NSF (grant 0812141) and the DARPA CSSG program (grant D11AP00242).

References

Babaian, T., and Schmolze, J. G. 2006. Efficient open world reasoning for planning. *Logical Methods in Computer Science* 2(3).

Bonet, B., and Geffner, H. 2001. GPT: a tool for planning with uncertainty and partial information. In *Proc. IJCAI-01 Workshop on Planning with Uncertainty and Partial Information*, 82–87.

Boutilier, C.; Dean, T. L.; and Hanks, S. 2011. Decision-theoretic planning: Structural assumptions and computational leverage. *CoRR* abs/1105.5460.

Chong, E.; Givan, R.; and Chang, H. 2000. A framework for simulation-based network control via hindsight optimization. In *IEEE Conference on Decision and Control*.

Cimatti, A.; Roveri, M.; and Bertoli, P. 2004. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence* 159(1–2):127–206.

Doshi, F. 2009. The infinite partially observable markov decision process. In *Neural Information Processing Systems*, volume 22, 477–485.

Etzioni, O., and Weld, D. S. 1994. A softbot-based interface to the internet. *Communications of the ACM* 37(7):72–76.

Joshi, S.; Schermerhorn, P. W.; Khardon, R.; and Scheutz, M. 2012. Abstract planning for reactive robots. In *Proceedings of IEEE ICRA*, 4379–4384.

Levesque, H. 1996. What is planning in the presence of sensing? In *Proceedings of AAAI*.

Levesque, H. J. 2005. Planning with loops. In *Proceedings of IJCAI*.

McDermott, D. 1987. A critique of pure reason. *Computational Intelligence* 3(1):151–160.

Mercier, L., and van Hentenryck, P. 2007. Performance analysis of online anticipatory algorithms for large multistage stochastic programs. In *Proceedings of IJCAI*.

Meuleau, N., and Smith, D. E. 2003. Optimal limited contingency planning. In *Proceedings of UAI*.

Phillips, M., and Likhachev, M. 2011. Planning in domains with cost function dependent actions. In *Proceedings of the fourth international symposium on combinatorial search (SoCS-11)*.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32:663–704.

Sanner, S. 2011. Relational dynamic influence diagram language (rddl): Language description. *NICTA, Australia*.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large POMDPs. In *In Advances in Neural Information Processing Systems 23*, 2164–2172.

Smith, D. E., and Weld, D. S. 1998. Conformant graphplan. In *Proceedings of AAAI*, 889–896.

Talamadupula, K.; Benton, J.; Schermerhorn, P.; Kambhampati, S.; and Scheutz, M. 2010. Integrating a closed world planner with an open world robot: A case study. In *Proceedings of AAAI*.

Wu, G.; Chong, E.; and Givan, R. 2002. Burst-level congestion control using hindsight optimization. *IEEE Transactions on Automatic Control*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of Conference on Artificial Intelligence (AAAI)*.

Yoon, S.; Ruml, W.; Benton, J.; and Do, M. B. 2010. Improving determinization in hindsight for on-line probabilistic planning. In *Proceedings of the Tenth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Younes, H. L., and Littman, M. L. 2004. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proceedings of the international planning competition*, 46.

Using Classical Planners for Tasks with Continuous Operators in Robotics

Siddharth Srivastava and **Lorenzo Riano** and **Stuart Russell** and **Pieter Abbeel**
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720

Abstract

The need for high-level task planning in robotics is well understood. However, interfacing discrete planning with continuous actions often requires extensive engineering of the solution. For instance, picking up an object may require removing many others that obstruct it. Identifying the exact obstructions requires geometric reasoning which is prohibitively expensive to precompute, with results that are difficult to represent efficiently at the level of a discrete planner. We propose a new approach that utilizes representation techniques from first-order logic and provides a method for synchronizing between continuous and discrete planning layers. We evaluate the approach and illustrate its robustness through a number of experiments using a state-of-the-art robotics simulator, accomplishing a variety of challenging tasks like picking objects from cluttered environments, where the planner needs to figure out which other objects need to be moved first to be able to reach the target object, and laying out a table for dinner, where the planner figures out effective tray-loading, navigation and unloading strategies.

1 Introduction

A robot capable of achieving high-level goals was one of the original motivations behind the automated planning problem (Fikes and Nilsson 1971). Since this early work, numerous advances have been made in automated planning. Discrete planners are able to automatically compute effective problem-specific heuristics for solving planning problems specified implicitly in terms of parameterized operators with preconditions and effects (e.g.,(Hoffmann and Nebel 2001; Helmert, Haslum, and Hoffmann 2007)). Continuous state space planners have also been developed for solving the lower level search problems in the configuration space of a robot to achieve desired motion trajectories (LaValle 2006). Techniques from both of these areas are required for a real-world robot to solve a high-level problem like preparing a table for dinner. However, using classical planners to solve planning problems encountered by a robot presents several fundamental challenges.

For instance, consider the task of picking up an object in blocks-world, a domain that is widely regarded as trivial for modern classical planners. If the table is cluttered, a continuous planner will fail to find a solution because other objects will have to be removed first (see Fig. 1). A classical planner could compute the sequence of operations required to clear

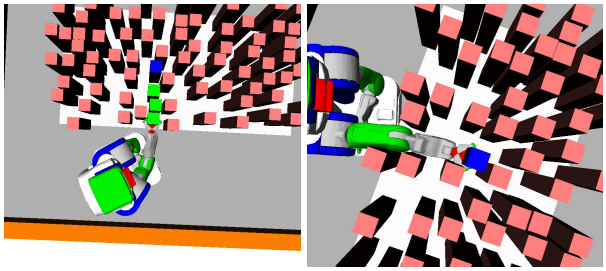


Figure 1: Example scenario with the target object in blue (L) and the desired solution after removing obstructing objects (R)

a path to the object, but only if it gets the set of obstructions. Superficially, this seems to be easily resolved by adding a high-level predicate, *obstructs*(c, b_1, b_2) where c ranges over the robot configurations and b_i over objects. However, any such formulation leads to two major problems. First, the robot configuration is usually a high dimensional real-valued vector (11-dimensional for the PR2 using only one gripper). Representing this problem for a discrete planner requires a discretization of the possible configurations which in itself is infeasible for the high dimensional spaces involved. Moreover, a pre-processing step will need to compute the graspability and obstruction facts specific to the given scenario, for every combination of discretized configurations and objects used as arguments to the corresponding predicates. This will lead to planner input files with thousands of configuration symbols at the least, making planning infeasible. Even if all of these representational steps are carried out, it is difficult to specify in a discrete problem definition how obstructions change on the application of a pick and place operation, due to the spatial reasoning involved. Finally, one of the problems with fixing a discretization a priori is that it may miss the points required for a solution.

These issues go beyond pick and place tasks and stem from the following fundamental problems: how to enable classical planners to efficiently utilize information provided by a continuous planner, and how to utilize them in situations where facts and operator effects over continuous variables are not available a priori. Our main contribution is a new approach addressing these problems. We show that this approach can solve number of challenging robot planning tasks.

Overview of Our Approach We assume that every high-

level action corresponds to a continuous implementation. The continuous implementation may use a variety of techniques for accomplishing the post-conditions of the high-level action. If the preconditions of the high-level action are satisfied, a trajectory is guaranteed to be found by the continuous implementation. However, as discussed in the introduction, the preconditions of high-level actions may involve spatial reasoning and are generally not computable a priori. Thus, the high-level planner may be wrong in selecting an action for execution at a certain step. If a solution trajectory cannot be found, the lower level returns the violated preconditions to the high-level planning layer, which incorporates the new facts and computes a new high-level solution plan. Throughout this paper, we will use the terms "discrete planning" interchangeably with high-level planning and "lower level" with continuous planning and execution.

While this overall approach is similar to re-planning (Talamadupula et al. 2010; Yoon, Fern, and Givan 2007), our focus is on the problem of continuous operator representations for high-level, discrete planning and communication between the continuous and discrete planning levels. The discrete problem is specified by replacing the domains of real-valued variables (required to express action preconditions and effects, even at the high-level) with finite sets of *Skolem symbols*. This translation is described in Sec. 2.1 The lower layer maintains an interpretation of these symbols (e.g., specific robot configurations). Failure messages from the lower level specify the failed preconditions of actions in the form of literals over such symbols. This interface as well as the details of planning methods used is described in Sec. 3. If the high-level planning layer finds the problem to be unsolvable, the lower level computes a new interpretation and the planning process resumes. This way the high-level planning process never has to deal with the continuous variables and the lower level does not have to deal with discrete task sequencing. The resulting approach allows easy modeling of various tasks. We present many experimental results in a simulated execution scenario developed using OpenRave (Diankov 2010) for the PR2 robot in Sec. 4. A summary of related work is presented in Sec. 5.

2 Problem Formulation

We assume that a propositional framework like typed PDDL (Fox and Long 2003) is used for specifying high-level planning domains and problems. We formalize the representation briefly as follows. A *planning domain* $\langle \mathcal{R}, \mathcal{A} \rangle$ consists of a set of relation symbols $\mathcal{R}$ with their arities and type signatures, and a set $\mathcal{A}$ of actions. Let the set of literals and atoms constructed using a set of relations $\mathcal{R}$ and a set of typed variables or constants V be *lit*$(\mathcal{R}, V)$ and *atoms*$(\mathcal{R}, V)$ respectively. An action is defined as $\langle$*Param, Pre, Eff*$\rangle$, where *Param* is a sequence of typed parameter variables; *Pre* is a first-order logic formula and *Eff* is a conjuction, both over *lit*$(\mathcal{R}, V)$ with only the variables in *Param* being free. With some syntactic limitations, such expressions can be represented in the PDDL input language used by most classical planners.

Given a finite set U of typed constants (informally constituting the "objects" in a problem), a state is an assignment of truth values to each atom in *atoms*$(\mathcal{R}, U)$; for compactness, we use the closed world assumption to represent states as sets of true atoms. We define ground actions as $a(\bar{c})$, where $\bar{c}$ denotes a mapping from the parameters of a to appropriately typed constants from U. The preconditions and effects of a ground action $a(\bar{c})$ are obtained using the variable assignments in $\bar{c}$ in the usual way.

We consider full observability and deterministic action effects, so that a ground action ga is applicable in a state s iff all of its preconditions are true in s.

Finally, a *planning problem* consists of a planning domain $\langle \mathcal{R}, A \rangle$, a set U of typed constants, an initial state s_0 and a set $g \subseteq$ *atoms*$(\mathcal{R}, U)$ denoting the goal condition. A solution to a planning problem is a sequence of ground actions $ga_1, \ldots ga_k$ such that ga_i is applicable in $ga_{i-1}(\cdots ga_1(s_o)\cdots)$ and $ga_k(\cdots ga_1(s_0)\cdots)$ satisfies g.

2.1 Domain Transformation

We introduce the central principle behind using Skolem symbols with an example. Consider an action like grasping in a pick and place domain. Its preconditions include real-valued vectors and preconditions require spatial reasoning. For clarity in this description, we consider a situation where objects are not stacked and assume that the target location where an object has to be placed is clear. The high-level grasp action can be specified as follows:

grasp(c, obj_1)
precon *graspable*$(c, obj_1) \wedge$ *robotAt*(c)
$\wedge \forall obj_2 \neg$ *obstructs*(c, obj_2, obj_1)
effect *in-gripper*(obj_1)

In this specification, the variable c represents robot configurations. In order to motivate our approach, consider the effect of this action in a framework like situation calculus (Levesque, Pirri, and Reiter 1998) but using a timestep rather than a situation to represent the fluents:
$\forall t, obj_1 \forall c(graspable(c, obj_1, t) \wedge robotAt(c, t) \wedge \forall obj_2 \neg obstructs(c, obj_2, obj_1, t) \rightarrow inGripper(obj_1, t+1))$

Note that this is not the complete *successor state axiom* for *in-gripper*, which will also have to include other actions that affect it and default conditions under which it doesn't change across timesteps. However, this implication is sufficient to illustrate the main representational device we will use. We first use the clearer, logically equivalent form:
$\forall t, obj_1(\exists c(graspable(c, obj_1, t) \wedge robotAt(c, t) \wedge \forall obj_2 \neg obstructs(c, obj_2, obj_1, t)) \rightarrow inGripper(obj_1, t)$

which asserts more clearly that *any* value of c that satisfies the preconditions allows us to achieve the postcondition. We can now use the standard technique of Skolemization to replace occurrences of c with a function of obj_1, t:

$$\forall t, obj_1((graspable(gp(obj_1, t), obj_1, t) \wedge robotAt(gp(obj_1), t) \wedge \forall obj_2 \neg obstructs(gp(obj_1), obj_2, obj_1, t)) \rightarrow inGripper(obj_1, t+1)$$

where the Skolem function $gp(x, t)$ is just a symbol of type *location* to the discrete planner. Intuitively, it represents a robot configuration corresponding to the grasping-pose of x. This representation will allow the discrete planner to treat the entire problem at a symbolic level, without the need for

creating a problem-specific discretization. A potential problem however, is that the Skolem functions will depend on t, or the current step in the plan. In practice however, at the discrete level, the time argument in a Skolem function $f(\bar{x}, t)$ can be ignored as long as it is possible to recompute (or reinterpret) f when an action's precondition is violated by its existing interpretation during the execution (as is the case in our implementation). We therefore drop the t argument from the Skolem functions in the rest of this paper.

The equivalence with an existential form as described above can be used for each action effect as long as the continuous variable being Skolemized is not free in the subformula on the right of the implication. For instance, we can add the effect that grasping an object removes all obstructions that it had created, regardless of robot configurations. Therefore, to represent the grasp operator for a discrete planning problem, rather than using a discretized space of configurations, we only need to add symbols of the form $f(\bar{o})$ in the planning problem specification, for each object argument tuple $\bar{o}$ consisting of the original objects, or constants in the problem. Since many classical planners don't support functions, they can be reified as objects of the form $f\_\bar{o}$ with an associated set of always true relations, e.g. $is\_f(f\_o_i, o_i)$. The discrete description of grasp thus becomes:

grasp(ℓ, obj_1)*:*
precon *is_gp*$(\ell, obj_1) \wedge$ *graspable*(ℓ, obj_1)
$\wedge$ *robotAt*(ℓ)
$\wedge \forall obj_2 \neg$ *obstructs*(ℓ, obj_2, obj_1)
effect *in-gripper*(obj_1)
$\wedge \forall \ell_2, obj_3 \neg$*obstructs*$(\ell_2, obj_1, obj_3)$

Here ℓ ranges over the finite set of constant symbols of the form gp_obj_i where obj_i are the original constant symbols in the problem. In this way, regardless of how many samples are used in the lower level process for interpreting these symbol, the discrete planner has a limited problem size to work with while computing the high-level plan.

Finally, consider the only remaining case for an action effect, when a continuous variable occurs freely in the subformula on the right, e.g. $\forall x, c(\varphi_{precon}(x, c) \rightarrow \varphi_{effect}(x, c))$. In this case we don't perform Skolemization. The symbol used for c in this case will be an action argument, and must range over the original objects in the domain or those already introduced via Skolemization.

Although this exhausts the set of actions commonly used in PDDL benchmark problems, the accurate description of an action may involve side-effects on symbols not used as its arguments. E.g., the putDown(obj_1, loc_1) action may deterministically introduce obstructions between a number of robot configurations and other objects. We don't encode such side effects in the high-level planning problem specification; these facts are discovered and reported by the lower level when they become relevant.

The putDown action has a similar specification, with the Skolem function $pdp(target_{\text{loc}})$ denoting the pose required for putting down an object o at location ℓ_1 when the robot is configuration ℓ_2.
putDown(o, ℓ_l, ℓ_2)*:*

precon *is_pdp*$(\ell_2, \ell_1) \wedge$ *poseForPlacing*(ℓ_2, o, ℓ_1)
InGripper(o) $\wedge$ *RobotAt*(ℓ_2)
effect $\neg$ *InGripper(o)* $\wedge$ *At*(o, ℓ_1)

The *moveto*(c_1, c_2) action changes the robot's configuration with the only precondition that it is at c_1. We assume that the environment is not partitioned and the robot can move between any two collision-free areas. These three actions constitute the high-level pick and place domain.

To summarize, we transform the given planning domain with actions using continuous arguments by replacing occurrences of continuous variables with symbols representing Skolem function application terms whenever the continuous variable occurs in the precondition but not in the effects. Every continuous variable or the symbol replacing one, of type τ gets the new type τ_{sym}. Planning problems over the modified domains are defined by adding finite set of constants for each such τ_{sym} in addition to the constants representing problems in the original domain. The added constants denote function application terms, e.g. *gp_obj17*, for each physical object and Skolem function application. This increases the size of the input only linearly in the number of original objects if the Skolem functions are unary. Note that the set of Skolem functions itself is fixed by the domain and does not vary across problem instances. The initial state of the problem is described using facts over the set of declared objects, e.g. "*is_gp(gp_obj17, obj17)*" denoting that the location name *gp_obj17* is a grasping pose for *obj17* and "*obstructs(gp_obj17, obj10, obj17)*", denoting that *obj10* obstructs *obj17* when the robot is at *gp_obj17*.

In this formulation, the size of the input problem specification is independent of the sampling-based discretization: we do not need to represent sampled points from the domains of continuous variables.

2.2 Discrete and Continuous States

A low-level state extends a discrete state by associating actual values (interpretations) with each Skolem symbol. Thus, every low-level state corresponds to a unique discrete state representation. Conversely, each discrete state represents a set of possible low-level states.

Since the low-level has complete information about states, it can compute (though not efficiently) the truth value of any atom in the vocabulary at any step, given an interpretation of every Skolem symbol used in the atom. E.g., the low-level can compute the truth value of *obstructs(***`cval`**`, obj10, obj17)` where **cval** is a numeric vector representing a robot pose. However, we wish to avoid the determining the value of such predicates as far as possible; the discretization approach discussed in the introduction is undesirable primarily due to its requirement that all such facts be precomputed.

If an action is not executable at a certain step in the low-level, then the low-level execution layer computes and reports a set of truth valuations of atoms showing that the action's preconditions do not hold.

Details of the low-level are presented in 3.3. In the next section, we present our overall algorithm and its properties.

3 Discrete and Continuous Planning

As noted above, the initial state for a planning problem is defined using the ground atoms which are true in the initial state. However, the truth values of ground atoms over Skolem symbols like *obstructs(gp_obj17, obj10, obj17)* are not known initially. Domain specific default truth values are assumed for such atoms. We discuss the relationship between the choice of defaults and completeness of the approach in Sec. 3.2.

3.1 Discrete Planning and the Interface

Our overall algorithm is shown in Alg. 1. In line 2, a classical planner is called with state s, which is initialized with the input problem state. If no solution is found, the stored values of Skolem symbols in the lower layer are cleared and all facts using Skolem symbols are reset to their default values using *clear_cache()*. The test in lines 3 and 5 ensure that this is done only once per iteration. If the problem is unsolvable even after clearing the cache, line 6 terminates the algorithm.

In line 7, the computed plan π is passed to the low-level, which uses sampling to estimate the values of Skolem symbols and RRT-based techniques to implement movements representing each action in the plan. Because the high initial state in line 2 used default values for facts over Skolem symbols, π may not be fully executable in the low level. If this is the case, the low-level reports the step/action of the plan that failed and the reason for that failure, represented by an assignment of atoms that violate the action precondition (a minimal violating assignment is sufficient). These components of the low-level are described in detail Sec. 3.3.

As noted earlier, the execution of an operator may void the interpretation of a symbol. E.g., moving an object invalidates the cached grasping pose. If the operators that invalidate a symbol's cached value are known, they can be provided to execute() within the cache voiding structure C. The low-level then reinterprets all affected Skolem symbols after executing an action.

In line 9, the UpdateState() subroutine uses the PDDL descriptions of actions to propagate s forward using π until *errStep* and adds *violatedPrecons* to the resulting state. The entire iteration then repeats with the resulting state unless a preset planning time/resource limit is reached.

Algorithm 1: Discrete-Continuous Interface

```
   Input: PDDL state s & domain D; cache-voiding ops C
1  repeat
2      π ← classicalPlanner(s, D)
3      if plan not found & not cleared_cache then
4          clear_cache(); continue
5      else if plan not found then
6          return "unsolvable"
       end
7      ⟨errStep, violatedPrecons⟩ ← execute(π, s, C)
       if errStep is null then
8          return "success"
       else
9          s = UpdateState(s, π, errStep, violatedPrecons)
       end
   until resource limit reached
```

3.2 Completeness

We now discuss the conditions under which our solution approach is complete. In doing so we show that under certain conditions, effective default assignments for atoms can be ontained easily. We use the notion of *probabilistically-complete* (LaValle 2006) to categorize sampling based algorithms that are guaranteed to find a solution with sufficiently many samples.

The following definition uses the concept of positive and negative occurrences of atoms in formulas. Intuitively an occurrence of an atom in a formula is negative (positive) if it is negated (non-negated) in the negated-normal-form of the formula. This notion of occurrence is sufficient for our purposes as we deal only with problems with finite universes and all quantifiers can be compiled into conjunctions or disjunctions. The following lemma follows from the definition of positive and negative occurrences:

Lemma 1. *Suppose an atom $p(c)$ occurs only positively (negatively) in a ground formula φ. If s is an assignment under which φ is true then it must also be true under an assignment s' that makes $p(c)$ true (false) and is the same as s for all other atoms.*

Definition 1. *A planning domain is* uniform *wrt a goal g and a set of predicates R if for every $r \in R$,*

1. *Occurrences of r in action preconditions and goal are either always positive, or always negative*
2. *Actions can only add atoms using r in the form (positive or negative) used in preconditions and the goal g*

Let P_S be the set of predicates whose atoms may use Skolem symbols as one of their arguments, and let $D = \langle \mathcal{R}, \mathcal{A} \rangle$ be a planning domain that is uniform wrt a goal g and P_S. In the following, consider atoms over a fixed set of constants U. Let s_{part} be an assignment of truth values to atoms over $R \setminus P_S$. Let $s_{default}$ be an assignment of truth values to atoms over P_S, assigning the atoms of each positively (negatively) occurring predicate the truth value true (false).

Proposition 1. *The state $s_{\text{part}} \cup s_{\text{default}}$ has a solution plan for a goal g iff there is some assignment s_0 of atoms over P_S such that $s_{\text{part}} \cup s_0$ has a solution plan for reaching g.*

Proof. Suppose there is an assignment s_0 under which $s_{part} \cup s_0$ has a solution π and which assigns an atom $p(\bar{c}_1)$ true, while all occurrences of p in action preconditions and g are negative. Consider the assignment s'_0 which assigns $p(\bar{c}_1)$ false, but is otherwise the same as s_0.

We show that π solves $s_{part} \cup s'_0$ as well. Suppose not, and that $a(\bar{c}_2)$ is the first action in π whose preconditions are not satisfied when π is applied on $s_{part} \cup s'_0$. In this failed execution, $a(\bar{c}_2)$ must have been applied on a state s'_k that differs only on $p(\bar{c}_1)$ from the corresponding state $s_k$in the execution of π on s_0. This is because all preceding action applications succeded and have the same deterministic effects in both executions. Let the ground formula representing the preconditions of this application of $a(\bar{c}_2)$ be φ. By

Lemma 1, φ must be satisfied by s'_k, and we get a contradiction: $a(\bar{c}_2)$ must be applicable on s'_k. The case for positive defaults is similar. □

Thus, in planning problems that are uniform wrt to the set of predicates which use Skolem symbols, it is easy to obtain a default truth assignment for atoms over these predicates, under which the problem is solvable if there is any assignment to those atoms under which it is solvable. The following result provides sufficient conditions under which our approach is complete. Let D be a planning domain and U a set of typed constants. A dead-end for $\langle D, U\rangle$ wrt g is a state over $atoms(\mathcal{R}, U)$ which has no path to g.

We say that two low-level states are physically similar if they differ only on the interpretation of Skolem symbols and atoms using Skolem symbols. We use the symbol $[s]$ to denote the discrete representation of a low-level state s, and $[s]_{default}$ to represent the version of $[s]$ obtained by using the default assignment for atoms using Skolem symbols, and the assignments in $[s]$ for all other atoms.

The following result presents sufficient conditions under which our approach solves any problem which is solvable under some evaluation of Skolem symbols.

Theorem 1. *Let D be a planning domain, U a set of typed constants, g a goal, and P_S the set of predicates in D that use Skolem symbols, such that $\langle D, U\rangle$ does not have dead-ends wrt g and D is uniform wrt g and P_S.*

If the low-level sampling routines for Skolem symbols are probabilistically complete, then for any low-level state s, if there exists a physically similar state s' such that $[s']$ is solvable by the high-level planner then the execution of Alg. 1 with inputs $[s]_{\text{default}}$ and D reaches the goal.

Proof. The proof follows from the following two points:
1. For any given interpretation of Skolem symbols represented by a low-level state s'_ℓ, Alg. 1 will eventually either discover the truth values of all atoms using Skolem symbols or identify the interpretation as unsolvable.

This is because in every non-final iteration of Alg. 1, line 7 must be executed and will return a non-empty assignment of atoms constituting a violated precondition. The entire set of atoms that violated preconditions are generated from is finite. Once they have been returned by the low level, they will never be generated again for this interpretation. This is because the atom is incorporated into the new discrete state accurately. Subsequent actions can only make it true (false) when it occurs positively (negatively) in preconditions. As a result, the discrete planning layer will never consider an atom that occurs positively to be true if its truth value was returned as false (true) by the low-level—unless an action set it to true in which case the low-level also considers it to be true.

If at any stage a discrete state s with default values for some atoms using Skolem symbols is unsolvable, then replacing them with any other truth values will not make the problem solvable. Thus, if at any stage the classical planner call in line 2 fails for a discrete state s with default values that are inconsistent with the low-level state for some atoms using Skolem symbols, the problem is indeed unsolvable under that interpretation.
2. As a result of probabilistic completeness, the low-level will discover all possible interpretations of Skolem symbols. □

The uniform property used in this result plays a role similar to *simple domains* defined by Bonet and Geffner (Bonet and Geffner 2011). Neither categorization is more general than the other. In contrast to simple domains, uniformity is less restrictive in not requiring invariants over initially unknown facts, while simple domains are less restrictive in not enforcing all occurrences of unknown predicates to be of the same polarity.

Extension of this result to more general situations is left for future work.

3.3 Low-Level Execution and Interpretation of Skolem Symbols

Arbitrary sampling techniques could be used for interpreting Skolem functions. For efficiency, we used methods that searched for interpretations satisfying required conditions which cannot be achieved by high-level planning alone. For instance, the symbol $gp(obj_1)$ needs to satisfy graspable$(gp(obj_1), obj_1)$ and $\wedge\forall obj_2\neg$ obstructs$(gp(obj_1), obj_2, obj_1)$. Since *obstructs* facts can be cleared by high-level actions but *graspable* is not achieved by any, we used a sampling process that achieves *graspable*.

We drew upon the wealth of approaches developed in the robotics literature for solving navigation, motion planning and obstacle avoidance problems. In this work we assumed that the continuous planner has complete knowledge of the state of the world. This can obtained using, for example, 3D sensors (Hsiao et al. 2010) and map-making techniques (Marder-Eppstein et al. 2010). Both arm motion planning and base movements between poses (i.e. the *moveto* action) are performed using Rapidly Exploring Random Trees (RRTs) (LaValle and Kuffner Jr 2000).

Interpreting Grasping Pose Symbols A robot pose from which an object can be grasped (i.e, a grasping pose or one that satisfies *graspable*) must satisfy two conditions: the robot base should be at a collision free configuration from where the object is reachable, and there must exist a collision-free trajectory from this configuration to one that brings the manipulator into the grasping configuration. Grasping an object requires finding a manipulator configuration that allows the robot to robustly hold it. We precalculated and stored the grasping configurations for each object in OpenRave (these configurations can be computed on-the-fly if necessary).

Our approach for finding a grasping pose is inspired by Probabilistic Roadmaps (Kavraki et al. 1996), and summarized in Algorithm 2. The first step in lines 2-3 is to generate a robot's base configuration that is within a reachability area around the object. The presence of obstacles, the nonlinearity of the robot actuators and the high-dimensionality of the search space render this problem very challenging (Stulp, Fedrizzi, and Beetz 2009; Berenson, Kuffner, and Choset 2008). While pre-calculating poses using inverse

Algorithm 2: Find a Grasping Pose

```
   Input: Object o
1  ikfound ← False
2  repeat
3      p ← sampleAround(o)
4      t ← sampleTorsoHeight()
5      gps ← generateGrasps(p, t, o)
6      for gp_i ∈ gps do
7          if IKsolution(p, t, gp_i) then
8              ikfound ← True; solnPose ← p
9              if obstaclesFreeArmPath(p, t, gp_i) then
10                 return (p, t, gp_i)
               end
           end
       end
   until max number of attempts reached
   /* No soln found, return obstructions   */
11 if ikfound then
12     objs ← FindObstructingObjects(o, solnPose)
13     return CreateObstructionAtoms(objs)
   end
```

kinematic (IK) caching could have been used, we found that sampling from a sphere centered around the object yields the same qualitative results. During sampling, poses outside the environment's boundaries or in collision with the environment are discarded.

Collisions can be efficiently calculated by approximating the robot's shape to be either a cube or a cylinder. At line 3 we generate candidate values for the PR2's torso height and retrieve the manipulator grasping configurations at line 4.

The conditions for satisfying a grasping pose are checked in lines 6 & 7. Previously found grasping solutions are cached and retained until an object is moved or when the discrete planner calls *clear_cache()*. If an IK solution exists but no obstacle-free solution is found within a maximum number of iterations then the simulator is used to calculate all the objects in collision with the robot's arm for a computed IK solution. Logical atoms are created using these objects and returned in line 13. This last step requires a few additional inputs like the predicate name and ordering of arguments, but are omitted for readability.

Interpreting Symbols for putDown By substituting the grasping manipulator poses at line 5 with a set of manipulator poses that lay above a tabletop, Algorithm 2 can be generalized to find base poses to put down objects on free areas of a surface. Our current implementation does not take into account collisions between objects held by the robot and the environment. Therefore only obstacle free manipulator configurations are considered when looking for an empty area above a tabletop. Although our simulator provides a realistic implementation of robots' kinematics and collision detection, to speed up the computations we did not simulate physics. Therefore once the robot opens the gripper holding an object we forced the object to fall down and rest on the target surface in a horizontal position.

4 Empirical Evaluation

We tested our proposed approach in two different scenarios, simulated using OpenRave[1]. The first scenario involves grasping objects from a cluttered tabletop (see Figure 2). This problem requires the robot to identify objects that prevent a collision free trajectory to the target object, and moving them to a separate side table. The second scenario sees the robot setting up a dining table with three pairs of kitchen objects (see Figure 3). The robot can make use of a tray to help carrying more than one object, but there are constraints on how the objects can be stacked on the tray.

Our approach can be used with any classical planner. For the first experiment we used a well-established satisficing planner, FastForward (FF) (Hoffmann and Nebel 2001). The second experiment required a cost-sensitive planner. We used FastDownward (FD) (Helmert 2006) and always chose the second plan that it produced in its anytime mode if it was produced within 350s, and the first plan otherwise. When reporting the execution time we did not calculate the time required to actually perform an action, e.g. move the robot's base or the arms. In other words the robot instantaneously moves all of its joints.

All the experiments discussed below were carried out on an AMD Opteron dual-core 2GHz machine with 4GB RAM.

Pick and Place Scenario An instance of the pick-and-place problem is illustrated in Figure 2. To guarantee that objects in the middle of the table are not easily reachable we approximated their shape with over-sized bounding boxes. We modeled this problem as a realistic simulation of a situation the robot might face, where the robot can make free use of the available empty space between objects to reach its goal. Thus a highly cluttered table with 80 objects may have obstructions from nearly every possible graspable pose, but the number of obstructions from each may be small. If grasping fails, the continuous planner returns a list of obstructions.

We performed 30 tests over 3 randomly generated environments. Each environment has the same configuration of static objects (tables and walls) but different configuration of movable objects. Each random environment has between 50 and 80 objects randomly placed on the tabletop. In these experiments we used 200 samples of base configurations generated while searching for grasping poses (outer loop of Alg. 2). Table 1 summarizes the results. Each row in the table is an average of 10 runs. For each run we randomly selected a target object with at least one obstruction to be the grasping goal. As a baseline for comparison we used the time a classical planner would take to solve a version of the same problem with all obstructions precomputed and a discretized set of sampled configurations. For each object in the environment, we sampled 200 poses and kept only those from where objects are graspable. This resulted in 80 to 130 configurations for our problems. The last column of Table 1 shows the time required to pre-calculate all these obstructions, which is significantly higher than the time required for the computation and execution of the whole plan in our

[1]Source code is available at `https://github.com/lorenzoriano/OpenRaving`

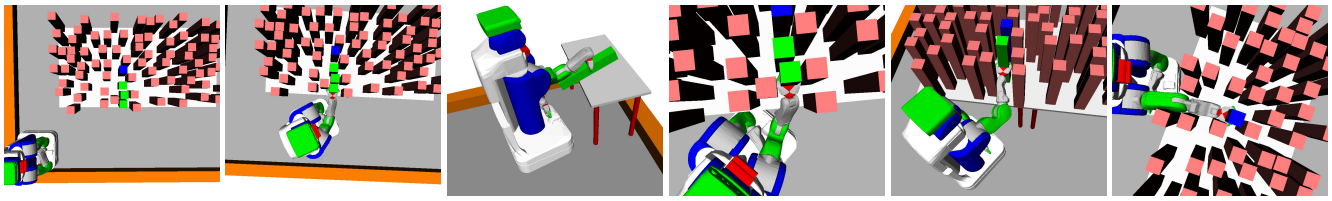


Figure 2: An example execution of a pick and place experiment. From left to right: 1) The initial disposition of the objects on the table. The blue object is the target, while the green objects are the ones identified as obstructing the target. 2) The robot selects the first object to remove. 3) The robot puts the object on a free area of an additional table. 4) The robot removes the second obstruction. 5) The robot removes the third obstruction. 6) The target object is finally reachable.

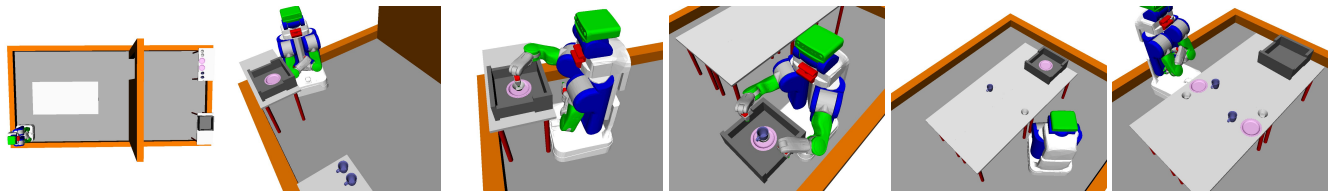


Figure 3: An example scenario of a dining set-up experiment. From left to right: 1) The initial configuration of the environment. 2) The robot places a plate on the tray. 3) Another plate and a bowl are placed on the tray. 4) The robot carries the tray with four objects to the dining table. 5) The first objects are removed from the top of the stack on the tray and placed on the appropriate locations on the table. 6) The final goal configuration is achieved.

approach.

Dining Table Set-Up Scenario The second experiment illustrates the general applicability of our approach to heterogeneous problems. The robot's task is to move six objects from an initial location to a dining table, as shown in Figure 3. The robot can only move objects between the "kitchen" and "dining" areas by placing them on a tray, moving the tray to the target table and then placing the objects from the tray to their desired configuration. Multiple round-trips are allowed. For this experiment we considered two sets of plates, bowls and mugs.

Placing objects on the tray is an action that can fail, thus generating violated preconditions. This simulates dynamical instability between pairs of objects which is difficult to pre-compute without execution. The lower level enforced this by not allowing a plate to be stacked over other objects, and not allowing anything to be stacked over a mug. These conditions are initially unknown to the high-level planner. We also made the cost of moving across rooms 100 times the cost of other actions.

As picking up the tray requires two-arm manipulation we pre-calculated both manipulators poses for grasping it. Grasping objects, placing them on the tray or on the table uses the same primitives described in Sec. 3.3.

Solving the dining table scenario required 230s, including execution. Given the violations induced by stacking objects on the tray, the classical planner had been called 3 times. The solution plans typically attempt to stack a few objects on the tray and transport them in batches.

#Objs	#Obstrns	Time(s)	#Replan	Precomp(s)
80	2.6	602	2.3	4245
65	2.0	228	2.6	3667
50	1.8	139	2.1	1777

Table 1: Results for the cluttered table scenario. For comparison, times for precomputing obstruction facts are shown in the last column. The central 3 columns are averages of 10 independent runs, showing the number of obstructions, total time for planning and execution and number of calls made to the classical planner.

5 Related Work

This work is related to, and builds upon a vast history of research in planning and robotics. The field of discrete planning has made several advances in scope and scalability, mainly through the development of efficient methods for computing heuristic functions automatically from a domain definition (Bonet and Geffner 2001; Hoffmann and Nebel 2001; Helmert, Haslum, and Hoffmann 2007). These advances are among the main motivations in developing our approach. An alternate representation of our high-level problem would be a partially observable problem with non-deterministic actions (Bonet and Geffner 2000). However, this is a much harder problem in general (Rintanen 2004) and an offline contingent planning process would be unfeasible in our setting.

Various researchers have investigated the problem of combining discrete and continuous planning. Alami et al. (1998) describe a system architecture that uses a translation module for converting high-level actions into continuous trajectories. Volpe et al. (2001) also use a similar module, referred to as "time-line" interaction. However, these approaches do not discuss specific methods for compiling away continuous parameters of operators in order to facilitate discrete planning. Plaku et al. (2010) propose a sampling based approach for guiding the low-level search process with information from a high-level planner. They also

allow non-deterministic actions. However, communication is only one-way: only the first action in the high-level plan is used, and only to bias low-level search process. In contrast, the low-level search process in our approach provides information back to the high-level planner which uses it to generate more relevant plans. Hauser (2010) observes almost the same problems in tasks like pick-and-place. His approach uses a STRIPS like language for specifying actions, but does not use a classical planner for the high-level search. Another approach that combines discrete and continuous planning is described in (Choi and Amir 2009). These authors propose including kinematics constraints among the information provided to the discrete planner. Their approach automatically generates logical actions from the output of a geometric motion planning algorithm. However this approach also has only one-way communication between the discrete and continuous planner. Moreover their approach is difficult to adapt to non-motion planning problems. Wolfe et al. (2010) use angelic hierarchical planning to define a hierarchy of high-level actions over primitive actions. However the high-level actions in their approach have continuous action arguments and need to be sampled. In contrast, we define a whole discrete problem, the size of which is independent of the discretization. They also do not model problems with preconditions like obstructions which require geometric reasoning. Our approach also includes an angelic interpretation: the skolemized functions in our discrete, high-level operators are assumed to have *a* value that satisfies the preconditions, and is selected by the agent. Kaelbling et al. (2011) combine discrete and continuous planning by relying upon a very specific regression-based planner with task-specific components. Levihn et al. (2012) present an approach for the problem of navigation among movable obstacles, but unlike our approach, they assume that the effects of discrete, high level actions on obstructions and geometric regions are known a-priori. They also do not address the problem of utilizing general purpose classical planners in conjunction with motion planning.

Approaches for planning modulo theories (Gregory et al. 2012) (PMT) and planning with semantic attachments (Hertle et al. 2012) address similar high level problems of planning in hybrid domains. However, these approaches use an extended planning language and require appropriately extended planners. Both of these approaches require deterministic functions for computing "external" predicates during search, and do not address the representational problem of using a classical planner to plan with actions and fluents that use continuous arguments. In contrast, our approach only requires information from the lower level when a high level plan fails and works with arbitrary classical planners. Our approach can also be used in situations where a model of the effects of low level actions is not available, and the action execution failures cannot be predicted accurately (as in the stacking problem) and are only known when they occur. As a consequence of this generality, the completeness of our approach depends on the problem domain (the presented analysis provides a sufficient but not necessary condition).

Grasping objects in a cluttered environment is still an open problem in robotics. In (Dogar and Srinivasa 2011) an approach is proposed that replaces pick-and-place with pushing objects away from the desired trajectory. This represents an interesting solution to the reachability problem that we aim at exploring as an application of our framework.

6 Conclusions

In this paper we presented an approach for planning in robotics that relies upon the synergy between a classical and a continuous planner. One of the advantages of our proposed approach is that not all the facts need to be pre-computed before planning, but only the ones that are required by the execution. The classical planner is also able to efficiently represent these facts without having to deal with representations that grow with the sampling process used in the lower layer. As our experiments show, this is significantly more efficient than solving precomputed discrete problems with sampling-based discretizations. Our solution also addressed a broad underlying problem in robotics: that the truth value of many predicates can only be found when the robot tries to perform an action. Both of our test scenarios featured this problem.

Although we simulated dynamical instability by using fixed rules, a different implementation could utilize a physics simulator for predicting if an object is unstable when placed over a stack. It could thus prune away actions that a planner alone would not know were wrong. While we conducted our main experiments in a pick-and-place scenario, the second experiment shows that our proposed approach is not limited to this particular application. Introducing costs, handling constraints and plans with heterogeneous actions are supported by our algorithm through its modularity.

Future work will include a more thorough analysis of optimality and termination guarantees on replanning. We also plan to study situations where action costs, in addition to facts, are made available during execution. Although our simulation is extremely faithful in terms of the capabilities and limitations of the robot and in terms of construction of the problem scenarios, the most natural next step in this work will be to test the plans with a real robot.

Acknowledgments

We thank Malte Helmert and Joerg Hoffmann for providing versions of their planners with verbose outputs that were very helpful in our implementation. We also thank John Schulman for helpful comments. This work was supported by the NSF under Grant IIS-0904672.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal Of Robotics Research* 17:315–337.

Berenson, D.; Kuffner, J.; and Choset, H. 2008. An optimization approach to planning for mobile manipulation. *2008 IEEE International Conference on Robotics and Automation* 1187–1192.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. of*

the 6th International Conference on Artificial Intelligence Planning and Scheduling, 52–61.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artif. Intell.* 129(1-2):5–33.

Bonet, B., and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI*, 1936–1941.

Choi, J., and Amir, E. 2009. Combining planning and motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 238–244. IEEE.

Diankov, R. 2010. *Automated Construction of Robotic Manipulation Programs*. Ph.D. Dissertation, Carnegie Mellon University, Robotics Institute.

Dogar, M., and Srinivasa, S. 2011. A framework for push-grasping in clutter. *Robotics: Science and Systems VII*.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. Technical report, AI Center, SRI International. SRI Project 8259.

Fox, M., and Long, D. 2003. PDDL2.1: an extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res.* 20(1):61–124.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning modulo theories: Extending the planning paradigm. In *ICAPS*.

Hauser, K. 2010. Task planning with continuous actions and nondeterministic motion planning queries. In *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. of the 17th International Conference on Automated Planning and Scheduling*, 176–183.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hertle, A.; Dornhege, C.; Keller, T.; and Nebel, B. 2012. Planning with semantic attachments: An object-oriented view. In *ECAI*, 402–407.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hsiao, K.; Chitta, S.; Ciocarlie, M.; and Jones, E. 2010. Contact-reactive grasping of objects with partial shape information. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 1228–1235. IEEE.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Proc. IEEE International Conference on Robotics and Automation*, 1470–1477.

Kavraki, L.; Svestka, P.; Latombe, J.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on* 12(4):566–580.

LaValle, S., and Kuffner Jr, J. 2000. Rapidly-exploring random trees: Progress and prospects.

LaValle, S. M. 2006. *Planning algorithms*. Cambridge University Press.

Levesque, H. J.; Pirri, F.; and Reiter, R. 1998. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence* 2:159–178.

Levihn, M.; Scholz, J.; and Stilman, M. 2012. Hierarchical decision theoretic planning for navigation among movable obstacles. In *Workshop on the Algorithmic Foundations of Robotics*, 19–35.

Marder-Eppstein, E.; Berger, E.; Foote, T.; Gerkey, B. P.; and Konolige, K. 2010. The Office Marathon: Robust Navigation in an Indoor Office Environment. In *International Conference on Robotics and Automation*.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *IEEE International Conference on Robotics and Automation*, 5002–5008.

Rintanen, J. 2004. Complexity of planning with partial observability. In *Proc. of the 14th International Conference on Automated Planning and Scheduling*, 345–354.

Stulp, F.; Fedrizzi, A.; and Beetz, M. 2009. Action-related place-based mobile manipulation. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* 3115–3120.

Talamadupula, K.; Benton, J.; Schermerhorn, P. W.; Kambhampati, S.; and Scheutz, M. 2010. Integrating a closed world planner with an open world robot: A case study. In *Proc. of AAAI*.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; and Das, H. 2001. The CLARAty architecture for robotic autonomy. In *Proc. of IEEE Aerospace Conference*, 121–132.

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined task and motion planning for mobile manipulation. In *Proc. of the International Conference on Automated Planning and Scheduling*, 254–258.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *Proc. of the International Conference on Automated Planning and Scheduling*, 352–.

Closed Loop Configuration Planning with Time and Resources

Maurizio Di Rocco and **Federico Pecora** and **Alessandro Saffiotti**
Center for Applied Autonomous Sensor Systems, Örebro University, SE-70182 Sweden
{modo, fpa, asaffio}@aass.oru.se

Abstract

We propose an approach to configuration planning in closed loop with sensing and actuation. Configuration plans are fine-grained action plans for robotic systems which specify the causal, temporal, resource and information dependencies between individual sensing, computation, and actuation components. The key feature which enables closed loop performance is that configuration plans are represented as constraint networks, which are shared between the planner and the executor and are continuously updated during execution. We report experiments run in a simulated multi-robot scenario which show that our approach yields robust closed loop performance in the face of four types of disturbances during execution: delays, resource collapse, exogenous events, and changing goals.

Introduction

Since its beginnings as an exploration of logical reasoning for robots (Fikes and Nilsson, 1972), the field of AI planning has progressed enormously. Yet, if you look inside a typical autonomous robot today you will see little evidence of this progress, and you may suspect that planning in AI has focused on issues which are not the main concerns of robot builders. Indeed, if you ever tried to program a robot to accomplish tasks in unstructured, everyday environments, you would expect an AI planner to provide the ability to act as a *knowledge-based controller* (Dean and Wellman, 1991) for the robot: given a goal, the planner continuously synthesizes actions which bring about the achievement of the goal as the state of the world changes during execution. This entails (*requirement 1*) that the planner should possess knowledge and reason about the physical aspects of the domain, like time, space, and resources. In the real world, plans are subject to execution-time perturbations, such as actions taking longer than expected, resources not being available, or assumed causal requirements not being met. Therefore, the planner should (*requirement 2*) generate plans that enable a sufficient degree of flexibility to accommodate these contingencies during execution whenever possible. Furthermore, robots today are not monolithic entities, rather a collection of sensors, drivers, actuators, and information processing components. This entails that the planner should (*requirement 3*) decide how to compose these enabling modules while taking into account their physical and logical dependencies.

Today's planners exhibit some of the above features. For instance, a considerable amount of work has been done to integrate metric time into planning (Knight et al., 2001; Do and Kambhampati, 2003; Gerevini, Saetti, and Serina, 2006; Doherty, Kvarnström, and Heintz, 2009; Barreiro et al., 2012; Eyerich, Mattmüller, and Röger, 2009). Including time has allowed some planning systems to perform *continuous planning*, i.e., continuously synthesize plans as new goals and contingencies become known. Execution monitoring techniques have been developed which leverage the explicit temporal representation of these plans (Finzi, Ingrand, and Muscettola, 2004; McGann et al., 2008b,a), thus effectively providing a few examples of planning for real robots. Although these constitute important steps towards obtaining planners that are appropriate for robots, they address only partially (i.e., in the temporal dimension) requirements 1 and 2. Some work has addressed the issue of including further dimensions into the planning problem, e.g., resources. In addition to addressing more fully requirement 1, some of these approaches (Köckemann, Pecora, and Karlsson, 2012; Fratini, Pecora, and Cesta, 2008; Ghallab and Laruelle, 1994) would also be well suited for use in closed loop with actuation and perception, as they maintain a certain level of least-commitment with respect to the timing of plan execution. Nevertheless, they are not proposed nor evaluated as closed-loop planning systems. Lemai and Ingrand (2004) propose an extension of the IxTeT planner (Ghallab and Laruelle, 1994) for closed loop execution monitoring with resources — however, the technique is exemplified on single robot navigation tasks.

Much of the work in AI planning has focused on action planning (Ghallab, Nau, and Traverso, 2004) — however, when planning for robots, actions are not the only aspect upon which a planner should reason. We are interested in *configuration plans*, namely fine-grained plans for robotic ecologies (Saffiotti et al., 2008) which specify the causal, temporal, resource and information dependencies between individual sensing, computation, and actuation components. Configuration planners have been proposed before: the AsymTre architecture (Parker and Tang, 2006) considers a set of robots equipped with software modules, called schemas, able to sense and modify the envi-

ronment. Schemas are interconnected through information channels which are decided by a planning process. Further refinements of this idea are presented by Zhang and Parker (2011); Parker and Zhang (2012): the former introduces metrics to estimate the information quality gathered by the chosen configuration, while the latter provides more sophisticated techniques to gather the needed information. This approach focuses on information exchange ignoring resource and temporal requirements. Furthermore, relying on an auction procedure to incrementally build the plan, this architecture doesn't manage explicitly the presence of shared actions in the planning process. In fact, as the authors point out, the backtracking procedure involving these schemas could lead to problems that are currently not addressed. Zhang and Parker (2013) address a limited form of resource reasoning, however the method does not provide closed-loop execution monitoring; interestingly, dependencies are used to enforce a basic form of temporal constraints, namely precedences. An explicit representation of the world is instead considered by Lundh, Karlsson, and Saffiotti (2008): differently from AsymTre, this system leverages a propositional logic description of the world based on standard planning techniques. Reasoning is performed by coupling an action planner together with a configuration planner: the former provides a sequence of actions that have to be further refined by the latter; in turn, the configuration planner chooses software modules to activate and decides the related communication linkage. This system allows a detailed representation of the evolution of the world — however, it is decoupled from execution, and therefore suffers from many of the aforementioned problems. An improved version of this work (Lundh, 2009) takes into account multiple goals through a merging sequence. Similarly to AsymTre, the approach lacks the ability to perform on-line plan monitoring and execution.

In this paper, we propose a configuration planner which fulfills the above requirements by combining three enabling factors: (1) representing a (configuration) plan as a constraint network; (2) defining the configuration planning process as a search in the space of such networks; and (3) sharing the constraint network between the planner and the executor.

Representation

Our approach is grounded on the notion of *state variable*, which models elements of the domain whose state in time is represented by a symbol. State variables, whose domains are discrete sets, represent parts of the real world that are relevant for the configuration planner's decision processes. These include the actuation and sensing capabilities of the robotic system, and the various aspects of the environment that are meaningful. E.g., a state variable can represent the capabilities of a robot, whose meaningful states might be "navigating" or "grasping", or the interesting states of the environment, e.g., a light which can be "on" or "off". Let $\mathcal{S}$ be the set of state variables in a given application scenario.

Some devices require resources when they are in given states. A *reusable resource* has a limited capacity which is fully available when not required by a device. An example is power: a maximum wattage is available, and devices can simultaneously require power so long as the sum of requirements is less than the maximum power. We denote with $\mathcal{R}$ the set of all resource identifiers, and with $\mathrm{Cap}(R) \in \mathbb{N}$ the capacity of $R \in \mathcal{R}$.

Devices in our domain may serve the purpose of providing or requiring *information contents*. For instance, a software component may require range data from a laser range finder, and provide localization information. We denote IC the set of all information contents.

Representing Configuration Plans and Goals

We employ *activities* to represent predicates on the possible evolution of state variables:

Definition 1 *An* activity a *is a tuple* $(\mathrm{x}, \mathbf{v}, I, u, \mathrm{In}, \mathrm{Out})$, *where*

- $\mathrm{x} \in \mathcal{S} = \{\mathcal{S}_{contr} \cup \mathcal{S}_{obs} \cup \mathcal{S}_{int}\}$ *is a* state variable*, where*
  - $\mathcal{S}_{contr}$ *is the set of controllable variables*
  - $\mathcal{S}_{obs}$ *is the set of observable variables*
  - $\mathcal{S}_{int}$ *is the set of internal variables*
- $\mathbf{v}$ *is a* possible state *of the state variable* x*;*
- $I = [I_s, I_e]$ *is a* flexible temporal interval *within which the activity can occur, where* $I_s = [l_s, u_s], I_e = [l_e, u_e], l_{s/e}, u_{s/e} \in \mathbb{N}$ *represent, respectively, an interval of admissibility of its start and end times;*
- $u : \mathcal{R} \to \mathbb{N}$ *specifies the* resources used *by the activity;*
- $\mathrm{In} \subseteq \mathrm{IC}$ *is a set of* required information contents*;*
- $\mathrm{Out} \subseteq \mathrm{IC}$ *is a set of* provided information contents*.*

The notation $(\cdot)^{(a)}$ indicates an element of the five-tuple pertaining to activity a. The pair $(\mathrm{x}^{(a)}, \mathbf{v}^{(a)})$ asserts a particular state $\mathbf{v}$ of the state variable x; $I^{(a)}$ represents possible temporal intervals of occurrence of the state $\mathbf{v}^{(a)}$ of state variable $\mathrm{x}^{(a)}$. Note that a pair of activities (a, b) is *possibly concurrent* if $I^{(a)} \cap I^{(b)} \neq \emptyset$. A pair (a, b) of possibly concurrent activities thus indicates that $\mathrm{x}^{(a)}$ and $\mathrm{x}^{(b)}$ can be, respectively, in states $\mathbf{v}^{(a)}$ and $\mathbf{v}^{(b)}$ at the same time.

Unspecified parameters of an activity are indicated with $(\cdot)$ — e.g., $(\mathrm{x}, \cdot, I, u, \mathrm{In}, \mathrm{Out})$ indicates a predicate asserting that state variable x can be in any state during interval I.

Activities can be bound by *temporal constraints*, which restrict the occurrence in time of the predicates. Temporal constraints can be of two types. *Binary* temporal constraints in the form $a\,\mathrm{C}\,b$ prescribe the relative placement in time of activities a, b — these constraints are relations in Allen's Interval Algebra (Allen, 1984), and restrict the possible bounds for the activities' flexible temporal intervals $I^{(a)}$ and $I^{(b)}$. *Unary* temporal constraints in the form $\mathrm{C}\,a$ prescribe bounds on the start or end time of an activity a — these constraints are commonly referred to as *release time constraints* and *deadlines*.

Allen's interval relations are the thirteen possible temporal relations between intervals, namely "precedes" (p), "meets" (m), "overlaps" (o), "during" (d), "starts" (s), "finishes" (f), their inverses (e.g., p^{-1}), and "equals" ($\equiv$).

When state variables are used to represent a system, the overall temporal evolution of such system is described by a *constraint network*:

Definition 2 *A* constraint network *is a pair* $(\mathcal{A}, \mathcal{C})$*, where* $\mathcal{A}$ *is a set of* activities *and* $\mathcal{C}$ *is a set of* constraints *among activities in* $\mathcal{A}$*.*

A constraint network can be used to represent a *configuration plan*. Configuration plans are said to be *feasible* if they are consistent with respect to the resource, state, and temporal requirements. Specifically,

Definition 3 *A configuration plan* $(\mathcal{A}, \mathcal{C})$ *is* feasible *iff:*

- *the constraint network is* temporally consistent*, i.e., there exists at least one allocation of fixed bounds to intervals such that all temporal constraints are satisfied;*
- *activities do not over-consume resources, i.e.,* $\sum_{a \in A} u^{(a)}(R) \leq \mathrm{Cap}(R), \forall R \in \mathcal{R}$*, where* $A \subseteq \mathcal{A}$ *is a set of possibly concurrent activities;*
- *activities do not prescribe that state variables assume different states in overlapping temporal intervals, i.e.,* $\mathbf{v}^{(a)} \neq \mathbf{v}^{(b)}, \forall (a, b) \in A \times A : \mathrm{x}^{(\mathrm{a})} = \mathrm{x}^{(\mathrm{b})}$*, where* $A \subseteq \mathcal{A}$ *is a set of possibly concurrent activities.*

A *goal* for a configuration planning problem is also represented as a constraint network, therefore expressing temporal, resource, state and information requirements. Typically, a goal (A_g, C_g) is an under-specified configuration plan. Initial conditions are feasible sub-networks of a goal.

Domain Representation

Given a goal (A_g, C_g) and a configuration plan $(\mathcal{A}, \mathcal{C})$ which contains the goal, the feasibility of the configuration plan is not a sufficient condition for achieving the goal. This is because feasibility does not enforce information and causal requirements. The way these requirements are to be enforced depends on a *domain*:

Definition 4 *A* configuration planning problem *is a pair* $((A_g, C_g), \mathcal{D})$*, where* (A_g, C_g) *is a goal constraint network, and* $\mathcal{D}$ *is a* domain*. The domain is a collection of* operators*, which describe the information and causal dependencies between activities.*

Definition 5 *An* operator *is a pair* $(a, (A, C))$ *where*

- $a = (\mathrm{x}, \mathbf{v}, \cdot, \cdot, \cdot, \mathrm{Out})$ *is the* head *of the operator;*
- $A = A_p \cup A_e \cup \{a\}$ *is a set of activities, where*
  - A_p *is a set of* preconditions*, i.e., requirements, in terms of state variable values, information input, and resource usage, needed to achieve the state* $\mathbf{v}^{(a)}$ *of state variable* $\mathrm{x}^{(a)}$ *and to produce* $\mathrm{Out}^{(a)}$*;*
  - A_e *is a set of* effects*, i.e., state variable values entailed by the achievement of state* $\mathbf{v}^{(a)}$ *of state variable* $\mathrm{x}^{(a)}$*;*
- C *is a set of temporal constraints among activities in* A*.*

Computing a configuration plan consists in selecting and instantiating operators form the domain into the goal constraint network. Unlike in classical planning, the relevance of an operator (denoted γ^{-1} in Ghallab, Nau, and Traverso, 2004) is not determined by unifying effects with sub-goals, rather by the unification of an operator's head with a sub-goal. The head of an operator is a non-ground activity which describes the value of a state variable and the information provided as a result of applying the operator. Preconditions and effects are used during execution, the former to determine the control action(s) given to the system (input regulation problem), the latter as a part of state estimation (see next Section).

An operator can specify the information requirements needed for achieving a particular functionality. For instance, the $\mathrm{MoveFromTo}$ operator, which does not provide any information content, requires the current position of the robot:

$$
\begin{aligned}
a &= (\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, \cdot, \cdot, \cdot, \emptyset) \\
A_p &= \{a_1, a_2\}, A_e = \{a_3\}, \text{where} \\
a_1 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{position}\}) \\
a_2 &= (\mathrm{RobotLocation}, \mathbf{kitchen}, \cdot, \cdot, \cdot, \cdot) \\
a_3 &= (\mathrm{RobotLocation}, \mathbf{livingroom}, \cdot, \cdot, \cdot, \cdot) \\
C &= \{a \,\mathrm{d}\, a_1, a \,\mathrm{m}^{-1}\, a_2, a \,\mathrm{m}\, a_3\}
\end{aligned}
$$

The head of the operator is a predicate on the functionality $\mathrm{MoveFromTo}$. The operator is considered relevant when the constraint network contains an activity $(\mathrm{MoveFromTo}, \mathbf{kitchen\_livingroom}, \cdot, \cdot, \cdot, \cdot)$, i.e., when a (sub-)goal stating that the robot must move from the kitchen to the living room is present in the network. The operator also prescribes the temporal relations that must exist between the activities, namely that the $\mathrm{MoveFromTo}$ functionality should occur during the availability of the position data ($a \,\mathrm{d}\, a_1$), that it should be met by the precondition of the robot being in the kitchen ($a \,\mathrm{m}^{-1}\, a_2$), and that it meets the effect of the robot being in the living room ($a \,\mathrm{m}\, a_3$).

An operator can also represent a means to achieve information requirements. For example, the operator

$$
\begin{aligned}
a &= (\mathrm{VisualSLAM}, \mathbf{running}, \cdot, u(\mathrm{CPU}) = 10, \cdot, \{\mathrm{position}\}) \\
A_p &= \{a_1, a_2\}, A_e = \emptyset, \text{where} \\
a_1 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{range\_data}\}) \\
a_2 &= (\cdot, \cdot, \cdot, \cdot, \cdot, \{\mathrm{ref\_frame}\}) \\
C &= \{a \,\mathrm{d}\, a_1, a \,\mathrm{m}^{-1}\, a_2\}
\end{aligned}
$$

specifies one way to achieve the necessary information requirement (position) for the $\mathrm{MoveFromTo}$ operator, namely through visual SLAM. This localization functionality requires (1) a functionality which provides range data, (2) a reference frame for the computation of the position estimate, and (3) 10% of the CPU resource. Also, the operator states that range data should be available during the entire duration of the localization process, and that the reference frame is needed at the beginning of the process.

The above operator does not specify how to obtain needed information inputs. For instance, range data might be provided through the following operator:

$$
\begin{aligned}
a &= (\mathrm{StereoCamDriver}, \mathbf{on}, \cdot, u(\mathrm{Cam1}) = 1, \cdot, \{\mathrm{range\_data}\}) \\
A_p &= \{a_1\}, A_e = \emptyset, \text{where } a_1 = (\mathrm{Light}, \mathbf{on}, \cdot, \cdot, \cdot, \cdot) \\
C &= \{a \,\mathrm{d}\, a_1\}
\end{aligned}
$$

An operator may also specify that the reference frame is obtainable by invoking a functionality of the stereo camera's pan-tilt unit:

$$a = (\text{PanTilt}, \mathbf{return\_ref\_frame}, \cdot, \cdot, \cdot, \{\text{ref\_frame}\})$$
$$A_p = \emptyset, A_e = \emptyset, C = \emptyset$$

The above operators can be applied to obtain a configuration plan from the following goal constraint network:

$$A = \{a_0 = (\text{MoveFromTo}, \mathbf{kitchen\_livingroom}, I_0, \cdot, \cdot, \cdot)\},$$
$$C = \emptyset$$

A particular application of the above operators may refine the given constraint network to the following:

$$\begin{aligned} A = \{&a_0 = (\text{MoveFromTo}, \mathbf{kitchen\_livingroom}, I_0, \emptyset, \emptyset, \emptyset) \\ &a_1 = (\text{VisualSLAM}, \mathbf{running}, I_1, u(\text{CPU}) = 10, \\ &\quad \{\text{ref\_frame}, \text{range\_data}\}, \{\text{position}\}) \\ &a_2 = (\text{RobotLocation}, \mathbf{kitchen}, I_2, \emptyset, \emptyset, \emptyset) \\ &a_3 = (\text{RobotLocation}, \mathbf{livingroom}, I_3, \emptyset, \emptyset, \emptyset) \\ &a_4 = (\text{StereoCamDriver}, \mathbf{on}, I_4, \\ &\quad u(\text{Cam1}) = 1, \emptyset, \{\text{range\_data}\}) \\ &a_5 = (\text{PanTilt}, \mathbf{return\_ref\_frame}, I_5, \emptyset, \\ &\quad \emptyset, \{\text{ref\_frame}\}) \\ &a_6 = (\text{Light}, \mathbf{on}, I_6, \emptyset, \emptyset, \emptyset)\}, \end{aligned}$$
$$C = \{a_0 \,\mathrm{d}\, a_1, a_0 \,\mathrm{m}^{-1}\, a_2, a_0 \,\mathrm{m}\, a_3, a_1 \,\mathrm{d}\, a_4, a_1 \,\mathrm{m}\, a_5, a_4 \,\mathrm{d}\, a_6\}$$

This network represents a temporally consistent configuration plan in which resources are never used beyond their capacity, and state variables are never required to assume different values in overlapping temporal intervals. The plan is therefore *feasible*. Furthermore, the plan contains activities providing the required information contents as determined by the operators in the domain. However, not all causal dependencies are necessarily achieved by construction. If, e.g., the initial condition does not state that the light is on, the configuration planner would regard the activity a_6 as yet another sub-goal to satisfy, and might do so through the following operator:

$$a = (\text{Light}, \mathbf{on}, \cdot, \cdot, \cdot, \cdot)$$
$$A_p = \emptyset, A_e = \{a_1\}, \text{where } a_1 = (\text{LightController}, \mathbf{on}, \cdot, \emptyset, \cdot, \cdot)$$
$$C = \{a \,\mathrm{p}^{-1}\, a_1\}$$

This operator models an actuation process (Light represents an environment variable), and its application would refine the configuration plan by adding an activity $a_7 = (\text{LightController}, \mathbf{on}, I_7, \emptyset, \emptyset, \emptyset)$ to the network, along with the constraint $a_6 \,\mathrm{p}^{-1}\, a_7$, prescribing that the LightController be in state **on** before the light is required to be on. Note that the light control functionality has no information requirements ($\text{In}^{(a_1)} = \emptyset$).

Planning in Closed Loop

Fig. 1 provides an overview of our approach. As a whole, our architecture can be seen as a controller for a dynamical system: the *controlled system* consists of the robot(s) and the environment in which they operate; the *controller* is composed of an observer and a regulator, and has the task to dispatch actuation commands to the controlled system so that its observed state S coincides with the desired state G.

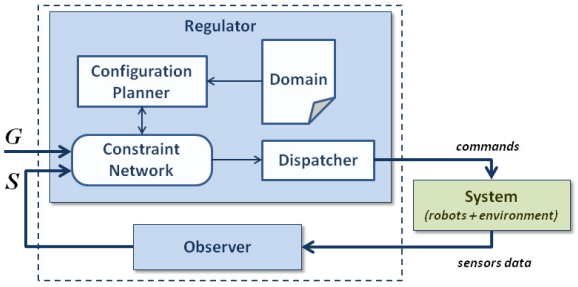


Figure 1: Overview of the control architecture.

At the core of the controller is a shared constraint network. The *observer* adds activities and temporal constraints to this network, which represent the current state of the environment as provided by sensors: if a sensor sensorX reports a new reading $\mathbf{v}$ at time t_{now}, the observer inserts a new activity $(\text{sensorX}, \mathbf{v}, I, \emptyset, \emptyset, \emptyset)$ and adds a temporal constraint restricting the beginning of I to be t_{now}; if the reading of sensorX changes, the previous activity is constrained to end at t_{now} and another activity is started.

The *regulator* includes the configuration planner, and a dispatcher that sends to the system the executable actions in the constraint network. An action $(\text{deviceY}, \mathbf{v}, [I_s, I_e], \cdot, \cdot, \cdot)$ is considered executable when two conditions are met: (1) $t_{\text{now}} \leq \min\{I_s\}$; and (2) its observable preconditions are verified in the current state. The latter condition is tested by attempting to unify each activity representing such a precondition with a sensed activity produced by the observer. If unification fails, the precondition is delayed by inserting a temporal constraint, and re-evaluated at the next iteration. If both conditions are met, the command $\mathbf{v}$ is transmitted to deviceY.

The configuration planner, which implements the control logic of the regulator, is a search procedure which continuously modifies the constraint network so as to guarantee the presence of a feasible plan given the goal $G = (A_g, C_g)$. The resulting constraint network represents one or more temporal evolutions of the state variables that guarantee the achievement of G under nominal conditions. In this sense, our planner follows a least commitment principle. Feasible and goal-achieving configuration plans are obtained by means of five interacting solvers. (1) a *temporal solver* propagates temporal constraints to refine the bounds $[l_s, u_s], [l_e, u_e]$ of activities, and returns failure if and only if temporally consistent bounds cannot be found; (2) a *resource scheduler*, which chooses and posts to the network temporal constraints so as to remove temporal overlap from sets of over-consuming, possibly concurrent activities (Cesta, Oddi, and Smith, 2002); (3) a *state variable scheduler*, which exploits the same constraint-posting mechanism of the resource scheduler to ensure that activities do not prescribe conflicting states in overlapping intervals; (4) an *information dependency reasoner*, which instantiates relevant operators (in the form of activities and temporal

constraints) so as to enforce the information dependencies modeled in the domain; (5) and a *causal planner*, which instantiates relevant operators so as to enforce the causal dependencies modeled in the domain.

Temporal feasibility enforcement is not subject to multiple choices, as the temporal constraints form a Simple Temporal Problem (Dechter, Meiri, and Pearl, 1991), which is tractable. Tractability also holds for information dependencies, which in our approach constitute an acyclic propositional Horn theory. Conversely, all other reasoners must search in the space of alternative choices for conflict resolution (e.g., alternative sequencing decisions, alternative operator selections). All of these choices are seen as *decision variables* in a high-level Constraint Satisfaction Problem (CSP). Given a decision variable d, its possible values constitute a finite domain $\delta^d = \{(A_r^d, C_r^d)_1, \ldots, (A_r^d, C_r^d)_n\}$, whose values are alternative constraint networks, called *resolving constraint networks*. The individual solvers are used to determine resolving constraint networks $(A_r^d, C_r^d)_i$, which are iteratively added to the goal constraint network (A_g, C_g).

Function `Backtrack` (A_g, C_g) : success or failure

```
1  d ← Choose((A_g, C_g), h_var)
2  if d ≠ ∅ then
3      δ^d = {(A_r^d, C_r^d)_1, ..., (A_r^d, C_r^d)_n}
4      while δ^d ≠ ∅ do
5          (A_r^d, C_r^d)_i ← Choose(d, h_val)
6          if (A_g ∪ A_r^d, C_g ∪ C_r^d) is temporally consistent then
7              return Backtrack(A_g ∪ A_r^d, C_g ∪ C_r^d)
8          δ^d ← δ^d \ {(A_r^d, C_r^d)_i}
9      return failure
10 return success
```

In order to search for resolving constraint networks, we employ a systematic search (see Algorithm `Backtrack`), which occurs through standard CSP-style backtracking. The decision variables are chosen according to a variable ordering heuristic h_{var} (line 1); the alternative resolving constraint networks are chosen according to a value ordering heuristic h_{val} (line 5). The former decides which (sub-)goals to attempt to satisfy first, e.g., to support a functionality by applying another operator, or to resolve a scheduling conflict. The latter decides which value to attempt first, e.g., whether to prefer one operator over another (recall that there may be more than one decomposition for a given activity.) Note that adding resolving constraint networks may entail the presence of new decision variables to be considered.

The possible values for resource contention or unique state decision variables are temporal constraints. In particular, our framework relies on the earliest time timelines to assess both resource over-consumption and multiple overlapping states. Values for information decision variables are ground operators, as shown in the previous Section. Lastly, values for causal decision variables are either ground operators, or unifications with activities that already exist in the constraint network. Search uses unification to build on previously added activities — e.g., leveraging that the light has already been turned on to support a previously branched-upon causal dependency. Unification also allows to accommodate on-going sensing and execution monitoring processes during planning. For instance, $(\text{Light}, \mathbf{on}, I^{(a)}, \emptyset, \emptyset, \emptyset)$ may be supported by unification with $(\text{Light}, \mathbf{on}, [[0,0][13,13]], \emptyset, \emptyset, \emptyset)$ which models the temporal interval within which a light source was sensed.

Since the configuration planner is in a closed loop with the system (through the observer and dispatcher), modifications of the constraint network necessary to achieve the goal can occur whenever a disrupting renders the network an infeasible plan. Note that due to the presence of the above solvers in this loop, the modifications made to the network in the face of disturbances can take on the form of temporal propagation, resource or state variable scheduling, or operator application, depending on the situation. If temporal reasoning and scheduling fail, replanning is needed: the planner empties the constraint network of all activities whose lower bound is greater than the current time, and Algorithm `Backtrack` is re-invoked. Note that all activities representing operators currently under execution, as well as those already executed remain in the network, as do all activities representing the sensed state of the physical world. As shown below, this mechanism also enables to deal with dynamically posted goals.

Experiments

Methodology. Planners are typically evaluated in terms of internal properties of the algorithm, such as completeness, complexity or optimality. In this work, we are interested in the closed loop behavior of the planner when coupled to the controlled system. Intuitively, the question that we want to answer is not "does the planner generate a correct plan to the goal?" but rather "does the planner bring the system to the goal — even in the face of discrepancies between its model and the actual world?". This question is similar to the one typically considered in control theory: will the controller bring the state of the controlled system to the desired set point? Accordingly, our experimental methodology is inspired by the methods used for closed loop controllers.

Referring to Fig. 1, we call E (for *error*) the distance $d(S, G)$ between the current state S of the system, and its desired state G. We are interested in the ability of our controller to keep E to zero, even in the face of disturbances. Disturbances include un-modeled factors, like exogenous events and delays, and changes in the goal G. If S and G are values in a metric space, e.g., vectors of real numbers, d is a standard distance function. In our case, S and G are constraint networks, so we define a distance in the space of such networks with respect to a given domain $\mathcal{D}$. We let $d(S, G)$ be the minimum number of network transformations needed to solve the configuration planning problem $(G, \mathcal{D})$ starting from the initial state S. That d depends on $\mathcal{D}$ is natural, since it depends on what actions the controlled system can perform. Operationally, we compute E by invoking our configuration planner on G from the initial state S and counting the number of decision steps: this gives an upper bound of the real $d(S, G)$.

Experimental setup. We test the controller's ability converge on the desired goal network $G = (A_g, C_g)$ in four scenarios. In the first three, G is fixed and the system is subject to (1) temporal perturbations (delays) of increasing magnitude, (2) resource collapse, and (3) exogenous events. These

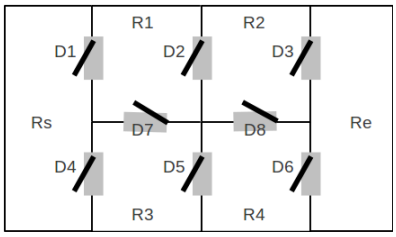


Figure 2: Testing environment used in the evaluation.

are disturbances that affect real robotic systems, e.g., navigation takes longer due to a crowded corridor, a room with limited capacity is occupied, or a door assumed to be open is found closed. In scenario (4), we measure the tracking performance of the system by varying the goal according to a fixed schedule and subjecting the system to temporal perturbations.

In all tests, we employ a domain in which three robots have to deliver goods within the indoor environment depicted in Fig 2. Rooms are connected through actuated doors. Rooms `Rs` and `Re` are, respectively, the pick up and drop-off locations. Rooms `R1-R4` have a maximum capacity of one robot, while rooms `Rs` and `Re` can accommodate up to three robots. Robot navigation requires several resources: motors, a localization device (either a laser or a camera) and the availability of source and destination rooms. Each goal has a deadline, whose violation implies plan failure.

Tolerance to disturbances. In the first test, each robot must pick up an object in `Rs`, release it in `Re`, and navigate back to its starting point. During execution, delays in $[0, D_{\max}]$ were applied to the start or end time or all activities. $D_{\max}$ was set to 0, 5, 10, 20 and 50 units, and 100 runs were performed for each value of $D_{\max}$.

With $D_{\max} = 0$, the system always managed to converge to the goal. The longest execution was 57% of the overall deadline, i.e., the system afforded a slack of 43%. When delays are introduced the system may not achieve the goal within the given deadline. This depends on the magnitude of the delay compared to the available slack. At 33%, the system diverged in 32% of the runs, while at 40% it diverged in 60% of the runs. Fig. 3 (a) plots the value of E over time for the cases where the system converged. This plot gives a qualitative impression of the behavior of the system when the deadline can be met: the distance to the goal monotonically decreases until convergence, albeit more slowly for increasing delays.

In the second test, we subject the system to resource collapses by lowering the capacity of every resource during an interval with random start and duration, where the duration is in $[0, D_{\max}]$ with $D_{\max} \in \{5, 10, 20, 50\}$. In light of Fig. 3 (a), we wish to assess the role of the heuristics used in the planner's search algorithm. Our planner follows a *least commitment* principle, as it does not commit to a particular fixed-time execution schedule during planning, rather it imposes constraints that prune out behaviors that do not comply with the goal (A_g, C_g). For this reason, we compare the controller's performance with two alternative heuristics for resource scheduling: h_{flex}, which gives priority to scheduling decisions that least impact the remaining temporal flexibility of the network (Cesta, Oddi, and Smith, 2002); and h_{rndm}, which decides to resolve resource contention without taking into account temporal flexibility, thus potentially overcommitting with respect to the current distance to the deadline. As an illustration, the Root Mean Square (RMS) rigidity of the constraint network (Hunsberger, 2002), which is inversely related to the amount of temporal slack in the network, is shown in Fig. 3 (b). Two runs were performed with the same magnitude of perturbation ($D_{\max} = 5$), one using using h_{flex}, the other using h_{rndm}. As shown, in the former run the network retains much more flexibility, testifying to the fact that least commitment affords more adjustments to the plan until goal achievement. This enhanced flexibility is reflected in the results shown in Table 1: injecting the same disturbances, h_{flex} performs significantly better than h_{rndm}, achieving convergence in a majority of scenarios with resource unavailability periods of up to $D_{\max} = 20$.

$D_{\max}$	h_{flex}	h_{rndm}
5	92%	60%
10	74%	58%
20	53%	37%
30	27%	19%

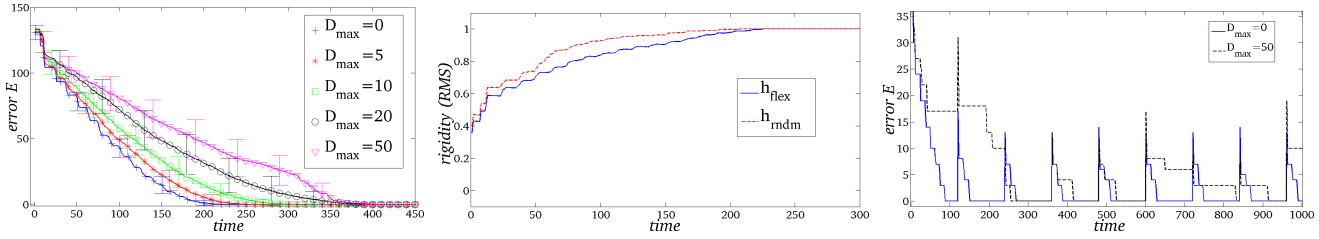
Table 1: Percentage of runs in which convergence was achieved for varying magnitudes of resource collapse.

In the third experiment, we perturb the system by introducing exogenous events. Specifically, at regular intervals T_{door}, each door can be closed with probability p. This implies that each door will be closed at least once during the whole plan with probability p_{close}, which can be computed from p and the nominal duration of the plan. Once a door is closed, re-planning may be necessary due to the discrepancy between the planned and sensed activities. Table 2 shows the rate of succeeded executions for different values of p_{close} and for $T_{door} = 10$.

p_{close}	0%	20%	40%	60%	100%
Success rate	100%	76%	47%	30%	3%

Table 2: Success rate when doors are closed with p_{close}.

Tracking a dynamic goal. The last experiment is related to a tracking problem: the controller receives on-line requests to put a room under surveillance. A room is considered such if it contains two robots. A domain specific heuristic is used to accomplish this task, namely assigning robots to rooms based on distance. All rooms have capacity two and doors have capacity one, as only one robot can pass the threshold at a time. The sequence of rooms to be put under surveillance is periodically posted to the controller,



(a) (b) (c)

Figure 3: (a) Distance to goal (E) over time for different amounts of delays (each curve is the average of 100 runs for a given value of $D_{\max}$; vertical bars are standard deviation); (b) RMS rigidity of the constraint network over time for h_{flex} and h_{rndm}; (c) tracking a moving goal, with $f = 3\bar{T}$ and with or without delay ($D_{\max} = 50$).

f	$D = 5$	$D = 10$	$D = 20$	$D = 50$
$1/120$	1	0.975	0.91	0.76
$1/90$	0.94	0.92	0.87	0.65
$1/60$	0.92	0.97	0.83	0.4
$1/40$	0.93	0.68	0.6	0.24

Table 3: Success rate for the tracking scenario — rows: goal posting period, columns: success ratio compared to the zero delay case

and each posting supersedes the previous one. To assess the tracking performance of the controller, we choose a more fine-grained metric than convergence, namely the number of surveillance tasks achieved. We vary the goal posting period T_{post} in the range $[40, 120]$ time units, and perturb the system with random delays between 0 and $D_{\max}$, with $D_{\max}$ up to 50 time units. The periodic sequence (`Re`, `R1`, `R4`) has been chosen such that plan execution time in the absence of another posted goal is, on average, a constant $\bar{T}$. Fig. 3 (c) shows the controller's behavior with a posting period of $3\bar{T}$ in two cases: when there are no delays, and when there are delays with $D_{\max} = 50$. As can be seen, in the absence of disturbances each goal is always achieved before another goal is posted (i.e., the curve always reaches 0 before rising back up in consequence of another goal being posted). Conversely, this is not the case when delays are introduced. The complete results are shown in Table 3: as expected, failures increase significantly when the posting period approaches $\bar{T}$.

Discussion and Conclusions

Our work aims at building a planner that can be used with real robotic systems. Experiments exploiting robotic platforms have been shown in (Di Rocco et al., 2013), this paper rather focuses on the performances achieved by the system in presence of disturbances. We have introduced a new formalism which provides sufficient expressiveness to capture the salient features of a multi-robot domain with resources and information dependencies. However, the novelty of this work is not in the formalism itself (which builds heavily upon previous work in constraint-based temporal planning (Cesta and Oddi, 1996; Do, 2012; Smith, Frank, and Cushing, 2008)), rather in catering explicitly to the real problems faced by roboticists: complexity of sensing and actuation may cause failed or delayed actions, incorrect localization, interrupted data flows — all of which warrant continuous plan adaptation.

This work also leverages another key idea present in architectures like EUROPA (Frank and Jónsson, 2003), OMPS (Fratini, Pecora, and Cesta, 2008), IDEA (Finzi, Ingrand, and Muscettola, 2004) and T-REX (McGann et al., 2008a), namely the use of a common constraint database onto which flaw-resolving decisions are posted. However, note that in our work the particular decisions posted to the constraint network are subject to a meta-level CSP-style search. This is the first system which truly adopts a meta-CSP approach in exploring the combined search-space of causal, information, resource and temporal reasoning. Using a principled algorithmic methodology as the backbone for integrating different solvers forebodes the possibility to include further domain features (e.g., spatial reasoning, ad-hoc procedures) and to orchestrate them properly through variable and value ordering heuristics with other decision processes. The implementation of the approach is based on the Meta-CSP Framework (Pecora et al., 2012), an open source Java API which facilitates the development of meta-CSP based problem solvers.

Through an experimental evaluation, we have shown how plans are adapted on-line and at different levels depending on the specific contingency (temporal adjustments, requiring polynomial computation; scheduling, requiring to incrementally solve an NP-hard problem; causal/information dependency resolution, requiring exponential time). A limitation of the current approach, which makes an interesting point for future work, is the lack of support for uncertainty about EVs; also, it may be interesting to support soft goals for some robotic applications. Another particularly significant direction of future work will be to explore issues related to the controllability of the temporal constraint network (in our present approach, all time points are assumed to be controllable from the point of view of temporal reasoning). In particular, it is worth investigating whether existing techniques for ensuring its controllability (e.g., Cimatti, Micheli, and Roveri, 2012) are suitable for configuration planning and multi-robot contexts.

On-going work is addressing performance analysis and deployment in physically instantiated scenarios, as well as validation through more extensive tests. We will also focus on re-planning strategies, address the issue of including pre-

dicted/perceived human plans dynamically, and taking into account spatial constraints.

Acknowledgments

This work was funded by the EC Seventh Framework Programme (FP7/2007-2013) grant agreement no. 288899 Robot-Era. The authors wish to thank the anonymous reviewers for their comments and suggestions.

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2):123–154.

Barreiro, J.; Boyce, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Smith, T.; and Do, M. 2012. EUROPA: A platform for AI planning. In *Proc of ICAPS-ICKEPS*.

Cesta, A., and Oddi, A. 1996. DDL.1: A formal description of a constraint representation language for physical domains. In Ghallab, M., and Milani, A., eds., *New Directions in AI Planning*. IOS Press: Amsterdam.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A constraint-based method for project scheduling with time windows. *Journal of Heuristics* 8(1):109–136.

Cimatti, A.; Micheli, A.; and Roveri, M. 2012. Solving temporal problems using smt: Strong controllability. In Milano, M., ed., *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science. Springer Berlin Heidelberg. 248–264.

Dean, T., and Wellman, M. 1991. *Planning and control*. Morgan Kaufmann series in representation and reasoning. M. Kaufmann Publishers.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Di Rocco, M.; Pecora, F.; Kumar, P.; and Saffiotti, A. 2013. Configuration planning with multiple dynamic goals. In *Proc. of AAAI Spring Symposium on Designing Intelligent Robots*.

Do, M. B., and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)* 20:155–194.

Do, M. 2012. Nddl reference manual. Available at `https://code.google.com/p/europa-pso/wiki/NDDLReference`.

Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Autonomous Agents and Multi-Agent Systems* 19(3):332–377.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. of the 19th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Fikes, R., and Nilsson, N. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

Finzi, A.; Ingrand, F.; and Muscettola, N. 2004. Model-based executive control through reactive planning for autonomous rovers. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Constraints* 8(4):339–364.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.

Gerevini, A.; Saetti, A.; and Serina, I. 2006. An approach to temporal planning and scheduling in domains with predictable exogenous events. *J. Artif. Int. Res.* 25(1):187–231.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *AIPS*, 61–67.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Hunsberger, L. 2002. Algorithms for a temporal decoupling problem in multi-agent planning. In *Eighteenth national conference on Artificial intelligence*, 468–475. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Knight, S.; Rabideau, G.; Chien, S.; Engelhardt, B.; and Sherwood, R. 2001. Casper: Space exploration through continuous planning. *Intelligent Systems* 16(5):70–75.

Köckemann, U.; Pecora, F.; and Karlsson, L. 2012. Towards planning with very expressive languages via problem decomposition into multiple csps. In *Proc. of the ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS)*.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, 617–622. AAAI Press.

Lundh, R.; Karlsson, L.; and Saffiotti, A. 2008. Autonomous functional configuration of a network robot system. *Robotics and Autonomous Systems* 56(10):819–830.

Lundh, R. 2009. Robert lundh. robots that help each-other: Self-configuration of distributed robot systems. In *PhD Thesis. rebro University, rebro, Sweden, May 2009.*

McGann, C.; Py, F.; Rajan, K.; Ryan, J. P.; and Henthorn, R. 2008a. Adaptive Control for Autonomous Underwater Vehicles. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence*.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008b. A Deliberative Architecture for AUV Control. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA)*.

Parker, L., and Tang, F. 2006. Building multirobot coalitions through automated task solution synthesis. *Proc of the IEEE* 94(7):1289–1305.

Parker, L., and Zhang, Y. 2012. Task allocation with executable coalitions in multirobot tasks. *Proc of the Int. Conf. on Robotics and Automation (ICRA)*.

Pecora, F.; Di Rocco, M.; Köckemann, U.; Mansouri, M.; and Ullberg, J. 2012. The meta-csp framework public repository. Available at `http://meta-csp-framework.googlecode.com`.

Saffiotti, A.; Broxvall, M.; Gritti, M.; LeBlanc, K.; Lundh, R.; Rashid, J.; Seo, B.; and Cho, Y. 2008. The PEIS-ecology project: vision and results. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2329–2335.

Smith, D. E.; Frank, J.; and Cushing, W. 2008. The ANML language. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Zhang, Y., and Parker, L. E. 2011. Solution space reasoning to improve iq-asymtre in tightly-coupled multirobot tasks. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 370 –377.

Zhang, Y., and Parker, L. E. 2013. Considering inter-task resource constraints in task allocation. *Autonomous Agents and Multi-Agent Systems* 26(3):389–419.

Deliberative Systems for Autonomous Robotics: A Brief Comparison Between Action-oriented and Timelines-based Approaches

Pablo Muñoz and María D. R-Moreno
Departamento de Automática, Universidad de Alcalá
Ctra. Madrid-Barcelona Km 33,600 E-28871
Alcalá de Henares, Madrid

Abstract

Autonomous robotics is a multidisciplinary and widely research area. Several approaches have been followed to make autonomous and reliable control architectures, especially for applications in environments inaccessible to humans. We are going to focus on two architectures designed to automate the operation of an exploration robot: on the one hand we have the Model-Based Architecture and on the other hand the Goal-Oriented Autonomous Controller.

On these architectures there is at least one deliberative entity, which is the responsible of managing the long term planning to accomplish the objectives of the mission for which it has been designed for. The first architecture implements an action-oriented planner that uses the Planning Domain Definition Language (PDDL) to describe the environment, available actions and goals. The second system employs a timelines approach, which reasons about temporal state variables using the Domain Definition Language (DDL) to model the physics interactions in the world. In this paper we present the deliberative models that these architectures require to perform a robotic exploration mission, and then, we present a briefly and initial analysis about the differences between the solutions given by both planners, focusing on the flexibility of the deliberative process and the semantics and representative capabilities of the languages employed.

Introduction

Since the birth of modern robotics, important efforts have been made in designing and implementing architectures that allow a robot to autonomously inter-operate with its environment to perform specific tasks. The field in which more research is conducted in autonomous control architectures is exploration robots, whether underwater or space, due to the extreme conditions that make them inaccessible to humans.

The long term planning and scheduling is a primary topic in the design of these autonomous architectures for robotics control. In this way, there are several approaches to solve the deliberative necessities for new science missions such as Curiosity or Mars Science Laboratory (MSL) from NASA or the ESA ExoMars, intended to be launched in 2018. The operation of these robots is a complex task, due to the hard environment conditions, the communications problems (i.e. delays between the ground operation team and the communication windows) and the hard constraints required to the survival of the mission. So, these constraints will require of increasingly capable systems to achieve the goals they are designed for.

In this paper we present a small comparison between the deliberative capabilities of two autonomous architectures designed to automate the operation of a rover-like robot: on the one hand the MoBAr (Model-Based Architecture) (Muñoz, R-Moreno, and Martínez 2011) and, on the other hand, the GOAC (Goal-Oriented Autonomous Controller) system (Ceballos et al. 2011). These architectures employ two distinct approaches to the planning problem as R-Moreno et al. (2008) differentiate:

- Action-oriented: it uses predicates logic and the world is seen as an entity that can be in different states. The domain specifies actions that can be performed to change the state of the world and only applicable when some particular states are set. The objective is to find a sequence of actions that, from an initial world state, through applying successive actions, the system achieves a desired goal state. This approach is followed by the MoBAr architecture deliberative layer.

- Timelines-based: this one is more recent than the action-oriented and it is based on the first order logic. It represents the world in terms of functions that describes the behavior of the system from a time perspective: a timeline is a logical structure used to represent and reason about the evolution of an attribute over a period of time. Rules must be defined to specify how the timelines can change, in order to obtain a sequence of decisions from the planner that bring the set of temporal functions to a final state in which a set of constraints are satisfied. The GOAC deliberator uses this approach.

The paper is structured as follows: first, there is a brief description of these two architectures, focused on the deliberative entity. Next section presents an example problem and the deliberative models that each control architecture uses to solve it. Then we present a brief analysis about the capabilities and lacks of each model. Finally some conclusions are outlined.

Model-Based Architecture

The MoBAr architecture initially developed for the Ptinto robot **??** corresponds to a three layers (3T) system (Gat 1998), in which the top tier will be in charge of the deliberative process, long-term memory and learning process as a function of events that occur in the environment. The middle level or execution system also has a short-term memory, as well as a series of rules that trigger the reactive behavior implemented, in order to response in a short time to eventual situations that may occur in both, the environment and the internal state of the robot. Finally, the low level or functional level, is responsible of providing the functionality of the robot, and relay the information collected by the sensors.

Each layer is based on a model with different levels of abstraction. A model defines the properties, capabilities and constraints of the robot at each layer. Starting from the functional layer, which represents the hardware abstraction level, that is, the definition of the internal state of the robot plus the abilities that it has, the upper layers have fewer detailed models, and thus, less coupled with the underlying hardware. In this way, the executor is in charge of taking high level actions coming from the deliberator and decomposing them into lower level commands supported by the functional layer. So, we want that the executor model has little relationship with the hardware, and the high level model only knows the hardware in terms of goal oriented actions and high level abstraction of constraints.

To implement the MoBAr architecture we have taken advantage of different general purpose technologies: for the deliberative layer we have used the PDDL language (McDermott 1998) and a PDDL-based planner. We have chosen the Universal Executive and its language, PLEXIL (PLan EXecution Interchange Language) (Verma et al. 2006) to model and control the executor, and, finally the G^{en}oM2 (Generator Of Modules) framework (Mallet, Fleury, and Bruyninckx 2002) for the definition and implementation of the functional layer. The first two layers, deliberator and executor, use models represented by a language that will be interpreted by a program, and thus, are easily interchangeable in order to adapt them to multiple robots. Instead, the functional layer is strongly dependent on the hardware.

The architecture follows a sense-(re)plan-act cycle (see fig. 1): at first time, the system takes the initial state of the world (state from the functional layer and the PDDL problem), the goals to achieve and it obtains a feasible plan. This plan is static: the executor takes each action, decompose it in a set of commands and send it to the functional layer. With the results of these commands execution, the executor checks if there is an unexpected situation, and, if it cannot handle that situation, it updates the information of the world, that is goals currently achieved and the actual state of the robot, and then the planner must obtain a new plan.

Focusing on the deliberative capabilities, PDDL-based planners are systems that use two input files to represent their knowledge base. One of the files contains a description of the actions that represent "what can/cannot be done" and the other file includes the three elements which define the problem: the known objects of the world, the initial state and the goals we want to achieve. With this information, the planner searches a sequence of actions that can reach the goals from the initial state. The optimal solution of the problem is a conjunction of two factors: the metric used and the resolution algorithm.

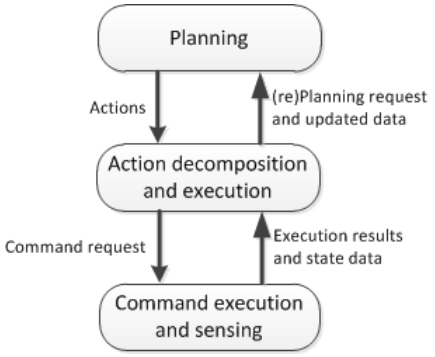


Figure 1: Execution cycle in MoBAr.

Currently we are using the SGplan$_6$ (Hsu and Wah 2008) PDDL-planner that accepts PDDL version 2.1 (Fox and Long 2003) and common features of PDDL 3 (Gerevini and Long 2005). With PDDL 2.1 we can determinate how long each action will take and, using fluents, we can establish a basic resource model for the energy consumption of the actions, and consistently decide whether a plan is feasible or not in terms of the total amount of energy consumed. Also, using version 3, we can employ goals as preferences, so if there is an unreachable goal, we can obtain a plan that ignores it.

Goal-Oriented Autonomous Controller

The GOAC system is the result of a multi-institutional[1] effort within the on-going Autonomous Controller Research and Development activity funded by ESA-ESTEC. GOAC is an architecture that integrates four mature technologies: the functional layer is implemented with the couple G^{en}oM3 (Mallet et al. 2010) plus BIP (Basu, Bozga, and Sifakis 2006) to ensure generation of reliable, modular and verifiable functional components for the hardware abstraction layer; T-REX (McGann et al. 2007; Rajan et al. 2009) is the responsible of the planning and execution dispatching using a timeline-based representation and an interleaving schema (which is similar to IDEA (Aschwanden et al. 2006)); and, finally, the deliberation process resides in the APSI planner (Fratini et al. 2011), a timeline-based, domain independent planner. The deliberative layer of GOAC is composed by a set of one or more deliberative reactors, that is, the couple of a T-REX reactor and an APSI planner. This schema could be seen in fig. 2.

Each deliberative reactor follows a sense-plan-act paradigm for goal oriented autonomy. This set is not fixed, for each robot or mission it could have a different design by defining various reactors and their interactions, giving a scal-

[1]GOAC involves these partners: LAAS-CNRS (France), VERIMAG (France), MBARI (United States of America), CNR-ISTC (Italy) and GMV (Spain).

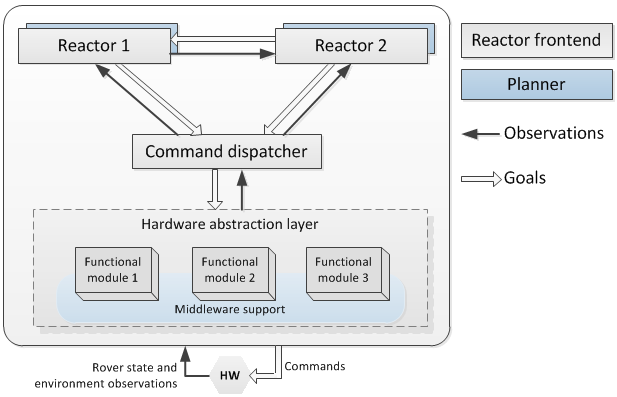


Figure 2: A possible instance of the GOAC architecture.

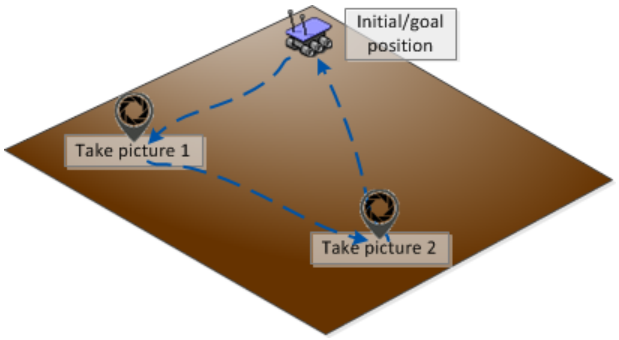


Figure 3: Problem representation.

able architecture. That allows a divide and conquer approach in which the scope of each deliberative reactor (those ones in charge of the planning process using the APSI planner) could be refined by other more specific reactors. To do this, reactors can interact with other sending goals and receiving observations.

In order to ensure the correct execution of the required commands to achieve the goals, it is possible to have one or more reactors in charge of the command dispatching between the deliberative reactors and the functional layer. Considering the fig. 2, there is only one of this kind of reactor, called "command dispatcher". This reactor implements a procedural executive that not only gives correctness in the command execution, it also gives a better level of abstraction and an interface between the functional layer and the higher level reactors. In fact, this reactor is responsible of gathering observation from the functional layer and to provide it in a convenient way to the deliberative reactors.

The deliberative process resides in the APSI planner. For each deliberative reactor there is one instance of an APSI planner, and every one has its own look-ahead window over which to deliberate. APSI uses a timelines representation, which encapsulates an evolution of a particular state variable over time. To define the world it implements the Domain Definition Language (DDL) language (Cesta and Oddi 1996; Fratini, Pecora, and Cesta 2008) that enables to specify the allowed state transitions as well as the causal and temporal relationships between state variables, so, the problem is modelled by identifying a set of relevant features whose temporal evolution needs to be controlled to obtain a desired behavior. Therefore, the result of a deliberative process is a sequence of state transitions for each timeline that achieves a specified condition as a planning goal.

Each deliberative reactor reasons about a specific part of the whole domain, being one the high level reactor, which has the more abstract states, and whenever we go down in the reactor "hierarchy", the reactors are more specific. In this way, the high level reactor puts goals in the timelines of other more specific reactors. Like that, a high level goal state could be decomposed into lower level goals states that must be reached by their respective timelines before the completion of the goal state in the high level. To do this, the reactors have two types of timelines: internal and external. The first ones are the timelines that the deliberative reactor can control directly, the reactor can change its value, but usually the value is dependent on one or more external timelines. These are not controllable from the reactor, but it can produce goal states in that timelines required to reach the goal state of its internal timelines.

Rover example problem

To analyse the deliberative capabilities for the two previous architectures, we define a basic scenario for an exploration rover. The scenario, represented in fig. 3, consists of a rover (such as the Dala[2] robot) that must achieve the acquisition of two pictures in different locations (and possibly with different pointing of the pan-tilt unit), and then, return to the start point. Also, during the traverse it could exist visibility windows in which the rover can transmit the data acquired to a station. The objective is to send all the possible pictures taken during these windows.

To safely operate the rover, there is a little set of constraints that must be included in the models:

- The rover is able to move between two points in space given their coordinates (x, y).
- The rover has two navigation modes: one for flat terrain (using a sick laser range finder) and other for rough terrain (using the stereo-vision cameras). The navigation mode is defined by this property of the terrain, which is continuously updated by a functional module. A change in the terrain triggers a signal that must be trapped by the upper layers to ensure the use of the correct navigation mode.
- The pan-tilt unit can move to reach a desired point, given by their angles (α, β).
- During the acquisition of a picture, the pant-tilt unit must be pointing at the desired site and the rover must stay still.
- When the rover is moving, the pan-tilt unit must be pointing to the front, that is, $(0, 0)$.
- It is possible to transmit pictures while the rover is stopped and exist a visibility window.

[2]http://homepages.laas.fr/matthieu/robots/dala.shtml

```
(:durative-action MoveTo
 :parameters (?r - rover ?p1 ?p2 - loc ?n - navmode)
 :duration (= ?duration (/ (distance_to_move ?p1 ?p2) (speed ?r ?n)) )
 :condition (and (over all (has_locomotion ?r))
            (over all (navigation_mode ?r ?n))
            (over all (platine_pos NavCam plat0_0))
            (at start (position ?r ?p1))
            (at start (>= (energy ?r)(*(power_per_m ?r ?n)(distance_to_move ?p1 ?p2)))) )
 :effect (and (at start (not (position ?r ?p1)))
         (at end (position ?r ?p2))
         (at end (decrease (energy ?r)(*(power_per_m ?r ?n)(distance_to_move ?p1 ?p2)))) )
)
(:durative-action MovePlatine
 :parameters (?r - rover ?c - cam ?p1 ?p2 - platpos)
 :duration (= ?duration (time_move_platine ?p1 ?p2))
 :condition (and (at start (platine_pos ?c ?p1))
                 (at start (>= (energy ?r)(*(platine_energy)(time_move_platine ?p1 ?p2)))) )
 :effect (and (at start (not (platine_pos ?c ?p1)))
              (at end (platine_pos ?c ?p2))
              (at end (decrease (energy ?r)(*(platine_energy)(time_move_platine ?p1 ?p2)))))
)
(:durative-action TakePicture
 :parameters (?r - rover ?p - loc ?c - cam ?a - platpos ?m - mode)
 :duration (= ?duration (time_to_picture ?c ?m))
 :condition (and (over all (camera_mode ?r ?c ?m))
                 (over all (position ?r ?p))
                 (over all (platine_pos ?c ?a))
                 (at start (>= (energy ?r) (camera_energy ?c ?m))) )
 :effect (and (at end (picture ?p ?m ?a))
              (at end (decrease (energy ?r) (camera_energy ?c ?m))) )
)
```

Figure 4: PDDL actions definition for the rover example.

We are interested in reviewing how both models designed for this scenario, one modelled in PDDL and the other in DDL, solve the problem. Taking into consideration real constraints in the scenario, such as partially known (and potentially dynamical) environment, energy requirements and possibility of failures; make that problem a good case of study for a basic comparison between models. Next sections briefly explain the high-level models for the MoBAr and the GOAC architectures respectively.

PDDL model

The PDDL model here presented is an adaptation of the employed in the MoBAr architecture (Muñoz, R-Moreno, and Martínez 2011). The problem and domain files contain the knowledge of the rover and the actions that it can perform. The domain, shown in fig. 4, specifies the actions that the robot perform to change both, its internal state and the environment. The actions representation include the duration and the energy consumption. The energy is treated as a fluent, but it is not the best solution; the energy model can be more effective if it were treated as a resource. The system only maintains the value of the fluent, modifying it during the planning, but the planner does not reason about how to deal with it in a efficient manner. For each action, there is a precondition that specifies the amount of energy required to performing the action, and therefore, there is an effect to decrease the energy level consequently.

The movement action (`MoveTo`) is based on the rover navigation mode and the travel distance. The time to travel is dependent on the speed of the rover in the current navigation mode. One relevant condition prior to move is to check that the pan-tilt unit of the navigation cameras is pointing to the front.

The other actions are related to the picture acquisition: `MovePlatine` and `TakePicture`. First one moves the pan-tilt unit associated to a specific camera from a start position to a desired position. The duration of this action depends on the movement, and the energy required is a fixed value multiply by the required time. To take a picture, we need a camera with a determined mode, orientation of its pan-tilt unit and the rover stopped in the desired location. Time spent and energy consumption of the action is a function of the camera and mode employed.

For the previous domain we need to define the present objects of the world, the initial state (including rover energy consumption and time spent to move between locations) and the goals to achieve. The problem presented in fig. 5 includes all the necessary data to solve the rover example problem explained previously. First, we need to define the possible objects, that is, the relevant locations, cameras and modes, pan-tilt positions and, finally, rover and its navigation modes.

For the initial state, the connections between locations must be set, defined by the distance that separates both points. If we want to define different times for each navi-

gation mode, we need to duplicate that info including the navigation mode involved as a parameter. Next there is the rover data. It contains the initial position, energy, navigation mode and the energy consumption of each subsystem. As specified in the domain model, the energy required for the navigation and camera subsystems depend on the current operation mode. Also, there are the functions that specify the duration of the movement for the pan-tilt unit.

The last element of the problem is the goal(s) definition. It defines the tasks that the rover must perform, such as, go to a desired location, or take an image from a location and with a particular orientation and mode. If the planner supports plan preferences, one or more targets cannot be satisfied by the planner, assuming a penalization for each unsatisfied goal. The goals defined for the rover example are: take two pictures and finish the plan in the initial location. The metric defined specifies the goodness of the solution in terms of the time spent and the unsatisfied goals.

DDL model

The domain and problem described here correspond to a case study scenario for the Dala robot employed in a particular instance of the GOAC architecture. For this example there are two deliberative reactors: one in charge of the mission control (the one that accepts the goals from the user), and the other composed of different timelines to represent the rover subsystems (navigation, camera, platine and terrain to define the navigation mode) at which the other reactor dumps lower level objective states necessary to reach high level goals. The domain model represents the physical interaction between the different state variables, and the problem defines the states at the start time, and a set of desired states that must be accomplished at the end of the execution.

A small fragment of the domain is shown in fig. 6. There are two transitions defined, one for the high level deliberator and the other related to the timeline that represents the evolution of the rover position. Both specify the relationship between the current state of the timeline and the requirements to switch to another state. In the first case, there is the definition for a picture acquisition controlled by the *MissionTimeline* timeline. As required by the problem defined previously, the rover must be in a particular location, determined by its coordinate pair, `(?x1, ?y1)` and the pan-tilt unit must be pointing with a desired pair of angles, `(?pan1, ?tilt1)` using the variables expressed in fig. 6. The picture acquired is stored using the `?file_id1` identifier. To express the relationship and the changes triggered when the *MissionTimeline* starts the execution of this transition, there is the definition of the state required in the other timelines: `cd1` implies that the timeline in charge of the camera must change to acquire a picture in the desired position and pointing; and `cd5` requires from the communication subsystem to transmit that picture to the station. When these two subgoals are achieved by their respective timelines, the high level timeline, *MissionTimeline*, go back to the idle state, as expressed in `cd2`. Obviously, prior to transmit the picture, it must be taken; this relationship is declared using `cd1 BEFORE [0,+INF] cd5`: when `cd1` is completed, in an interval that starts immediately after the conclusion of `cd1`

```
(define (problem RoverDemoProb)
 (:domain RoverDemo)
 (:objects
     C0_0 C6_0 C5_-5 - loc
     NavCam - cam  lowRes - mode
     plat0_0 plat15_30 - platpos
     laser stereo - navmode
     dala - rover
 )
 (:init
  ;LOCATIONS INTERCONNECTION
  (= (distance_to_move C0_0 C6_0) 10)
  (= (distance_to_move C6_0 C0_0) 10)
  (= (distance_to_move C0_0 C5_-5) 8)
  (= (distance_to_move C5_-5 C0_0) 8)
  (= (distance_to_move C5_-5 C6_0) 6.4)
  (= (distance_to_move C6_0 C5_-5) 6.4)
  ;ROVER DATA AND CONFIG
  (position dala C0_0)
  (= (energy dala) 1.44)
  (has_locomotion dala)
  (navigation_mode dala laser)
  (= (speed dala laser) 0.2)
  (= (speed dala stereo) 0.1)
  (= (power_per_m dala laser) 0.002)
  (= (power_per_m dala stereo) 0.034)
  (camera_mode dala NavCam lowRes)
  (= (camera_energy NavCam lowRes) 0.008)
  (= (time_to_picture NavCam lowRes) 15)
  (platine_pos NavCam plat0_0)
  (= (platine_energy) 0.003)
  (= (time_move_plat plat0_0 plat15_30) 10)
  (= (time_move_plat plat15_30 plat0_0) 7)
 )
(:goal (and
 (preference PIC1
   (picture C6_0 lowRes plat15_30))
 (preference PIC2
   (picture C5_-5 lowRes plat15_30))
 (preference POSF (position dala C0_0)) )
)
(:metric minimize (+
 (*(is-violated PIC1) 100)
 (*(is-violated PIC2) 100)
 (*(is-violated POSF) 900)
 (total-time) ))
)
```

Figure 5: PDDL problem for the rover example.

and infinity, `cd5` can occur. The `TakingPicture` state in the *camera* timeline implies also a decomposition of states to other timelines prior to the picture acquisition: first, the rover must be at the desired location, then the pan-tilt unit needs to be pointed to a specific angle and, finally, the picture can be taken.

The other definition presented here is the one that controls the behavior of the rover movement. In this case, is related to the movement over flat surfaces using the laser navigation mode. So, there is a check condition (`cd3`) to ensure that the terrain is flat prior to move using the laser navigation. Also, as specified in the domain definition, the pan-tilt

```
SYNCHRONIZE MissionTimeline.mission_timeline {
 VALUE TakePicture(?file_id1, ?x1, ?y1, ?pan1, ?tilt1) {
   cd1 Camera.camera.TakingPicture(?file_id2, ?x2, ?y2, ?pan2, ?tilt2);
   cd5 Communication.communication.Communicating(?file_id3);
   cd2 MissionTimeline.mission_timeline.Idle();
   CONTAINS [0,+INF] [0,+INF] cd1;
   CONTAINS [0,+INF] [2,2] cd5;
   MEETS cd2;
   cd1 BEFORE [0, +INF] cd5;
   ?x1 = ?x2;
   ?y1 = ?y2;
   ?pan1 = ?pan2;
   ?tilt1 = ?tilt2;
   ?file_id1 = ?file_id2;
   ?file_id1 = ?file_id3;
} }
SYNCHRONIZE RobotBase.robot_base {
 VALUE GoingToLaser(?x1, ?y1) {
   cd2 Platine.platine.PointingAt(?pan1 = 0, ?tilt1 = 0);
   cd3 Terrain.terrain.FlatTerrain();
   DURING [0, +INF] [0, +INF] cd2;
   DURING [0, +INF] [0, +INF] cd3;
} }
```

Figure 6: Fragment of the DDL domain definition.

```
PROBLEM RoverDemoProb (DOMAIN RoverDemo) {
mtl0 <fact> MissionTimeline.mission_timeline.Idle() AT [0,0] [1,+INF][1,+INF];
comvw1 <fact> Communication.communication_windows.Visible() AT [70,70][90,90][20,20];
comvw2 <fact> Communication.communication_windows.Visible() AT [150,150][180,180][30,30];
pos0 <fact> RobotBase.robot_base.At(?x1=0, ?y1=0) AT [0,0] [1,+INF] [1,+INF];
or0 <fact> Platine.platine.PointingAt(?pan1=0, ?tilt1=0) AT [0,0][1,+INF][1,+INF];
cam0 <fact> Camera.camera.CamIdle() AT [0,0][1,+INF][1,+INF];
com0 <fact> Communication.communication.CommIdle() AT [0,0][1,+INF][1,+INF];
tp1 <goal> MissionTimeline.mission_timeline.TakePicture
    (?gfile_id2=1, ?gx2=6, ?gy2=0, ?gpan2=15, ?gtilt2=30) AT [10,10][80,90][70,80];
tp2 <goal> MissionTimeline.mission_timeline.TakePicture
    (?gfile_id1=2, ?gx1=5, ?gy1=-5, ?gpan1=15, ?gtilt1=30) AT [80,95][160,185][80,90];
gp <goal> MissionTimeline.mission_timeline.At(?gx3=0, ?gy3=0) AT[160,190][161,+INF][1,+INF];
}
```

Figure 7: Problem definition for the rover example coded in PDL.

unit must be pointing at $(0, 0)$ in order to proceed to move. This is expressed in the condition `cd3`, and if the observation from the timeline that manages the pan-tilt unit shows that the orientation is not the required one, a transition is injected in that timeline prior to the navigation transition. These two conditions must be satisfied during all the time that the `GoingToLaser` state is active.

The problem definition expressed in the Problem Definition Language (PDL) for the example case we are dealing with is shown[3] in fig. 7. This, as well as the PDDL model, establishes the initial state of the world and the goals. To define the initial state is required to set the state of all the timelines employed. For the high level deliberator there are the *MissionTimeline* and *Communication* timelines. First one starts in an idle state, and the other sets the temporal intervals in which there is a communication window available to transmit the pictures, for the example there are two windows, `comvw1` and `comvw2`. The rest of the time is assumed that there is not visibility with the station. For the other rover subsystems there are four timelines that control the rover position, the camera, pan-tilt unit and the communication module. Camera and communication timelines start in idle state, position of the rover is set to the start location $(0, 0)$ and the pan-tilt unit is oriented to the front. For each initial fact (and goal) there are three time intervals: start, end and duration. Every interval is defined by the minimum and maximum time, that is, the transition must start after the minimum time and before the maximum. This applies to the start and end intervals. The duration specifies the minimal and

[3] Although we present the initial facts of all timelines, these values are taken from the state of the functional modules in the GOAC system, they are presented here only to provide an initial state of the system. To work with GOAC there is only required the goal specification and the evolution of uncontrollable timelines, that is, the ones that the system cannot modify, such as the timeline which specifies the visibility windows.

maximal time that the transition can take.

The goal state is declared as a set of states that must be satisfied in a defined temporal interval and, if necessary, in a particular order. Here are defined three objective states, `tp1` and `tp2` and `gp`. First two define the state of acquiring a picture using the same angles for the pan-tilt unit, but in different locations. The constraints defined are that the picture 1, `tp1`, must be taken and transmitted before the second one, `tp2`, and the goal position, `gp`, must be reached after both pictures are taken. The precedence order is provided by the start interval of each goal.

Brief analysis of deliberative models

We are not interested in discussing the optimality of the solution or the time spent to get it, because both architectures use different approaches, and the unique way to obtain a real comparison is to make a hard test-bench and to monitor all the relevant data: energy consumption, required time, etc. Particularly, we are going to push the focus on the flexibility of the deliberation process, which not only resides in the planning/scheduling algorithm, due to the restrictions added by the language employed. In this way, a flexible deliberator must manage system failures and unpredictable elements in the environment such as obstacles in the path or opportunistic science situations. Also, the possibility of including goals during the plan execution is a desirable competence on those systems.

Using the previous models for the action-oriented planning using PDDL and SGplan planner; and the timelines approach with DDL and T-REX/APSI deliberative reactors, we have obtained the plans presented in figures 8 and 9 respectively.

First, we need to analyze how both planners obtain a solution: SGplan (and PDDL-based planners in general) takes a domain and a problem and performs a planning process that achieves all the goals, and then, it returns the sequence of actions to reach the goal state. In the opposite side, APSI timeline-based planner deliberates over a temporal horizon, that is, it gives a partial plan that grows during its execution. Reasoning about that, allows us to see what happens if there is an unexpected situation, for example, if the rover detects a change in the terrain while moving in laser mode and thus, the navigation mode must change to stereo navigation mode to continue:

- The plan obtained with the PDDL model is fixed, so, if the movement action cannot continue in the nominal way, the executor system must manage the situation. In that case, the only way is updating the data of the problem and requesting the planner a new plan. This implies some modifications: first, the necessity of changing the rover data, that is, the current position and energy. But the problem defines only the relevant locations for the initial problem, so the new location needs to be added to the objects and the connections between locations, as well as the destination of the unsuccessful move. Finally, if there are goals already reached, they must be erased from the problem. But, what happens with the navigation mode? We can change the initial state of the navigation mode, but then, the rest of the navigation is performed using the stereo mode, so, when the functional layer triggers another change in the terrain, this process must be repeated.
- The GOAC deliberative reactors contains a set of timelines that control the state of the different subsystems. The current location is managed by a specific timeline (*RobotBase* in fig. 9) that checks another timeline which contains the state of the terrain (*Terrain* in the figure). This state could be flat or rough. When the rover detects the terrain change, there is an observation that modifies the state of that timeline to rough. Considering that *RobotBase* is `GoingToLaser`, as previously explain, this transition required that the *Terrain* is permanently in flat state. So, the transition currently executing in the *RobotBase* timeline cannot continue. But there is another transition defined in that timeline, `GoingToStereo`, that employs the stereo navigation mode. *RobotBase* changes its state to that one, and continues moving to the desired location. In that case, there is no problem to change between laser and stereo navigation modes, the planner can change it dynamically based on the observation of the state of the timelines, which are continuously updated. Also, if the change in the navigation mode does not exceed the time to reach the destination, there is no necessity to perform other changes in high level timelines.

We found that the plan obtained for the MoBAr architecture is a sequence of actions with a fixed duration. What happens if an action takes more or less time to execute? Using a PDDL model, we cannot deal with a flexible model of time, so the only possibility is to use the worst time case for the actions and to delegate the control of the time to the executor. As an example of this problem, we can mention the visibility windows to transmit the pictures. In PDDL we cannot define a specific time in which the rover can transmit the data acquired because the language does not support this type of constraints. It is possible to use a fluent to control the time but it is not practical: it requires to include more complexity to all actions, expressing the evolution of the time manually (the planner does not manage it as it does with the `duration` variable) and to define the communication windows in the problem in terms of start time and end time. However, our experience with this solution do not give good results, if the communication intervals are smaller, the planner does not provide a solution, although it really exits. In contrast, the timelines have a flexible time behavior, when a state transition finishes another transition waiting for the last one can begin. In the same way, we can define the interval in which there are visibility windows, so the transmission of the pictures can be performed in the correct time.

For the case of the goal injection during the execution, a typical situation for exploration missions is to exploit unexpected science targets. We found the same problem as when the rover gets stuck for the PDDL model: we need to modify the initial state and the goals of the problem and perform a replanning to obtain a new plan that includes the new goal. For the timelines approach, a new goal implies the inclusion of the new state and a dynamically replanning process, propagating new subgoals to the different timelines to reach the

```
  0.001: (MOVE_TO  C0_0 C6_0 LASER) [50.0000]
 50.002: (MOVEPLATINE NAVCAM PLAT0_0 PLAT15_30) [10.0000]
 60.003: (TAKEPICTURE C6_0 NAVCAM PLAT15_30 LOWRES) [15.0000]
 75.004: (MOVEPLATINE NAVCAM PLAT15_30 PLAT0_0) [7.0000]
 82.005: (MOVE_TO C6_0 C5_-5 LASER) [32.0000]
114.006: (MOVEPLATINE NAVCAM PLAT0_0 PLAT15_30) [10.0000]
124.007: (TAKEPICTURE C5_-5 NAVCAM PLAT15_30 LOWRES) [15.0000]
139.008: (MOVEPLATINE NAVCAM PLAT15_30 PLAT0_0) [7.0000]
146.009: (MOVE_TO C5_-5 C0_0 LASER) [36.0000]
```

Figure 8: Solution obtained by SGplan for the rover PDDL model.

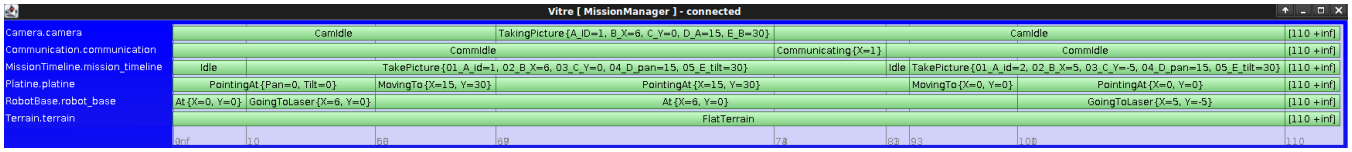


Figure 9: Part of the solution obtained by a particular instance of GOAC using the DDL model previously presented[4].

new objective.

Another difference that appears is the semantic of these approaches: in the case of the PDDL language we need to define all the objects in the environment, what makes it hard to work with continuous values, such as the position of the rover or the orientation angles for the pan-tilt unit. However, DDL is designed to work in continuous domains, defining the interval in which the coordinates or the pan-tilt unit can work, so there is no necessity of defining objects, only set the initial value for the relevant variables. This is more natural than the definition of objects in PDDL, in which we cannot use numerical values.

Finally, an important question in these systems are the resources. For the unitary reusable resources such as a camera or a sample collection tool, both systems could manage the situation in a good manner. In PDDL is easy to define a predicate that indicates when a resource is busy and for the timelines approach, the timeline in charge of the resource expresses when the resource is working or idle, being visible to other timelines that could require it. But for cumulative resources the problem is a more complex question and is not solved yet. Focusing on the rover energy, for the DDL models there is no presence of it, but some work is being carried out in this topic (Diaz et al. 2011; 2012). Meanwhile, the PDDL allows us to define the energy as a fluent, which is possible to consume or renew it within the actions. Model it is not easy and, as is the case of the time, the values are fixed and only allows to represent maximum energy consumption allowed for each action. Also it is possible to maintain the instantaneous power below a threshold using another fluent (here we have not done it). With the PDDL model employed for the battery we can ensure that the plan obtained does not spend more energy than the available.

[4]The image is acquired using the Vitre reactor included in T-REX. This reactor shows the timelines evolution for the deliberative reactors.

Conclusions

This paper has presented the deliberator models for a rover case study for two architectures focused on autonomy: MoBAr and the GOAC architecture, result of the effort of multiple institutions under the ESA supervision.

On the one hand, we have a PDDL-based planner and its corresponding model. The propositional logic is easy to understand and allows us to model basic domains and problems that are action-oriented. The inherent problems are both, the fixed treatment of the time and the static solution provided by the planner. On the other hand, we have deliberators that implement the first order logic in a timelines based approach, which can use the DDL language to model the world. Models are quite complex, but allows us to work with time and continuous variables, giving us more realistic representations of the physics interactions that occur in the environment. Also, dealing with time allows the planning algorithm to deliberate over a dynamic temporal horizon, being more robust to unexpected changes in the nominal conditions of the environment.

So, we can conclude that the deliberative layer of the GOAC system (the couple T-REX/APSI timeline-based planner) give us a better way to model the complex behavior of a robot and its interactions with the environment than the MoBAr deliberator (SGplan a PDDL-based and action-oriented planner) in terms of flexibility of the deliberation process and the semantic employed by their respective modeling languages.

Acknowledgments. Pablo Muñoz is supported by the European Space Agency (ESA) under the Networking and Partnering Initiative (NPI) *Cooperative systems for autonomous exploration missions*. This work was partially supported by the Spanish CDTI project COLSUVH. Authors want to thank Dr. Andrea Orlandini and Dr. Amedeo Cesta for their support with the GOAC architecture.

References

Aschwanden, P.; Baskaran, V.; Bernardini, S.; Fry, C.; R-Moreno, M. D.; Muscettola, N.; Plaunt, C.; Rijsman, D.; and Tompkins, P. 2006. Model-unified planning and execution for distributed autonomous system control. In *AAAI 2006 Fall Symposia*.

Basu, A.; Bozga, M.; and Sifakis, J. 2006. Modeling hterogeneous real-time components in BIP. In *4th IEEE Int. Conference on Software Engineering and Formal Methods*.

Ceballos, A.; Bensalem, S.; Cesta, A.; Silva, L. D.; Fratini, S.; Ingrand, F.; Ocón, J.; Orlandini, A.; Rajan, F. P. K.; Rasconi, R.; and Winnendael, M. V. 2011. A goal-oriented autonomous controller for space exploration. In *ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Cesta, A., and Oddi, A. 1996. *DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains*. Amsterdam: IOS Press.

Diaz, D.; R-Moreno, M. D.; Cesta, A.; Oddi, A.; and Rasconi, R. 2011. Toward a csp-based approach for energy management in rovers. In *4th IEEE International Conference on Space Mission Challenges for information technology (SMC-IT 2011)*.

Diaz, D.; R-Moreno, M. D.; Cesta, A.; Oddi, A.; and Rasconi, R. 2012. An integrated constraint-based, power aware control system for autonomous rover mission operations. In *i-SAIRAS 12 - International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *AI Research* 20:61–124.

Fratini, S.; Cesta, A.; Rasconi, R.; and Benedictis, R. D. 2011. APSI-based deliberation in goal oriented autonomous controllers. In *ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.

Gat, E. 1998. Three-layer architectures. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *Mobile Robots and Artificial Intelligence*, 195–210. AAAI Press.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. In *The Language of the Fifth International Planning Competition*.

Hsu, C., and Wah, B. 2008. The SGPlan planning system in IPC-6. In *Sixth International Planning Competition*.

Mallet, A.; Pasteur, C.; Herrb, M.; Lemaignan, S.; and Ingrand, F. 2010. GenoM3: Building middleware-independent robotic components. In *2010 IEEE International Conference on Robotics and Automation*.

Mallet, A.; Fleury, S.; and Bruyninckx, H. 2002. A specification of generic robotics software components: future evolutions of GenoM in the orocos context. In *International Conference on Intelligent Robotics and Systems*.

McDermott, D. 1998. The PDDL planning domain definition language. *The AIPS-98 Planning Competition Comitee*.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2007. T-REX: A model-based architecture for auv control. In *3rd Workshop on Planning and Plan Execution for Real-World Systems (ICAPS07)*.

Muñoz, P.; R-Moreno, M. D.; and Martínez, A. 2011. A first approach for the autonomy of the exomars rover using a 3-tier architecture. In *ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

R-Moreno, M. D.; Cesta, A.; and Kurien, J. 2008. Innovative AI technologies for future esa missions. In *ASTRA 2008 - 10th Symposium on Advanced Space Technologies in Robotics and Automation*.

Rajan, K.; Py, F.; McGann, C.; Ryan, J.; Reilly, T. O.; Maughan, T.; and Roman, B. 2009. Onboard adaptive control of AUVs using automated planning and execution. In *International Symposium on Unmanned Untethered Submersible Technology*.

Verma, V.; Jnsson, A.; Pasareanu, C.; and Iatauro, M. 2006. Universal Executive and PLEXIL: Engine and language for robust spacecraft control and operations. In *American Institute of Aeronautics and Astronautics Space Conference*.

Delegating Geometric Reasoning to the Task Planner

Fabien Lagriffoul
AASS Cognitive Robotic Systems Lab
Örebro University, S-70182 Örebro, Sweden
fabien.lagriffoul@aass.oru.se

Abstract

Different architectures for combining task and motion planning (CTAMP) have been proposed. They differ in the way the symbolic and geometric layers interact with each other, but all of them use the symbolic layer to reason about causal and/or temporal relationship between actions, while geometric reasoning is entirely delegated to the geometric layer. We argue that the domain of humanoid robot manipulation has specific features which allow to delegate a part of the geometric reasoning to the task planer, by enriching the planning domain with coarse geometric representations. With this approach, the task planner generates alternative sequences of actions which lead to the same geometric result. A heuristic is required to choose the right sequence of actions, because some sequences are much more costly than others in terms of motion planning. We report a failed attempt in designing such a heuristic, and propose an alternative solution based on local search.

Introduction

Specifying high-level goals for robotic systems has long been a goal in robotic research. The first attempts in this direction were based on hierarchical architectures, with typically three levels of abstraction: a deliberative layer on top, a path planning layer, and a control layer for execution. As the ability of robotic platforms to manipulate the environment increased, these architectures became obsolete, because the deliberative layer could not reason about the consequences of abstract actions in the real world. As a result, a plan which is logically sound may fail when the robot executes it. To address this problem, a new paradigm emerged, in which task and motion planning are not decoupled, but combined. Recently, several approaches for CTAMP were proposed. In SamplSGD (Plaku and Hager 2010), the planner mainly works on a path planning problem, but a symbolic interpretation of the domain is used to structure the path planning problem and determine where to direct the search. Similarly in aSyMov (Cambon, Alami, and Gravot 2009), the planner alternates between symbolic search and geometric search according to cost functions which determine if it is worth investing time in motion planning or exploring alternative symbolic actions. In another type of approaches, the task planner is steering the search, while dedicated geometric reasoners are called to geometrically evaluate the preconditions and effects of symbolic actions. These approaches include (Guitton and Farges 2009), SAHTN (Wolfe, Marthi, and Russell 2010), semantic attachments (Dornhege et al. 2009), and (Kaelbling and Lozano-Perez 2010).

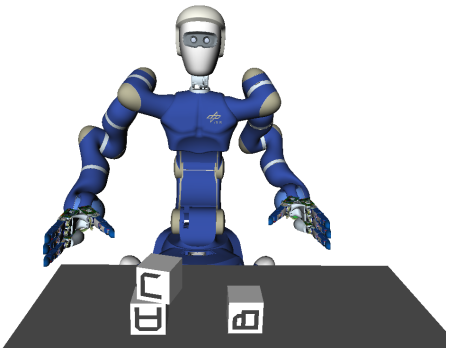


Figure 1: The blocks-world domain implemented on a "real" robot (a simulation of Justin ((Ott and al. 2006) was used.

However, the work cited above is based on so called *mobile* manipulation scenarios, i.e. a mobile robot (often a forklift robot) performs manipulation tasks in a static environment (except Asymov, which can deal with several robots). Working with more complex platforms, like *humanoid robots* (see Fig. 1) changes the nature of the problem. Essentially, geometric reasoning becomes non-trivial, which makes its integration in the symbolic/geometric loop problematic. For a mobile robot for instance, the symbolic action `place object1 kitchen` results in a call to a path planner which computes a path from the current robot position to the kitchen. In the case of a humanoid robot, the symbolic action `place object1 table` cannot be directly mapped to a single motion planning query. This was early pointed out in the pionner work done on the robotic platform *Handey* (Lozano-Pérez et al. 1989). Their work shows that picking and placing an object in a desired pose often requires to decompose the problem into several re-grasp operations. In order to deal with this, an ad-hoc *re-grasp planner* (Tournassoud, Lozano-Perez, and Mazer 1987) had to be called when the action could not be achieved with a single movement. More recently, probabilistic roadmaps(PRM)-based methods have been developed which can solve pick-and place problems for a *single* object (Simeon et al. 2002). Dealing with several objects is a more complex problem called *navigation among movable obstacles* (NAMO), and requires specific techniques (Stil-

man and Kuffner 2006).

Another major difference between the two domains is that a mobile robot performs its task *within* a workspace, whereas the humanoid robot *itself* is part of the workspace. Each arm of the robot is potentially an obstacle for the movements of the other arm. Collisions between arms can be avoided by using a motion planner which works in the combined configuration space of both arms, but the computational cost of such planner is higher, and the resulting motions often look unnatural. In addition to this, some tasks may require dual-arm regrasping, which is also computationally demanding, see for instance (Vahrenkamp et al. 2009), (Saut et al. 2010).

In summary, working with a humanoid robot raises two additional difficulties compared to a mobile robot:

- some actions have to be decomposed into sequences of re-grasping operations by a dedicated planner;
- motion planning with two manipulators is generally more prohibitive.

This is problematic in CTAMP, because geometric reasoning is repeatedly invoked during the planning process. Next, we describe our approach for combining task and motion planning, which naturally works around these problems: the task planner is used to plan for re-grasping operations, and to split actions involving two arms into sequences of single-arm motions. Then, we report on some failed attempts to design heuristics for such a system, and sketch a possible solution based on balancing the search effort between the task level and the geometric level.

Our approach

We implemented a symbolic planning domain which, given a high-level problem description, breaks down the task into a sequence of pick and place operations. We used the hierarchical task network (HTN) planner JShop2 (Nau et al. 2001) to generate these plans. HTN planning allows to write methods to guide task decomposition. We implemented methods which handle the cases where re-grasping is required, hence no specialized re-grasp planner is needed. Dual-arm re-grasping is also handled by dedicated methods, hence no dual-arm re-grasp planner is needed. The task is decomposed into actions where one manipulator is moved at a time, hence only 7-degrees of freedom (DOF) motion planner is needed (see (Zacharias et al. 2010), where the authors show that with a simple strategy, using independent motion planners for each arm (2×7 DOF) is more effective than using a 14-DOF dual-arm motion planner).

In summary, we propose an approach to CTAMP in which the task planner is also used to break down difficult motion planning problems into simpler sub-problems. This reduces the complexity of motion planning, but increases the complexity at the task level, because these sub-problems can be combined in multiple ways. Since the task planner does not take into account geometric constraints (robot geometry, collisions), some action sequences found by the task planner are not executable on the real robot, or lead to complex motion planning problems, e.g., narrow passages, or intricate arm positions. Hence, a heuristic is needed to choose among the possible task decompositions, which ones are the easiest to implement geometrically. Next, we describe the planning domain used, and then report some attempts in designing such a heuristic.

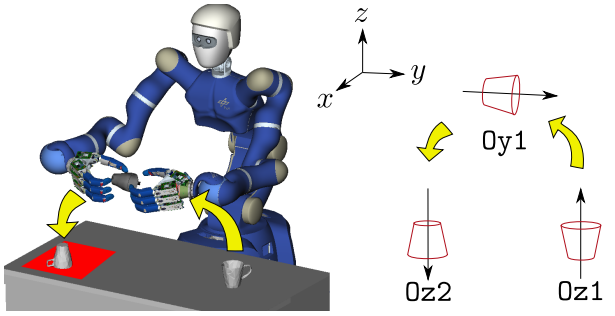


Figure 2: Example of action sequence, with the corresponding coarse object orientations used by the task planner.

Description of the planning domain

As test bed for our approach, we chose a real world version of the well known AI benchmark domain *blocks-world* with three blocks (see Fig. 1). The problem is similar (building a pile with A on top of B, and B on top of C), but additional geometric constraints have to be considered: the blocks have to be in upright position and with the same orientation. In this section, a glimpse of the planning domain is given rather than an exhaustive description.

Operators

There exists different implementations for the blocks-world domain. The point to stress is that in the original domain, the `pick` and `stack` operators were only parametrized by the object to be moved, and by the underlying object. In order to reason about geometry, more parameters are used in our domain. In particular, we use a parameter which we call *coarse_object_orientation*, which represents the alignment of an object with a reference axis in the real world. For instance, we use the symbols `Ox1`, `Oy1`, `Oz1` to represent the fact that an object has its main axis aligned with respectively the x, y, z axes of the world frame, and `Ox2`, `Oy2`, `Oz2` denote alignment with the opposite axes $-x, -y, -z$ (see Fig. 2). For instance, the predicate:

```
is_oriented object1 z2
```

represents the fact that `object1` in upside-down position. We define the following parameters for the actions:

A `pick`-like action (`pick`, `pick-regrasp`) is parametrized by:

- *side*: `left`, `right`;
- *grasp*: `side`, `top`, `bottom` ;
- current *coarse_object_orientation*: `Ox1`, `Ox2`, `Oy1`, ...
- *object*.

A `place`-like action (`place`, `place-regrasp`, `stack`) is parametrized by:

- *side*: `left`, `right`;
- *grasp*: `side`, `top`, `bottom` ;
- *target_location*;
- target *coarse_object_orientation*: `Ox1`, `Ox2`, `Oy1`, ...
- *object*.

To illustrate this with an example, consider the sequence of actions depicted in Fig. 2, where the robot grasps a cup with the left manipulator, re-grasps it with the right manipulator, and places it on the tray. The corresponding symbolic action sequence with our action parametrization scheme is:

```
(!pick left top Oz1 cup1)
(!place_regrasp left top Oy1 cup1)
(!pick_regrasp right bottom Oy1 cup1)
(!place right bottom tray Oz2 cup1).
```

The task planner is not aware of the exact orientation of objects: it only knows if an object is aligned with the x, y, or z axis, and in which direction. Similarly for grasps, the task planner reasons about the type of grasp without knowing the exact orientation of the tool center point (TCP) relative to the object. The exact angular values are handled at the geometric level.

Predicates

Using coarse object orientations together with specific predicates in the preconditions of operators allows for basic geometric reasoning. For instance, the operator `!pick` has the following preconditions (among others):

```
(graspable ?obj ?grasp)
(is_oriented ?obj ?axis)
(allow_pick ?axis ?grasp ?side) .
```

The predicate `graspable` indicate which types of grasp can be performed on different objects, e.g., the blocks used in our experiments can be grasped by top and bottom, but a side-grasp is not feasible because the hand of the robot is too large, so it would collide with the table for such grasps. The predicate `allow_pick` represents commonsense knowledge about actions which are not feasible. For instance, `(allow_pick z1 top left)` means that when an object is in upright position, it can be grasped with a top-grasp with the left arm, which implicitly means that a bottom grasp is not possible. Similarly, the `!place` action has a predicate `allow_place` in its preconditions.

For re-grasping operations, similar predicates are used to determine which operations can be done or not. For instance, the re-grasp operation depicted in Fig. 2 is allowed because the operator `!pick_regrasp` has the preconditions:

```
(graspable ?obj ?grasp)
(is_oriented ?obj ?axis)
(allow_regrasp ?axis ?grasp ?side) ,
```

and because the following predicates are defined:

```
(allow_regrasp y1 top left)
(allow_regrasp y1 bottom right) .
```

Note that when the object is in this position, it may in principle be feasible to use a bottom-grasp with the left hand, and a top-grasp with the right hand. However, this would require a very intricate position of the arms, which is likely to be infeasible in practice. Hence, these predicates filter out actions which could in any case not be geometrically feasible. The following predicates are also used for re-grasping operations:

```
(opposite_side right left)
(opposite_side left right) .
```

HTN methods

The methods are used to decompose an abstract task into a sequence of actions. In this domain, a common abstract task is to *move* an object to a desired location. A simple way of decomposing the abstract task `move ?obj ?loc` is the sequence:

```
(!pick ?side ?grasp ?axis ?obj)
(!place ?side ?grasp ?loc ?axis ?obj) .
```

Note that the axis is preserved. We need alternative methods to decompose the *move* task, because this one may not be feasible in all cases. Imagine for example that the location and the object are not both reachable by the same arm. Then the object has to be placed in a temporary location to achieve the task:

```
(!pick ?side ?grasp1 ?axis ?obj)
(!place ?side ?grasp1 ?temp_loc ?temp_axis
?obj)
(!move_arm_away ?side)
(!pick ?o_side ?grasp2 ?temp_axis ?obj)
(!place ?o_side ?grasp2 ?location
?final_axis ?obj) .
```

The action `!move_arm_away` is used so that the second `!pick` action is not hindered by the presence of the hand, which remains above the object after the first `!place` action. The method has a precondition `(opposite_side ?side ?o_side)` to ensure that both arms are used consistently. Note that the type of grasp can be changed with this method. However, if the object can only be grasped by the bottom or the top, these two methods do not allow us to flip an object. In order to do this, dual-arm re-grasping is needed. We also defined a method that decomposes the *move* task in this way:

```
(!pick ?side ?grasp1 ?axis1 ?obj)
(!place_regrasp ?side ?grasp1 ?axis2 ?obj
(!pick_regrasp ?o_side ?grasp2 ?axis2 ?obj
(!place ?o_side ?grasp2 ?location ?axis3
?obj) .
```

The actions `!pick_regrasp` and `!place_regrasp` are similar to `!pick` and `!place`, but without a location. Depending on the current orientation `?axis1` of the object, and according to the types of grasp allowed for this object, different sequences can be produced which place the object with orientation `?axis3`.

System architecture

A simple architecture was developed, in which task and motion planning are decoupled, i.e., a symbolic plan is generated first, and geometrically evaluated afterward (see Fig. 3). We decoupled task planning and geometric reasoning because we wanted to sort the symbolic plans *before* evaluating them with the geometric reasoner. This approach turned out to fail, we report on this at the end of the section. Ignoring the sorting step, the system simply consists in generating a list of symbolic plans, and evaluating them one after another with the geometric reasoner until one of them succeeds. The role of the filtering step is to reject the plans that are not feasible, e.g., because an object is out of reach by the robot. All the generated plans achieve the same geometric goal, but they differ with respect to the choice of

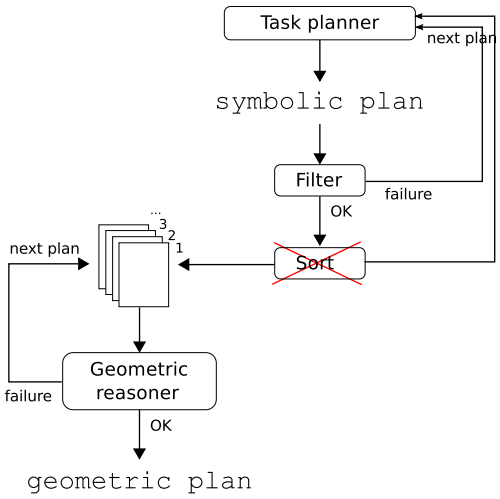


Figure 3: The task planner generates symbolic plans which are geometrically evaluated by the geometric reasoner. Sorting the plans was abandoned.

arms, types of grasp, coarse orientations, and re-grasping operations.

The geometric reasoner is a complex component that takes in input a symbolic plan, and returns (if possible) a sequence of motion paths. The details of this component can be found in (Lagriffoul et al. 2012). In short, each action needs to be converted into a motion path. The final configuration for the motion path is found by sampling different possibilities. For a `!pick` action, several orientations of the TCP are sampled, for a `!place` action, several positions for the object are sampled. When an arm configuration achieving the action is found, the motion planner is called. If no path is found, another sample position is tried. If all the samples fail, previous actions have to be reconsidered, because they may be the cause of the failure. We refer to this procedure as *geometric backtracking* (see (Karlsson et al. 2012)). Geometric backtracking is computationally expensive compared to task planning, because the motion planner is called at each backtrack. By comparison, finding one hundred symbolic plans takes a few milliseconds, while evaluating one single plan geometrically takes several seconds.

A failed attempt for sorting plans

Since geometric reasoning takes time, and since several symbolic plans need to be evaluated before one is found that works, the first idea that comes to mind is to sort the plans according to some criteria, so that the ones which are more likely to fail are evaluated last. According to our experience, a plan is generally rejected by the geometric reasoner because a motion path cannot be found for one of the actions. This occurs when the scene is cluttered, or if an action requires an arm to be close to its joint limits. It can also occur if the symbolic plan assigns the left arm to an object situated on the right and vice versa, which causes one of the arms to be "locked" by the other. We implemented a simple procedure that computes a geometric configuration for each symbolic action, ignoring motion planning. Then, for each geometric configuration, we computed the following criteria, which aimed at measuring the level of intricacy between the objects and the robot:

- distances between the links of left and right arm;
- distance between left and right TCP;
- distances from the joint limits;
- distances between all objects and TCPs;
- "crossing" of the arms.

We ran hundreds of experiments with different symbolic plans and different initial positions of the objects. For each run, the list of geometric configurations was computed, and the average value and minimum value of these criteria were evaluated over the list. Then, the geometric reasoner was called and the running time was measured. The statistical analysis of the data showed that correlations between these criteria and the time needed for geometric reasoning exist, but the correlation is *too weak* for using these criteria as a heuristic or predict which plans are most likely to fail.

This failed attempt demonstrates that it is not possible to evaluate the difficulty of geometric reasoning based on the simple geometric criteria we used. Geometric reasoning is not trivial because of geometric dependencies between actions. Motion planning is often counter-intuitive for humans: some actions which seem trivial are sometimes executed in complicated ways by the robot, and some actions which seem complicated sometimes require a short motion planning time. This suggests a more complex procedure for computing this heuristic, although fast enough so that it does not take more time than geometric reasoning itself. A common technique for designing heuristics is to solve a relaxed version of the problem. We applied this idea to our problem, by using a simplified version of the geometric reasoner, with limited backtracking capacity and lower cutoff time for motion planning. When this simplified geometric reasoner fails, it means that a solution *may* exist but that more time is required to find it. Hence, this is not strictly speaking a heuristic, because it prunes out symbolic plans which may be feasible, but it can be used to jump over infeasible or geometrically difficult symbolic plans, and reach the easy ones. Hence, we traded completeness against the hope to have simpler motion planning problems to solve. The results of our experiments are presented in the next section.

Experiments

In order to test this approach, we defined two modes for the geometric reasoner which correspond to two different parametrization schemes. In the *normal mode*, the geometric reasoner backtracks geometrically until a solution is found, and the motion planner is set a cutoff time of 10 seconds. These are the usual parameters to solve common manipulation tasks. In the *simple mode*, geometric backtracking is limited to one level, and the motion planner cutoff time is set to 1 second, which is insufficient to solve complex problems, but terminates in a reasonable time. Hence, in the *normal mode*, more effort is spent on geometric reasoning, while in the *simple mode*, part of this effort is delegated to the task planner.

Experimental setup

We used two scenarios for the experiments. In the first scenario, only the blocks are on the table (see Fig. 1), while in

the second scenario the space was cluttered with additional objects acting as obstacles (see Fig. 4). In both scenarios, one hundred runs were conducted. The task is the same (stacking the blocks), but the initial poses of objects and obstacles were randomized.

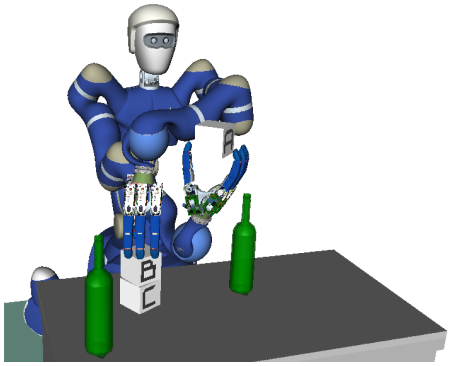


Figure 4: An example of task decomposition which led to a geometrically difficult motion planning problem.

Results

We compared the approach using the *normal mode* and the approach using the *simple mode* for the geometric reasoner. For all experiments, we have measured the total time in seconds needed to reach a solution, i.e., the time spent on pruning out symbolic plans, plus the time needed to compute the actual geometric solution. Out of the 100 runs, 23 were not solved in the first scenario, and 18 in the second scenario. The results are shown in Fig. 5. The runs are ordered according to the time spent using the *normal mode*.

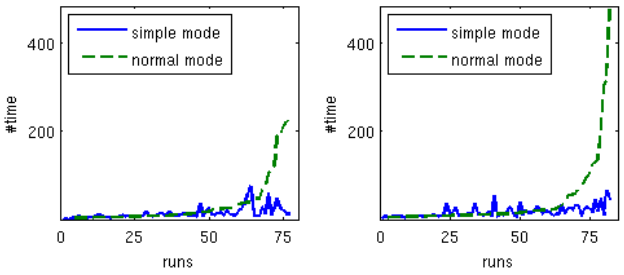


Figure 5: Results (in seconds) for the first scenario (left) and the second scenario (right).

The performance of both approaches is similar for most problems, but in 20% of the cases, when using the *normal mode* is demanding, it is faster to try alternative symbolic plans for which a geometric solution is found more easily. Fig. 4 is a good illustration of a symbolic plan which is difficult to achieve geometrically: after stacking block B on block C, the symbolic plan says that block A should be placed in upright position for a re-grasp with the left arm by the top. This is difficult in terms of motion planning because the left arm and the bottle are in the way. In this case, choosing at the symbolic level a different orientation for re-grasping, or picking block A with the left arm would most probably make the problem geometrically easier.

The *simple mode* evaluates more symbolic plans on average (12.5) than the *normal mode* (7.6), but this pays off because this evaluation is fast. As a result, the average time for solving a problem using the *simple mode* is 15 s, versus 35 s with the *normal mode*. Note also that in complicated problem instances, the *simple mode* remains in its average time, while the time used in *normal mode* reaches several minutes. Regarding completeness, it is clear that the *simple mode* is not complete. It jumps over symbolic plans after limited effort spent at the geometric level, which can be seen as a kind of local search algorithm. However, in the presented experiments, only 2 problems out of 159 were not solved by the *simple mode* which the *normal mode* could solve. Hence, an easy way to cope with this problem is to run *simple mode* first, and *normal mode* in case of failure.

Conclusion

We proposed a way of delegating part of the geometric reasoning to the task planner in a framework combining task and motion planning for a humanoid robot. This was done by enriching the planning domain with coarse geometric representations. With this approach, only simple motion planners are needed, but choosing among symbolic plans which one is the simplest to compute geometrically is not straight-forward. We presented our initial experiments in this direction, and the problems we encountered. We also wanted to point out the differences between the domain of mobile robotics, where heuristics based on distances can be used for robot actions, compared to the domain of humanoid robots, where the complexity of motion planning and the geometric dependencies between actions make the computation of heuristics more tedious.

Acknowledgments

This work was partially supported by EU FP7 project "Generalizing Robot Manipulation Tasks" (GeRT, contract number 248273). We would like to thank in particular Florian Schmidt from the Robotics and Mechatronics Center of DLR, which developed for us functionalities in Justin's simulation environment that made this work possible.

References

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *Int. J. Rob. Res.* 28(1):104–126.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS09)*, 114–122.

Guitton, J., and Farges, J.-L. 2009. Taking into account geometric constraints for task-oriented motion planning. In *Proc. Bridging the gap Between Task And Motion Planning, BTAMP'09 (ICAPS Workshop)*.

Kaelbling, L. P., and Lozano-Perez, T. 2010. Hierarchical planning in the now. In *Proc. of Workshop on Bridging the Gap between Task and Motion Planning (AAAI)*.

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 *Proceedings of TAMPRA: Combining Task and Motion Planning for Real-World Applications (ICAPS workshop)*.

Lagriffoul, F.; Dimitrov, D.; Saffiotti, A.; and Karlsson, L. 2012. Constraint propagation on interval bounds for dealing with geometric backtracking. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 957 –964.

Lozano-Pérez, T.; Jones, J.; Mazer, E.; and O'Donnell, P. A. 1989. Task-level planning of pick-and-place robot motions. *IEEE Computer* 22:21–29.

Nau, D.; Muoz-avila, H.; Cao, Y.; Lotem, A.; and Mitchell, S. 2001. Total-order planning with partially ordered subtasks. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 425–430.

Ott, C., and al. 2006. A humanoid two-arm system for dexterous manipulation. In *2006 IEEE Int. Conf. on Humanoid Robots*, 276–283.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *Proceedings of ICRA10*.

Saut, J.-P.; Gharbi, M.; Cortés, J.; Sidobre, D.; and Siméon, T. 2010. Planning pick-and-place tasks with two-hand regrasping. In *IROS*, 4528–4533.

Simeon; Cortes; Sahbani; and Laumond. 2002. A manipulation planner for pick and place operations under continuous grasps and placements. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 2, 2022 – 2027 vol.2.

Stilman, M., and Kuffner, J. 2006. Planning among movable obstacles with artificial constraints. In *In Workshop on the Algorithmic Foundations of Robotics*, 1–20.

Tournassoud, P.; Lozano-Perez, T.; and Mazer, E. 1987. Regrasping. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, 1924 – 1928.

Vahrenkamp, N.; Berenson, D.; Asfour, T.; Kuffner, J.; and Dillmann, R. 2009. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, 2464–2470. Piscataway, NJ, USA: IEEE Press.

Wolfe, J.; Marthi, B.; and Russell, S. J. 2010. Combined task and motion planning for mobile manipulation. In Brafman, R. I.; Geffner, H.; Hoffmann, J.; and Kautz, H. A., eds., *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS10)*, 254–258.

Zacharias, F.; Leidner, D.; Schmidt, F.; Borst, C.; and Hirzinger, G. 2010. Exploiting structure in two-armed manipulation tasks for humanoid robots. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 5446 –5452.

RealTime GPU-based Motion Planning for Task Execution in Dynamic Environments

Chonhyon Park and Jia Pan and Ming Lin and Dinesh Manocha

***Abstract*— We present a realtime GPU-based motion planning algorithm for robot task executions. Many task execution strategies break down a high-level task planning problem into multiple low-level motion planning problems, and it is essential to solve those problems at interactive rates. In order to achieve high performance for the planning, our method exploits a high number of cores on commodity graphics processors (GPUs). We describe a parallel formulation of a RRT-based motion planning algorithm, which is highly suited for single query motion planning. Our approach uses the properties of Poisson-disk samples to achieve a high parallelism to exploit the computational capabilities of GPUs. Our approach can obtain 10-20X speedup over prior CPU-based motion planning algorithms and we demonstrate the performance on a number of benchmarks.**

I. Introduction

Planning and scheduling techniques are widely used in robotics. The underlying planning problems in robotics are characterized by geometric constraints that affect specification and how the goals are satisfied. For example, before deciding to pick up any objects in an environment, we need to figure out whether it is geometrically feasible for the robot (or its arm) to move to an appropriate location in the environment. At a high level, planning problems can be decomposed into task planning and motion planning. Task level planning refers to high level specification and execution of tasks (e.g. pick up a can). On the other hand, motion planning refers to computing a collision-free trajectory for a real or virtual robot.

Over the last few few years, there is considerable research effort on combined task and motion planning for different applications, including robot navigation, manipulation, search-and-rescue, game design, air-traffic control, and surgical procedures, etc. While some of them deal with discrete motions, others tend to compute a continuous motion trajectory. Most of these planning problems reduce to searching for a solution into the appropriate space. Furthermore, these solutions needs to account for various complex geometries of the obstacles. physical constraints, motion dynamics (e.g. maximum velocity or acceleration that robot joints can take), collision avoidance, and robot-environment interactions. It is important to develop fast planning algorithms that can satisfy these constraints and compute an appropriate solution in realtime in new, uncertain environments.

It turns out that many high-level planning or complex tasks involve repeated calls to motion planning algorithms to perform various subtasks. In many cases, motion planning and subtask execution steps are performed in an interleaved manner, and the result of previous planning steps may be used in the later planning steps. [2]. In this regard, a key challenge is task execution in dynamic or uncertain environments, which boils down to repeated computation of collision-free motion planning. A dynamic environment may have moving obstacles [3] or sensors with uncertainty [4]. In this environment, the robot only has a partial observation of the environment and it is hard to precisely predict the location of all the obstacles in the environment in the future. Therefore, interleaved planning is used to update the environment representation and robot task execution. As shown in Fig. 1, the algorithm repeatedly performs motion planning with the latest sensor information and executes the planning result as a subtask. Many previous works use similar approaches [5], [6] to compute a collision-free trajectory in dynamic environments.

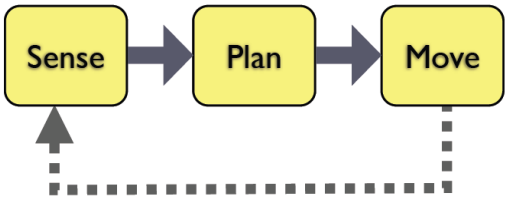


Fig. 1: The task planning repeatedly performs sensing, motion planning and execution steps in an interleaved manner.

Some task planning algorithms decompose complex tasks into multiple subtasks to reduce the complexity of planning problem. [1] The subtasks are arranged in appropriate order by the algorithm and solved in order using motion planning algorithms. A subtask graph is used to formalize the task planner [7]. Complex tasks of humanoid robot can be decomposed into simple motion planning problems of multiple parts [8].

A key challenge in this high-level task planning is to develop fast (almost real-time) techniques for motion planning algorithms, which are invoked multiple times during task executions. Real-time planning makes it possible to compute a collision-free trajectory for robots in a short time and improves the safety of the task planning in dynamic environments.

In this paper, we present a parallel motion planning algorithm that exploits the computational capability of GPUs for single-query motion planning. The real-time nature of our algorithm also makes it possible to handle dynamic obstacles

Chonhyon Park, Jia Pan, Ming Lin, and Dinesh Manocha are with the Department of Computer Science, University of North Carolina at Chapel Hill. E-mail: {chpark, panj, lin, dm}@cs.unc.edu.

or partial sensor data. We use Poisson-disk sampling for node generation, which reduces the generation of redundant nodes. This sampling allows the algorithm to expand a large number of nodes in parallel and exploit the high number of computational cores. Moreover, we describe efficient parallel strategies for RRT planning, including sample generation and tree expansion. We highlight the speedup of the parallel algorithm on different benchmarks, which is a 10-20X improvement over prior CPU-based RRT planning algorithms; we also demonstrate our algorithm's scalability. Our real-time planning algorithm can be used to accelerate high-level task executions.

The remainder of the paper is organized as follows: Section II provides an overview of our approach; we describe the details of parallel algorithms in Section III; in section IV, we highlight our performance on different motion planning benchmarks.

II. Overview

In this section, we describe our GPU-based planning algorithm's underlying framework and how it derives from sampling-based planning algorithms.

A. RRT Planning

The basis or our algorithm is rapidly-exploring random trees (RRT) [9], one of the most widely used planning algorithms used in solving motion-planning problems. Sampling-based algorithms (including RRT) formulate the planning problem as a search problem in the configuration space of a robot. In the configuration space, a configuration of the robot is represented as a point. The configuration space $\mathcal{C}$ has the region $\mathcal{C}_{free}$, where the robot configurations in the region has no collision, and the region $\mathcal{C}_{obs}$, where a collision occurs between the robot and one of the obstacles. The RRT algorithm grows the RRT tree to compute a continuous collision-free path from an initial configuration $\mathbf{x}_{init}$ to a goal configuration $\mathbf{x}_{goal}$.

The RRT tree $\mathbf{T}$ is initialized with the root node of $\mathbf{x}_{init}$, and the algorithm expands the tree incrementally. Each iteration of RRT planning executes two main procedures, `sampling` and `expansion`. The `sampling` procedure generates a new random configuration $\mathbf{x}$, which determines the direction of the tree expansion. The `expansion` procedure includes two steps, 1) nearest node search and 2) local planning. With the configuration $\mathbf{x}$, nearest node search finds a node $\mathbf{v}$ in $\mathbf{T}$, the closest node to $\mathbf{x}$. The local planning step checks whether the shortest path between $\mathbf{v}$ and $\mathbf{x}$ lies in $\mathcal{C}_{free}$, (i.e., the configurations on the path do not collide with the obstacles). If the path is collision-free, $\mathbf{x}$ is added to $\mathbf{T}$ as a new node connected to the node $\mathbf{v}$. If the path has a collision, the collision-free configuration $\mathbf{x}_{new}$ on the path thiat is farthest from $\mathbf{v}$, is added to $\mathbf{T}$ instead of $\mathbf{x}$. In many applications, the local planning procedure checks for collisions from $\mathbf{v}$ to $\mathbf{x}_t$ (which has a distance of Δ from $\mathbf{x}$) if the distance between $\mathbf{v}$ and $\mathbf{x}$ is longer than the problem-specific length Δ. In this approach, the maximum distance of the edge which connects the new node and $\mathbf{T}$ is limited to Δ. The RRT algorithm repeats this iteration until $\mathbf{x}_{goal}$ is added to $\mathbf{T}$, when the collision-free path from $\mathbf{x}_{init}$ to $\mathbf{x}_{goal}$ can be extracted from $\mathbf{T}$.

B. Parallel RRT Planning using Poisson-disk Sampling

The RRT algorithm is efficient for single-query problems, since the algorithm incrementally expands the RRT tree to the unexplored regions and terminates when the solution is found. However, this incremental expansion of the tree means that it is difficult to make an efficient parallel algorithm for planning.

Our algorithm is based on the AND parallel RRT algorithm [10], which adds multiple nodes simultaneously. In an AND parallelization, each thread independently executes `sampling` and `expansion` procedures, and the new nodes are added to a single RRT tree shared by all threads. The AND parallelization can expand the tree faster than the original RRT. However, as the number of threads increases, the algorithm results in more redundant nodes in the RRT tree, degenerating the performance of overall planning.

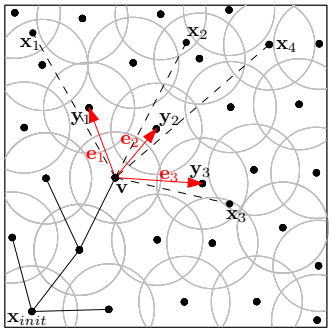


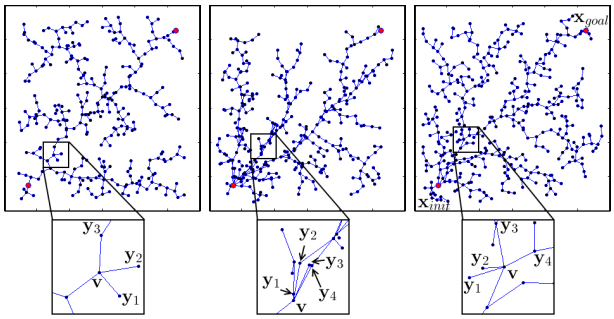
Fig. 2: Parallel tree expansion using 4 threads. The i-th thread expands the tree toward sample $\mathbf{x}_i$, $i = 1, 2, 3, 4$. The vectors $\mathbf{e}_i$ (in red) show the new RRT edges added. Since $\mathbf{x}_2$ and $\mathbf{x}_4$ correspond to the identical Poisson-disk sample ($\mathbf{y}_2$), both of them result in adding the edge $\mathbf{e}_2$ to the tree; only 3 nodes are added and no redundant node is generated.

In order to lessen the overhead caused by the redundant nodes, our algorithm uses precomputed Poisson-disk samples in the tree expansion. In this case, the samples satisfy the free disk property:

$$\forall \mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}, \mathbf{x}_i \neq \mathbf{x}_j : \|\mathbf{x}_i - \mathbf{x}_j\| \geq r, \qquad (1)$$

where $\mathbf{X} = \{\mathbf{x}_i\}$ is set of samples and r is a predefined minimum distance between any of two samples. Unlike the standard RRT, which does local planning between the nearest node $\mathbf{v}$ and the configuration $\mathbf{x}_t$, our algorithm chooses a Poisson-disk sample $\mathbf{x}_{nbr}$ that is closest to $\mathbf{x}$ among $\mathbf{v}$'s neighboring Poisson-disk samples. The free disk property ensures that the chosen sample is at least a minimum distance, denoted here by r, from $\mathbf{v}$. Our approach eliminates the problem of multiple threads of the algorithm choosing the same direction, which would generate redundant nodes that would be too close to each other in the standard RRT tree expansion. In our algorithm, the threads will instead choose the same Poisson-disk sample and stop the redundancy

problem from developing. We add the sample only once to the tree. An example of tree construction in our algorithm is shown in Fig. 2.



(a) RRT (b) Parallel RRT (c) Our Algorithm

Fig. 3: Comparison of RRT trees generated using different planning approaches. (a) The tree corresponding to the original RRT algorithm is generated according to the Voronoi bias of the sequential algorithm. (b) The parallel RRT tree generated by AND parallelism tends to have many redundant nodes that are close to other nodes in the tree (e.g., the new nodes $\mathbf{y}_2$, $\mathbf{y}_3$, and $\mathbf{y}_4$ are close to $\mathbf{y}$). (c) The tree generated with Poisson-disk sampling has fewer redundant nodes due to the free disk property of samples.

Fig. 3 shows the RRT trees generated by an original RRT, an AND parallelization RRT, and our algorithm. The tree generated by AND parallelization has many redundant nodes that are close to other tree nodes, while the tree generated using Poisson-disk sampling has efficiently spaced nodes.

The precomputed Poisson-disk samples may not correspond to a large set of samples that can always find a collision-free solution. As a result, the algorithm performs adaptive Poisson subsampling at runtime to generate more samples with reduced distance amongst the samples.

III. Parallel RRT Planning using GPUs

In this section, we present our planning algorithm's specific components, and explain how this setup efficiently exploits the parallel computational resource of GPUs.

A. Hierarchy Computation

In order to accelerate collision checking, we compute bounding volume hierarchies (BVH) for the robot and the obstacles in the environment. We construct the oriented bounding boxes (OBB) trees [11] for the triangle model representations of the robot and obstacles using a GPU-based construction algorithm [12]. The OBB trees improve the performance of collision checking by using the high culling efficiency of OBBs.

B. Precomputation of Poisson-disk Samples

We use precomputed Poisson-disk samples in tree expansion. Since the samples are generated on the entire configuration space $\mathcal{C}$, including $\mathcal{C}_{obs}$, the precomputation is performed offline, and the generated sample set can be used for all motion planning problems with the same configuration space dimension. Recently, parallel algorithms that use GPUs for fast computation of Poisson-disk sampling have been suggested ([13] and [14]); these algorithms can be used to compute samples for high-dimensional spaces.

C. Sampling

The algorithm first generates multiple uniform random configurations so that it can decide the directions of the tree expansion. Each dimension value of the generated configurations is computed in a separated thread, and the high number of threads utilize the massive parallel processors of the GPU.

D. Tree Expansion

After computing the random samples in the `expansion` procedure, each thread needs to find the nearest tree node for the random sample. From the tree node, we find the Poisson-disk sample within the precomputed sample set, which is in the direction given by the random set and is the closest sample to a node. These two steps utilize the nearest neighbor search. There has been extensive work done on the nearest neighbor search using GPUs [15], [16], [17]. We use the algorithm proposed by Pan et al. [17], which uses Locality-Sensitive Hashing (LSH) for clustering nearby points in high-dimensional spaces. The algorithm generates the same hash value for points near one another; points with the same hash value are stored in the same bucket of the hash table. Using this data structure, the nearest neighbor search for a point can be computed in nearly constant time by lookup only one bucket in the hash table.

When the tree node and the nearest Poisson-disk sample are computed, the algorithm performs local planning to check for a feasible path between the two configurations. We use discrete collision detection (DCD), which discretizes the path between two configurations into multiple steps, and then check collisions for each step. The collision checking performed during local planning is regarded as the most time-consuming part of the overall algorithm. In order to accelerate the computations, we perform this multiple-step collision checking in parallel, using multiple threads on GPUs.

If a local planning finds a collision-free path, the Poisson-disk sample is added to the RRT tree as a new node. The tree expansion is repeated until the goal configuration is added to the tree as a new node.

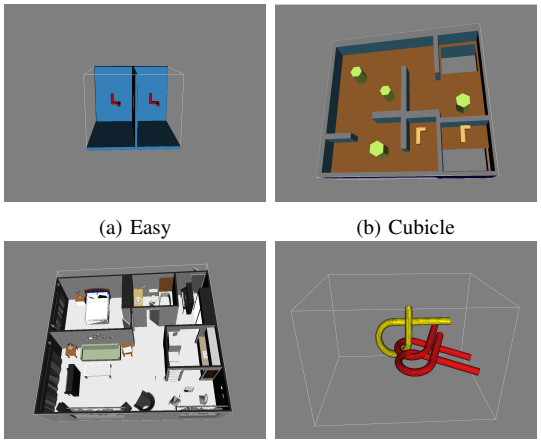
IV. Results

In this section, we present our experimental results and highlight the performance of our parallel planning algorithm on different benchmarks. We implemented the algorithm using OMPL [18] and NVIDIA CUDA libraries. All the timings described in this section were generated on a commodity PC with a NVIDIA GTX 680 GPU.

We used four well-known benchmark scenarios from OMPL (shown in Fig. 4). The planning problems used vary in their level of difficulty; some include narrow passages, which are notoriously hard to navigate using sample planning methods.

For each benchmark, we evaluated the performance of our GPU-based RRT with the following existing CPU-based RRT variant algorithms available in OMPL:

- Standard RRT [9] : Sequential RRT that uses random uniform sampling.



(a) Easy (b) Cubicle

(c) Apartment (d) Alpha Puzzle

Fig. 4: The planning problems used in our planner benchmark. *Easy* moves a robot from the left room to the right room by passing a small window; *Apartment* moves the piano to the hallway near the door entrance; *Cubicles* moves the robot in a cluttered office environment; *Alpha puzzle* contains a narrow passage.

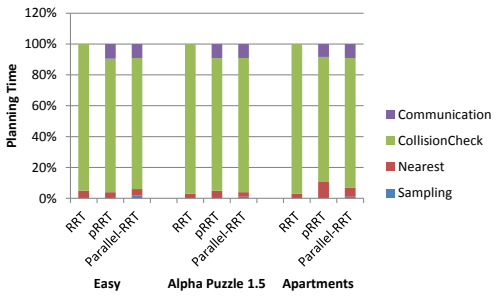


Fig. 5: Timing breakdown among various components for RRT, pRRT, and GPU-based RRT algorithms for different benchmarks.

- RRT-Connect [19] : Bidirectional algorithm which expands trees from both the initial and the goal configurations.
- Lazy-RRT [20] : Algorithm which defers collision checks until it finds a solution.
- pRRT [21] : CPU-based parallel RRT algorithm.

In general, our GPU-based RRT is faster than other CPU-based algorithms, providing up to 20X speedup over the original RRT.

Fig. 5 shows the timing breakdown of the planning for original RRT, pRRT, and GPU-based RRT. The percentage of time spent in nearest neighbor computation is reduced in GPU-based RRT computation, because of its exploitation of the disk free properties of Poisson-disk samples. On the other hand, nearest neighbor computation takes a higher percentage of the total time in pRRT; this `nearestNode` computation is a major source of inefficiency for our method.

V. Conclusions and Future Work

In this paper, we have proposed a new GPU-based motion planning algorithm. Our algorithm can improve the performance of the executions of complex tasks, especially when motion planning is used in an interleaved manner. Our algorithm is based on RRT motion planning algorithm and uses Poisson-disk sampling to to exploit the multiple cores on GPUs. Furthermore, we describe efficient parallel strategies to accelerate the RRT-based motion planning. Our approach can be easily parallelized on GPUs, and it provides up to 20X speedup over previous CPU-based planning algorithms.

There are many avenues for future work. The performance of our planning algorithm can be considerably improved by various optimizations, including bidirectional search similar to that of RRTConnect. We are also interested in extending our planning algorithm to high-DOF articulated models. It would be useful to apply our planning algorithm on robots to solve complex task execution problems in challenging scenarios with dynamic environments or having multiple constraints.

References

[1] J. Guitton and J.-L. Farges, "Taking into account geometric constraints for task-oriented motion planning," in *ICAPS Workshop on Bridging the Gap Between Task and Motion Planning*, 2009, pp. 26–33.

[2] K. Talamadupula, J. Benton, and P. Schermerhorn, "Integrating a closed world planner with an open world," in *ICAPS Workshop on Bridging the Gap Between Task and Motion Planning*, 2009.

[3] L. Jaillet and T. Siméon, "A prm-based motion planner for dynamically changing environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 1606–1611.

[4] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1448–1465, 2009.

[5] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2210–2215.

[6] K. Hauser, "On responsiveness, safety, and completeness in real-time motion planning," *Autonomous Robots*, vol. 32, no. 1, pp. 35–48, 2012.

[7] K. Hauser and J.-C. Latombe, "Integrating task and prm motion planning: Dealing with many infeasible motion planning queries," in *ICAPS Workshop on Bridging the Gap Between Task and Motion Planning*, 2009, pp. 19–23.

[8] L. Zhang, J. Pan, and D. Manocha, "Motion planning of human-like robots using constrained coordination," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. IEEE, 2009, pp. 188–195.

[9] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[10] S. Carpin and E. Pagello, "On parallel RRTs for multi-robot systems," in *Italian Association for Artificial Intelligence*, 2002, pp. 834–841.

[11] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: a hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 171–180.

[12] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast BVH construction on GPUs," in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 375–384.

[13] L. Wei, "Parallel poisson disk sampling," *Transactions on Graphics*, vol. 27, no. 3, p. 20, 2008.

[14] M. Ebeida, S. Mitchell, A. Patney, A. Davidson, and J. Owens, "A simple algorithm for maximal poisson-disk sampling in high dimensions," *Computer Graphics Forum*, vol. 31, no. 2, pp. 785–794, 2012.

[15] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, 2008, pp. 1–6.

[16] J. Pan, C. Lauterbach, and D. Manocha, "g-planner: Real-time motion planning and global navigation using GPUs," in *AAAI Conference on Artificial Intelligence*, 2010, pp. 1245–1251.

[17] ——, "Efficient nearest-neighbor computation for GPU-based motion planning," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 2243–2248.

[18] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012, http://ompl.kavrakilab.org.

[19] J. Kuffner Jr and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *International Conference on Robotics and Automation*, vol. 2, 2000, pp. 995–1001.

[20] R. Bohlin and L. Kavraki, "Path planning using lazy prm," in *International Conference on Robotics and Automation*, vol. 1, 2000, pp. 521–528.

[21] D. Devaurs, T. Siméon, and J. Cortés, "Parallelizing RRT on distributed-memory architectures," in *International Conference on Robotics and automation*, 2011, pp. 2261–2266.

Integrated Planning and Execution for an Aerial Service Vehicle

Jonathan Cacace and **Alberto Finzi** and **Vincenzo Lippiello** and **Giuseppe Loianno** and **Dario Sanzone**
DIETI, Università degli Studi di Napoli Federico II,
via Claudio 21, 80125, Naples, Italy

Abstract

We propose a high-level control system designed for an Aerial Service Vehicle capable of performing tasks in close interaction with the environment. We designed a hybrid control architecture which integrates task, path, motion planning/replanning, and execution monitoring. The high-level system relies on a continuous monitoring and planning cycle to suitably react to events, user interventions, and failures. In this paper, we present the applicative domain, the high-level control architecture, along with preliminary empirical results.

Introduction

In this paper, we present a high-level control system designed for an Aerial Service Vehicle (ASV) operating in close interaction with the external environment. This work is framed within the The AIRobots project (AIR ; Marconi et al. 2012a) whose aim is to develop a new generation of unmanned service helicopters, equipped with sensors and end-effectors, and capable not only to fly, but also to achieve robotic tasks in proximity and in contact with the surface (e.g. site inspections, simple manipulations, etc.).

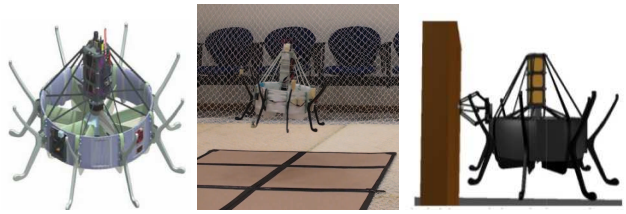


Figure 1: Robotic Platform: ducted-fan ASV

In our scenario, the autonomous control system should orchestrate a new set of operations like wall approach, docking, undocking, wall scanning etc.. These operations represent different operative modes, each associated with a different controller with specific control laws and performance the high-level control system should be aware of. Each switch from one operative mode to the other should be suitably prepared and planned to keep smooth control trajectories. Since the system flies close to the obstacles in cluttered and unknown environments, fast planning and replanning engines are required to generate (or to adjust) trajectories in real-time. On the other hand, the system should be able to regulate the trade off between fast planning and accurateness of the generated trajectories depending on the operative mode and the context. Moreover, since the system operates with the man in the loop, the planning/executive system should be able to manage sliding autonomy, from autonomous to teleoperated mode, depending on the humans' interventions. This applicative domain is challenging and novel and has not been investigated in depth in the UAV literature which is mainly focused on free flight tasks and simultaneous localization, mapping, and path planning problems (Bloesch et al. 2010; Hrabar 2006; Stentz 1995). High-level architectures for UAVs have been proposed in literature (Doherty et al. 2000; Gancet et al. 2005; Doherty, Kvarnström, and Fredrik 2009), none of these addresses the challenges of the ASV domain proposed in this paper. The aim here is mainly to present this novel scenario along with the solutions adopted in the AIRobots project.

System Requirements and Architecture

The applicative scenario described so far requires a high-level control system with following features:

- The air vehicle operates in close interaction with the environment, hence reactive, adaptive, and flexible planning/replanning capabilities are needed;
- Both autonomous and human-in-the-loop control modalities should be supported to allow human interventions and teleoperation;
- High-level control strategies should be defined taking into account the low-level operative modes and constraints.

In particular, the high-level system should orchestrate the activations of a set of low-level controllers, modeled as hybrid automata (Naldi, Marconi, and Gentili 2011), switching to the appropriate controller according to the operative mode and the task (see Figure 2) feeding the selected controller with suitable data (e.g. state and references).
To match these requirements we proposed the layered architecture depicted in Figure 2. Here, two layers are distinguished: the high-level supervisory system is responsible

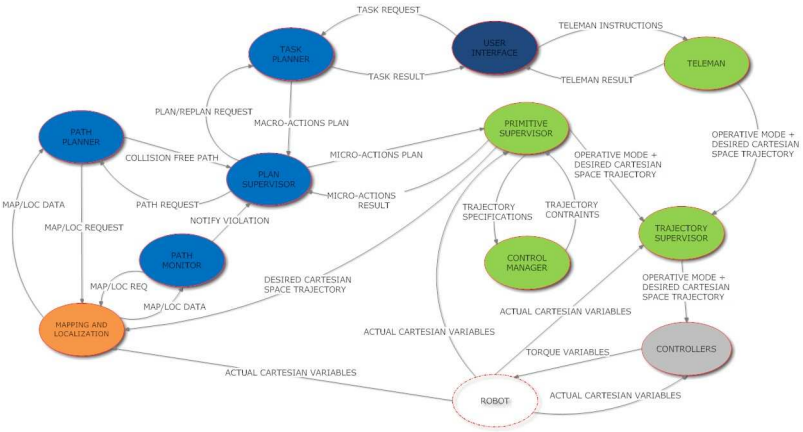


Figure 3: High Level Architecture: high level, low level, and reactive level modules are respectively in blue, green, and gray

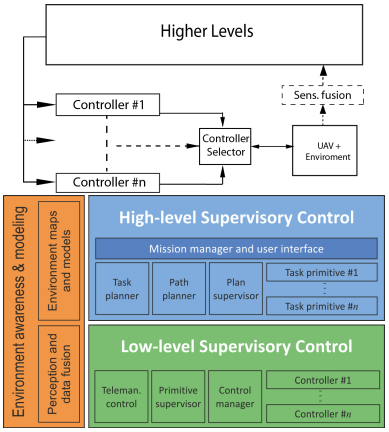


Figure 2: Interaction between the high level system and the low-level controllers (left); the high level control system is composed of high-level and low-level supervisory systems.

for user interaction, task planning, path planning, execution monitoring, while the low-level supervisory system manages the low-level execution of control primitives setting the controllers and providing control references. This architecture is detailed in Figure 3.

The robot activities are represented at different levels of abstraction: *mission-level tasks* representing mission goals; *macro-actions* representing primitive tasks (e.g. $TakeOff$); At a lower level of abstraction we introduce the set of commands that can be sent to the low-level supervisory system - specifically, to the *Primitive Supervisory* (PR)- (micro-actions). The *Task Planner* (TP) provides a plan composed of *macro-actions*. The *User* module (US) allows us to specify high-level goals (e.g. $Inspect(p)$) or lower level tasks (e.g. $TakeOff$) or to directly teleoperate. That is, the user can continuously interact with the system both by providing new high-level tasks/actions and by adjusting the low-level execution in a mixed-initiative control modality. Each task/goal is delivered to the TP which expands a task into an abstract plan composed of *macro-actions*. This plan is then sent to the *Plan Supervisor* (PS) for high-level execution. Each task or macro-action can be interrupted and pre-empted by new tasks provided by the user, provoking task replanning. That is, high-level mixed-initiative control is managed trough mixed-initiative planning (Finzi and Orlandini 2005). The PS generates, for each macro-action in the high-level plan, a set of *micro-actions* to be executed by the *Primitive Supervisor* (PR). Each *macro-action* is further decomposed into a sequence of *micro-actions* which are endowed with detailed information about the associated geometrical paths. The PR exploits the *Control Manager* (CM) to select the low-level controller responsible for the micro-action execution. This module is the main responsible for the high-level/low-level control integration: given the operative constraints provided by the high-level supervisor and given the low-level controller features, the CM is to decide the best controller for the execution. Finally, the PR generates the control trajectory passing it to the *Trajectory Supervisor* (TS) to generate control references at a suitable frequency. The PR exploits concatenations of fifth-order polynomials to provide smooth trajectories between waypoints (Macfarlane and Croft 2003) while ensuring the velocity and tolerance constraints. Depending on the required reactivity (first requirement), the PS regulates the number of geometric waypoints to be processed by the PR. When a micro-action fails, the PS can either call the PP to generate an alternative path or call the TP to generate a different plan of macro-actions. Furthermore, it can be interrupted by the *Path Monitor* (PM) which checks for trajectory deviations and unexpected obstacles. Finally, the operator can always switch to a manual control mode, in this case the TS should monitor the trajectory provided by the *Teleman*. Once the autonomous control is restored, a replanning process is needed to recover the execution of the current task.

Task Decomposition and Executive Control

The high-level executive system coordinates task decomposition and plan monitoring (see Figure 4). In order to meet the reactivity and flexibility needed in the SAV domain, the system relies on a PRS engine (Ingrand, Georgeff,

$TakeOff(Pos)$	Take off from the current pose and hover in the pose Pos;
$Land(Pos)$	Land from the current position to Pos;
$Hover(Pos)$	Keep the pose Pos;
$MoveTo(Pos)$	Move from the current pose to Pos;
$MoveCircular(Pos, I)$	Circular movement around P with radius in I ;
$Scan(Srf)$	Scan the surface Srf;
$Inspect(Obj, P)$	Observe the object Obj in position P;
$Brake(C)$	Execute a hard brake from the current position;
$Approach(P)$	Approach the target position P;
$Dock(P)$	Dock to a target position P;
$UnDock(C)$	Undock from the current position;
$Manipulate(Obj, P)$	Manipulate an object Obj in position P.

Table 1: Macro actions considered in the operative domain.

and Rao 1992) that manages a BDI-like execution cycle (Rao and Georgeff 1991) and hierarchical task decomposition. The high-level executive system responds to events generated by the US, PS, or TP itself by committing to handle one pending goal, selecting a method from a plan library, managing the hierarchical decomposition to extract/update the macro-actions plan. Once a plan is generated, the PS should manage the actual execution of each *macro-action* providing the action results to the TP module. During this execution process, user interventions are treated in a uniform way: at any time the user can interrupt/suspend the current task, or the execution of alternative tasks can be invoked. In this case, the executive system reacts by replanning from the current state: it selects alternative methods and generates an alternative plan. This enables mixed initiative task planning (Allen and Ferguson 2002; Finzi and Orlandini 2005).

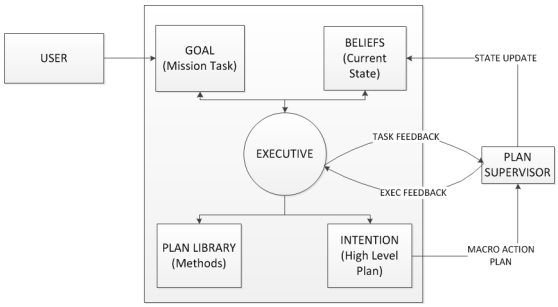


Figure 4: The high-level executive control.

Path Planning and Replanning

The *Path Planner* expands each *macro-action* into a set of *micro-actions* representing a path that respects geometric and operative constraints. The path generation algorithm is based on a Rapidly-exploring Random Tree (RRT) algorithm (Lavalle 1998) which is particularly suitable in highly unstructured and dynamic domains. In this work, the RRT algorithm generates collision-free paths composed of sequences of waypoints (x, y, z, θ), where (x, y, z) is a point and θ is the yaw. More specifically, it generates a path as a sequence of (x, y, z) points in a 3D search space (3D grid map), while the yaw θ is obtained as the direction pointing towards the next waypoint. The generated path should satisfy a set of additional control, safety, and temporal constraints:

- *Maximum angle* for pitch and yaw;
- *Minimal distance* from the obstacles (this parameter is also associated with the operative mode and the accuracy of the selected controller);
- *Maximum Time* for the path generation processes, if the algorithm cannot find a feasible path before the timeout, it should provide the best partial path. Moreover, that RRT path planner can generate several solutions to refine the path, until one of the following conditions are satisfied:
- *timeout*, i.e. the available time for path planning expires;
- *interrupt*, i.e. a replanning request or an exogenous event interrupts plan generation;
- *cost threshold*, i.e. as soon as the current path cost is below a suitable threshold, the generated plan is considered as satisfactory.

In order to refine the generated path, the RRT algorithm is iterated until the current generated path is not satisfactory as illustrated in the Algorithm 1. This refinement process can be interrupted at any time. Usually the RRT is reactive enough to produce at least one solution, however, if the $timeout$ occurs before the generation of the first solution, the $solveRTT$ function generates the $path$ that arrives closer to the target. In this case, the PS will start the execution of this incomplete path providing the path planner with additional time to complete and refined it.

Algorithm 1 Refine_RRT(q_{init},q_{goal},$threshold$,$timeout$)

```
initialize(path,time);
while ((time < timeout) ∧ (preempted = false) ∧
(pathCost ≥ threshold)) do
   newPath ← solveRRT(q_init,q_goal,timeout);
   if C(newPath) < path then
      path ← newPath;
      pathCost ← C(newPath);
   end if
end while
return path
```

The path cost is defined as follows:

$$\begin{aligned}\mathbf{c}(path) =& \mathbf{c}_{lng}(path) \cdot p_{lng} + \mathbf{c}_{ang}(path) \cdot p_{ang} + \\ & \mathbf{c}_{way}(path) \cdot p_{way} + \mathbf{c}_{obs}(path) \cdot p_{obs} + \\ & \mathbf{c}_{unk}(path) \cdot p_{unk}\end{aligned} \quad (1)$$

where the p_i are suitable weights and $\mathbf{c}_i$ are defined as follows. $\mathbf{c}_{lng}(path)$ is a cost associated with the path length; $\mathbf{c}_{ang}(path)$ represents the cost associated with angular (yaw and pitch) variations, by minimizing this cost a straight path should be preferred to a path with angular turns; $\mathbf{c}_{way}(path)$ counts the generated waypoints and allows us to minimize the segments in the path; $\mathbf{c}_{obs}(path)$ is associated with obstacle proximity and penalizes paths close to obstacles; $\mathbf{c}_{unk}(path)$ penalizes paths through -or close to- unexplored cells. Once a path is generated, the path planner defines a set

of constraints $cst = (ms, md, et)$ associated with each generated segment. Roughly, for each segment, we set the maximum speed ms directly proportional to the obstacle minimal distance mo along the corresponding segment; ms is also associated with a proportional error et, therefore we set md as mo-et (if this value is not positive, the speed limit is lowered). These constraints cst are also accessible to the human operator which can manually reset them. Note, that cst are just rough limits used by the CM and the PR to select the right controller and to generate the trajectory associated with the path. Once the controller is selected, the generated path is incrementally expanded into a motion trajectory by the PR (using fifth-order polynomials concatenations (Macfarlane and Croft 2003)) under the supervision of the PS that regulates the number of waypoints to be processed depending on the control mode (and the associated requested reactivity).

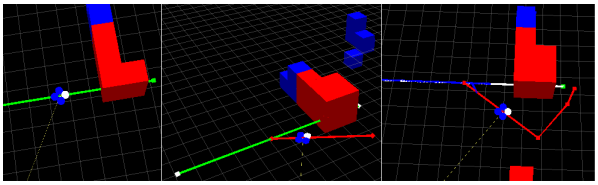


Figure 5: (left) *Brake* to avoid collision; (center) *Escape* path to avoid the obstacle; (right) *Replan* a new path generated to reach the target.

Path replanning is managed with different strategies depending on the time available for path generation. The urgency associated with the replanning activity depends on the position of the collision point p_{obs} and the estimated time to collision t_{ttc}. This one is estimated by considering the obstacle distance d_{obs} along the trajectory and the mean velocity v_{mean} along the path. Given the time to collision t_{ttc}, we introduce two thresholds $T_b < T_e$ used to distinguish the following three cases:

- *Brake.* If $t_{ttc} \leq T_b$ then the obstacle is too close for replanning, hence the PS directly sends a $Brake$ command to the PR to stop the robot in $hovering$ (Figure 5 up-left).
- *Escape.* If $T_b < t_{ttc} \leq T_e$, the PP is invoked by the PS to find an escape path that allows the robot to avoid the obstacle; the escape trajectory represents a fictituos detour that provides the planner with additional time to generate the new path on-the-fly (Figure 5 up-right).
- *Replan.* If $t_{ttc} > T_e$ then the time is sufficient for safe replanning, hence the PS calls the PP to replan, on-the-fly, a trajectory from a suitable deviation point along the previous path (Figure 5 down).

The PP is called in the case of *Escape* and *Replan*. In the case of *Escape*, the path planning task is simple: it is to select a close and safe target point q_{target} in the free space, far enought to enable safe on-the-fly replanning, and to generate a path to reach it (Figure 5 right). That is, *Escape* provides a path that not only permits to avoid the obstacle, but also provides the time for replanning a new path to the goal.

The interesting case is the third one, where the path planning process should find an alternative path that connects the old trajectory with a new one while the robot is flying. The replanning algorithm is illustrated in Algorithm 2. Given the target q_{goal}, the old path $path_{old}$, the collision point q_{obs}, and the t_{ttc} time, the replanning process first estimates the time needed to replan t_{rp} (*estimatedRepTime*); then it selects a waypoint wp_{rp}, along the old path $path_{old}$, from which it is possible to safely calculate the deviation $path_{new}$ from $path_{old}$ (*selectDeviationWP*); finally, upon setting a suitable threshold (*setThreshold*), the replanning process calls RRT_refine to generate the new path $path_{new}$ from the deviation waypoint wp_{rp} to the target q_{goal}. $path_{new}$ should allow the PR to generate a new trajectory connecting the old one with a smooth deviation from wp_{rp}.

Algorithm 2 Replan(q_{goal}, $path_{old}$, q_{obs}, t_{ttc})

$q_c \leftarrow$ getPosition();
$t_{rp} \leftarrow$ estimatedRepTime(q_c, q_{goal}, $path_{old}$, q_{obs});
$wp_{rp} \leftarrow$ selectDeviationWP(q_c, q_{obs}, $path_{old}$, t_{rp});
$threshold \leftarrow$ setThreshold(wp_{rp}, q_{goal}, t_{rp}, t_{ttc});
$path_{new} \leftarrow$ Refine_RRT(wp_{rp} , q_{goal}, $threshold$, t_{rp});
return $path_{new}$

To select the deviation waypoint wp_{rp} we defined the following strategy. Given the estimated time needed to replan t_{rp}, we estimate the robot position q_{pr} at time t_{rp} (assuming that it keeps following the old path $path_{old}$ during replanning), if there exists a waypoint wp in $path_{old}$ that follows p_{rp} and precedes q_{obs} (keeping a suitable range the we assume as $maxRange$), then we select wp as the deviation waypoint wp_{rp}, otherwise, q_{rp} is on the path segment that intersects the obstacle, hence we select wp_{rp} as the point q_m in the middle of the segment that connects q_{rp} and a point q'_{obs} which is at $maxRange$ distance from q_{obst}. In Figure 5 (center), we find an example of replanning from a waypoint after the collision detection (left).

3D Mapping

The environment for mapping and path planning is a 3D grid-map of cells which is run-time generated given the robot pose and the 3D point clouds extracted from the cameras. We deploy the well known pin-hole camera model (Hartley and Zisserman 2004). Our pose is either obtained by using libviso2 (Geiger, Ziegler, and Stiller 2011) coupled with a Kalman filter or, alternalively, by directly deploying an optitrack motion capture system. Given the pose, the associated point cloud map should be suitably processed into a 3D occupancy grid. This is obtained by discretizing the vehicle's workspace with elementary cubes of equal size. In our case, we employed a vehicle of $50 \times 50 \times 20$ cm hence, we used cubes of 10 cm. For each cube we stored: the number of inliers (3D triangulated points) fell into the cube volume, the last camera position which an inlier had been collected, and the state of the cube. The number of inliers represents the number of different points from which the same obstacle has been detected. Each cell can be associated with one of the following values: $free$, $occupied$, $obstacle$, $target$, $ignored$

or $unknown$. Initially each cube is set to $free$. When a 3D point is detected to belong to a given cube, the value of the corresponding cube is set to $occupied$. When the number of points inside a cube reaches a given treshold, the state is set to $obstacle$. On the other hand, when a target is identified, the corresponding cube is set to $target$. Moreover, from each position that had generated a valid target view point, all the cubes laying along the optical rays are set to $ignored$. For wide environments, a sparse representation of the occupancy grid map is associated with a spatial/temporal vanishing criterion. This determines whether an occupancy cube is sill reliable or if it has to be discarded (depending on the distance traveled by the vehicle or on the time last after its previous update). In fact, due to the drift of the vehicle pose estimation, obstacles which have been observed a long time before or far from the current position cannot be considered reliable anymore in the current map representation, therefore they should be refreshed. With these solutions the reliability and scalability of the map representation can be suitably tuned.

Experimental Results

In this section, we present preliminary experimental results on planning, replanning, and obstacle avoidance, both in real-world scenarios and in simulated environments.

Real-world planning and execution.

Our architecture has been tested in a real scenario of dimension $400 \times 400 \times 300\ cm^3$ considering the two environments depicted in Figure 7 (up and down). In the two testing scenarios, the task was the following: inspect a target point in pose $(380, 350, 50, 90)$ from the pose $(40, 40, 50, 0)$ with maximum and minimum speed set at $0.3\,m/s$ and $0.1\,m/s$ respectively. The obstacles are detected on the fly and this can provoke task/path replanning, escape, or brake. For each scenario, we executed each test 10 times collecting mean, max, min, and standard deviation (STD) of: time spent during planning (Tp), time spent in replanning (Tr), number of replanning episodes (Nr), length of the executed path (Lp), and total time for execution (including replanning time) (Te). For computation and simulation we used an Intel Core Duo, 1.40GHz, 3GB ram, Ubuntu 10.04. The high-level architecture was developed in ROS. As for 3D mapping, we used cameras ueye with hardware synchronized images, compressed on-board using atom 1.6 GHz, and sent to a ground station. The stereo images are streamed at around 15 Hz at the ground station. The vision algorithm can track around 120 image features correspondences on 4 images working at the streaming frequency. Each camera provides images with resolution of 752×480 and an angle of view of around $50°$.

Table 2 reports the results for the two scenarios (Test 1 and Test 2 in Figure 7). For both these settings, initially, the obstacles are not visible, hence the generated plan is simple and planning time is low (Figure 7 (left)). Once the obstacles are discovered on the fly, replanning is needed to adjust on-line the trajectory (i.e. without hovering during the replanning phase). Replanning and execution time are slightly

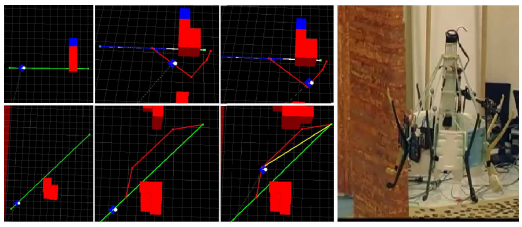


Figure 6: Replanning: generated and executed path (left) real platform during plan execution (right).

higher in the first scenario which is more complex. Instead, Tr seems negligible when compared with Te. The final trajectory length (Te) is similar in both the settings and comparable with the distance between the starting and target point, hence the final trajectory seems not affected by the continuous replanning process. In these tests, Tp and Tr are mainly due to path and trajectory planning (task planning is negligible). We never experienced brake or escape episodes. Overall, the system task/path planning performance seems compatible with the operative scenario requirements.

Simulated planning and execution.

We tested our planning and execution system in simulated environments. To test continuous replanning, we considered a larger space of dimension $100 \times 100 \times 50\ m^3$ with 4 and 9 obstacles (see Figure 7). To decouple replanning from map bulding, we assumed a know map associated with a visibility horizon (not visible obstacles are detected on the fly causing replanning). For each test, the task was to inspect a target point in pose $(90, 90, 5, 90)$ starting from $hovering$ in the pose $(5, 5, 5, 0)$ (in meters); the robot maximum and minimum velocity was set at $0.5\,m/s$ and $0.1\,m/s$ respectively. By changing the visibility horizon (green cells in Figure 7) of the planner (15 or 25 m) and the complexity of the environment (4 or 9 obstacles) we obtained 4 scenarios. Table

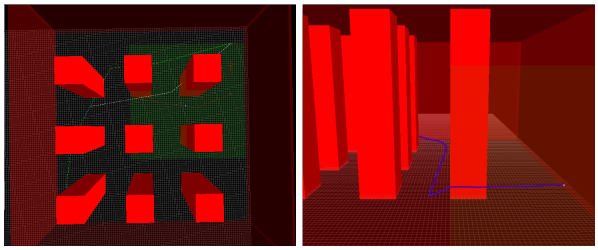


Figure 7: Replanning: generated (left) and executed path (right).

2 collects means and STD of 10 tests for each entry (time and length are in $sec.$ and m, LL, HL, etc. are for Low complexity and Low visibility, High complexity and Low visibility). Here, we can see that Tp increases with the obstacles (HL,HH) and decreases with short visibility (LL,HL). Indeed, in these cases the planning problem is simpler. However, short visibility is associated with additional replanning time which, in turn, decreases with the number of obstacles. The lower the replanning time, the lower is the execution

	Test 1				Test 2			
	Mean	**STD**	**Max**	**Min**	**Mean**	**STD**	**Max**	**Min**
Tp	0.075	0.014	0.08	0.04	0.017	0.002	0.03	0.01
Tr	0.614	0.41	1.20	0.01	0.067	0.04	0.11	0.005
Te	60.5	10.12	75	42	49.9	8.18	60	40
Lp	14.4	1.54	18	12	13.18	1.11	15	11

Table 2: Planning and execution results (in seconds) in the real scenario.

Res/Env	LL		LH		HL		HH	
	Mean	**STD**	**Mean**	**STD**	**Mean**	**STD**	**Mean**	**STD**
Tp	0.21	0.11	0.39	0.03	0.25	0.10	0.31	0.14
Tr	0.12	0.03	0.07	0.01	0.20	0.04	0.23	0.03
Te	308.39	3.1	211.88	2.4	718.57	5.2	720.45	7.6
Lp	79.09	13.76	78.04	9.63	86.79	12.65	85.24	13.12
Nr	0.9	0.21	0.3	0.12	3.4	1.71	2.5	1.10

Table 3: Planning and execution results(time in seconds, length in meters)

time and the shorter the executed path. A similar effect is due to visibility: short visibility causes frequent replanning events (Nr) and longer paths (Lp) and execution times (Te). Furthermore, the variance is enhanced with short visibility that enhances the uncertainty. In these tests, the task planning time is usually negligible (Tp and Tr mainly due to path and trajectory). Also in this case, we never experienced brakes or escapes.

Simulated and real inspection.

As for operations closer to the surface, we considered two typical inspection scenarios: physical (Pi) and visual inspection (Vi). In both these cases the system has to move in a pose which faces a vertical surface hovering at a close and fixed distance (approach), in this case 50 cm. As for Pi (see Figure 8, left), the robot executes a docking maneuver (docking) and slides (keeping the contact) along a linear trajectory (p-inspect) of $225\ cm$. In the case of Vi, an inspection trajectory (v-inspect) should be planned and executed. Here, the goal is to scan a $150 \times 100\ cm^2$ surface with step $50\ cm$ distant 50 cm from the wall (see Figure 8, right). In Tab. , we collect the results of 10 tests in a simulated environment, for each scenario considering planning time (Tp) divided in trajectory (Tm) and path planning (Tpp) time (task planning is negligible). For each test and scenario, both path and task planning times are compatible with the operative scenario requirements.

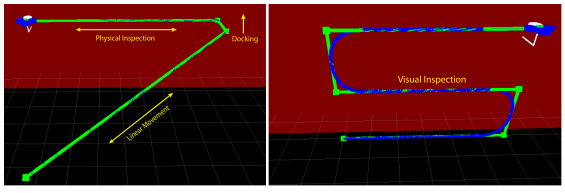


Figure 8: Physical inspection (left) and visual inspection (right)

We performed physical manipulation and visual inspection tasks also in a real environments (see Figure 9) obtaining similar results for path and task planning. Currently, we are carrying on additional experiments to better analyze the overall system performance with the real platform (e.g. failures, recoveries, replanning time, execution time, etc.).

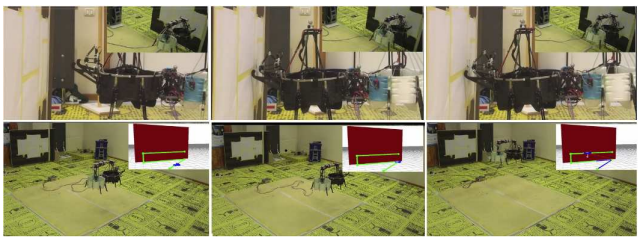


Figure 9: Physical inspection (up) and visual inspection (down)

Conclusions

Aerial Service Robotics is a challenging and novel application for autonomous systems. The close and physical interaction with the environment and the frequent user interventions requires a high-level control system which integrates fast and reactive planning engines working at different levels of abstraction and sensitive to low-level operative mode constraints. The aim of this paper was to present the challenges of the ASV domain along with the solutions provided within the AIRobot project. The proposed high-level system combines hierarchical task decomposition, mixed-initiative control, BDI execution, RRT path planning/replanning to allow reactivity, flexibility, and sliding autonomy. We described the system at work both in real and simulated environments providing preliminary results. Future work will focus on additional real-world experiments and on the extension to multi-aerial vehicles (Marconi et al. 2012b).

Acknowledgments

The research leading to these results has been supported by the AIRobots and SHERPA FP7 projects under grant agree-

	Physical Inspection				Visual Inspection			
	Mean	**STD**	**Max**	**Min**	**Mean**	**STD**	**Max**	**Min**
Tp	0.798	0.012	0.019	0.009	0.734	0.47	1.25	0.42
Tm	0.324	0.17	1.07	0.12	0.329	0.22	0.57	0.3
Tpp	0.473	0.27	0.71	0.14	0.405	0.07	0.49	0.39

Table 4: Physical inspection and visual inspection

ments ICT-248669 and ICT-600958 respectively.

References

Eu collaborative project ict-248669, "airobots", *www.airobots.eu*.

Allen, J., and Ferguson, G. 2002. Human-machine collaborative planning. In *NASA Workshop on Planning and Scheduling for Space*.

Bloesch, M.; Weiss, S.; Scaramuzza, D.; and Siegwart, R. 2010. Vision based mav navigation in unknown and unstructured environments. In *ICRA 2010*, 21–28.

Doherty, P.; Granlund, G.; Kuchcinski, K.; S, E.; Nordberg, K.; Skarman, E.; and Wiklund, J. 2000. The witas unmanned aerial vehicle project. In *In Proceedings of the 14th European Conference on Artificial Intelligence*, 747–755.

Doherty, P.; Kvarnström, J.; and Fredrik, H. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. In *AAMAS*, 332–377.

Finzi, A., and Orlandini, A. 2005. Human-robot interaction through mixed-initiative planning for rescue and search rovers. In *AI*IA-05*, 483–494.

Gancet, J.; Hattenberger, G.; Alami, R.; and Lacroix, S. 2005. Task planning and control for a multi-uav system: architecture and algorithms. In *IROS*, 1017–1022.

Geiger, A.; Ziegler, J.; and Stiller, C. 2011. Stereoscan: Dense 3d reconstruction in real-time. In *IEEE Intelligent Vehicles Symposium*, 963–968.

Hartley, R., and Zisserman, A. 2004. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge U.K.: Cambridge Univ. Press.

Hrabar, S. 2006. Vision-based 3d navigation for an autonomous helicopter.

Ingrand, F.; Georgeff, M. P.; and Rao, A. S. 1992. An architecture for real-time reasoning and system control. *IEEE Expert: Intelligent Systems and Their Applications* 34–44.

Lavalle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. *Computer Science Dept., Iowa State University, Tech. Rep*.

Macfarlane, S. E., and Croft, E. A. 2003. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on Robotics* 19:42–52.

Marconi, L.; Basile, L.; Caprari, G.; Carloni, R.; Chiacchio, P.; Huerzeler, C.; LIPPIELLO, V.; Naldi, R.; Janosch, N.; Siciliano, B.; Stramigioli, S.; and Zwicker, E. 2012a. Aerial service robotics: The airobots perspective. In *2nd International Conference on Applied Robotics for the Power Industry, Zurich, Switzerland*.

Marconi, L.; Melchiorri, C.; Beetz, M.; Pangercic, D.; Siegwart, R.; Leutenegger, S.; Carloni, R.; Stramigioli, S.; Bruyninckx, H.; Doherty, P.; Kleiner, A.; Lippiello, V.; Finzi, A.; Siciliano, B.; Sala, A.; and Tomatis, N. 2012b. The sherpa project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments. In *Proc. of the IEEE Int. Workshop on Safety, Security and Rescue Robotics (SSRR)*.

Naldi, R.; Marconi, M.; and Gentili, L. 2011. Modelling and control of a flying robot interacting with the environment. *Journal of IFAC* 4(12):2571–2583.

Rao, A. S., and Georgeff, M. P. 1991. Deliberation and its role in the formation of intentions. In *UAI*, 300–307.

Stentz, A. 1995. Optimal and efficient path planning for unknown and dynamic environments. *Int. J. of Robotics and Automation* 10(3):89–100.

Optimization of Aerial Surveys using an Algorithm Inspired in Musicians Improvisation

João Valente, Antonio Barrientos and Jaime Del Cerro
Robotics & Cybernetics Research Group (CAR UPM-CSIC)
joao.valente@upm.es

Abstract

This paper tackles the problem of computing safe coverage trajectories for a fleet of unmanned aerial vehicles (UAVs) on large areas. The solution proposed is based on a evolutionary optimization algorithm denoted as Harmony Search (HS). The resulting algorithm has been entitled as m-CPP (Meta-heuristic Coverage Path Planning) algorithm. Results obtained by applying this approach have been compared with former heuristic-based methods. Finally, safety restrictions have been applied to allow near optimal and safe coverage flights cooperatively. Complete safe coverage missions have been planned in order to be performed by teams of quad-rotors by applying the proposed approach.

Introduction

Currently, small Unmanned Aerial Vehicles (UAV) play an important role both in military and civil missions. Due to the limited autonomy of these small vehicles, when the area to cover is large, several robots can be used working as teams, performing their mission in a collaborative way.

These teams of UAV's have become emerging technologies for multipurpose remote sensing, i.e. remote imagery, where high-resolution images of the overall area are required. Some real applications of these techniques are Precision Agriculture (PA) and Critical Infrastructure Protection (CIP).

Therefore, the problem addressed in this paper can be summarized as the problem of finding a set of trajectories for flying robots that completely survey an area in order to build high-resolution mosaics with the minimum cost in a safe way.

Previous work shows that mission completion time decreases if the number of heading movements during area survey is reduced [1]. In addition to this, a reusable aerial planner has to deal with pre-defined mission setup positions, forbidden or avoidable areas and different robot capabilities. Taking into account these considerations, aerial coverage path-planning problem (ACPP) applied to a team of quad-rotors endowed with digital cameras can be literally defined as: To compute a complete aerial coverage trajectory for each quad-rotor with the minimum number of heading movements (i.e. changes of direction).

The planner has to generate trajectories in order to fly over all the required points in the workspace just once as well as deal with additional restrictions such as predefined initial and goal positions. Additionally, some details have to be taken in consideration for solving the problem. Firstly, each point in the environment corresponds to a way-point (WP) where a picture has to be taken by using the on-board camera.

In order to solve this problem, a CPP algorithm based on evolutionary strategies is proposed. The algorithm named m-CPP is based on a quite novel technique known as Harmony Search (HS). The motivations behind this choice are the simplicity of the technique and the rapid replication of the findings.

Next section enhances the problem background and reviews some methodologies previously applied. Next the HS algorithm is explained. After that the m-CPP algorithm is introduced. In section Results a comparison between previous results and obtained by applying the m-CPP algorithm is summarized. Finally conclusion is presented.

Background

Covering task for mobile robots has many different meanings. Nevertheless, it usually refers to a task in which a robot endowed with a specific sensor is able to cover (sense) partially or completely an area in an optimal way. General coverage approaches can be found in [2, 3].

Coverage Path Planning (CPP) is a sub-field of motion planning that addresses the problem of finding a robot path within the workspace that allows performing full environment coverage by using an end-effector that can be a sensor, an actuator or even both.

CPP algorithms have been developed mostly for ground vehicles [4, 5]. On the other hand, regarding UAVs, fewer

approaches have been presented so far. However, this type of planning has been recently considered in [6, 7, 8].

In [6] an area decomposition and coverage approach with fixed-wing aerial vehicles is presented. The problem presented in this work is different from previous one mainly due to the following reasons: 1) It is applied to be used with a team of vehicles instead of only one; 2) It takes more common restrictions into consideration; and 3) It also considers forbidden or undesirable areas in order to avoid them.

This problem was already addressed by Valente et al. [7] where a solution based on heuristic and non heuristic approaches was proposed by using computer graphic techniques and a wave-front planner algorithm with a backtracking procedure.

Although the coverage strategies previously mentioned were reliable and admissible, safety has to be taken into consideration during the mission generation. Thus, it is required to reach complete area coverage and, at the same time, to ensure the vehicle's safety prevails. Risk analysis study reported in [9] shown that security borders between robots would dismiss if the workspace agree with some conditions (e.g. enough distance between way-points). The aforementioned report is focus on providing a solution for the full simultaneous coverage problem taking into account the safety operation of the drones.

Following the same research line, a novel ACPP algorithm is proposed in the following sections in order to improve the results previously obtained.

Harmony Search Algorithm

The HS algorithm is a meta-heuristic algorithm based on musician's improvisation proposed in [10]. It is also considered as a member of the evolutionary algorithms family. This technique has already been applied to solve many optimizations problems in different fields, e.g., telecommunications [11, 12], economy [13], computer vision [14], etc.

Let us imagine a group of Jazz musicians, each one playing a different instrument. They start to pitch some notes in order to compose a new song. As they search for a sequence of musical notes that provides a good musical harmony, the harmonies achieved up to the moment are kept in mind. If a new harmony sounds better than a previously one, the new harmony will replace it. From an optimization point of view, each player represents a variable and each pitch, a candidate value.

The main body of the algorithm is the Harmony Memory (HM) matrix, defined in (1), where rows are candidate solutions vectors, and columns decision variables. The last column of the HM matrix is the cost function value. The main parameters of the HS algorithm are: Harmony Memory Size (HMS), Harmony Memory Considering Rate (HMCR), and Pitch Adjustment Rate (PAR). HMS is the number of rows or candidate solutions considered. HMCR is the probability of choosing a variable value from the HM. Finally; a variable value is adjusted (switched by a neighboring value) with probability PAR where uniform distribution U(0,1) is commonly used.

$$HM = \begin{bmatrix} X_1^1 & \cdots & X_N^1 & J(X^1) \\ \vdots & \ddots & \vdots & \vdots \\ X_1^{\{HMS\}} & \cdots & X_N^{\{HMS\}} & J(X^{\{HMS\}}) \end{bmatrix} \quad (1)$$

The algorithm can be structured in four steps: Step 1, HM initialization; Step 2, improvisation of new harmony vector $X^{'}=\{X^{'}_1,\ldots, X^{'}_N\}$ from the HM (with or without PAR) or by randomness; Step 3, Harmony replacement if the new harmony turns out to be better than the worst; Step 4, Checking if stop criterion has been reached (e.g. number of iterations, cost function); if not, to go to Step 2, else return the best solution.

The m-CPP algorithm

A step-by-step procedure of the algorithm as well as some definitions is described in this section.

Definitions

Let's consider $a \in \mathbb{R}^2$, a sub-area from the global area $\mathcal{A}$ to be covered. It can be written as, $\mathcal{A} = \bigcup_{a \epsilon \mathcal{A}} a$.

Let p be an admissible coverage path that exists for each sub-area, expressed as $\forall a \in \mathcal{A}, \exists\, p$. If p is a set of points, herein denoted as way-points w, it can be written that $p = \bigcup_{w \epsilon p} w$. Considering a typical coverage mission, a specific task has to be performed (e.g. taking a picture or taking some data from onboard sensors) in each single way-point.

The scope of the action (e.g. photograph) is defined by a cell c. A set of cells is defined by $\mathfrak{c} = \bigcup_{c \epsilon \mathfrak{c}} c$. In this way, it can be said that, p is an admissible coverage trajectory, if:

1. The number of elements of set p is equal to the number of elements of set $\mathfrak{c}$
2. There are no duplicated elements in the set, therefore: $w' \neq w, \forall w \in p$
3. p is subject to a certain w order

The set of coverage trajectories p that cover $\mathcal{A}$ is denoted by $\mathcal{P}$, such that, $\mathcal{P} = \bigcup_{p \epsilon \mathcal{P}} p$, and the set of regular adjacent cells that map the overall area $\mathcal{A}$ is given by $\mathcal{C} = \bigcup_{c \epsilon \mathcal{C}} \mathfrak{c}$.

As previously mentioned, the variable of interest to minimize is the number of heading movements. A heading

movement, is referred to a displacement from the UAV around it center of mass (also known by heading movement). Considering regular cells and bearing in mind do not visit a waypoint twice, these rotations have a lower and an upper bound of 45º and 135º respectively (absolute value). The cost function that specifies this constraint can be written as:

$$J_1 = K_1 \times \sum_{i=1}^{m} \psi_k^{\{i\}} + K_2, \quad k \in \{135°, 90°, 45°, 0°\}$$

Where ψ_k are weights defined for angles k as follow,

$$\psi_{\pm 135^\circ} > \psi_{\pm 90^\circ} > \psi_{\pm 45^\circ} > \psi_{\pm 0^\circ}$$

and K_n are weights,

$$K_2 > K_1, \quad K_{1,2} \in \mathbb{R}$$

On the other hand, as a primary requirement, the coverage mission has to be accomplished in a safe way. Therefore, an effort was made to introduce another restriction in order to avoid simultaneous adjacent positions during the mission. This means that two or more UAVs are not allowed to stay in adjacent positions at any time (considering homogenous systems, with the same features and capabilities). Fig. 1 illustrates this situation.

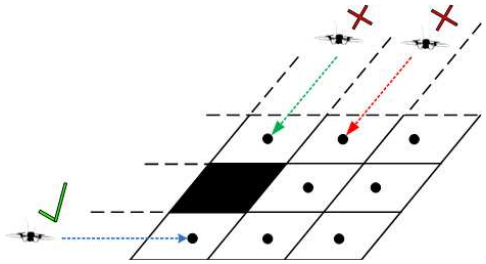


Figure 1. Three quad-rotors flying over a decomposed area. Red and green trajectories are in conflict, due to adjacency. A safe trajectory is shown in blue.

The cost function has been defined by using a Boolean-type function and a real number, and is given by,

$$J_2 = J_2^{'} \times K_3,$$

where K_3 is a weighting parameter,

$$K_3 >> K_1, K_2, \qquad K_3 \in \mathbb{R}$$

and,

$$J_2' = \mathcal{S}_1 \vee \mathcal{S}_2 \ldots \mathcal{S}_{n-1} \vee \mathcal{S}_n = \bigvee_{i=1}^{n} \mathcal{S}_i$$

where $\mathcal{S}$ denotes a conflict alarm. If two or more UAVs are nearby each other (in adjacent cells), this alarm is activated. Finally, the final cost function can be written as,

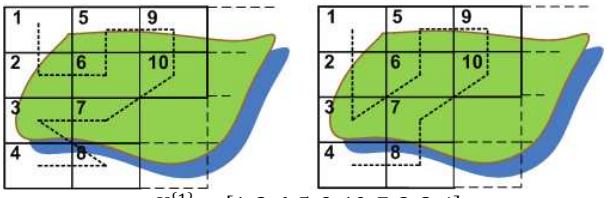
$$J = J_1 + J_2 \qquad (2)$$

Since the new cost function considers the position of the all drones at any time, some important changes in the computational procedure have been required. It should be highlighted that previous approach do not required considering time as a variable. Thus, simple serial computation was enough to execute the algorithm. By the other hand, the requirement of considering the position of all drones before performing the improvisation step, make parallel computation necessary.

From HS to m-CPP

As referred in Section 3, HM is the main body of the HS algorithm. Candidate solutions are herein stored, represented by a vector of dimension N, made up of decision variables from the optimization problem. Each decision variable addresses a cell to be visited by the aerial robot, corresponding to a way-point coordinate x_i, such that $x_i \in p$ with i = 1, .., n, where n = dim(p).

Fig. 2 shows how the variables are managed in the problem by considering HM with HMS=2. The dotted lines are the path transverse by the quad-rotors, which start from the cell with index 1 to the cell with index 4 in both cases.



$X^{\{1\}} = [1, 2, 6, 5, 9, 10, 7, 3, 8, 4]$
$X^{\{2\}} = [1, 2, 3, 6, 5, 9, 10, 7, 8, 4]$

Figure 2. HM vectors represented over a sample field.

Initialization

The first step of the HS algorithm is the initialization. In the first iteration, the Harmony vectors (i.e. solutions) are usually generated through a random process. As can be observed from the aerial CPP problem, the flight time represents the main constraint of the problem, which causes other constraints to arise, such as the number of revisited places in the environment. It is obvious that by reducing the number of revisited cells in the environment, the path will also shorten, and consequently, the coverage time.

As a result, a HM matrix with HMS permutation vectors with N elements without repetitions is initially generated randomly.

Improvisation

An iterative process called improvisation starts after generating the initial HM by using the method previously described. In order to ensure the Harmony vectors permutation, the new vector must be carefully obtained by slightly changing the conventional HS algorithm mechanism.

Herein, each element of the new vector X' is either selected from the HM or the entire possible range of values. According to the previously mentioned probability HMCR. The mechanism for selecting each probability will be explained in detailed in following sections.

In the conventional HS algorithm, a new X'_i value is randomly chosen with 1-HMCR probability from the possible range of values. On the other hand, an X'_i value is typically chosen from the ith column of the HM with HMCR probability. In the present approach, the same reasoning is applied. However, the trajectory continuity must be ensured, which means that jumps over the cells are not allowed.

The problem can be solved by introducing some changes: a new X'_i value is randomly chosen with 1-HCMR probability from the set of the nearest neighbors of that decision variable (i.e. all free cells adjacent to the cell addressed by the decision variable). If the new value is chosen from the entire possible range of values, the trajectory continuity in not ensured. Moreover, a random value from the ith column is selected according to the unvisited neighbor cells with HMCR probability. If there are not unvisited neighbor cells to choose from, a random neighbor in the HM is chosen (see Eq. 3). It should be notice that each node can or cannot have neighboring, the set that contains all siblings is denoted by S, such that,

$$S = \bigcup_{s \in S} s$$

$$X'_i \leftarrow \begin{cases} X'_i \in \begin{cases} S_i \in X_i \ \exists s \in X_i \\ S_i \in X_i \ \nexists s \in X_i \end{cases}, & w.p\ HMCR \\ X'_i \in S_i, & w.p\ 1 - \mathrm{HMCR} \end{cases} \quad (3)$$

In addition, each time the probability falls on HMCR, a new pitch adjusted rate (PAR) is applied to the decision variable. Else the decision variable remains unchanged (1-PAR). Usually, the adjustment is based on the displacement of K neighboring values in the candidate set of values. In such case, the pitch adjustment is the displacement of one neighbor within the neighborhood, by adding or subtracting a unit in the admissible neighbors set (see Eq. 4).

$$X''_i \leftarrow \begin{cases} X'_i \pm 1, & w.p\ PAR \\ X'_i, & w.p\ 1 - PAR \end{cases} \quad (4)$$

Update

Every time that a new X'_i vector is created, the Harmony vector cost is computed by using the cost function defined in Equation 1. If the cost computed is better than the worst Harmony vector cost in the HM, the new vector will be added to the HM, and consequently, the Harmony vector with the worst cost will be discarded from the HM matrix. If not, the HM will remain unchanged.

Finalization Criterion

As in other optimization algorithms, the role of the finalization criterion is to stop the optimization process when a determinate condition is achieved.

Since the goal is to improve the results obtained by applying the wave-front planner with backtracking approach, stop criterion can be set when reaching a reasonable number of turns, or even a determinate number of iterations.

The stop criterion was initially set in order to stop when the numbers of turns were lower than resulting previous work. After that, the number of iterations was set to 1000. This iteration upper-bound was obtained by trial, and it was high enough to optimize all the aerial trajectories presented in the next Section.

Results

The new algorithm described in the previous Section has been validated by comparison with the results obtained in [7]. Since the problem deals with the same constraints, variables and workspace, this is the best case study to quantify the improvements in comparison to the method previously presented.

The workspace was based on a rectangular agricultural field with an irrigation system in the middle that is considered as no interest area. An understandable real scenario is shown in Fig. 3.

However, the same methodology could be applied to the surveillance of a CIP area. A feasible example is a nuclear plant. Where the cooling tower area might be understand as an area not to cover.

Figure 3. Agricultural field locate in Madrid, Spain.

Moreover, since the mission has to be performed by 3 drones, each color stroked corresponds to a sub-area from the previous decomposed workspace [7]. The thin red lines map the previous covered trajectories.

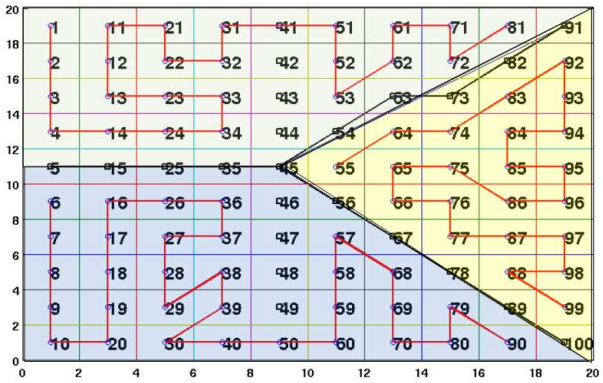


Figure 4. Results previously obtained in [7]. Area 1 (green); Area 2 (blue); Area 3 (yellow).

The new coverage trajectories were computed considering the same restrictions, mainly the assigned discretized sub-area and the predefined starting and ending positions. The m-CPP problem has made possible to reduce the number of heading movements for each single coverage trajectory above the same principles shown in shown in Fig. 5.

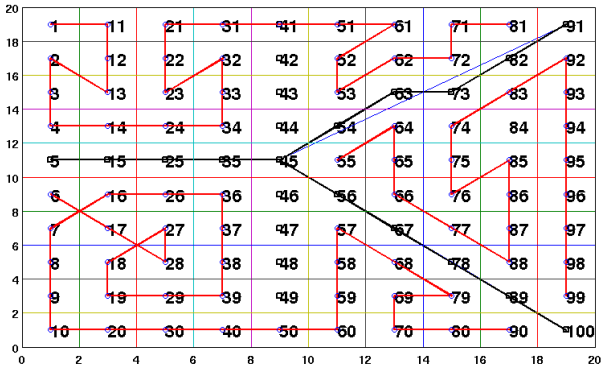


Figure 5. Results obtained with m-cpp method. The areas numeration the same of Fig. 3.

Table I compares both methods regarding to the number of heading movements to carry out the complete coverage trajectory. It should be highlighted that the improvement with respect to the previous method is significant. This is mainly due to the results obtained in Area 2. In this particular case, the number of times that the quad-rotor has to change its direction during a flight was reduced by half.

	Area 1	Area 2	Area 3
Approach in [5]	15	17	16
m-CPP	14	7	14
OR (%)	6.7	59	12.5

Table I. Number of turns (i.e. heading movements) obtained after computing the coverage trajectories in each sub-area with both methods. The optimization rate (OR) enhances the improvement relative to the previous method.

The m-CPP algorithm not only allows optimizing the solution but also shows a rapid evolution during the optimization process. This outstanding point could be better appreciated in Fig. 6.

Finally, the main goal was to achieve a complete (considering no borders) and safe coverage. Therefore, the safety borders were removed, as proposed in [9].

Under these conditions, a complete coverage trajectory was computed with m-cpp in a safe mode, as shown in Fig. 7.

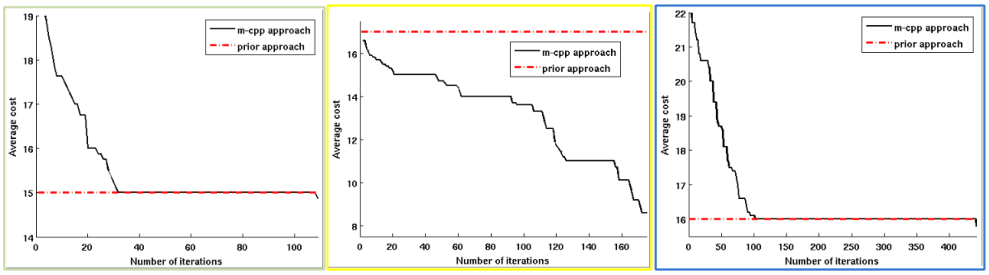


Figure 6. Number iterations versus cost, regarding to results shown in Fig. 5.

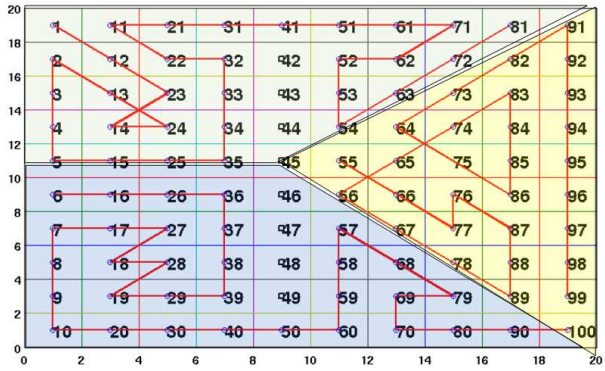


Figure 7. Complete safe coverage with m-cpp.

The last results obtained are summarized in Table II. The three complete coverage trajectories were computed in a Intel's Core 2 Duo at 2.26GHz, and 4GB of memory.

Although computing time is not a hard priority in offline mission planner, the maximum computing time achieved was 2 min approximately, which can be considered as negligible according to the problem complexity, number of nodes, etc.

Concerning to the number of turns - and this is the main cost during the flight - it should be notice that in Areas 1 and 3, this cost was minimized. On the other hand, in Area 2 the number of turn was increased. These coverage cost not depend from the algorithm at all, but from the sub-area/area configuration, shape, etc.

Withal, the total number of turns that the fleet has to perform is 35, with or without safety borders. Thus, the flying cost remains constant. As expected, in this last coverage mission, a full coverage - no cells are left to cover - was obtained in a safe way, which makes the mission optimal in terms of coverage.

	Area 1	Area 2	Area 3
Minimum Turns	13	9	13
Nodes	31	29	32
Time (s)	117	109	51

Table II. Results obtained from the complete coverage with three UAVs.

Conclusions

A novel algorithm based on a computational technique known as Harmony Search has been presented. Results have shown that it is adequate for solving Coverage Path planning problem with aerial vehicles obtaining better result that previous approaches.

The enhancements introduced in the algorithm allows considering time-sealed restrictions such as the position of all the drones with a very reasonable performance.

In opposite to previous approach, the solution of the algorithm can be considered as a safe flight plan for UAV's, a basic requirement in any application is.

A weakness of the algorithm is that the computation time highly grows with the workspace dimension; some improvements in parallel computing should be introduced to minimize this effect.

Acknowledgments

This work have been supported by the Robotics and Cybernetics Research Group at Technique University of Madrid (Spain), and funded under the projects 'ROTOS: Multi-Robot system for outdoor infrastructures protection', sponsored by Spain Ministry of Education and Science (DPI2010-17998), and 'Robot Fleets for Highly Effective Agriculture and Forestry Management', sponsored by the European Commission's Seventh Framework Programme (NMP-CP-IP 245986-2 RHEA). The authors want to thank all the project partners as well.

References

[1] João Valente, Antonio Barrientos, and Jaime Del Cerro. Coverage path planning to survey a large outdoor areas with aerial robots: A comprehensive analysis. In Daisuke Chugo and Sho Yokota (eds.), Introduction to Modern Robotics II, chapter 12. iConcept Press, Annerley, Australia, 2011.

[2] L. Jiao and Z. Tang, A Visibility-based Algorithm for Multi-robot Boundary Coverage, International Journal of Advanced Robotic Systems, Vol. 5, No. 1, pp. 63-68, 2008.

[3] M. Ozkan, G. Kirlik, O. Parlaktuna, A. Yufka and A. Yazici, A Multi-Robot Control Architecture for Fault-Tolerant Sensor-Based Coverage, International Journal of Advanced Robotic Systems, Vol. 7, No. 1, pp. 67-74, 2010.

[4] Choset, H. Coverage for robotics - a survey of recent results, Annals of Mathematics and Artificial Intelligence, Vol. 31, No.1-4, pp. 113-126, 2001.

[5] Marija Đakulovic, Ivan Petrovic, Complete coverage path planning of mobile robots for humanitarian demining, Industrial Robot: An International Journal, Vol. 39, Issue 5, 2012.

[6] Y. Li, H. Chen, M. Joo Er, X. Wang, Coverage path planning for UAVs based on enhanced exact cellular decomposition method, Mechatronics, Vol. 21, Issue 5, pp. 876-885, 2011.

[7] J. Valente, A. Barrientos, J. Del Cerro, C. Rossi, J. Colorado, D. Sanz, M. Garzon, Multi-robot visual coverage path planning: Geometrical metamorphosis of the workspace through raster graphics based approaches, in: B. Murgante, O. Gervasi, A. Iglesias, D. Taniar, B. Apduhan (Eds.), Computational Science and Its Applications – ICCS 2011, Vol. 6784 of Lecture Notes in Computer Science, Springer Berlin/Heidelberg, pp. 58-73, 2011.

[8] A. Barrientos, J. Colorado, J. d. Cerro, A. Martinez, C. Rossi, D. Sanz, J. Valente, Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots, Journal of Field Robotics, Vol. 28, issue 5, pp. 667-689, 2011.

[9] J. Valente, A. Barrientos, J. del Cerro, C. Rossi, D. Sanz, and M. Garzón, C. Rossi, Practical issues and improvements in farmland coverage with aerial vehicles, First Workshop on Research, Development and Education on Unmanned Aerial Systems (RED-UAS 2011), Seville, Spain, November 2011.

[10] Z. W. Geem, J. H. Kim, G. Loganathan, A new heuristic optimization algorithm: Harmony search, Simulation, Vol. 76, no.2, pp. 60–68, 2001.

[11] I. Landa-Torres, S. Gil-Lopez, S. Salcedo-Sanz, J. Del Ser, J. A. Portilla-Figueras, A Novel Grouping Harmony Search Algorithm for the Multiple-Type Access Node Location Problem, Expert Systems with Applications, vol. 39, no. 5, pp. 5262–5270, 2012.

[12] J. Del Ser, M. Matinmikko, S. Gil-Lopez and M. Mustonen, Centralized and Distributed Spectrum Channel Assignment in Cognitive Wireless Networks: A Harmony Search Approach, Applied Soft Computing, vol. 12, no. 2, pp. 921-930, 2012.

[13] I. Landa-Torres, E. G. Ortiz-Garcia, S. Salcedo-Sanz, M. J. Segovia, S. Gil-Lopez, M. Miranda, J. M. Leiva-Murillo, J. Del Ser, Evaluating the Internationalization Success of Companies using a Hybrid Grouping Harmony Search – Extreme Learning Machine Approach, IEEE Journal on Selected Topics in Signal Processing, Vol. PP., N. 99 (early access), 2012.

[14] J. Fourie, S. Mills, R. Green, Harmony filter: A robust visual tracking system using the improved harmony search algorithm, Image and Vision Computing, Vol. 28, Issue 12, 2010.

Multi-Robot Exploration in the Polygonal Domain

Tomáš Juchelka, Miroslav Kulich, and Libor Přeučil
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Technicka 2, 166 27 Prague
Czech Republic

Abstract

This paper addresses the problem of multi-robot exploration, where the particular robots are equipped with range sensors (e.g. laser range finders). Contrary to the majority of current methods, the presented approach uses a polygonal representation of the explored environment since it is memory effective and planning algorithms are fast on it. The approach is based on the Vatti algorithm for polygon clipping which is modified in order to store and manage information relevant to the exploration process. Moreover, several state-of-the-art exploration strategies were implemented for the polygonal representation and the developed framework was used to quantitatively compare and to evaluate the implemented strategies in various environments.

1 Introduction

Exploration of an unknown environment by a team of mobile robots is a fundamental problem in mobile robotics as it consists of standard tasks being studied by the robotic community – from localization and mapping, motion and path planning to high-level planing, cooperation, coordination, and communication. In this task, mobile robots are autonomously driven according to acquired sensory information in order to build a map of the environment. The exploration algorithm can be defined as an iterative procedure consisting of model updating with newly measured sensory data, selection of a new goal for each robot based on the current knowledge of the environment, and subsequent navigation to this goal. A natural condition is to perform the exploration with a minimal usage of resources, e.g. trajectory length, time of exploration, or energy consumption.

In this paper, several assumption are made in order to simplify the problem and to focus on representation of the actual knowledge about the environment in which the robots operate. We particularly assume that positions of the robots are known with enough precision, the robots are equipped with laser range finders, operate in 2D, and the system is centralized, i.e. the robots share a common map and there is a central element assigning goals to the particular robots. Moreover, we do not solve communication problems. Instead, we suppose that communication between the robots and the central element is always established and faultless.

The most popular approach to both single-robot and multi-robot exploration is frontier-based exploration introduced by Yamauchi (Yamauchi 1998) and further extended by many researchers (Wurm, Stachniss, and Burgard 2008; Burgard et al. 2005; Amigoni 2008; Holz et al. 2010). The approach is based on occupancy grids where the working space is divided into small cells and each cell stores information about the corresponding piece of the environment in the form of a probabilistic estimate of its state.

Several authors do not build an exact metric map. Instead, they incrementally create topological information about the space in the form of a graph. Frontier-based modification of *Sensor-based Random Tree*, a probabilistic strategy, which represents a roadmap of the explored area with an associated safe region is presented in (Freda and Oriolo April). The approach has been generalized in (Franchi et al. April), where *Sensor-based Random Graph* is constructed. Feature-based map is used in (Newman, Bosse, and Leonard Sept). Moreover, a free space is represented by a set of so-called markers, which are connected based on visibility constrain.

Combination of metric (in the form of occupancy grid) and topological maps is presented in (Poncela et al. 2002). The metric map is built first, while a hierarchical structure is created over it leading to topological map construction. The opposite approach (*Spatial Semantic Hierarchy*) defines distinctive places and paths in order to build a topological description, while geometric knowledge is assimilated onto the elements of this description (Kuipers and Byun 1991).

(Shen, Michael, and Kumar May) propose a stochastic differential equation-based algorithm. They use a system of particles with Newtonian dynamics to determine regions for further exploration in 3D for unmanned aerial vehicle.

Also, exploration based on a polygonal representation is not new, although it is used for a single robot only. (González-Baños and Latombe 2002) introduce a concept of a *Safe Region*, the largest region guaranteed to be obstacle-free given the history of sensor readings. A map is iteratively built by executing union operations over successive safe regions. (Dakulović, Ileš, and Petrović 2011) extends Ekman's approach (Ekman, Torne, and Stromberg Apr). For each scan, a polygon is created using line-fitting on scan points, then Vatti's algorithm (Vatti 1992) is used to compose particular polygons. However, quantitative evaluation and performance comparison are missing in these papers, they present only few pictures with obtained polygonal maps.

The presented general framework for multi-robot exploration based on a polygonal representation of the operating environment brings the following contributions:

- To the best of authors' knowledge, there is no other exploration system that builds and uses a polygonal representation for multiple robots.
- Several multi-robot exploration strategies have been implemented within the presented framework. This shows that existing strategies designed for occupancy grids can be adopted for a polygonal representation in a straightforward way.
- The implementation of several strategies within an unifying framework allowed us to perform a comprehensive evaluation and comparison of the strategies in various environments and present quantitative results of this comparison.

The rest of the paper is organized as follows. The problem definition is presented in Section 2, while the approach itself is introduced in Section 3. The implemented strategies are described in Section 4. Evaluation of the results and discussions are presented in Section 5. Finally, Section 6 is dedicated to concluding remarks.

2 Problem definition

Exploration is the process in which robots autonomously operate in an unknown environment with the aim to built a map of it. The map is built incrementally as actual sensor measurements are gathered and it serves as a model of the environment for further exploration steps.

The exploration algorithm consists of several steps that are repeated until some unexplored area remain. The process starts with reading actual sensor information by individual robots. After some data processing, the existing map is updated with this information. New goal candidates are then determined and goals for particular robots are assigned using a defined cost function. This assignment is called exploration strategy and can be formalized as follows.

Let the current n goals be located at positions $\boldsymbol{G} = \{g_1, \ldots, g_n\}$ and the current robot poses be $\boldsymbol{R} = \{r_1, \ldots, r_m\}$. The problem is to determine a goal $g \in \boldsymbol{G}$ for each robot $r \in \boldsymbol{R}$ that will minimize the total time spent (or the maximal traveled distance) by individual robots to explore the whole environment.

Having assigned the goals to the robots, the shortest path from the robots to the goals are found. Finally, the robots are navigated along the paths. The whole exploration process is summarized in Algorithm 1.

while *unexplored areas exist* **do**
    read current sensor information;
    update map with the obtained data;
    determine new goal candidates;
    assign the goals to the robots;
    plan paths for the robots;
    move the robots towards the goals;

Algorithm 1: The exploration algorithm

In this paper, we follow Yamauchi's frontier based approach (Yamauchi 1998), which assumes that the next best view (goal) lies on the border between free and unexplored areas (this border is called *frontier*).

3 Polygonal domain

In the presented approach, the information about the environment is approximated by a polygon with holes (i.e., the outer polygon representing a border of the working area and containing obstacles – holes). This polygon $\mathcal{P}$ is, similarly to (Dakulović, Ileš, and Petrović 2011), incrementally created as a union of polygons $\mathcal{P}_i$ representing sensor measurements (scans) taken during the mission: $\mathcal{P} = \cup_{i=0}^{t} \mathcal{P}_i$, where t is the actual time.

The particular polygon $\mathcal{P}_i$ is created from a range data $\mathcal{R}_i$ that are typically represented as a vector of points. This is a standard task and many approaches for polygon building from sensory data have been developed. The combination of three algorithms is used:

- *Successive Edge Following* (Siadat et al. 1997) splits scan data into clusters representing particular objects,
- *Ramer–Douglas–Peucker* algorithm (Hershberger and Snoeyink 1992) smooths objects' boundaries into piecewise linear curves (polylines),
- *Least Squares Fit* finds parameters of lines to best fit the scan data.

Finally, the position of the sensor is added between the first and the last point and successive polylines are connected. The resulting polygon represents a free space as detected by the measurement. We distinguish between two types of edges: those representing an obstacle and those that were added in the last step, which represent frontiers.

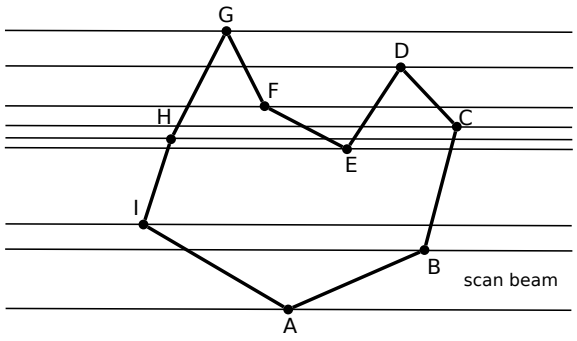


Figure 1: Vatti's representation of a polygon. The vertices *A,B,C,D* form the right bound and the vertices *A,I,H,G* form the left bound. The vertex *A* is the local minimum and the vertex *D* is the local maximum.

The union of polygons obtained from different measurement poses can be computed by Vatti clipping algorithm (Vatti 1992)[1]. The algorithm can handle large sets of

[1]In our implementation, we use the Clipper library (Johnson 2012), which is an open-source polygon clipping library based on Vatti clipping algorithm. The library performs the boolean clipping operations - intersection, union, difference, and XOR. Moreover, it performs polygon offsetting.

polygons, polygons with holes, and self-intersecting polygons. On the other hand, adding information about edge type is not straightforward since this information has to be preserved by clipping operations.

The Vatti algorithm processes both involved polygons by a sweep line starting at the lowermost vertex and going upwards passing through all vertices of the polygons. Bounds — sequences of consecutive edges starting at a local minimum and ending at a local maximum — are formed during this process. Left and right bounds are distinguished with respect to their positions to the polygon's interior. A polygon described with this notation is shown in Fig. 1

Modifications of clipping

The straightforward approach how to add and preserve information about the edge type lies in modifications of specific parts of the Vatti algorithm. In this case, information about the edge type is stored in vertices adjacent to it. Unfortunately, this approach fails. One of the main reasons is that a relation between output vertices and the input edges can not be determined easily. For example, if two different bounds share their local minimum, a new vertex is added into the output polygon. There is no guarantee which bound the algorithm takes first. When the second bound is processed, its vertex is skipped, because it was already added. The example situation is illustrated in Fig. 2, where v_{min} is the local minimum which is added to the output either in processing edges e_1 and e_2 or e_3 and e_4. Because the first pair of edges is a frontier, while the second one is not, the parameters of the added vertex may be different.

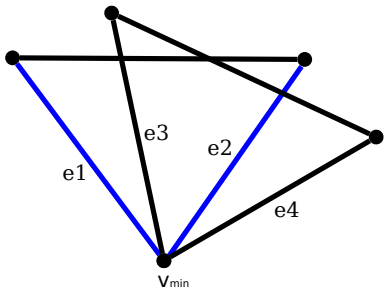


Figure 2: Problem in a local minimum.

Our approach post-processes edges of the output polygon, compares them with edges of the input polygons and assigns them the correct type. The comparison of each output edge is made by computing a penalty function for all edges in the input polygons. The penalty value can be expressed as the sum of all the distances depicted in Fig. 3.

$$P = p_1 + p_2 + |d_1| + |d_2| \quad (1)$$

The distances p_1,p_2 are the perpendicular distances from vertices of e_{orig} to e_{out} and d_1,d_2 are differences in y-axis. Notice that the distance d_1 is considered only if the bottom vertex of e_{out} has lower y-coordinate than the bottom vertex of e_{orig} and the distance d_2 is considered only if the top vertex of e_{out} has higher y-coordinate than the top vertex of e_{orig}. The best input edge, i.e., the edge with the lowest

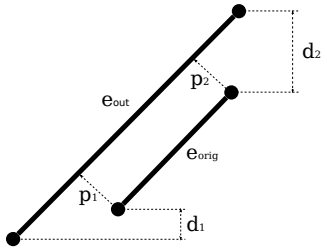


Figure 3: Comparison of edges.

penalty, is considered as correct and the information from the found input edge is copied into the output edge.

This process is much more computationally complex than the clipping algorithm itself. From the knowledge about the clipping algorithm it is possible to do some simplifications that speed up the matching. The clipping algorithm creates the bounds with the edges in a bottom-up fashion starting at the local minimum. The most important fact is that the edges are ordered by y-coordinate. The bounds are also ordered by y-coordinate of their local minimum. These internal structures can be used instead of the original polygons. The edges of the input polygons are processed based on their order in bounds while the following rules are applied:

- The bounds with y-coordinate of its local minimum higher than y-coordinate of the top vertex of the output edge can be completely skipped.
- If y-coordinate of the bottom vertex of the edge from the bound is higher than the top vertex of the output edge then the rest of one bound can be skipped.
- If the penalty value is zero then skip further comparison of the output edge.

These criteria improve the speed of the algorithm significantly. Fig. 4 shows how the simplifications affect the algorithm performance. Although the modifications even with simplifications slow-down the clipping, the approach can be applied on real problems. For example, the biggest map used in the experiments (Hospital section map) contains approximately 1300 vertices and although the clipping with the accelerated modifications is approximately two times slower than the original clipping algorithm, it takes few milliseconds.

Polygon offsetting

The map created by the clipping algorithm is useful for path planning for a point-robot only. If a robot is approximated by a disk, the map (i.e., each obstacle) has to be enlarged by constructing the Minkowski sum of the map with this disk. This can be done by polygon offsetting operation. Management of the edge type is similar to the clipping process. Notice that enlarging can lead to intersecting or self-intersecting polygons and therefore the same post-processing as for polygon clipping is used.

Map representation

It is possible to maintain the knowledge about the environment as the all-in-one map, but this is not robust. Both data

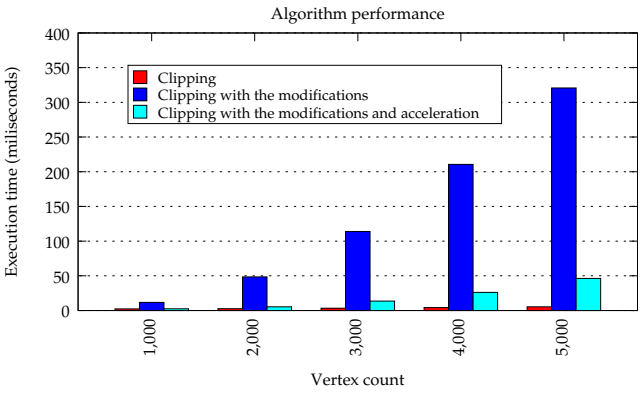


Figure 4: Performance of the modified algorithm.

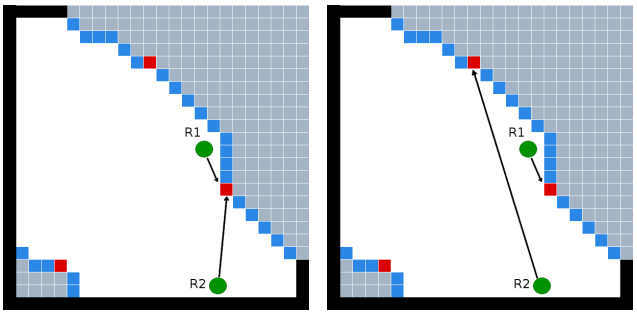


Figure 5: Greedy assignment, (a) two robots exploring the same goal (b) an inefficient assignment of goals.

from a laser range finder and odometry can be noisy, which causes that some frontiers may be generated nearby or inside obstacles. The more robust approach is to represent the maps of a free-space and obstacles separately. Whenever a new scan is added into the map it is added into the free-space map as it is. The scan is next checked whether it contains obstacles. If so, the obstacles are offset (proportionally to the map size and noise) and added into the obstacle map.

Before the map is used for planning, both maps are temporarily combined together into a single map. The modified clipping is performed here and the resulting map contains the information about frontiers. If the resulting edge comes from the free-space map it is considered as a frontier and it is marked as an obstacle otherwise.

The created map contains a large number of vertices. Ramer–Douglas–Peucker algorithm is thus used after each clipping or offsetting in order to reduce this number.

The map is used for goal candidates selection, planning, and evaluation of the candidates in each exploration step. The selection process determines the candidates so that they lie on frontiers, the distance of a candidate to its neighbors is twice a sensor range, and all frontiers are inside the union of circles with centers in the candidates and radius equal to the sensor range. This guaranties that all frontiers will be explored (i.e., it will be detected whether a frontier lies in a free space or in any obstacle) after visiting all candidates.

Planning consists of two steps: a visibility graph is computed first, followed by several runs of Dijkstra algorithm, which computes shortest distances among each robot and a goal candidate[2]. The cost function for evaluation of goal candidates is then simply this distance.

4 Exploration strategies

Several exploration strategies have been implemented within the presented exploration framework. The following paragraphs give an overview of these methods and discuss necessary modifications for the polygonal domain.

[2]Usage of all-pairs shortest path algorithm (e.g. Floyd-Warshall or Johnson's algorithm) is not effective in this case since we don't need to compute distances among the goal candidates.

Greedy approach

A simply and easily implementable strategy is described in (Yamauchi 1998) – each robot greedily heads towards the best (according to a cost function) goal without any coordination between robots. The strategy is not much optimal since one goal can be selected and explored by many robots as depicted in Fig. 5(a). To avoid this inefficiency it is possible to discard already selected goals from the further selection. This is used in the Broadcast of Local Eligibility (BLE) assignment algorithm developed by Werger & Mataric (Werger and Mataric 2001), see Algorithm 2.

while *any robot remains unassigned* **do**
  find the robot-goal pair (i, j) with the highest utility;
  assign the goal j to the robot i and remove them from the consideration;

Algorithm 2: BLE assignment algorithm.

Nevertheless, it is still a greedy algorithm, which not necessarily produces the optimal solution. The solution depends on the order of the robot-goal assignments. Fig. 5(b) depicts an example of an inefficient targets assignment.

Hungarian method

The Hungarian method firstly introduced in (Kuhn 1955) is more sophisticated. It is an optimization algorithm which solves the worker-task assignment. The assignment can be written in a form of the $n \times n$ matrix C, where the element $c_{i,j}$ represents the cost that the j-th task has been assigned to the i-th worker. The Hungarian method finds the optimal assignment for the given cost matrix C in $O(n^3)$.

The algorithm requires the number of robots to be the same as the number of goals which can not be guaranteed throughout the exploration. If the number of robots or goals is lower it is possible to add imaginary robots or goals to satisfy the assumption. They have assigned a fixed cost, so they don't affect the real robots/goals. In the selection, the imaginary robots and targets are skipped. This strategy doesn't assign the same goal to different robots and it doesn't depend on the order of selection.

K-means clustering

In the majority of multi-robot tasks, robots start from the same area, e.g., from the building entrance. It leads to an exhaustive exploration of the starting area during the first phase of exploration. In search and rescue it is preferable that the robots quickly create an outline of the map and then they focus on the individual parts of the environment. (Solanas and Garcia 2004) present a strategy based on K-means clustering that divides an unknown space into K regions, where K is the number of robots. The particular regions are then assigned to the closest robots. After the assignment, each robot chooses a frontier according to a predefined cost function. The cost of the frontier F_j for the robot R_i assigned to the region ζ_i is defined as:

$$c_{i,j} = \begin{cases} \Delta + e(F_j, C_i) + o_{i,j} & F_j \notin \zeta_i \\ d(F_j, R_i) + o_{i,j} & F_j \in \zeta_i \end{cases}$$

where Δ is a constant penalization representing the diagonal length of the map, e is the euclidean distance, C_i is the centroid of the region, d is the real path cost defined by any path planning algorithm and $o_{i,j}$ is the accumulated penalization increasing the cost when the frontier has been already selected.

The frontier that does not belong to the assigned region receives a high penalization Δ, so it can happen that there is no frontier in the assigned region, in that case robots select the closest frontier to their region. As the result, robots tend to work separately in their assigned regions. If the assigned region is not directly accessible, other regions are explored on the way to the assigned one. Robots explore all these separated regions simultaneously because each robot heads to its own region. This leads to a dispersion of robots in the environment and different parts of the environment are explored at similar speeds.

In general, the K-means algorithm consists of the following steps.

1. Randomly choose K centroids C_i where $1 \leq i \leq K$.
2. Classify each cell to the class ζ_i of its closest centroid C_i.
3. Determine a new centroid for each class.
4. If all the centroids didn't change, finish. Otherwise, continue with step 2.

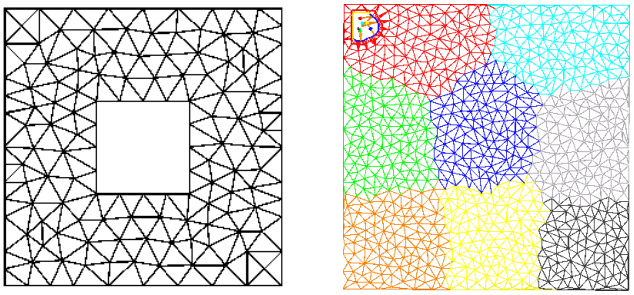
The algorithm works with a set of points, which is not directly available for the polygonal representation. The idea is therefore to sample the polygons with points using a triangular mesh generator[3]

Fig. 6(a) shows an example of a generated triangle mesh on a polygon with hole, while Fig. 6(b) visualizes a triangle mesh with applied K-means clustering.

5 Results

The proposed framework together with the aforementioned strategies has been implemented using ROS (Quigley et al. 2009) as a communication middleware and the strategies have been evaluated. The experiments have been performed in simulation using maps with various sizes and structures, see Figure 7.

[3]We use the Triangle library (Shewchuk 1996).



(a) (b)

Figure 6: Triangular meshes: (a) an example of triangle mesh; (b) a triangle mesh with applied K-means clustering algorithm at the beginning of an exploration with 8 robots using the K-means strategy;

The Empty map 7(a) has been created to simulate a trivial case of a big room without obstacles. The Arena map 7(b) represents a slightly structured environment with large corridors and rooms. The Jari-huge map 7(c) represents the real administrative building with many separated rooms. The hospital small map 7(d) is a part of the hospital-section map from the Stage simulator representing another building.

All the experiments were examined on the same hardware with a quad-core processor on 3.30 GHz, 8 GB RAM running x86_64 GNU/Linux kubuntu 3.0.0-20, ROS version electric and gcc version 4.6.1.

The considered numbers of robots are $m = \{4, 6, 8\}$, while the sensor range is set to $\rho = 5$ meters with 270° field of view. The robots are controlled using the SND driver (Durham and Bullo 2008). The planning period has been set to 1 second. Each experiment was repeated 30 times, which means that the number of experimental runs is 1440 in total.

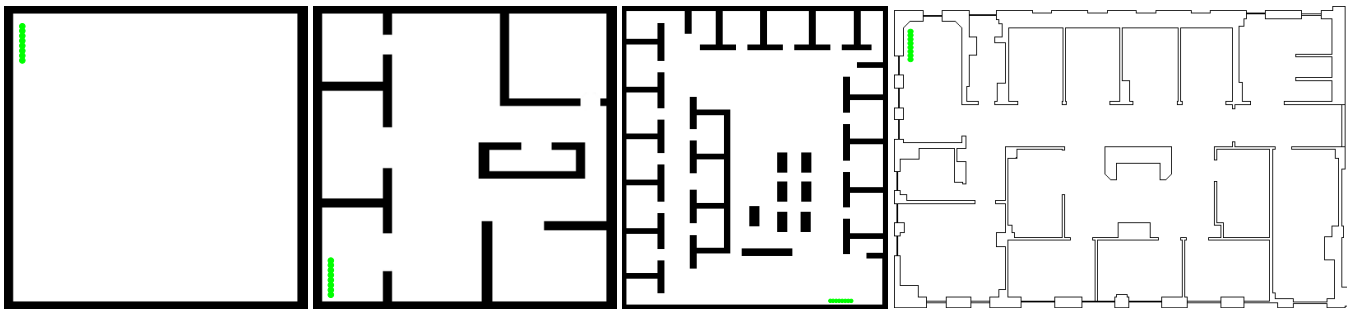
As the computations are not time consuming, the experiments are speeded up 3 times in the Stage's configuration file. This acceleration has no effect on the quality of the exploration but it has its limits in the computational complexity. The chosen acceleration is therefore a trade-off between the run time and the system load.

The strategies were evaluated from several points of views. The first criterion is the number of planning steps t_{exp} which corresponds to the exploration time (i.e., the time needed to explore the whole environment) assuming that the planning period is constant. The maximal distance d_{max} travelled by a robot is the second criterion. The results are depicted in Tables 1–5 and on Figs. 8–12 .

Comparison of the strategies

Empty map There is not big difference in the exploration time of the Hungarian and the K-means strategy. A surprise is a good performance of the Greedy strategy. Because there is no obstacle in the map the robots naturally disperse in the environment as they head directly towards goals with no need of coordination.

The maximal distance d_{max} apparently indicates the ef-



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

Figure 7: Maps used in the experiments. The starting positions are marked with the green circles. (a) Empty map with dimensions 50x50 m; (b) Arena map with dimensions 50x50 m. (c) Jari-huge map with dimensions 52.5x60 m; (d) Hospital-small map with dimensions 138x110.75 m.

fect of the map segmentation in the K-means strategy. The robots focus their regions which reduces the maximal travelled distance by any robot that is smaller by 7% than in Hungarian strategy but it has longer exploration time by 5.4% for 6 robots. For 8 robots, the K-means strategy is better in both of the parameters. It has also the lowest standard deviation so the solution has not a big variance.

Arena map The results are similar to the results on Empty map with one difference. Although for 4 and 6 robots the best results are achieved by the K-means strategy, an interesting effect can be seen for 8 robots where the results are worse. The map partitioning forces the robots to explore small regions partially spread over several rooms, which slows down the exploration while for the lower number of robots the regions are bigger and contain the whole rooms.

Jari-huge map The K-means strategy behaves poorly on this map. Small rooms generate regions split over several separated rooms, so each robot must explore the assigned rooms in the region. The behavior that each robot visits a single room is in fact desired, but it requires correctly generated regions including the whole rooms, not only their parts. The Hungarian strategy significantly outperforms other strategies, e.g. for 8 robots it is faster by 28% than the Greedy strategy, by 16% than the Greedy-ble strategy, and also by 12% than the K-means strategy.

Hospital-small map The results are quite balanced except the Greedy strategy. The exploration time and the maximal traveled distance is slightly better for the K-means strategy and 4 robots. The K-means strategy is better in the mean effort parameter. This experiment confirms that there exists a combination of the map size and the number of robots in which the K-means strategy yields bad results (6 robots in this case) because the number of robots corresponds to sizes of regions that may be inappropriate for the given map. The Hungarian strategy again seems to be the best strategy.

"Extreme" case The Hospital-section map is used to simulate an extreme case with many robots on a huge map. The exploration is performed by 10 robots with no other change against the previous experiments. This experiment can not be speeded up because of the computational complexity, so the number of repetitions is lower, i.e., 10 trials.

Fig. 12(a) shows an example of the created map at the end of the exploration.

The results are in accordance with the results of the previous experiments. The worst and the most varying strategy is the Greedy strategy. The Greedy-ble strategy is better but still not optimal. The best results are achieved by the Hungarian strategy which has also the shortest d_{max}. With this number of robots and the K-means strategy it is possible that a robot visits a lot of goals from different regions until it reaches its region because its assigned region lies on the other side of the map. Such a robot travels a long distance especially when new shorter paths are frequently found.

Complexity

Several experiments were performed to compare memory consumption and time complexity of a polygonal representation with a grid approach. Assume the Hospital-section map, which occupies an area of $14253\ m^2$ and its polygonal representation contains 1226 vertices (see also Fig. 12(b)). If a grid with a cell size equal to $5\ cm$ is used to represent the same environment, 12076800 cells is needed. This leads to much higher computational complexity of the exploration process as shown in Table 6. This table depicts times for environments of various sizes.

	Polygonal		Grid	
S $[m^2]$	t_{plan} $[ms]$	t_{total} $[ms]$	t_{plan} $[ms]$	t_{total} $[ms]$
1000	10	35	274	471
3000	22	50	985	1224
5000	14	70	2141	2410
10000	45	116	3369	3854
13000	36	132	4902	5611

Table 6: Comparison of polygonal and grid representations.

6 Conclusion

The paper introduces a polygonal approach for multi-robot exploration. The mapping process is based on a standard library for polygon clipping, so it is robust and fast. This enables to perform exploration with higher number of robots, in larger experiments and make re-planning faster than possible in a grid-based approach. The most challenging task was to modify the clipping library in order to work with the frontiers. After several unsuccessful attempts to change the clipping algorithm a compromise solution was found in the edge matching.

Several goal-selection strategies were implemented and compared according to various criteria. The best results were achieved by the Hungarian strategy. The K-means strategy was expected to dominate among the other strategies especially in the mean effort parameter. Unfortunately, this criterion does not distinguish locations in the map in which the exploration happens and therefore the difference against the other strategies is not so significant.

7 Acknowledgments

This work has been supported by the Technology Agency of the Czech Republic under the project no. TE01020197 "Centre for Applied Cybernetics 3".

References

Amigoni, F. 2008. Experimental evaluation of some exploration strategies for mobile robots. In *ICRA*, 2818–2823.

Burgard, W.; Moors, M.; Stachniss, C.; and Schneider, F. 2005. Coordinated multi-robot exploration. *IEEE Transactions on Robotics* 21(3):376–378.

Dakulović, M.; Ileš, Š.; and Petrović, I. 2011. Exploration and mapping of unknown polygonal environments based on uncertain range data. *Automatika–Journal for Control, Measurement, Electronics, Computing and Communications* 52(2).

Durham, J. W., and Bullo, F. 2008. Smooth nearness-diagram navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 690–695.

Ekman, A.; Torne, A.; and Stromberg, D. Apr. Exploration of polygonal environments using range data. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 27(2):250–255.

Franchi, A.; Freda, L.; Oriolo, G.; and Vendittelli, M. April. The sensor-based random graph method for cooperative robot exploration. *Mechatronics, IEEE/ASME Transactions on* 14(2):163–175.

Freda, L., and Oriolo, G. April. Frontier-based probabilistic strategies for sensor-based exploration. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 3881–3887.

González-Baños, H. H., and Latombe, J.-C. 2002. Navigation strategies for exploring indoor environments. *International Journal of Robotic Research* 21(10-11):829–848.

Hershberger, J., and Snoeyink, J. 1992. Speeding up the douglas-peucker line-simplification algorithm. Technical report, University of British Columbia, Vancouver, BC, Canada, Canada.

Holz, D.; Basilico, N.; Amigoni, F.; and Behnke, S. 2010. Evaluating the efficiency of frontier-based exploration strategies. In *ISR/ROBOTIK*, 1–8.

Johnson, A. 2012. Clipper - an open source freeware polygon clipping library. `http://www.angusj.com/delphi/clipper.php`.

Kuhn, H. W. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2:83–97.

Kuipers, B., and Byun, Y.-T. 1991. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and Autonomous Systems* 8(1-2):47–63.

Newman, P.; Bosse, M.; and Leonard, J. Sept. Autonomous feature-based exploration. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 1, 1234–1240 vol.1.

Poncela, A.; Pérez, E. J.; Bandera, A.; Urdiales, C.; and Hernández, F. S. 2002. Efficient integration of metric and topological maps for directed exploration of unknown environments. *Robotics and Autonomous Systems* 41(1):21–39.

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

Shen, S.; Michael, N.; and Kumar, V. May. Autonomous indoor 3D exploration with a micro-aerial vehicle. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 9–15.

Shewchuk, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*. Springer-Verlag. 203–222.

Siadat, A.; Kaske, A.; Klausmann, S.; Dufault, M.; and Husson, R. 1997. An optimized segmentation method for a 2d laser-scanner applied to mobilerobot navigation. In *3rd IFAC Symposium on Intelligent Components and Instruments for Control Aapplications*, 153–158.

Solanas, A., and Garcia, M. A. 2004. Coordinated multi-robot exploration through unsupervised clustering of unknown space. In *International Conference on Intelligent Robots and Systems*.

Vatti, B. R. 1992. A generic solution to polygon clipping. *Communications of the ACM* 35:56–63.

Werger, B. B., and Mataric, M. J. 2001. Broadcast of local eligibility for multi-target observation. In *Distributed Autonomous Robotic Systems 4*, 347–356. Springer-Verlag.

Wurm, K.; Stachniss, C.; and Burgard, W. 2008. Coordinated multi-robot exploration using a segmentation of the environment.

Yamauchi, B. 1998. Frontier-based exploration using multiple robots. In *Proc. of the Second International Conference on Autonomous Agents*, 47–53.

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$
4	greedy	960.2	790	1186	93.39	567.99	55.904
	greedy-ble	909.8	824	1042	52.65	538.32	39.45
	hungarian	893.0	804	1001	51.06	526.85	33.50
	kmeans	903.5	783	998	44.37	484.84	30.12
6	greedy	744.7	595	923	89.02	458.50	54.94
	greedy-ble	682.6	607	838	49.54	422.43	32.12
	hungarian	667.7	561	921	63.20	412.40	30.15
	kmeans	705.6	665	779	25.96	383.51	21.32
8	greedy	693.6	564	839	74.54	429.67	49.41
	greedy-ble	629.4	517	846	75.36	377.19	45.07
	hungarian	579.2	512	641	34.91	357.16	29.63
	kmeans	575.3	517	618	22.50	310.19	17.03

Table 1: Empty map: Comparison

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$
4	greedy	1278.4	1046	1567	138.76	588.33	65.39
	greedy-ble	1197.5	994	1708	143.63	572.64	58.33
	hungarian	1155.5	988	1305	81.86	556.12	50.08
	kmeans	1139.2	1053	1394	84.95	546.10	50.27
6	greedy	1023.0	741	1382	143.27	473.39	55.86
	greedy-ble	900.4	761	1014	66.75	435.57	36.44
	hungarian	867.1	783	1050	65.23	429.18	31.700
	kmeans	848.4	748	1112	74.34	427.65	37.85
8	greedy	987.5	791	1378	150.74	439.90	57.56
	greedy-ble	807.3	677	977	82.61	395.00	47.62
	hungarian	748.3	627	985	77.04	374.67	44.65
	kmeans	788.0	681	925	72.37	403.98	31.93

Table 2: Arena map: Comparison

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$
4	greedy	558.4	459	750	78.75	247.85	34.89
	greedy-ble	489.2	433	568	36.75	218.17	19.55
	hungarian	473.4	406	553	39.80	210.93	19.68
	kmeans	524.0	470	573	31.10	235.21	15.68
6	greedy	488.8	365	848	138.15	208.14	52.23
	greedy-ble	369.9	311	439	31.62	165.14	17.27
	hungarian	336.2	301	402	26.84	150.98	13.34
	kmeans	365.7	334	453	26.59	163.22	12.77
8	greedy	400.9	285	573	79.82	176.38	30.51
	greedy-ble	345.6	281	471	42.30	159.77	20.61
	hungarian	289.8	258	331	19.70	133.43	9.58
	kmeans	328.5	285	371	21.96	149.48	12.66

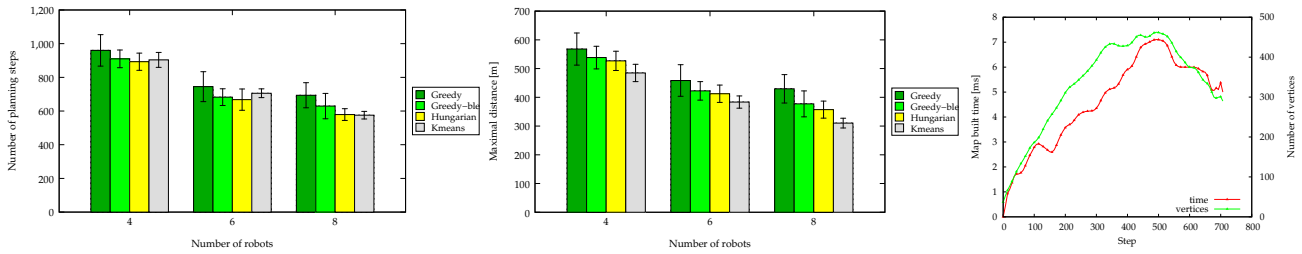
Table 3: Jari-huge map: Comparison

Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$
4	greedy	798.5	704	901	62.97	455.31	39.09
	greedy-ble	736.2	658	876	64.56	425.88	40.18
	hungarian	726.1	648	861	55.39	430.05	30.26
	kmeans	728.1	686	781	28.88	423.44	19.47
6	greedy	644.4	538	779	77.01	374.48	53.31
	greedy-ble	605.9	543	672	45.99	364.79	25.34
	hungarian	576.1	532	611	29.13	350.71	14.66
	kmeans	637.2	605	667	19.04	384.39	13.18
8	greedy	568.9	500	653	49.18	343.54	31.74
	greedy-ble	567.7	519	628	35.23	345.94	25.43
	hungarian	533.9	458	613	44.92	331.44	23.53
	kmeans	560.7	519	606	26.92	346.78	18.69

Table 4: Hospital-small map: Comparison

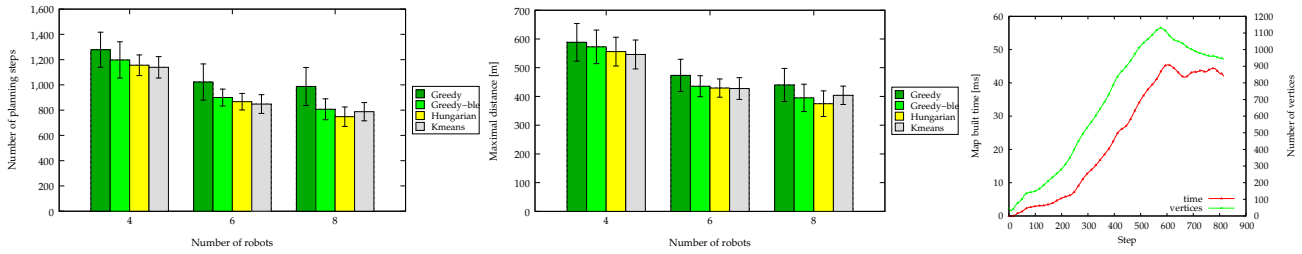
Robots	Strategy	t_{exp}	t_{min}	t_{max}	$\sigma_{t_{exp}}$	d_{max}	$\sigma_{d_{max}}$
10	greedy	1365.7	1195	1507	158.05	710.13	81.54
	greedy-ble	1221.7	1148	1294	40.17	697.28	35.22
	hungarian	1098.8	1051	1168	33.27	625.72	24.71
	kmeans	1222.7	1142	1325	48.95	691.16	26.91

Table 5: Hospital-section map: Comparison



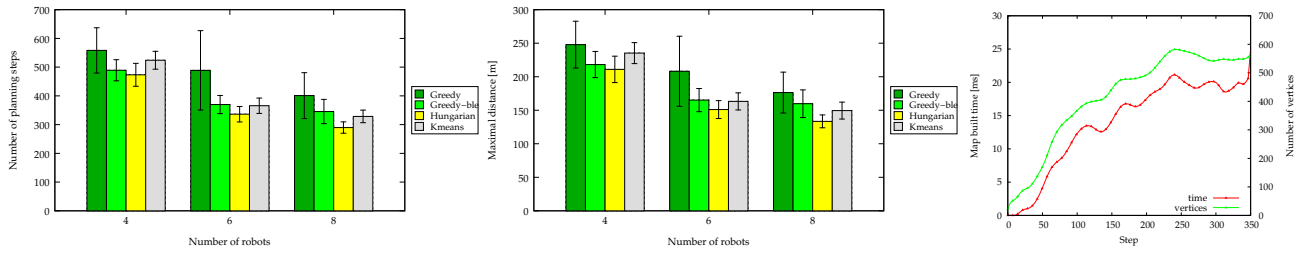
(a) (b) (c)

Figure 8: Empty map (a) - planning steps comparison. (b) - maximal distance comparison. (c) - average map built time and the number of vertices.



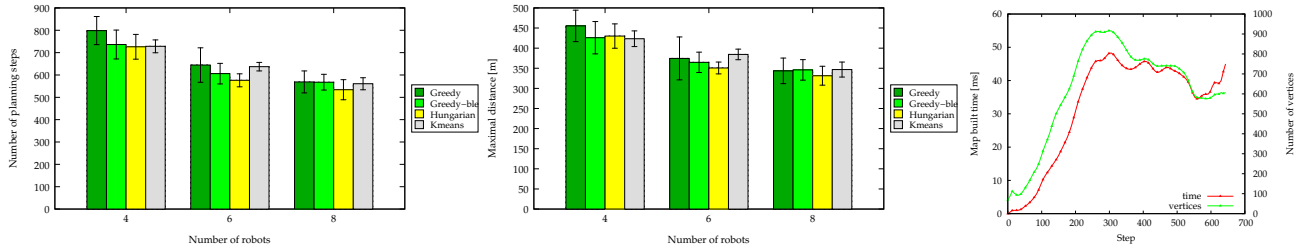
(a) (b) (c)

Figure 9: Arena map (a) - planning steps comparison. (b) - maximal distance comparison. (c) - average map built time and the number of vertices.



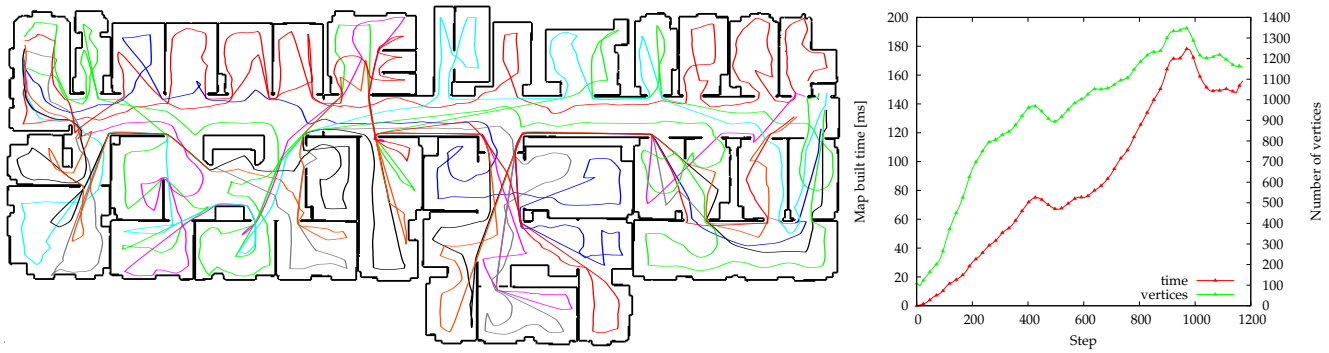
(a) (b) (c)

Figure 10: Jari-huge map (a) - planning steps comparison. (b) - maximal distance comparison. (c) - average map built time and the number of vertices.



(a) (b) (c)

Figure 11: Hospital-small map (a) - planning steps comparison. (b) - maximal distance comparison. (c) - average map built time and the number of vertices.



(a) (b)

Figure 12: Hospital section map: (a) traversed trajectories for 10 robots and Hungarian strategy; (b) the average map built time with the number of vertices.

Path Planning in Dynamic Environments with the Partially Observable Canadian Traveller's Problem

Mikko Lauri and **Risto Ritala**
Department of Automation Science and Engineering
Tampere University of Technology
P.O. Box 692
33101 Tampere, Finland
mikko.lauri@tut.fi, risto.ritala@tut.fi

Abstract

We formulate a path planning problem in a dynamic environment as a Canadian traveller's problem (CTP) with partial observability. The partially observable CTP is a shortest path problem on a directed graph where edge traversal costs depend on partially observable environmental variables. The environment variables evolve according to two-state Markov chains, and information on them is obtained via noisy measurements. We derive an online planning scheme as a combination of finite-step lookahead and a set of cost-to-go problems, for which the underlying graph structure is exploited to find efficient approximate solutions. Empirical evaluation of the planning scheme via simulation studies is presented, and a proposed real-world implementation is discussed.

Introduction

Consider the problem of an agent seeking a shortest path between a start and goal vertex in a graph, where each edge has a cost of traversal. The task is to find a path which reaches the goal while minimizing the sum of edge traversal costs. In case the costs are static and known, the shortest path problem may be solved by static graph search methods such as Dijkstra's algorithm or heuristic-guided search methods such as the A* algorithm. If edge costs vary with time, but are still fully deterministic for any time instant, the problem may be solved with static graph search by expanding it into a time-expanded network (Ahuja et al. 2003). In case the changes in edge costs are not known in advance but incrementally revealed e.g. for one time step ahead, the problem may be solved by repeatedly applying static graph search. A more efficient solution can be reached by noting that the new shortest path problem on the modified graph is typically similar to the one previously solved, and applying incremental search techniques that take advantage of this property. Such methods include e.g. the D* (Stentz 1994) or D* Lite (Koenig and Likhachev 2005) algorithms.

In more general cases, some or all of the edges' traversal costs are initially uncertain. In the Canadian traveller's problem (CTP), first introduced by Papadimitriou and Yannakakis (1991), the cost of traversing an edge is revealed only when a vertex incident to the edge is visited. Once a cost is revealed, it assumes either a finite or infinite value, denoting traversable and non-traversable edges, respectively. As prior information, the probability of each edge being traversable is known. The objective is to minimize the expected total cost of traversal.

Several extensions of the CTP have since been studied. In (Polychronopoulos and Tsitsiklis 1996), edge costs were considered random variables assuming non-negative values. Dynamic programming algorithms to solve the problem were derived in two cases: when edge costs are independent of each other or when they are dependent random variables. Bnaya, Felner, and Shimony (2009) extended the CTP to allow remote sensing. In the remote sensing variant, the agent may query the cost of any edge, whether it is incident to the agent's current vertex or not.

In all of the above examples, the edge costs remain fixed after assuming their initial value, although some of the values are not immediately known by the agent. In (Psaraftis and Tsitsiklis 1993), edge traversal costs are described by a known function dependent on an environmental variable at the source vertex of the edge. The environmental variables are fully observable and independent of each other and evolve in discrete time according to finite state Markov chains. The problem is then to decide when and which edge to traverse or when to wait for a more favourable environmental state.

In this paper, we combine characteristics of the problems presented above to yield a more general partially observable CTP, where the edge costs depend on environment variables which evolve according to a Markov chain and are only partially observable. In addition, there may be multiple sensing modalities for the agent to choose from in order to obtain information on the environment variables. The planning problem is to find a policy for sensing and edge traversal actions that minimizes the expected cost to reach the goal. Such a problem may be considered as a partially observable Markov decision process (POMDP) and it can model shortest path problems in dynamic environments e.g. in robotics. To find a good yet suboptimal solution in a feasible time, we divide the planning into two parts: a k-step lookahead, and a set of cost-to-go problems on the underlying graph. This approach is compared with a state-of-the art online POMDP solution method. Finally, we present an overview of a system design to implement our planning scheme in a dynamic real-world environment.

Partially observable Markov decision processes

A partially observable Markov decision process (POMDP) (Kaelbling, Littman, and Cassandra 1998) may be defined as a tuple $\langle S, A, Z, T, O, R, b_0, \gamma \rangle$, where

- S is a set of possible states the system may be in,
- A is a set of possible actions that may be executed,
- Z is a set of possible observations that may be perceived,
- $T : S \times A \times S \to [0, 1] \equiv p(s'|s, a)$ is the state transition model,
- $O : S \times A \times Z \to [0, 1] \equiv p(z|s', a)$ is the measurement model,
- $R : S \times A \to \mathbb{R}$ is a reward function, specifying the immediate reward when action $a \in A$ is executed in state $s \in S$,
- $b_0 \equiv p(s_0 = s)$ is a probability distribution over S, describing the initial information on the system state at time $t = 0$, and
- $\gamma \in [0, 1)$ is a discount factor.

We assume that S, A and Z are finite sets. In a POMDP, the objective is to maximise the expected cumulative sum of discounted rewards over a specified horizon of time, while the state s_t evolves according to the transition model T. The expected rewards are calculated under the probability distributions, or belief states, b_t, which summarize information on the system state at time t. The space of all possible belief states B is called the belief space of the POMDP. The belief state b is updated based on the action a at time t and the observation result z at time $(t + 1)$, through the state transition and measurement models as

$$\tau(b, a, z)(s') = \frac{1}{\eta} p(z|s', a) \sum_{s \in S} p(s'|s, a) b(s), \quad (1)$$

where η is the normalizing factor denoting the prior probability of observing z;

$$\eta = p(z|b, a) = \sum_{s' \in S} p(z|s', a) \sum_{s \in S} p(s'|s, a) b(s). \quad (2)$$

Value iteration based on Bellman's principle of optimality is a typical method for finding actions that maximize the expected reward in POMDPs (Hauskrecht 2000). Value iteration begins from a horizon $t = 1$ value function $V_1 : B \to \mathbb{R}$ defined as

$$V_1(b) = \max_{a \in A} \sum_{s \in S} R(s, a) b(s) = \max_{a \in A} R_B(b, a), \quad (3)$$

where the term to be maximized is the expected immediate reward of performing action a in belief state b, denoted $R_B(b, a)$. The value function for horizon t is constructed from the value function for horizon $(t - 1)$ by the recursive equation

$$V_t(b) = \max_{a \in A} \left[R_B(b, a) + \gamma \sum_{z \in Z} p(z|b, a) V_{t-1}(b') \right], \quad (4)$$

where $b' = \tau(b, a, z)$. The value function V_t defines the expected sum of discounted rewards over t time steps, for any belief state b. It has been shown (Smallwood and Sondik 1973) that the optimal finite horizon value function is convex and piecewise linear. Therefore it may be represented by a set of $|S|$-dimensional hyperplanes, each defining a linear value function over B for one particular action.

The optimal policy $\pi_t^* : B \to A$ for a finite horizon t specifies an optimal action to perform in belief state b, found by finding the argument a maximizing $V_t(b)$ in equation (4).

Partially observable CTP

We now define a shortest path problem for a directed graph, where edge weights are known functions of partially observable environment variables and there are multiple sensing modalities available to an agent traversing the graph. We call this problem a partially observable Canadian traveller's problem. In (Blei and Kaelbling 1999), it was first shown that a Canadian traveller's problem with dynamic edge traversability may be considered as a POMDP. To see that the problem with multiple sensing modalities is a special case of a general POMDP as well, we make the following redefinitions for the state and actions sets of the POMDP described in the previous section.

Let $G = (V, E)$ denote a directed graph with a finite set of vertices V with cardinality $|V| = n$ and a finite set of edges E. The agent is initially located at the start vertex $v_0 \in V$ and wants to reach the goal vertex $g \in V$. Throughout the paper we shall assume that a path from v_0 to g exists. For every $v \in V$ there exists an environment state w_v. A vector $w = [w_1 \ldots w_n]^{\mathrm{T}} \in W$ then describes the environment state on all vertices.

The state space is $S = V \times W$, such that any state may be represented by an ordered pair $s = (v, w)$ where $v \in V$ and $w \in W$. The possible actions are defined by a collection of subsets $A_v \subset A$ for each $v \in V$. To account for multiple sensing modalities, we factor the action subsets as $A_v = E_v \times M_v$, where $E_v \subset E$ is the subset of edges incident to vertex v, denoting possible edge traversal actions, and M_v is the set of possible measurement actions at v, defined e.g. as a collection of subsets of V on whose environment variables it is possible to obtain measurements on when the agent is at v. The possibility to wait in place is accounted for by adding an edge from v to v for every $v \in V$. Thus, any action may be represented by an ordered pair $a = (e, a_m)$, where $e \in E_v$ and $a_m \in M_v$.

The state transition model T now consists of a deterministic part $T_d : V \times E_v \to V$, and a stochastic part $T_s : W \times W \to [0, 1] \equiv p(w'|w)$. The measurement model is defined as $O : S \times M_v \times Z \to [0, 1] \equiv p(z|w', a_m)$.

The deterministic part T_d of the state transition model describes how the current vertex of the agent in the graph is updated as function of edge traversal actions. The goal vertex g is absorbing, such that the agent may not transition out of g with any action $a \in A_g$. The stochastic part T_s defines a Markov process according to which the environment states evolve, independent of the actions of the agent.

The reward of each action consists of two parts: the state-dependent edge traversal cost $c_c(w, e) \geq 0$ and the state-

independent measurement cost $c_m(a_m) \geq 0$. Thus, the reward function in the partially observable CTP is

$$R(s,a) = -c_c(w,e) - c_m(a_m). \tag{5}$$

We have given a general formulation where $c_c(w,e)$ may depend on any subset of the environment variables. The special case of interest is where the dependence is on a single environment variable, i.e. $c_c(w_u,e)$ where the vertex $u \in V$ may be e.g. the source or target vertex of the edge e. As the goal state is absorbing, we set all rewards in the goal state equal to zero, i.e. $R(s,a) = 0 \, \forall a \in A_g$.

The current vertex of the agent is known and evolves deterministically according to T_d. Therefore, the belief state may also be written as an ordered pair of form $b_t = (v_t, p_t(w))$, where v_t denotes the agent's vertex and $p_t(w)$ is a probability distribution over W describing information on the environment state at time t. The distribution $p_t(w)$ is updated applying the stochastic model T_s. The belief update equation (1) now also consists of a deterministic and stochastic part.

The partially observable CTP is related to two other problem formulations found in the literature. If g is absorbing and always reachable from v_0, i.e. proper policies exist, setting the discount factor γ equal to 1 results in a formulation of the problem as a goal-oriented POMDP (Bonet and Geffner 2009), where the task is to go from an initial belief state to a goal belief state while minimizing the cost of traversal. The belief state of the partially observable CTP is composed of a fully observable part, the location of the agent, and a partially observable part, the environment state. Such mixed-observability MDPs were studied in (Ong et al. 2010) and the factorization of the belief state was exploited in design of an offline planner.

Online planning for the partially observable CTP

Consider the partially observable CTP presented above. If the number of vertices in G is n, and if each environment variable may take K different values, the number of possible states in the problem is $|S| = n \cdot K^n$. As n increases, the number of states quickly increases to a number intractable even for modern offline POMDP solvers.

The key idea of online POMDP planning algorithms (Ross et al. 2008) is to find an optimal action repeatedly for a single belief state during task execution rather than for all belief states before execution. Online planning algorithms interleave planning and execution stages, focusing computation of the value function on the subset of belief states reachable from the current belief state by bounded length sequences of actions and observations.

We now derive an online planning scheme that takes into account the special features of the partially observable CTP. We consider the full tree of action and observation possibilities until a finite lookahead horizon is reached. We then assume that no observations on the system state will be available beyond the lookahead horizon. This enables approximation of the value of belief states reached at the end of the lookahead horizon via static graph searches.

The objective in the partially observable CTP may be restated as

$$\max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \gamma^t R_B(b_t, a_t) \tag{6}$$

i.e. find an action policy that maximizes the sum of discounted expected rewards. The expected rewards are calculated under the belief state b_t which is determined by b_0, the sequence of actions till time $(t-1)$, and the sequence of observation data till time t, updated according to equation (1).

We reformulate the problem (6) into an online planning problem, where for a finite online time horizon $H_1 \geq 1$ the full tree of action and observation data possibilities is considered. Formally, this leads to

$$\max_{\{a_t\}_{t=0}^{H_1-1}} \left[\sum_{t=0}^{H_1-1} \gamma^t R_B(b_t, a_t) + \max_{\{a_t\}_{t=H_1}^{\infty}} \sum_{t=H_1}^{\infty} \gamma^t R_B(b_t, a_t) \right]. \tag{7}$$

The first summation corresponds to a lookahead for horizon H_1, while the inner maximization problem corresponds to a cost-to-go problem to reach the goal vertex. The inner maximization problem is a POMDP, as can be seen by comparing it to equation (6). In fact, equation (7) instructs us to solve this POMDP for each possible belief state b_{H_1} that may be reached at time H_1 to find the optimal solution.

We make a simplifying approximation that during time steps $t \geq H_1$ the agent no longer has access to any measurements and hence receives no observation data but has to plan based on information received till H_1. The justification for this approximation is that we are more concerned of the quality of the information that we base our planning on in the short term. A particular sequence of observations tends to be less likely the longer the length of the sequence is, as the number of possible sequences increases. When determining an optimal action, the expected value of information that we gain in the near future is higher than that of information we may gain later. We assume that the horizon H_1 is set long enough such that if surprising information (observations with a low probability) are encountered, the agent has sufficient time to react.

Consider the inner optimization problem of equation (7), a cost-to go problem

$$\max_{\{a_t\}_{t=H_1}^{\infty}} \sum_{t=H_1}^{\infty} \gamma^t R_B(b_t, a_t) \tag{8}$$

starting from any of the belief states $b_{H_1} = (v_{H_1}, p_{H_1}(w))$ that may be reached at end of the on-line horizon at time H_1. With the simplifying assumption made above, the expected reward at time $t \geq H_1$ is

$$R_B(b_t, a_t) = -\sum_{w \in W} c_c(w, e_t) p_t(w). \tag{9}$$

As the Markov process T_s by which $p_t(w)$ evolves is known and we assume no observations are available for $t \geq H_1$,

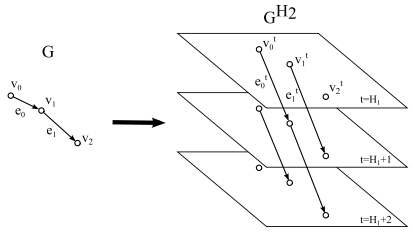


Figure 1: Construction of a time-expanded network G^{H_2} with horizon $H_2 = 2$ (right) from a directed graph G (left).

equation (8) defines a cost-to-go problem in a graph whose edge costs evolve with time in a known manner described by equation (9).

A cost-to-go problem in a graph whose edge costs vary in time in a known manner may be transformed into a shortest path problem in a static graph known as a time expanded network (Ahuja et al. 2003). An offline time horizon $H_2 > 1$ is selected, over which time-varying edge costs are considered. For a graph $G = (V, E)$ and offline horizon H_2, the time expanded network is denoted $G^{H_2} = (V^{H_2}, E^{H_2})$.

For each vertex $v_i \in V$ there are $H_2 + 1$ copies $v_i^{H_1}, v_i^{H_1+1}, \ldots, v_i^{H_1+H_2}$ in V^{H_2}, with v_i^t representing vertex v_i at time t. For each edge $e \in E$, there are at most H_2 edges in E^{H_2}, corresponding to the different possible times of traversing the edge. Thus if there is an edge between v_i and v_j in G, there is an edge between v_i^t and v_j^{t+1} in G^{H_2} for each $H_1 \leq t \leq H_1 + H_2 - 1$, which represents traversing an edge between v_i and v_j starting at time t. In Figure 1, an example of a time-expanded network is shown for a horizon $H_2 = 2$. When solving the cost-to-go problem (8) as a deterministic shortest path problem in a time-expanded network, the costs of the edges in E^{H_2} are set according to the discounted expected values with respect to $p_t(w)$.

The cost-to-go problem with known time-varying edge costs in G is a cost-to-go problem in G^{H_2} with static edge costs that may be solved using any suitable graph search method. The start vertex is the copy $v_{H_1}^{H_1}$ of the agent's vertex at time H_1 and the goal vertex is any of the copies g^{H_1+k} of the goal vertex with $k \leq H_2$.

The offline horizon H_2 should be selected so that it is large enough that we can guarantee that there exists a path in G^{H_2} from $v_{H_1}^{H_1}$ to some g^{H_1+k}. On the other hand, it is useful to limit the horizon to save both memory and computational effort. If we construct the time-expanded network G^{H_2} dynamically while searching for the shortest path by a method known to find an optimal solution, e.g. Dijkstra's algorithm or A* with an admissible heuristic, it is not necessary to set a value for H_2 before starting the search. We may simply run the search algorithm until the optimal path is found.

A partially observable CTP formulation of a navigation problem

As an example of a problem formulated as a partially observable CTP, consider a robot navigating in a dynamic environment where obstacles appear and disappear according to Markov processes. The objective of the robot is to reach a goal location while avoiding collisions with obstacles.

The graph $G = (V, E)$ describes a map with V corresponding to distinct spatial locations and E corresponding to possible transitions between spatial locations. The environment variables w_v describe if there is an obstacle present or not at location $v \in V$, corresponding to $w_v = 1$ and $w_v = 0$, respectively.

At any $v \in V$, the robot may choose to wait in place or move to any of the vertices adjacent to v. As the measurement action a_m, the robot may choose to sense the presence of obstacles at any of the vertices adjacent to the vertex it chose to move to.

The environmental variables describe if there are obstacles present in any of the spatial locations V. We have assumed that the environment variables at each vertex are independent of each other. The obstacles may appear and disappear at any location according to a two-state Markov chain with state transition probability

$$p(w'_v = 1 | w_v = j) = \begin{cases} p^{o|o} & \text{if } j = 1 \\ 1 - p^{u|u} & \text{if } j = 0 \end{cases} \tag{10}$$

where $p^{o|o}$ and $p^{u|u}$ are the probabilities that an occupied spatial location remains occupied and that an unoccupied spatial location remains unoccupied, respectively. The Markov chain parameters are identical for all vertices, an assumption that may be relaxed without adding complexity.

The sensor the robot applies to measure the presence of obstacles in any spatial location $v \in V$ is noisy. With probability p_1^f the sensor outputs a false positive observation, and with probability p_0^f a false negative observation. The measurement model for the sensor is defined by the equations

$$p(z = 0 | w'_v = m) = \begin{cases} 1 - p_1^f & \text{if } m = 0 \\ p_0^f & \text{if } m = 1, \end{cases} \tag{11}$$

and $p(z = 1 | w'_v = m) = 1 - p(z = 0 | w'_v = m)$. We require that $1 - p_1^f > 0.5$ and $1 - p_0^f > 0.5$ so the observations carry information and the meaning of false positives and false negatives is retained.

A constant wait or move cost $C(e)$ is incurred for each action, defined

$$C(e) = \begin{cases} C_{wait} & \text{if } trg(e) = src(e) \\ C_{move} & \text{otherwise,} \end{cases} \tag{12}$$

where $src(e), trg(e) \in V$ denote the source and target vertices of the edge e, respectively. An additive cost β is incurred if the target vertex is occupied. Thus, the reward function is $R(s, a) = -C(e) - \beta w_{trg(e)}$. We set $\beta > C_{move} > C_{wait} > 0$. The reward function indicates that due to the need to avoid collisions, moving to a location occupied by an obstacle is more costly than moving to an unoccupied location. We have set the measurement cost to zero. Note that the selection of measurement still has an effect on the optimization problem (7), as it influences the belief states that are reached.

In the initial belief state b_0, the start vertex v_0 is set equal the start location of the robot. The distribution of environment states $p(w_v)$ for each $v \in V$ is set to reflect initial information of presence of obstacles in the environment.

Likhachev and Stentz (2009) studied planning problems with clear preferences on missing information, i.e. situations where a particular value of a hidden variable is preferable to any other possible value. In our navigation example, the situation is similar: it would be preferable if all spatial locations were unoccupied. However, the problems studied by Likhachev and Stentz are narrower in the sense that they assume perfect sensing of hidden variables and completely deterministic underlying problem dynamics.

Simulation results

The navigation problem presented above was solved by applying the online planning scheme derived earlier. In a manner similar to the Real-time Belief Space Search (RTBSS) algorithm presented in (Ross et al. 2008), we applied an upper bound for the value function to prune the search tree. Use of such branch-and-bound pruning gives a computational advantage, as it enables ignoring parts of the search tree that are known to be non-optimal. The upper bound we applied was the so-called optimistic bound. The optimistic bound has been applied in fully observable CTPs (Bnaya, Felner, and Shimony 2009; Eyerich, Keller, and Helmert 2010), where it is derived by defining that if it is possible that an edge is traversable it is assumed traversable. In a partially observable CTP, the optimistic bound is derived by assuming all edges have a cost equal to their minimum possible cost over all possible environment states.

We compared the results with those of a state-of-the-art online POMDP solution method known as Partially Observable Monte Carlo Planning (POMCP) (Silver and Veness 2010) that has been successfully applied to POMDPs with large state spaces. POMCP interleaves the construction of a search tree of action choices and observation data possibilities with estimation of their value by Monte Carlo simulations. Simulations are run until the time allocated for planning or a specified number of simulations is exceeded, and the action with the greatest value is returned. The value of a tree node is estimated by the mean of sums of discounted rewards for all simulations starting from the node. For further information on the POMCP algorithm, we refer the reader to (Silver and Veness 2010).

The graph G was defined as a four-connected two-dimensional grid graph with 8 vertices in both dimensions. This corresponds to $|V| = 8 \cdot 8 = 64$, and a state space size $|S| = 64 \cdot 2^{64}$. For our online planning scheme, the online horizon H_1 was varied from 1 to 3. For POMCP, the number of simulations was varied from 2^1 to 2^{11} and the exploration parameter (Silver and Veness 2010) was set to $\beta + C_{move}$ to scale it to a range comparable with the value estimates. The other parameters were defined as follows: $H_2 = 50$, $\gamma = 0.95$, $p^{o|o} = 0.9$, $p^{u|u} = 0.95$, $p_1^f = 0.05$, $p_0^f = 0.05$, $C_{wait} = 0.5$, $C_{move} = 1$, $\beta = 3$. For reaching the goal vertex, a one-time reward of +10 was set. The initial belief state b_0 was set equal to the stationary distribution of the Markov chain. The start and goal vertices were the bottom left corner and the top right corner of the grid graph, respectively.

H_1	**Sum of discounted rewards**
1	-10.16 $\pm$ 1.93
2	-10.70 $\pm$ 1.46
3	-9.78 $\pm$ 1.12

Table 1: Summary of results with online planning while varying the online horizon H_1. Values shown are means with 95% confidence intervals.

Each of the experiments was repeated 20 times for both planning methods. Table 1 shows results for the online planning with branch-and-bound pruning. The mean and its 95% confidence interval is shown for the sum of discounted rewards. We note that a longer online horizon H_1 slightly improves the average performance, although the improvement is not very significant. However, performance is more consistent with longer online horizons as indicated by decreasing confidence intervals as function of H_1. The improvement in average performance as function of H_1 is likely greater in problems where longer sequences of actions are required to reach a state with a favourable cost-to-go.

The means and their 95% confidence intervals for the sum of discounted rewards with POMCP are shown in Figure 2 as function of the number of simulations. Comparing the data with Table 1, we note that to achieve comparable performance in terms of the accumulated reward, POMCP requires more than 2^{11} simulations.

The two planning methods differ in how they estimate the cost-to-go to the goal (inner maximization problem of equation 7). POMCP employs a series of Monte Carlo simulations to iteratively extend the horizon up to which possible action choices and observation data possibilities are considered and to improve the value estimates. Our online planning approach applies a graph search in the time-expanded network to approximate the cost-to-go after the online horizon H_1 is reached. Paths to the goal consist of actions that must be taken in the correct order. As POMCP applies a random rollout policy to simulate the task and to improve value estimates, it is difficult for it to find a sequence that leads to a high reward. However, our online planning scheme takes advantage of the known graph structure and finds action sequences leading to a higher reward even with a short online horizon H_1.

System design for practical implementation

We are pursuing experimental validation of our proposed planning algorithm in a dynamic real-world environment. Navigation and learning of environment dynamics was studied in (Meyer-Delius, Beinhofer, and Burgard 2012), where the A* algorithm was repeatedly applied for path planning. Formulating the navigation problem as a partially observable CTP allows planning for several time steps ahead avoiding possible problems caused by the myopic replanning approach. We have designed a system level structure of the planning algorithm for an autonomous ground vehicle.

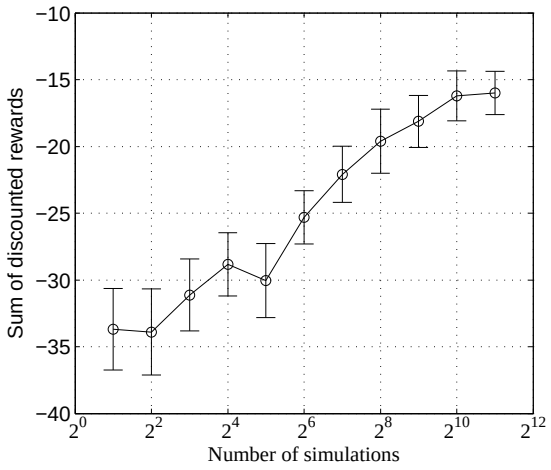


Figure 2: The mean sum of discounted rewards with 95% confidence intervals for POMCP as function of the number of simulations.

As mentioned earlier, the size of the state space in a partially observable CTP is exponential w.r.t. the number of vertices. Applying the method with metric map representations, such as e.g. occupancy grids, results in extremely large state spaces. Our planning formulation is therefore better suited for topological map representations, where each graph vertex represents a distinct spatial area.

Path planning on such topological graphs returns the order in which spatial locations are to be visited to reach the goal, but does not provide actual control signals that may be input to the actuators responsible for the movement of the vehicle. We have chosen to integrate the topological level planning with a local planner (Fox, Burgard, and Thrun 1997), which produces control signals that execute the topological level plan. The best control signal is selected based on minimizing an objective function which is a weighted sum of the vehicle's distance from the next waypoint specified by the topological level plan and the vehicle's distance from any obstacles detected near its current location.

In addition, a belief state estimator is employed to estimate the occupancy states of the vertices of the topological map, based on the state transition and measurement models and the sensor data perceived.

As initial information for the system, the topological graph structure of the environment, an initial belief state and the state transition and measurement models must be provided. Learning methods may be applied to learn the state transition models either from pre-collected datasets or online. Additionally, information on the pose of the vehicle is required, and may be obtained e.g. via a Simultaneous Localization and Mapping (SLAM) method (Thrun, Burgard, and Fox 2006).

The proposed system structure and operation are shown in Figure 3. When a desired goal position is provided, the vehicle's current pose is mapped onto a graph vertex which is given to the partially observable CTP planner. A topological

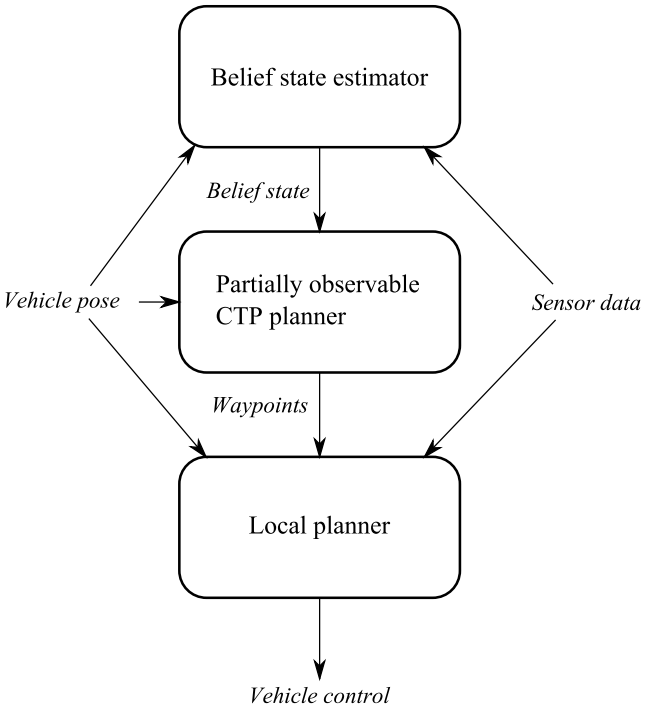


Figure 3: Overview of the system structure for implementing the proposed path planning algorithm. Boxed elements depict software modules and italicized terms indicate data and signals.

level plan leading to the goal is obtained. This plan is revised as new pose information is received and the belief state is updated by new sensor data. Depending on the current pose of the vehicle, the next waypoint on the planned path to the goal is given as target position for the local planner. The local planner plans a feasible trajectory for the vehicle that reaches this target position. Once the vehicle is near the target position, the next waypoint is provided by the partially observable CTP planner.

Conclusions and future work

We studied a special case of a partially observable Markov decision process, a partially observable Canadian Traveller's problem (CTP). In a partially observable CTP, there is an underlying shortest path problem in a directed graph, where the edge costs depend on partially observable environment variables. Shortest path problems in dynamic environments may be modelled as partially observable CTPs.

An online planning scheme was formulated consisting of a finite step lookahead and a set of cost-to-go problems, and compared with the partially observable Monte Carlo planning method (POMCP). Our results show that the online planning scheme achieves good performance with short online horizons, whereas POMCP requires a large number of Monte Carlo simulations.

Future work includes conducting experiments to prove practical applicability of our planning method. We are also

investigating possible extensions to the problem formulation, e.g. incorporating exploration by modifying the reward function to include quantities such as information entropy, and possibilities to further improve planning performance in partially observable CTPs by taking advantage of the underlying graph structure. Further study is also needed to compare our online planning method e.g. to methods exploiting the Goal-POMDP structure (Bonet and Geffner 2009) or factorization of the belief space (Ong et al. 2010) in a broader range of domains with varying parameter values.

References

Ahuja, R. K.; Orlin, J. B.; Pallottino, S.; and Scutellà, M. G. 2003. Dynamic Shortest Paths Minimizing Travel Times and Costs. *Networks* 41(4):197–205.

Blei, D., and Kaelbling, L. 1999. Shortest Paths in a Dynamic Uncertain Domain. In *Proceedings of the IJCAI Workshop on Adaptive Spatial Representations of Dynamic Environments*.

Bnaya, Z.; Felner, A.; and Shimony, S. 2009. Canadian Traveler Problem with Remote Sensing. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 437–442.

Bonet, B., and Geffner, H. 2009. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *Proceedings of the Twenty-Second International Joint Conference on Artificla Intelligence (IJCAI)*, 1641–1646.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-Quality Policies for the Canadian Traveler's Problem. In *Proceedings of the 24th Conference on Artificial Intelligence*, 51–58.

Fox, D.; Burgard, W.; and Thrun, S. 1997. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics & Automation Magazine* 4(1):23–33.

Hauskrecht, M. 2000. Value-function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research* 13(1):33–94.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence* 101(1-2):99–134.

Koenig, S., and Likhachev, M. 2005. Fast Replanning for Navigation in Unknown Terrain. *IEEE Transactions on Robotics* 21(3):354–363.

Likhachev, M., and Stentz, A. 2009. Probabilistic planning with clear preferences on missing information. *Artificial Intelligence* 173(5-6):696–721.

Meyer-Delius, D.; Beinhofer, M.; and Burgard, W. 2012. Occupancy Grid Models for Robot Mapping in Changing Environments. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2024–2030.

Ong, S. C. W.; Png, S. W.; Hsu, D.; and Lee, W. 2010. Planning under Uncertainty for Robotic Tasks with Mixed Observability. *The International Journal of Robotics Research* 29(8):1053–1068.

Papadimitriou, C. H., and Yannakakis, M. 1991. Shortest Paths without a Map. *Theoretical Computer Science* 84(1):127–150.

Polychronopoulos, G. H., and Tsitsiklis, J. N. 1996. Stochastic Shortest Path Problems with Recourse. *Networks* 27(2):133–143.

Psaraftis, H., and Tsitsiklis, J. 1993. Dynamic Shortest Paths in Acyclic Networks with Markovian Arc Costs. *Operations Research* 41(1):91–101.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-Draa, B. 2008. Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32(1):663–704.

Silver, D., and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems (NIPS) 23*, 2164–2172.

Smallwood, R., and Sondik, E. 1973. The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Operations Research* 21(5):1071–1088.

Stentz, A. 1994. Optimal and Efficient Path Planning for Partially-known Environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 3310–3317.

Thrun, S.; Burgard, W.; and Fox, D. 2006. *Probabilistic Robotics*. Cambrdige, MA: The MIT Press.

On the Traveling Salesman Problem with Simple Temporal Constraints

T. K. Satish Kumar*
Computer Science Department
University of Southern California
California, USA
tkskwork@gmail.com

Marcello Cirillo
AASS Research Centre
Örebro University
Sweden
marcello.cirillo@aass.oru.se

Sven Koenig
Computer Science Department
University of Southern California
California, USA
skoenig@usc.edu

Abstract

Many real-world applications require the successful combination of spatial and temporal reasoning. In this paper, we study the general framework of the Traveling Salesman Problem with Simple Temporal Constraints. Representationally, this framework subsumes the Traveling Salesman Problem, Simple Temporal Problems, as well as many of the frameworks described in the literature. We analyze the theoretical properties of the combined problem providing strong inapproximability results for the general problem, and positive results for some special cases.

Introduction

Tasks are usually situated in both time and space. While temporal and spatial reasoning are individually well studied, their combination is not straightforward. For example, Simple Temporal Networks (STNs) and Traveling Salesman Problems (TSPs) are two frameworks, for temporal and spatial reasoning respectively, which have been studied extensively over the years. However, little is known about the theoretical properties resulting from combining them.

A unified framework is important in many real-life domains. Imagine a surveillance vehicle which needs to autonomously decide in which order to perform the observation tasks it has been assigned. The tasks, however, are not completely independent of one another. For instance, the vehicle may be instructed to observe nearby areas allowing for a certain amount of time to elapse between observations. This dependence between tasks can be captured by temporal constraints. Under some assumptions on the nature of the temporal constraints, this problem is easily solved – or identified as unsolvable – even for a large number of tasks. Realistically, however, different tasks must be performed in different locations, and the vehicle must move from one place to the next. This adds a new dimension to the problem: a schedule which completely satisfies the temporal constraints between tasks could be infeasible because of the time the vehicle spends traveling from one location to the next. On the other hand, if we want to minimize the time (or distance) traveled by the vehicle, we still have to satisfy the temporal constraints among the tasks. Finding a schedule which satisfies the temporal constraints and minimizes the distance traveled, while taking into account the time necessary for the vehicle to move from location to location, is a problem for which there exists no general efficient solution technique.

In this paper, we first recount the two commonly used frameworks for solving temporal and spatial reasoning problems separately. We then introduce a unified framework, the *Traveling Salesman Problem with Simple Temporal Constraints* (TSP-STC). We analyze the combinatorial properties of the TSP-STC in light of recent results from the theoretical computer science community. This analysis yields both positive and negative results, which allow us to identify those aspects of the combined problem which require further research to obtain efficient solutions.

Temporal Reasoning Problems

In this section, we recount well established formalisms for reasoning about temporal constraints. Many kinds of temporal relations, including the ones considered in this paper, can be represented on a directed graph $\mathcal{G} = \langle \mathcal{X}, \mathcal{E} \rangle$, where a vertex $X_i \in \mathcal{X}$ is an event and a directed edge $e = \langle X_i, X_j \rangle \in \mathcal{E}$ is a constraint on the relative execution times of X_i and X_j. Conventionally, a special event X_0 is used to represent the "beginning of time" and its execution time is set to 0.

The simplest formalism for temporal reasoning is *precedence ordering*, which is commonly encoded by a directed edge $e = \langle X_i, X_j \rangle$, indicating that event X_i should be executed before event X_j. Although their representational power is limited, precedence constraints are useful in practice since they are able to represent causal relationships. For instance, precedence constraints can model a plan whose actions are causally ordered. In our previous example of the surveillance vehicle, causal relationships would dictate that the vehicle should first reach the target area before observing it. Producing a total ordering for a set of precedence constraints, or alternatively identifying that no such ordering exists, can be done in polynomial time.

A more expressive but still tractable formalism for temporal reasoning is the framework of *Simple Temporal Problems* (STPs). Here, each directed edge $e = \langle X_i, X_j \rangle \in \mathcal{E}$, annotated with the bounds $[LB(e), UB(e)]$, is a *simple temporal constraint* between X_i and X_j, indicating that the relative execution times of events X_i and X_j are constrained

*Alias: Satish Kumar Thittamaranahalli

by the pair of inequalities $LB(e) \leq X_j - X_i \leq UB(e)$.[1] A solution to an STP is an assignment of execution times to all events such that all simple temporal constraints are satisfied. STPs are one of the most widely used formalisms for reasoning about metric time. They are fairly rich in their expressiveness, although they cannot represent disjunctions. STPs can be solved in polynomial time using shortest path computations on their *distance graph* representations. In the distance graph representation, the constraint $X_j - X_i \leq w$ is represented as an edge from X_i to X_j annotated with a cost w. Each simple temporal constraint in the STP is therefore represented as a pair of edges in the distance graph. The absence of negative cost cycles in the distance graph characterizes the consistency of the temporal constraints (Dechter, Meiri, and Pearl 1991), that is, the existence of a solution. Shortest paths in the distance graph are commonly calculated using the Bellman-Ford algorithm. However, recent, more efficient algorithms can be employed for solving STP instances with additional structure (Planken, De Weerdt, and van der Krogt 2008).

There also exist more expressive formalisms for temporal reasoning, such as *Disjunctive Temporal Problems* (DTPs). However, their higher expressiveness comes at the cost of a higher complexity. In particular, DTPs are NP-hard problems, and all known procedures for solving them require exponential time. Here, we limit our analysis of combining temporal and spatial reasoning to cases with precedence and simple temporal constraints. The negative results proved here for the combined problem carry over to extensions in which DTPs are used instead of STPs.

Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is an established formalism for reasoning about spatial problems and has been extensively studied by different communities. Many variants of the problem exist. In this section, we recount the results associated with those variants which are relevant to our formal definition of TSP-STCs.

The classical TSP is the problem of finding a *Hamiltonian cycle* of minimum cost on an edge-weighted complete undirected graph. A Hamiltonian cycle is a cycle in which each vertex of the graph is visited exactly once. The TSP is NP-hard and even hard to approximate within any polynomial factor. However, many of its variants can be approximated in polynomial time because they allow for tours instead of cycles. A tour visits all vertices of the graph, like a Hamiltonian cycle, but any vertex can be visited more than once. This relaxation is equivalent to the metric assumption, where the triangle inequality holds for the distances between vertices (Chekuri and Pál 2007).

The most common TSP variants consist of different combinations of assumptions and requirements, such as: a. Whether we assume a metric distance function; b. Whether distances between vertices are symmetric; c. Whether we are interested in calculating a path between given start and goal vertices or a cycle (in both cases all vertices should be visited); d. Whether a subset of the vertices should be visited in a given order; and e. Whether a subset of the vertices should be visited in an order consistent with specified precedence constraints. The difference between requirements (d) and (e) is that we have a total ordering over the vertices of the subset for (d), while we have a partial ordering for (e). Other variants based on different assumptions and requirements are well studied in different communities, such as the relaxation of the requirement that every vertex should be visited or the assumption that a price is assigned to each vertex. However, these variants are out of scope for our analysis.

Table 1 summarizes the known results for the TSP and some of its most common variants which are relevant to us (Charikar et al. 1997). There is no polynomial-time approximation algorithm for the classical TSP (unless $P = NP$). However, under the metric assumption, polynomial-time approximation algorithms can be designed. In particular, for the symmetric TSP, where the distance from one vertex to another is the same as that in the opposite direction, there exists a factor-1.5 polynomial-time approximation algorithm. In the case of the Asymmetric TSP (ATSP), the distances are not necessarily symmetric. The ATSP is amenable to an $O(\log n)$ polynomial-time approximation algorithm.[2] Next, the table lists both the symmetric and asymmetric *path* variants of the TSP, TSP-Path and ATSP-Path, respectively, where the start and end vertices are given. While the approximation factor of $O(\log n)$ carries over to the ATSP-Path, the best known polynomial-time algorithm for the TSP-Path has a slightly worse approximation factor of $5/3$.

Two other notable variants are the TSP and ATSP with precedence constraints. These variants allow the specification of precedence constraints between vertices, which can be encoded as a directed acyclic graph and interpreted as a partial order. A feasible solution is a total ordering on the vertices which is consistent with the partial order. The cost of a feasible solution is equal to the cost of the tour that it induces. An optimal solution is a feasible solution with minimum cost. There are fairly strong inapproximability results for the TSP and ATSP with precedence constraints, which hold even under the metric assumption. Finally, the TSP and ATSP with path constraints are special cases of the TSP and ATSP with precedence constraints, respectively, where the precedence constraints induce a total ordering on a subset of the vertices. These last two variants have factor-3 and $O(\log n)$ approximation algorithms, respectively.

TSP-STC: A Formal Definition

Having reviewed STPs and TSPs, we now study a combination of the two, a spatial problem with temporal constraints where traveling from vertex to vertex takes time. In the example from the Introduction, the traversal times depend on both the terrain and the speed of the surveillance vehicle. Thus, the solution of the spatial part of the problem generates temporal constraints in addition to the ones already specified by the STP. The complexity of the TSP-STC cannot be smaller than the one of TSPs or STPs individually.

[1] Here, for convenience, we use the same notation to indicate events and their execution times.

[2] Here, and in the rest of the paper, n is the number of vertices.

Variant	Assumptions	Tractable approximations
TSP	No assumptions	–
TSP	Symmetric distances, metric domains	1.5 (Christofides 1976)
ATSP	Asymmetric distances, metric domains	$O(\log n)$ (Frieze, Galbiati, and Maffioli 1982)
TSP-Path (given start and goal vertices)	Symmetric distances, metric domains	$5/3$ (Hoogeveen 1991)
ATSP-Path (given start and goal vertices)	Asymmetric distances, metric domains	$O(\log n)$ (Chekuri and Pál 2007)
TSP with path constraints	Symmetric distances, metric domains	3 (Bachrach et al. 2005)
ATSP with path constraints	Asymmetric distances, metric domains	$O(\log n)$ (Chekuri and Pál 2007)
TSP with precedence constraints	Symmetric distances, metric domains	Inapproximability results (Charikar et al. 1997)
ATSP with precedence constraints	Asymmetric distances, metric domains	Inapproximability results (Charikar et al. 1997)

Table 1: A summary of the complexity results associated with different variants of the Traveling Salesman Problem.

Formally, a TSP-STC is a sextuplet $\langle \mathcal{V}, d, t, \mathcal{X}, c, \mathcal{E} \rangle$, where:

- $\mathcal{V}$ is the set of vertices, each of which represents a location;
- d is the distance function, that maps an ordered pair of vertices to a non-negative real number ($d : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}_{\geq 0}$), which represents the distance from one vertex to another;
- t is the traversal function, that maps an ordered pair of vertices to a non-negative real number ($t : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}_{\geq 0}$), which represents the time required to move from one vertex to another;
- $\mathcal{X}$ is the set of events, as defined for a standard STP;
- c is the function mapping events to vertices ($c : \mathcal{X} \rightarrow \mathcal{V}$);
- $\mathcal{E}$ is the set of directed edges of the form $e = \langle X_i, X_j \rangle$, annotated with the bounds $[LB(e), UB(e)]$. Each e is a simple temporal constraint between two events X_i and X_j.

A feasible solution of a TSP-STC is a total ordering on events in $\mathcal{X}$ and an assignment of execution times to all of them such that: a. The execution times are consistent with the total ordering; b. The execution times are consistent with the constraints in $\mathcal{E}$; and c. The execution times of two consecutive events X_i and X_{i+1} in the total ordering satisfy the induced constraint $X_{i+1} - X_i \geq t(c(X_i), c(X_{i+1}))$. The last condition accounts for the traversal time between vertices (locations) of consecutive events while not penalizing waiting time at any location.[3]

Since every event in $\mathcal{X}$ is mapped to a unique location in $\mathcal{V}$, a feasible solution defines a visit sequence on the locations. The cost of a feasible solution of a TSP-STC is equal to the cost of the induced tour as derived from the distance function d. An optimal solution of the TSP-STC is a feasible solution of minimum cost.

[3]In the rest of the paper, we use "vertex" and "location" synonymously, as one corresponds to the other in our definition of the TSP-STC.

Just like for TSPs, we assume metric distances between vertices for TSP-STCs as well. Unless otherwise specified, we also assume that the distance function of TSP-STCs is symmetric. If this is not the case, we refer to these problems as Asymmetric Traveling Salesman Problems with Simple Temporal Constraints (ATSP-STCs).

The above definition is only one possible way to combine TSPs and STPs. However, it is general enough to encode many real-world problems. In fact, both TSPs and STPs are special cases of TSP-STCs. Given a TSP with a set of vertices $\mathcal{V}$ and a distance function d, it can be represented in this framework by imposing the following conditions: $t(V_i, V_j) = 0$ for all $1 \leq i, j \leq |\mathcal{V}|$; $|\mathcal{X}| = |\mathcal{V}|$, where each $X_i \in \mathcal{X}$ is a fictitious event; c is a bijective function mapping each event to a unique location and vice versa; and $\mathcal{E} = \emptyset$. These conditions entail that every vertex must be visited in the tour, but no temporal constraints need to be considered. Conversely, a standard STP with a set of events $\mathcal{X}$ and a set of constraints $\mathcal{E}$ can be represented in this framework by imposing the following conditions: $\mathcal{V} = \{V_0\}$; $d(V_0, V_0) = 0$; $t(V_0, V_0) = 0$; and each event $X_i \in \mathcal{X}$ is associated with the same location, meaning that $c(X_i) = V_0$ for all $1 \leq i \leq |\mathcal{X}|$.

Computational Analysis of TSP-STCs

In this section, we present a complexity analysis of TSP-STCs. We show that this class of problems is subject to strong inapproximability results, even under simplifying assumptions commonly made for TSPs. We first address the complexity of TSP-STCs when temporal constraints are limited to precedence constraints (Theorems 1 and 2). Next, we prove that TSP-STCs are NP-hard to approximate within any polynomial factor, even under the assumption that both the distance and traversal functions are metric and symmetric. As TSP-STCs are a special case of ATSP-STCs, the inapproximability results for the former carry over to the latter.

Theorem 1. *For the TSP-STC, there is no polynomial-time $|\mathcal{V}|^{\alpha}$-approximation algorithm, for some $\alpha > 0$, unless $P = NP$.*

Theorem 2. *For the TSP-STC, there is no polynomial-time $(\log |\mathcal{V}|)^{\delta}$-approximation algorithm, for any $\delta > 0$, unless $NP \subseteq DTIME(|\mathcal{V}|^{\log \log |\mathcal{V}|})$.*

Proof. Consider TSPs with precedence constraints. Since precedence constraints are a special case of simple temporal constraints, TSPs with precedence constraints are a subclass of TSP-STCs and therefore inapproximability results for the former carry over to the latter. Theorems 1 and 2 correspond to Theorems 5 and 9 in (Charikar et al. 1997) after setting $k = n = |\mathcal{V}|$. □

A Closer Look at TSP-STCs with Precedence Constraints

Theorems 1 and 2 dictate that we cannot design polynomial-time approximation algorithms for TSP-STCs or ATSP-STCs with precedence constraints. However, we can precisely characterize the instance complexity of solving TSP-STCs and ATSP-STCs with only precedence constraints.

The first, trivial case encompasses instances without any temporal constraints, resulting in classical TSPs and ATSPs, for which there exist applicable polynomial-time approximation algorithms (see Table 1).

The second case encompasses instances where the subset of events for which precedence constraints are specified is totally ordered, resulting in TSPs and ATSPs with path constraints, for which there exist factor-3 and $O(\log n)$ approximation algorithms, respectively.

Finally, the third case encompasses those instances where precedence constraints are specified, but whose events cannot be uniquely ordered. For such cases, we can still reduce TSP-STCs and ATSP-STCs with only precedence constraints to multiple instances of TSPs and ATSPs with path constraints, where each instance corresponds to a total ordering over a subset of the vertices consistent with the original precedence constraints. The best guaranteed result obtained by evaluating all possible total orderings is a factor-3 approximation for the given original instance of the TSP-STC with precedence constraints or an $O(\log n)$ approximation for the ATSP-STC instance with precedence constraints. Obviously, the overall running time to find an approximate solution depends on the number of total orderings for that specific subset which are consistent with the specified precedence constraints.

The tractability of approximating TSP-STCs and ATSP-STCs with only precedence constraints depends on the number of total orderings over all events allowed by the precedence constraints. As a rule of thumb, a large space of possible total orderings (as in the first case) and a small space of total orderings (as in the second case) are both amenable to efficient approximations.

Strong Inapproximability Results

We now prove a strong inapproximability result for TSP-STC. Theorem 3 is stronger than Theorems 1 and 2 as it proves the inapproximability of the problem within any polynomial factor. Moreover, it proves that the negative results hold even under the assumption that both the distance and traversal functions are metric and symmetric. It is worth comparing this with the original TSP, which, although inapproximable within any polynomial factor, becomes amenable to tractable approximations under the metric assumption.

Theorem 3. *The TSP-STC is NP-hard and also NP-hard to approximate within any polynomial factor, even under the assumptions that both the distance and traversal functions are metric and symmetric.*

Proof. We reduce the Hamiltonian path problem to the TSP-STC. The Hamiltonian path problem is the problem of finding a path in a given undirected graph in which each vertex of the graph is visited exactly once. Consider a Hamiltonian path problem over an undirected graph $G = \langle \mathcal{N}, \mathcal{A} \rangle$, where $\mathcal{N}$ is a set of vertices ($|\mathcal{N}| = n$) and $\mathcal{A}$ is a set of undirected edges ($|\mathcal{A}| = k$). We associate each $N_i \in \mathcal{N}$ with a vertex $V_i \in \mathcal{V}$. For each $1 \leq i \leq n$, we define a unique event $X_i \in \mathcal{X}$ associated with V_i. We define t and d as follows: $t(V_i, V_j) = d(V_i, V_j) = t(V_j, V_i) = d(V_j, V_i)$; $t(V_i, V_j) = 1$ if there is an edge between N_i and N_j in G; $t(V_i, V_j) = 1.5$ otherwise. Note that both t and d are symmetric and metric, as the triangle inequality holds by construction. We complete the construction of the TSP-STC instance by defining $2\binom{n}{2}$ temporal constraints of the form $e_{ij} = \langle X_i, X_j \rangle$ for all $1 \leq i, j \leq n$ and $i \neq j$, with $LB(e_{ij}) = -\infty$ and $UB(e_{ij}) = n - 1$. A Hamiltonian path exists in G *iff* we can visit all vertices in the TSP-STC and satisfy the temporal constraints. This is so because the temporal constraints dictate that, regardless of the starting point of the path, every vertex should be visited within $n-1$ time units. The traversal function indicates that it takes exactly one time unit to go from one vertex to another if there is an edge between the two in G. It is easy to see that a solution of the Hamiltonian path problem maps to a solution of the TSP-STC instance constructed above. In addition, a solution of the TSP-STC instance maps to a solution of the Hamiltonian path problem: no vertex can be visited twice and no edge can be traversed in the TSP-STC which was not present in G because the time constraints would be violated. Finally, we can view the TSP-STC as a constraint optimization problem. The optimization component of the problem is the minimization of the sum of the distances in the tour induced by the visitation order. The satisfaction component is to respect the simple temporal constraints and the constraints induced by the traversal function. If the satisfaction component is itself NP-hard, it follows that the TSP-STC is NP-hard to approximate within any polynomial factor. As we have demonstrated, the reduction from the Hamiltonian path problem is only to the satisfaction component of the TSP-STC, hence proving the Theorem. □

A Notable Special Case

We now analyze an important special case of the TSP-STC which is reducible to TSPs with precedence constraints and

to which the analysis presented above applies. This notable case occurs when the traversal function is degenerate, i.e., maps all combinations of its arguments to zero, and all simple temporal constraints can be expressed in the form of *time windows*. A time window represents an interval during which an event must be executed. This is more restrictive than simple temporal constraints because, in the latter case, we can constrain the relative execution times of two different events, while, with time windows, we only constrain the execution times of individual events. A time window is defined by the interval $[a_i, b_i]$ for event X_i, where a_i is the start time and b_i the end time, with $a_i \leq b_i$. A degenerate traversal function captures the case in which the agent which has to travel between locations can move fast enough so that the traversal times can be disregarded as a factor in the problem domain. The case we analyze can be viewed as a special case of the well studied TSP with Time Windows (TSPTW) which also considers non-degenerate traversal functions (Melvin et al. 2007).

A time window for event X_i is modeled in the STP component of the TSP-STC as a simple temporal constraint $\langle X_0, X_i \rangle$ annotated with the bounds $[a_i, b_i]$, where X_0 is a special event which is used to represent the "beginning of time" and is conventionally set to 0. A TSP-STC instance with only time window constraints and a degenerate traversal function can be reduced to an instance of the TSP with precedence constraints as follows. The set of locations $\mathcal{V}$ and the distance function d remain unchanged, but locations are duplicated, if needed, so that at most one event is assigned to each location. For all events X_i and X_j with $b_i < a_j$, a precedence constraint $c(X_i) \prec c(X_j)$ is added.

A solution of the original TSP-STC instance is retrieved from a solution of the corresponding instance of the TSP with precedence constraints as follows. We assign execution times to the events in the TSP-STC using the total ordering obtained as a solution of the TSP with precedence constraints. This assignment of execution times is constructed to satisfy the invariant that the execution time of any event X_i is always equal to the starting point of the time window of an already scheduled event X_j. Assume that the event that corresponds to the initial location of the TSP with precedence constraints is X_i. We set the execution time of X_i to a_i. Thus, it holds that the execution time of X_i corresponds to the starting point of the time window of an already scheduled event (namely, X_i). Now, assume that the solution of the TSP with precedence constraints moves from the location that corresponds to event X_i to the location of the next event X_j in the total ordering. It cannot be the case that $b_j < X_i$ for the following reason: X_i corresponds to the starting point of the time window of an already scheduled event X_k, which the solution of the TSP with precedence constraints visits no later than event X_i. But this is impossible since $X_i = a_k$ and $b_j < X_i$ mean that there is a precedence constraint $c(X_j) \prec c(X_k)$. Thus, a solution of the TSP with precedence constraints cannot visit X_k and then later move to X_j. Thus, $X_i \leq b_j$. We now have two distinct cases:

- $a_j \leq X_i$, that is, $X_i \in [a_j, b_j]$. In this case, we set $X_j = X_i$, which satisfies the time window constraint of X_j. It holds that X_j corresponds to the starting point of the time window of an already scheduled event (namely, X_k).
- $a_j > X_i$. In this case, we set $X_j = a_j$, which satisfies the time window constraint of X_j. Here, too, it holds that X_j corresponds to the starting point of an already scheduled event (namely, X_j).

We claim that any solution of the TSP-STC instance is also a solution of the corresponding instance of the TSP with precedence constraints and vice versa. It would therefore follow that the optimal solution of the corresponding instance of the TSP with precedence constraints is also an optimal solution of the original instance of the TSP-STC. Consider any solution of the TSP-STC. Assume, for proof by contradiction, that it does not satisfy some precedence constraint $c(X_i) \prec c(X_j)$ of the TSP with precedence constraints because $X_i \geq X_j$. This is a contradiction since $X_i \leq b_i < a_j \leq X_j$ due to the semantics of the precedence constraint $c(X_i) \prec c(X_j)$. Conversely, consider any solution of the TSP with precedence constraints. The procedure described above already provides an algorithmic construction of a corresponding solution for the TSP-STC.

Related Work

The combination of spatial and temporal aspects is important in different domains, among which are robotics and vehicle routing. Thus, problems similar or identical to TSP-STCs have been studied in operations research, theoretical computer science, artificial intelligence and robotics. Different combinations of spatial and temporal aspects are possible. The spatial aspects can be expressed by minimizing distances or maximizing the (uniform or non-uniform) total reward of the visited vertices. The temporal aspects can be expressed via precedence constraints, absolute time windows or more general temporal constraints. The resulting combined problems are often solved with constraint programming, branch-and-bound search and dynamic programming as exact algorithms; or genetic algorithms, tabu search, cooperative auctions and insertion or interchange heuristics as heuristic algorithms. In this section, we present a general overview of the solution techniques adopted in different research areas.

Robotics researchers have studied multi-robot routing problems with rewards and disjoint time windows (that do not overlap), where robots have to visit targets during given time windows. The objective is to maximize the sum of the rewards of the visited targets minus the sum of the costs incurred for moving from target to target (Melvin et al. 2007). The problem is solvable in pseudo polynomial time for a single robot but is NP-hard for multiple robots, although special cases can be solved in polynomial time (including the case where the robots are identical and the targets have singleton time windows). Robotics researchers have also studied multi-robot routing problems where robots have to visit targets in the presence of precedence constraints between targets. The objective is to maximize the sum of time-decreasing rewards of the targets (Jones, Dias, and Stentz 2011). Cooperative auctions and genetic algorithms have

been proposed as heuristic algorithms to solve this problem. An alternative approach for solving multi-robot routing problems is to use multiple constraint solvers which progressively refine trajectory envelopes for each vehicle according to mission requirements, by leveraging the notion of least commitment to obtain easily revisable trajectories for execution (Pecora, Dimitrov, and Cirillo 2012).

Theoretical computer science researchers have studied prize-collecting traveling salesman (or vehicle routing) problems with time windows, where a salesperson (or vehicle) has to visit customers during given time windows. The objective is to maximize the sum of the rewards of the visited customers. Several scheduling problems with sequence-dependent setup times can be reduced to this problem, which can be solved with $O(\log n)$ approximation algorithms (Bansal et al. 2004).

Operations researchers have studied time-constrained TSPs, where a salesperson has to visit customers during given time windows. The objective is to minimize the travel distance or time (Baker 1983). It is already NP-hard to decide whether a feasible solution exists (Savelsberg 1985). Constraint programming, branch and bound search and dynamic programming have been proposed as exact algorithms to solve this problem, and greedy heuristics or interchange heuristics as heuristic algorithms (Pesant et al. 1998).

Conclusions

In this paper, we defined the framework of the TSP-STC, which combines the temporal aspects of STPs and the spatial aspect of TSPs. We recounted known computational results from the theoretical computer science community for the spatial and temporal aspects of the problem separately. We then analyzed the problem in its entirety, proving strong inapproximability results. These results hold even under common assumptions (such as the metric assumption) which allow for tractable approximations for TSPs.

Despite these negative results, we were able to present special cases which are amenable to tractable approximations with low-order instance complexities. Given the complexity of the TSP-STC and the strong inapproximability results, the most promising avenues for future investigations are heuristic and knowledge engineering approaches. From the heuristic perspective, we can potentially generalize well established heuristics for TSPs (e.g., the 2-opt heuristic). From the knowledge engineering perspective, we can potentially employ mixed-initiative approaches.

Although the TSP-STC is only a particular way of combining spatial and temporal reasoning, it is general enough to capture the requirements of many real-world applications. Therefore, our results bear strong implications on a variety of problems, namely the combination of temporal and spatial planning in agent-based systems and robotics. As we have seen in the previous sections, there are many existing methods for addressing more restrictive cases, whose applicability is, however, very limited. Our analysis covers the cases in which a single agent or robot is present, and therefore the inapproximability results carry over to multi-agent systems as well, which are more complex.

Our future work will focus on the adaptation of well established heuristic techniques developed in the TSP framework to TSP-STCs and on the generalization of our framework to multi-agent systems.

Acknowledgments. This paper is based upon research supported by a MURI under contract/grant number N00014-09-1-1031 and by the Swedish Knowledge Foundation (KKS) under project Safe Autonomous Navigation (SAUNA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

References

Bachrach, A.; Chen, K.; Harrelson, C.; Mihaescu, R.; Rao, S.; and Shah, A. 2005. Lower bounds for maximum parsimony with gene order data. In *Proceedings of the International Conference on Comparative Genomics*.

Baker, E. 1983. An exact algorithm for the time-constrained traveling salesman problem. *Operations Research* 31(5):938–945.

Bansal, N.; Blum, A.; Chawla, S.; and Meyerson, A. 2004. Approximation algorithms for deadline-TSP and vehicle routing with time-windows. In *Proceedings of the Annual ACM Symposium on Theory of Computing*.

Charikar, M.; Motwani, R.; Raghavan, P.; and Silverstein, C. 1997. Constrained TSP and low-power computing. In Dehne, F.; Rau-Chaplin, A.; Sack, J.-R.; and Tamassia, R., eds., *Algorithms and Data Structures*, volume 1272 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 104–115.

Chekuri, C., and Pál, M. 2007. An $O(\log n)$ approximation ratio for the asymmetric traveling salesman path problem. *Theory of Computing* 3(1):197–209.

Christofides, N. 1976. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1):61–95.

Frieze, A. M.; Galbiati, G.; and Maffioli, F. 1982. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks* 12(1):23–39.

Hoogeveen, J. 1991. Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters* 10(5):291–295.

Jones, E.; Dias, M.; and Stentz, A. 2011. Time-extended multi-robot coordination for domains with intra-path constraints. *Autonomous Robots* 30(1):41–56.

Melvin, J.; Keskinocak, P.; Koenig, S.; Tovey, C.; and Ozkaya, B. Y. 2007. Multi-robot routing with rewards and disjoint time windows. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Pecora, F.; Dimitrov, D.; and Cirillo, M. 2012. On mission-dependent coordination of multiple vehicles under spatial and temporal constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Pesant, G.; Gendreau, M.; Potvin, J.-Y.; and Rousseau, J.-M. 1998. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science* 32(1):12–29.

Planken, L.; De Weerdt, M.; and van der Krogt, R. 2008. P^3C: A new algorithm for the simple temporal problem. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS)*.

Savelsberg, M. 1985. Local search in routing problems with time windows. *Annals of Operations Research* 4:285–305.

On the Many Interacting Flavors of Planning for Robotics

Kartik Talamadupula[†] and **Matthias Scheutz**[§] and **Gordon Briggs**[§] and **Subbarao Kambhampati**[†]

[†]**Dept. of Computer Science and Engg.**
Arizona State University
Tempe AZ
{krt,rao} @ asu.edu

[§]**HRI Laboratory**
Tufts University
Boston MA
{mscheutz,gbriggs} @ cs.tufts.edu

Abstract

Automated planners have been employed in a variety of robotics applications. However, there still remains a distinct divide between task planning, or high-level planning, and its counterparts in robotics. In particular, navigation and dialogue planning have emerged as important concerns in the quest to make realistic end-to-end robotic systems a reality. However, in the absence of a unifying problem to solve, collaborations between these three fields have been sparse and mostly narrow and project-driven. In this paper, we discuss Human-Robot Teaming as that unifying problem, and outline via a simple example the various sub-fields of the different kinds of planning that interact naturally to produce a solution to the overall problem. Our hope is to spur the various, fragmented planning communities into further collaboration by highlighting the rich potential of these interactions.

1 Introduction

Automated planning systems have come a long way since the days of the STRIPS planner (Fikes and Nilsson 1972) and Shakey the Robot. Specifically, the evolution of fast heuristics and various compilation methods has enabled the application of state-of-the-art planners to cutting edge research problems in robotics. Planners – in one form or the other – now regularly guide robotic systems that hitherto had to rely on painstakingly pre-programmed scripts in a robust and real-time manner. The idea that robotic agents need to be endowed with autonomy is not new – from depictions in popular culture to actual deployed agents, robots are assumed to be autonomous and independent in many crucial ways. However, it is the meaning of this autonomy that is constantly changing. Much remains to be done is defining a full taxonomy for the usage of the word *planning* when it comes to robotic systems. Such an effort would go a long way towards recognizing the various disciplines and subfields that have hitherto been treated as separate from each other, and spur research on further bridging the divide between the automated planning and robotics communities.

Apart from automated planning – alternatively known as *task* planning – robotic systems have also had to contend with navigation or *path* planning. Increasingly, *dialogue* planning is gaining importance as well, reflecting the key role that speech and dialogue play in interaction with humans. All three of these fields house thriving research communities that report progresses at highly rated venues year after year. However, collaborations between these fields are few and far in between, and usually only come about as a result of integrated systems that are task or project specific.

In this paper, we discuss a motivating problem that brings these different types of planning together, and outline a simple example task that demonstrates the need for different kinds of planning. We conclude by pointing out various connections between existing work in the field of automated planning, and important and outstanding problems in other fields associated with robotics.

2 Planning for Human-Robot Teaming

Human-Robot Teaming scenarios are defined as those that involve humans working with autonomous robotic agents to achieve high-level goals that are determined and specified by a human (Talamadupula et al. 2011). Alternatively, any general problem that considers symbiotic interaction between humans and robots (Rosenthal, Biswas, and Veloso 2010) can be used to illustrate the point that we wish to make, too. Here we present a simple example of an HRT task.

A Motivating Example Consider a robot, *Cindy*, that must deliver a medical kit to *Commander Z*. Cindy is told that there is such a kit in a room at the end of the hallway, what the kit looks like, and instructed to remain undetected by the enemy while performing this task. Just before entering the room, Cindy encounters *Commander Y*, who asks her to void her earlier, more important goal in order to follow him. Cindy declines while indicating urgency and interruption in her voice, and negotiates a commitment to meet Commander Y wherever he happens to be when she achieves her current goal. Arriving outside the room where Commander Z is located, she senses that the door is closed, thus triggering a further query to her handler. Cindy is instructed to try a new action – pushing the door open with her hand. She enters the room and delivers the kit to Commander Z, who reinforces the commitment that she must go meet Commander Y at his current location at once.

Even in this simple task, various planning modalities must interact and occur in parallel to enable the script.

1. **Task Planning**: Agents must be able to plan for changing or conditional goals like the medical kit (Talamadupula et al. 2010), elaboration of the goals associated with the task (Baral and Zhao 2008) as well as trajectory constraints like 'remain undetected' on the form of the plan (Mayer et al. 2007). Additionally, the task planner may have to deal with updates to the model that are either learned, or specified by humans (Cantrell et al. 2012).

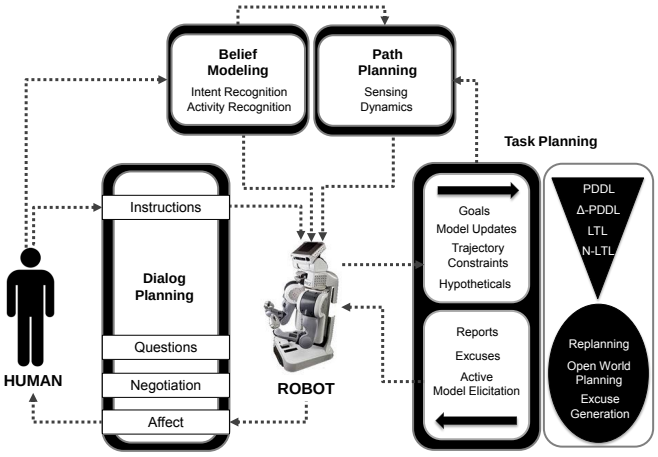


Figure 1: *A schematic of the various interactions present in a simple Human-Robot Teaming task.*

2. **Path Planning**: Autonomous robots must be endowed with capabilities of planning their paths. These may include planning with goal-oriented actions like looking for the medical kit (Simmons and Koenig 1995), finding the shortest path to the room that holds the kit (Koenig, Likhachev, and Furcy 2004), obeying constraints on the trajectories of the path (Saffiotti, Konolige, and Ruspini 1995) or planning for agents that exhibit different dynamics, like UAVs and AUVs (McGann et al. 2008).
3. **Dialogue Planning**: Robots need to skilled at both recognizing and producing subtle human behaviors vis-a-vis dialogue (Briggs and Scheutz 2013) – for example, in the above scenario, Cindy needs to both understand the superiority in Commander Y's voice when requesting a new task, as well as inflect her own response with urgency in order to indicate that the task at hand cannot be interrupted. Negotiation is another possibility, for which the robot needs to be informed by the task planner regarding excuses (Göbelbecker et al. 2010) and other hypotheticals.
4. **Mental Modeling**: The agent must be in a position to model the beliefs and mental state of other agents that are part of the scenario (Briggs and Scheutz 2012); in this case, Cindy may want to model Commander Y's mental state to determine his location at the end of the first task.
5. **Intent and Activity Recognition**: Closely tied in to both dialogue and mental modeling is the problem of recognizing the intents of, and activities performed by, other agents (Vail, Veloso, and Lafferty 2007). Humans are endowed with these capabilities to a very sophisticated degree, and agents that interact and team with humans must possess them as well.
6. **Architecture**: Finally, the integrated architecture that all these processes execute in plays a big role in determining the planning capabilities of the autonomous system. A good control structure must display programmability, adaptability, reactivity, consistent behavior, robustness, and extensibility (Alami et al. 1998). By dint of having to interact with humans, it must also fulfill the notions of attending and following, advice-taking, and tasking (Konolige et al. 1997). Finally, it must be able to detect and recover from failure, and tide all the other planning components over that failure.

3 Planning, and More Planning

As the above list shows, even a simple task requires various kinds of planning components in order to present humans with a seamless teaming experience. However, the mere presence of these components is not enough – they must interact in order to process data that comes in both from the human team-member as well as the world, so that the scenario objectives may be furthered. Figure 1 presents an outline of the various components described previously, and the interactions among them. Here, we look at three of the most important ones:

Task and Motion Planning The interaction between these two kinds of planning is well-established; in the above scenario, task planning sets the waypoints that must be visited in order to fulfill the higher level goals, and these waypoints are then passed on as goals themselves to the motion planning process. In return, the motion planning provides updates to the task planner on new objects in the world, and the current location of the robot.

Task and Dialogue Planning Real-world robotics applications are seeing an increase in the use of dialogue managers as more and more systems try to interact with and engage humans gainfully. On the one hand, dialogue systems provide more information and context to task planners in the form of instructions (goals), actions models and user preferences. In the reverse direction, task planners can be used to inform robotic agents about relevant questions to ask in order to elicit more information from humans, and to help determine the tone and affect of the interaction.

Mental Modeling and Task Planning Many human-robot teaming applications consider just the presence of one agent in the world; however, the real world is more closely approximated by multi-agent scenarios. Although a given agent will not interact with *all* of the agents in the world with the same degree of closeness, sometimes it is useful to model those other agents. In the example scenario described in Section 2, for instance, Cindy the robot must model Commander Y's mental state to some level in order to know his possible location when she achieves her prior goal, so that she may rendezvous with him. Similarly, given a model as well as the goals of Commander Y, a task planner can be used to simulate a mental model (in lieu of a real mental or belief model).

The challenge – and opportunity – for the automated planning community is to develop algorithms and systems that take all these interactions into account explicitly, in order to support more real-world robotics applications.

Acknowledgments

This research is supported in part by the ARO grant W911NF-13-1-0023, the ONR grants N00014-13-1-0176, N00014-09-1-0017, N00014-07-1-1049 and N00014-13-1-0519, and the NSF grant IIS201330813.

References

[Alami et al. 1998] Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *The International Journal of Robotics Research* 17(4):315–337.

[Baral and Zhao 2008] Baral, C., and Zhao, J. 2008. Non-monotonic temporal logics that facilitate elaboration tolerant revision of goals. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI*, 13–17.

[Briggs and Scheutz 2012] Briggs, G., and Scheutz, M. 2012. Multi-modal belief updates in multi-robot human-robot dialogue interaction. In *Proceedings of 2012 Symposium on Linguistic and Cognitive Approaches to Dialogue Agents*.

[Briggs and Scheutz 2013] Briggs, G., and Scheutz, M. 2013. A hybrid architectural approach to understanding and appropriately generating indirect speech acts. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, (forthcoming).

[Cantrell et al. 2012] Cantrell, R.; Talamadupula, K.; Schermerhorn, P.; Benton, J.; Kambhampati, S.; and Scheutz, M. 2012. Tell me when and why to do it!: Run-time planner model updates via natural language instruction. In *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*, 471–478. IEEE.

[Fikes and Nilsson 1972] Fikes, R., and Nilsson, N. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3):189–208.

[Göbelbecker et al. 2010] Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up With Good Excuses: What to do When no Plan Can be Found. In *Proc. of ICAPS 2010*.

[Koenig, Likhachev, and Furcy 2004] Koenig, S.; Likhachev, M.; and Furcy, D. 2004. Lifelong Planning A*. *Artificial Intelligence* 155(1):93–146.

[Konolige et al. 1997] Konolige, K.; Myers, K.; Ruspini, E.; and Saffiotti, A. 1997. The saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence* 9(2-3):215–235.

[Mayer et al. 2007] Mayer, M. C.; Limongelli, C.; Orlandini, A.; and Poggioni, V. 2007. Linear temporal logic as an executable semantics for planning languages. *Journal of Logic, Language and Information* 16(1):63–89.

[McGann et al. 2008] McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for AUV control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 1049–1054.

[Rosenthal, Biswas, and Veloso 2010] Rosenthal, S.; Biswas, J.; and Veloso, M. 2010. An effective personal mobile robot agent through symbiotic human-robot interaction. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 915–922.

[Saffiotti, Konolige, and Ruspini 1995] Saffiotti, A.; Konolige, K.; and Ruspini, E. H. 1995. A multivalued logic approach to integrating planning and control. *Artificial intelligence* 76(1):481–526.

[Simmons and Koenig 1995] Simmons, R., and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *International Joint Conference on Artificial Intelligence*, volume 14, 1080–1087.

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

[Talamadupula et al. 2011] Talamadupula, K.; Kambhampati, S.; Schermerhorn, P.; Benton, J.; and Scheutz, M. 2011. Planning for Human-Robot Teaming. In *ICAPS 2011 Workshop on Scheduling and Planning Applications (SPARK)*.

[Vail, Veloso, and Lafferty 2007] Vail, D. L.; Veloso, M. M.; and Lafferty, J. D. 2007. Conditional random fields for activity recognition. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 235. ACM.

Planning Surface Cleaning Tasks by Learning Uncertain Drag Actions Outcomes *

David Martínez, Guillem Alenyà and Carme Torras
Institut de Robòtica i Informàtica Industrial (CSIC-UPC)
Llorens i Artigas 4-6, 08028 Barcelona, Spain

Abstract

A method to perform cleaning tasks is presented where a robot manipulator autonomously grasps a textile and uses different dragging actions to clean a surface. Actions are imprecise, and probabilistic planning is used to select the best sequence of actions. The characterization of such actions is complex because the initial autonomous grasp of the textile introduces differences in the initial conditions that change the efficacy of the robot cleaning actions. We demonstrate that the action outcome probabilities can be learned very fast while the task is being executed, so as to progressively improve robot performance. The learner adds only a little overhead to the system compared to the improvements obtained. Experiments with a real robot show that the most effective plan varies depending on the initial grasp, and that plans become better after only a few learning iterations.

1 Introduction

Robotized household environments are a promising field where action planning can be useful. Typically to solve a task, a set of perceptions and a set of actions are defined, and a planner is used to choose the sequence of actions to execute. In this paper we put the attention on repetitive actions that are difficult to model offline because some initial conditions can change their outcomes. We use as an example the task of cleaning surfaces with a robot using an autonomously grasped textile, where the initial grasping of the textile clearly changes the effectiveness of the actions.

The actions used to clean are represented with rules which, given some preconditions on the state, define the expected behaviour of an action as a set of possible outcomes with probabilities associated. To obtain the best results we should use rules that accurately characterize any available action in the current environment.

As initial conditions may introduce large variability, we incorporate a learning system that improves the rules after the execution of each action, leading to better plans in the long term. Common learning methods usually need many executions until good rules are obtained, but we aim to finish the task quite fast after a few action executions. We propose starting with very simple handcrafted rules and update them with a new heuristic based on the m-estimate (Cestnik 1990) to get more accurate rules that will perform quite well after a few executions. This is intended for the initial steps until enough experience is obtained to apply more complex methods that can refine the rules with more details. The criteria of the *number of actions executed* and *time to complete the whole task* will be used to measure the success of the method. A more simple approach, like a reactive action execution, will not provide a proper solution as the combination of different actions cannot be taken into account.

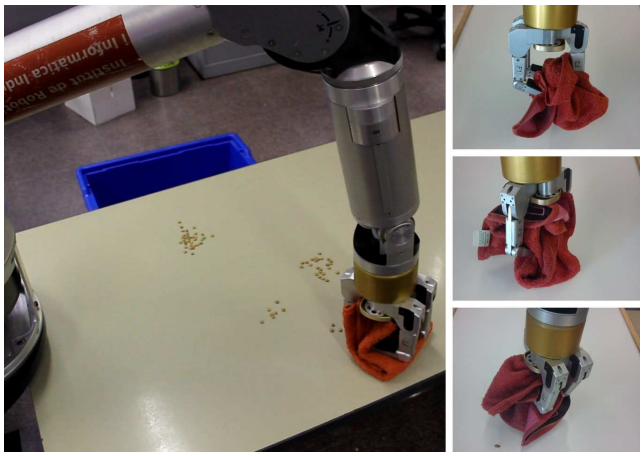


Figure 1: On the left the WAM robot arm is cleaning lentils from a table to a container. On the right three different grasps of the textile used for cleaning, each one leading to different behaviours of the cleaning actions.

The presented approach uses a WAM robot arm to perform the actions, and a Kinect camera to get a representation of the state, which is assumed to be completely observable. Figure 1 shows the environment used for the task, and different grasping configurations. As the actions are stochastic, an online probabilistic planner is used. As mentioned, a learner updates the planning rules to adapt to the grasped textile. The performance of the system depends on the quality of the rules, so learning will be critical to get the best results.

*This work was supported by the Spanish Ministry of Science and Innovation under project PAU+ DPI2011-27510, by EU Project IntellAct FP7-ICT2009-6-269959 and by the Catalan Research Commission through SGR-00155.

This paper is structured as follows. Section 2 presents some related work and where our idea fits in state of the art on automatic learning. The proposed algorithm is introduced in Sec. 3, where perceptions, actions and planning are presented. Section 4 explains the details of the learning procedure. Section 5 shows some experiments of cleaning a surface and the improvements attained when using our approach. Finally, Sec. 6 is devoted to draw some conclusions and future work.

2 Previous work

There are a few recent works in which a robot performs similar tasks to the one presented in this paper. In (Kormushev et al. 2011) and (Sato et al. 2011) robot skills to clean a whiteboard are presented. The robot is trained using imitation learning with hybrid position/force control to learn and execute trajectories trying to maintain the force of the hand against the whiteboard. Force feedback has also been used to learn dynamic motion primitives that ensure that the robot maintains contact and applies the desired force in tasks such as wiping a table (Gams et al. 2010). Moreover, methodologies to sequence motion primitives have been proposed (Nemec and Ude 2012). The surface cleaning strategy is fixed and thus the robot is unable to adapt to different layouts of dirt that could be cleaned more efficiently with simpler trajectories.

For this, a perception system is necessary to acquire the scene state, and actions should be selected accordingly. Then a model-based planner can generate sequences of actions to clean efficiently all kinds of dirt. As we have stochastic actions a probabilistic planner is needed. For large state spaces, a very common planning technique is UCT (Kocsis and Szepesvári 2006), which uses bandit ideas to guide a Monte-Carlo planner. The algorithm finds near-optimal solutions in finite-horizon or discounted MDPs. PRADA (Lang and Toussaint 2010) is a probabilistic planner that handles the uncertainty by converting the rules into a dynamic Bayesian network for state representation, and predicts the effects of action sequences by using an approximate inference method to efficiently propagate beliefs. PRADA has a better performance than classic UCT, so it will be the planner used.

Creating models manually is tedious as it has to be repeated for every task and environment, but learning methods exist to generate models based on experience. Two different approaches can tackle the problem of learning models. The first one is reinforcement learning (RL). Model-based bayesian RL that aims to obtain optimal behaviour as it chooses actions that maximize the expected reward as a function of the belief-state. However, this optimization problem is intractable so near optimal solutions have been proposed (Asmuth and Littman 2011).

The second approach is learning actions models. Using the accumulated knowledge of all executions of an action, a set of rules defining the model is generated. These rules define relational worlds which allow to generalize much better over different states, as the same rule may apply to several similar objects. Methods for learning stochastic actions (Pasula, Zettlemoyer, and Kaelbling 2007) and partially observable domains (Mourao, Petrick, and Steedman 2010) are available. These approaches lack an exploration-exploitation behaviour, and they require several samples for each action before they can get useful models.

(Lang, Toussaint, and Kersting 2010) propose a solution to this exploration-exploitation problem using a strategy based on the Explicit Explore or Exploit E^3 (Kearns and Singh 2002) algorithm and updating the rules with Pasula's algorithm. Although in the end good results are obtained, the algorithm just explores and uses vague rules until enough experience is acquired, getting bad results during the initial executions.

As executing actions in real robots has a large cost, we like our robot to perform as well as possible also during the first executions until enough experience is obtained to learn the action models. We propose using a very simple set of initial rules which can adapt with fast learning heuristics until enough samples are obtained to apply action learning algorithms. The initial rules will begin with very optimistic outcomes, getting the advantages of the optimism under uncertainty bias (Brafman and Tennenholtz 2003).

The rules are provided to the learner which has to update their probabilities online to improve the estimated outcomes of the actions. Several heuristics have been proposed and their performances have been compared (Janssen and Fürnkranz 2010). The family of parametrized heuristics, and in particular the m-estimate (Cestnik 1990), allows to adjust the trade-off between learned estimations and a priori probabilities. Similar to the work of (Agostini, Torras, and Wörgötter 2011), we want to produce confident estimates with a few examples by regulating the influence of the m value, but having stochastic actions we can't know a priori the number of actions needed to cover all possibilities. Therefore we propose a new heuristic that decreases the m value as experiments are carried out so as to adapt quickly the rule outcomes, but the decrease of m is gradually slowed down so that a small influence of the a priori probability is maintained for a long time to take into account unexperienced outcomes.

3 Proposed Method

The method proposed in this paper is aimed at cleaning a surface using a calibrated RGB-D camera and robot arm grasping a cloth.

Observations are continuously acquired with the camera and processed to have an updated representation of the environment. The robot has a set of actions consisting of sequences of movements to clean or displace dirt. Given a representation of the environment and a set of rules defining the available actions, a planner chooses a sequence of actions to clean the surface efficiently. An action is then performed by the robot, and once it is completed, the learner analyzes the changes in the state to update the rules of the executed action accordingly. The system keeps replanning, executing actions and learning if necessary until the task is complete.

Actions

A set of actions is designed to clean a surface containing small objects like lentils. These actions are parametrized

(a) Move to container (b) Join 2 groups (c) Group scattered

Figure 2: Cleaning actions.

with ellipses representing the dirty areas on the scene, and generate a sequence of points defining the cleaning movements. Lentils are detected with a simple RGB segmentation algorithm. The cleaning tool is always oriented perpendicular to the direction of motion to get effective moves.

Actions can be divided in two groups.

- **Cleaning actions** that remove dirt from the surface, moving it towards a container positioned near the edge of the surface (Fig. 2a).
  - `Fast move to container (ellipse)`:
    - Pushes the lentils in a dirty area to the container position using a grasped cloth.
  - `Straight move to container (ellipse)`:
    - It is equivalent to *Fast move to container*, but ensures that the trajectory is straight at the cost of being a little slower.
  - `Short move to container (ellipse)`:
    - It is equivalent to *Fast move to container*, but only does a short movement towards the container, ensuring that the trajectory is straight, although it won't reach the container if the dirt is far from it.
- **Grouping actions** that rearrange the dirty areas on the surface, so that they become easier to clean by means of future actions.
  - `Join 2 Groups(ellipse1, ellipse2)`:
    - The movement pushes the lentils of ellipse1 to ellipse2 joining them (Fig. 2b).
  - `Join 3 Groups(ellipse1, ellipse2, ellipse3)`:
    - Moves ellipse1 and ellipse2 to the position of ellipse3.
  - `Grouping scattered dirt(ellipse)`:
    - Moves scattered groups together to get compact groups that are more manageable (Fig. 2c).

These actions are stochastic due to several factors:

- Actions rely on the accuracy of the depth information provided by RGB-D cameras, which may have some errors. Although using a cloth provides some compliance, sometimes actions may fail to move the dirt as expected.
- The same action may get different outcomes for similar dirty areas. For example, some dirt may spread during the trajectory in some cases, while they may move successfully in other similar cases.
- The cloths used for cleaning produce different results depending on the way they are grasped.

Action:
straightMoveToContainer(X)
Preconditions:
dirt(X), mediumSize(X), ¬scattered(X)
Outcomes (Success probability: predicate changes):
0.4: ¬dirt(X), clean(X)
0.3: ¬mediumSize(X), smallSize(X)
0.2: ¬mediumSize(X), smallSize(X) scattered(X)
0.1: noise

Figure 3: Rule example for removing lentils.

Planning

The planner selects the set of actions to execute based on the perceptions and the probabilistic effects of action sequences. The task is quite complex, and selecting the fastest action sequence is challenging. For example, plans beginning with *grouping* actions may *penalize* in the beginning (remove no dust) compared to *cleaning* actions, but they can provide the best results in the long run.

Using a probabilistic planner is important when actions have several possible outcomes with different probabilities. Deterministic planners (Little and Thibaux 2007) only consider the most probable outcome for each action, while a probabilistic planner takes into account all outcomes.

The planner takes as input the state representing the scene and a set of rules defining the expected results of the actions, and it outputs a plan consisting in a sequence of actions. These actions will be converted into motions that the robot will perform to clean the dirty areas.

State representing the scene: The state s is defined as

$$s = (d_1, ..., d_n, near(d_i, d_j), ..., near(d_k, d_l)) \quad (1)$$

where d_i are the dirty areas represented by ellipses and $near(d_i, d_j)$ indicates dirty areas whose positions are close to each other.

Each dirty area is defined as $d_n = (Id, s, \sigma)$

- Id: Identifier.
- s: Size, where $s \in \{big, medium, small\}$.
- σ: Scattered, where $\sigma \in \{true, false\}$ accounts for compact or scattered distributions.

Action rules: We are using noisy indeterministic deictic (NID) rules (Pasula, Zettlemoyer, and Kaelbling 2007), where each outcome has associated the list of predicates that change when the rule is applied. There may be several rules with different preconditions for every action, and several outcomes for every rule. An example is shown in Fig 3.

4 Learning

The planner needs a set of rules defining the actions that may be performed, and the quality of the plans will depend on the precision of these rules. As our environment is stochastic, it is difficult to define accurate rules for every surface, type of dust, robot and grasping of cleaning tools. To solve this problem, we will learn the actual outcome probabilities for each configuration while performing the cleaning task with

initially inaccurate rules. The learner updates the expected rule outcomes for every action that is executed to reflect the result of the execution. Also, it saves a record of previously executed actions and their results to get better estimates of the outcomes.

Having stochastic actions means that the rule defining the action may have several outcomes for just one set of preconditions. After executing an action that we want to learn, the robot will have to take a new perception and look carefully for differences in the state to know which outcome was obtained. When no outcome matches the result of the action, the noise outcome probability will be increased.

Learning heuristics

We want the system to rapidly refine the outcomes to adapt to the new environment, but we also want to avoid wrong premature estimations to degrade the performance of the system, as it is learning at the same time that it is solving the task.

A learning heuristic to prevent these premature wrong estimations is the m-estimate (Cestnik 1990), that includes a parameter m to implement a trade-off between learned outcomes and a priori probabilities in rule outcomes

$$P = \frac{p + mP_0}{p + n + m}, \tag{2}$$

where P is the estimated probability, P_0 the a priori probability, p the number of positive examples and n the number of negative examples.

The problem with this heuristic is that small values of m may yield wrong estimates of rule outcomes, while a high value of m would entail the system taking too much time to converge to the learned estimates. We propose to use a different heuristic:

$$P = \frac{p + (m/\sqrt{p+n})P_0}{p + n + (m/\sqrt{p+n})}. \tag{3}$$

This decreasing-m-estimate is similar to the m-estimate when there are only a few examples, favouring a priori probabilities. But as the number of examples increases, its influence decreases, leading to better estimates that have little influence from a priori probabilities. The value of m should depend on the stochasticity of the task. In our experiments a value of 10 provided good estimates and was low enough to converge fast to the learned estimates.

Stop learning: Learning adds some overhead to the system. After executing an action to be learned, the arm has to leave the visual field of the camera to get a good perception of the surface and estimate correctly the outcome obtained. Otherwise, planning would rely on partial perceptions that may have occlusions.

Therefore, we only learn actions until we have enough examples to consider that the learned estimate is quite accurate. Using the Hoeffding inequality, we can have a bound for having a high probability $(1 - \delta)$ that our estimate $\widehat{p}$ is accurate enough $|\widehat{p} - p| \leq \epsilon$. The number of trials required is

$$T \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}. \tag{4}$$

Initial rules

The initial rules were written manually, defining only the basic expected behaviours of the actions. As the proposed learner is intended only for the first executions, little effort should be spent on these initial rules, as they will be changed later to more precise ones when enough experience is obtained. Based on the optimism under uncertainty bias (Brafman and Tennenholtz 2003), all rules are defined with a probability 1.0 of getting the best outcome, and as actions get executed, this probability tends to the actual one.

Another possible set of optimistic rules is learning the probabilities of outcomes for the best scenario, which in our case is a very good cloth grasp, and use them as the initial rules. These rules will converge faster to the actual ones as they already have learned some of the dynamics of the system and their outcome probabilities will be closer to the actual current ones.

5 Experimental results

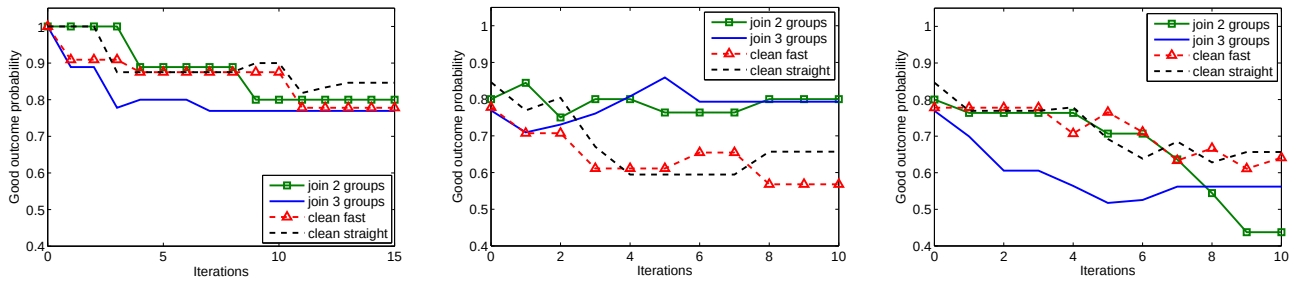
We have carried out two experiments to analyze the learner performance in the task of surface cleaning. Both experiments involved cleaning a surface with 30 lentils spread over it. The robot had to move the lentils to a container positioned near an edge of the surface. The task was repeated a number of iterations in each experiment to analyze the learning process. The value of m was set to 10 and T to 12 in both experiments.

First experiment: rules evolution over time. We analyzed the estimated probabilities of the rules over time (Fig 4). For clarity, only 4 rules are shown. Figure 4a shows the learning process starting with optimistic rules with a 1.0 probability for the best outcome, and along 15 iterations. Then the resulting rules were used in new situations with two different graspings (Figs. 4b and 4c).

As can be observed, these two grasps produce different rules. The grasp of Fig. 4b yields very good results with *join* actions, while the grasp in Fig. 4c gets very bad results with them, particularly the one joining two groups. Also the *clean fast* action has significant differences between both grasps.

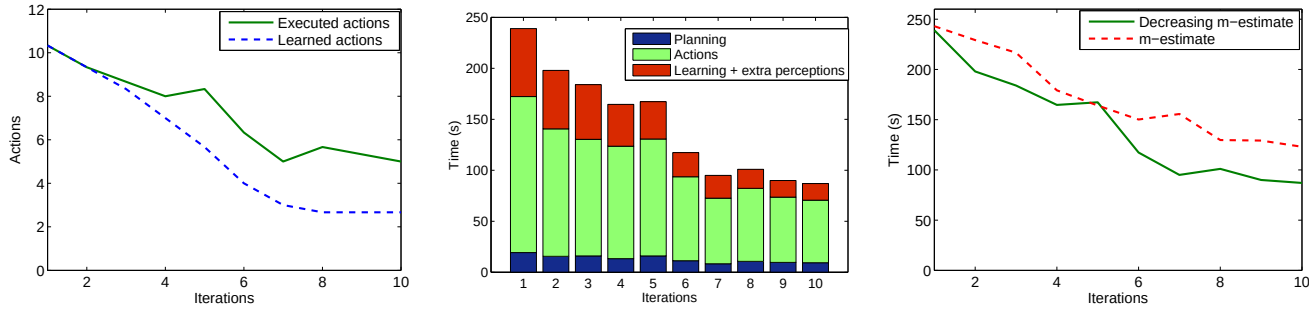
Second experiment: improvement using learning. Using the rules obtained during the previous experiment, we measured the number of actions and time taken to clean with new grasps. The experiment was repeated 4 times and the average of the results is shown in Fig. 5. As can be seen, the number of actions required to complete the tasks decrease as the rules improve. Also, the learner stops refining actions once it gets enough examples of them, reducing the learning time after a few iterations.

Moreover, the same experiment was repeated using the original m-estimate to compare its performance with our proposal. Although the m-estimate also improves the rules, the decreasing m-estimate obtains better results with fewer iterations as shown in Fig. 5c.



(a) Initial rule learning. (b) Refining rules from (a) with a new grasp. (c) Refining rules from (a) with another grasp.

Figure 4: Rule learning over time with m=10. In (a) initial rules are obtained with a common grasp. In (b) and (c) rules from (a) are refined with new grasps. Observe that each particular grasp changes the outcome probabilities of the different rules, e.g. join 2 groups performs well in (b) but bad in (c).



(a) (b) (c)

Figure 5: Improvements using the learner. **(a)** Number of actions executed and the number of them that required learning as they were considered unknown. **(b)** Distribution of time between planning, action execution and learning. **(c)** Time taken to clean the board using the proposed decreasing m-estimate and the original m-estimate.

6 Conclusions

In this work we have shown the use of a learner integrated in a surface cleaning system where a planner is used to choose good sequences of actions to clean efficiently with very little experience. Different grasps of the cloth vary significantly the rule outcomes probabilities, which makes the learner a very important piece to get accurate rules for the planner. As seen in the experiments, good rules improve the plans obtained, which allow the system to clean faster.

The learner produces quite accurate rules after a few executions using the decreasing m-estimate heuristic. It also adds a little overhead to the system, which almost disappears after a few executions when most used rules get already learned. Overall we can conclude that the inclusion of a learner for rule refinement is highly recommendable in dynamic environments where accurate rules are not available. Once enough experience is obtained, more complex methods for refining the rules preconditions and outcomes (Pasula, Zettlemoyer, and Kaelbling 2007) can be used to get more accurate rules.

A future improvement to the system would be integrating the initial learning heuristic with the more complex action model learner to incrementally update the rules preconditions and outcomes, thus getting the best of using both learning methods simultaneously.

References

Agostini, A.; Torras, C.; and Wörgötter, F. 2011. Integrating task planning and interactive learning for robots to work in human environments. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2386–2391.

Asmuth, J., and Littman, M. L. 2011. Learning is planning: near bayes-optimal reinforcement learning via monte-carlo tree search. In *Proc. of Conference on Uncertainty in Artificial Intelligence (UAI)*, 19–26.

Brafman, R. I., and Tennenholtz, M. 2003. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research* 3:213–231.

Cestnik, B. 1990. Estimating probabilities: A crucial task in machine learning. In *Proc. of European Conference on Artificial Intelligence*, 147–149.

Gams, A.; Do, M.; Ude, A.; Asfour, T.; and Dillmann, R. 2010. On-line periodic movement and force-profile learning

for adaptation to new surfaces. In *Proc. of the IEEE-RAS Int. Conf. on Humanoid Robots*, 560–565.

Janssen, F., and Fürnkranz, J. 2010. On the quest for optimal rule learning heuristics. *Machine Learning* 78(3):343–379.

Kearns, M., and Singh, S. 2002. Near-optimal reinforcement learning in polynomial time. *Machine Learning* 49(2):209–232.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *Proc. of the European Conference on Machine Learning*, 282–293.

Kormushev, P.; Nenchev, D. N.; Calinon, S.; and Caldwell, D. G. 2011. Upper-body kinesthetic teaching of a free-standing humanoid robot. In *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 3970–3975.

Lang, T., and Toussaint, M. 2010. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research* 39:1–49.

Lang, T.; Toussaint, M.; and Kersting, K. 2010. Exploration in relational worlds. *Machine Learning and Knowledge Discovery in Databases* 178–194.

Little, I., and Thibaux, S. 2007. Probabilistic planning vs replanning. In *Proceedings of the ICAPS07 Workshop on the International Planning Competition: Past, Present and Future*.

Mourao, K.; Petrick, R. P.; and Steedman, M. 2010. Learning action effects in partially observable domains. In *Proc. of European Conference on Artificial Intelligence (ECAI)*, 973–974.

Nemec, B., and Ude, A. 2012. Action sequencing using dynamic movement primitives. *Robotica* 30(5):837.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29(1):309–352.

Sato, F.; Nishii, T.; Takahashi, J.; Yoshida, Y.; Mitsuhashi, M.; and Nenchev, D. 2011. Experimental evaluation of a trajectory/force tracking controller for a humanoid robot cleaning a vertical surface. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3179–3184.

Robot Location Estimation in the Situation Calculus*

Vaishak Belle and **Hector J. Levesque**
Dept. of Computer Science
University of Toronto
Toronto, Ontario M5S 3H5, Canada
{vaishak, hector}@cs.toronto.edu

Abstract

Location estimation is a fundamental sensing task in robotic applications, where the world is uncertain, and sensors and effectors are noisy. Most systems make various assumptions about the dependencies between state variables, and especially about how these dependencies change as a result of actions. Building on a general framework by Bacchus, Halpern and Levesque for reasoning about degrees of belief in the situation calculus, and a recent extension to it for continuous domains, in this paper we illustrate location estimation in the presence of a rich theory of actions using an example. We also show that while actions might affect prior distributions in nonstandard ways, suitable posterior beliefs are nonetheless entailed as a side-effect of the overall specification.

Introduction

The situation calculus (McCarthy and Hayes 1969; Reiter 2001) is an important representational formalism for reasoning about actions. Equipped with a simple ontology, the formalism serves as a foundation for numerous planning languages (Lin and Reiter 1997; Claßen et al. 2007; 2008) and methodologies, such as loopy plans (Levesque 2005) and execution monitoring (Fritz and McIlraith 2007), among others (Gabaldon 2006; Fritz, Baier, and McIlraith 2008). More recently, the agent programming proposal Golog (Levesque and Reiter 1998), formulated as a situation calculus action theory, shows how plans can be filtered based on various (user-specified) control imperatives.

However, when imagining robots operating in the real world, a number of issues arise. The fundamental concern perhaps is that the world is uncertain, and the robot operates in such an environment with noisy sensors and effectors. It then becomes imperative that the underlying formalism, at least in terms of a *specification*, cope with the problems of how the robot is to modify its beliefs based on the actions performed and the results returned by its sensors. At one extreme, calculi such as the situation calculus, are expressive for rich domains, and exploit regularities in the effects that actions have on propositions to describe physical laws compactly. For example, the breaking of a fragile object held by the robot against a broken one getting fixed is written as:

$$\forall a, s.\ Broken(x, do(a,s)) \equiv a = drop(x) \wedge Holding(x,s) \wedge Fragile(x) \vee Broken(x,s) \wedge a \neq repair(x).$$

At the other extreme, sensor fusion and similar phenomena is effortlessly addressed using probabilistic models such as Kalman filtering and Dynamic Bayesian Networks (Dean and Kanazawa 1988; 1989; Dean and Wellman 1991). Unfortunately, while belief update of known priors over Gaussian and other continuous error models is treated appropriately here, very little is said about how actions might change values of certain state variables while not affecting others. These formalisms also assume a full specification of the dependencies between variables, making it difficult to deal with other forms of incomplete knowledge, and strict uncertainty in particular.

The above issues bring to forefront concerns about integrating high-level actions, incomplete information, and probabilistic state estimation in a general way. To see a simple example in the larger context of robots operating and manipulating objects in a 2-dimensional world, imagine the agent located at a certain distance h to the right of a wall, as in Figure 1. The robot might initially believe that h is drawn from a uniform distribution on $[2, 12]$. Among the robot's many capabilities, we imagine the ability of moving left, which might be predicated on the ground's slipperiness. A leftwards motion of 1 unit would shift the uniform distribution on h to $[1, 11]$, but a leftward motion of 4 units would change the distribution more radically. The point $h=0$ would now obtain a weight of .2, while $h \in (0, 8]$ would retain their densities. This mixed distribution would then be preserved by a subsequent rightward motion. Likewise, we might imagine the robot to be equipped with two onboard sensors: a sonar unit aimed at the wall estimating h, and a GPS (global positioning system) device sensing both h and the robot's vertical position. Each of these might be characterized by Gaussian error models, and the effect of a reading from any sensor would revise the distribution on h from uniform to an appropriate Gaussian. The robot is now left with the difficult task of adjusting its beliefs as it moves and obtains competing (perhaps conflicting) measurements from individual sensors. Apart from these concerns, the robot might have to reason about goals, and about how those are to be achieved.

*A version of this paper appears in the *Eleventh International Symposium on Logical Formalizations of Commonsense Reasoning*, 2013.

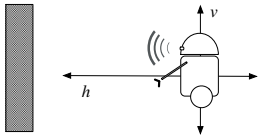


Figure 1: Robot operating in a 2-dimensional world.

Perhaps the most general formalism for dealing with probabilistic belief in formulas, and how that should evolve in the presence of noisy acting and sensing, is a logical account by Bacchus, Halpern and Levesque (BHL) (1999). In the BHL approach, besides quantifiers and other logical connectives, one has the provision for specifying the *degrees of belief* in formulas in the initial state. This specification may be compatible with one or very many initial distributions and sets of independence assumptions. All the properties of belief will then follow at a corresponding level of specificity.

Subjective uncertainty is captured in the BHL scheme using a possible-world model of belief (Kripke 1963; Hintikka 1962; Fagin et al. 1995). Intuitively, the degree of belief in ϕ is defined as a normalized sum over the possible worlds where ϕ is true of some nonnegative *weights* associated with those worlds. To reason about belief change, the BHL model is then embedded in a rich theory of action and sensing provided by the situation calculus (McCarthy and Hayes 1969; Reiter 2001; Scherl and Levesque 2003). The BHL account provides axioms in the situation calculus regarding how the weight associated with a possible world changes as the result of acting and sensing. The properties of belief and belief change then emerge as a direct logical consequence of the initial specifications and these changes in weights.

However, in contrast to the earlier mentioned Bayesian formalisms, one of the limitations of the BHL approach is that it is restricted to fluents whose values are drawn from discrete countable domains. One could say, for example, that $h \in \{2, 3, \ldots, 11\}$ is given an equal weight of .1, but stipulating a continuous uniform distribution and Gaussian sensor error models (and not discrete approximations thereof) is quite beyond the BHL approach. In (Belle and Levesque 2013a), we show how with minimal additional assumptions this serious limitation of BHL can be lifted.

In this paper, we illustrate how the (generalized) BHL scheme is utilized in location estimation using our example consisting of a robot's position in XY-plane, a sonar and a GPS device. Our example supposes that the robot is capable of deterministic physical actions, while the sensors are characterized by continuous error models. We stipulate that the GPS device operates problematically when the robot approaches the wall, perhaps due to signal obstructions, in which case readings are subject to systematic bias. Thus, the domain formalization illustrates belief change with respect to shifting densities as logical properties of actions, competing sensors, and situation-specific bias, among others. Since no assumptions need to be made in general regarding the kind of distributions that initial state variables are drawn from, nor about dependencies between state variables, this work illustrates how beliefs about the robot's location would change after acting and sensing in complex uncertain domains. Owing to these features, we also believe the formalism might be useful to study extensions of planning languages for uncertainty, using the situation calculus as a formal basis (Claßen et al. 2007; 2008). Note that the paper focuses *only* on illustrating the logical specification, that is, properties studied are *entailments* of the background theory. In particular, computational considerations are discussed as part of future work.

The paper is structured as follows. In the next section, we briefly review formal preliminaries, such as the situation calculus, the BHL scheme as well as the essentials of its generalization to continuous domains. We then model the robot domain and illustrate properties about belief change. In the final sections, we discuss related and future work.

Preliminaries

The language $\mathcal{L}$ of the situation calculus (McCarthy and Hayes 1969) is a many-sorted dialect of predicate calculus, with sorts for *actions*, *situations* and *objects* (for everything else, and includes the set of reals $\mathbb{R}$ as a subsort). A situation represents a world history as a sequence of actions. A set of initial situations correspond to the ways the world might be initially. Successor situations are the result of doing actions, where the term $do(a, s)$ denotes the unique situation obtained on doing a in s. The term $do(\alpha, s)$, where α is the sequence $[a_1, \ldots, a_n]$ abbreviates $do(a_n, do(\ldots, do(a_1, s)\ldots))$. Initial situations are defined as those without a predecessor:

$$Init(s) \doteq \neg\exists a, s'.\ s = do(a, s').$$

We let the constant S_0 denote the actual initial situation, and we use the variable ι to range over initial situations only.

In general, the situations can be structured into a set of trees, where the root of each tree is an initial situation and the edges are actions. In dynamical domains, we want the values of predicate and functions to vary from situation to situation. For this purpose, $\mathcal{L}$ includes *fluents* whose last argument is always a situation. Here we assume without loss of generality that all fluents are functional.

Basic action theory Following (Reiter 2001), we model dynamic domains in $\mathcal{L}$ by means of a *basic action theory* $\mathcal{D}$, which consists of [1]

1. axioms $\mathcal{D}_0$ that describe what is true in the initial states, including S_0;
2. precondition axioms that describe the conditions under which actions are executable;
3. successor state axioms that describe the changes to fluents on executing actions;
4. domain-independent *foundational* axioms, the details of which need not concern us here. See (Reiter 2001).

An agent reasons about actions by means of the entailments of $\mathcal{D}$, for which standard Tarskian models suffice. We as-

[1] As usual, free variables in any of these axioms should be understood as universally quantified from the outside.

sume henceforth that models *also* assign the usual interpretations to $=, <, >, 0, 1, +, \times, /, -, e, \pi$ and x^y (exponentials).[2]

Likelihood and degree of belief The BHL model of belief builds on a treatment of knowledge by Scherl and Levesque (2003). Here we present a simpler variant based on just two distinguished binary fluents l and p.

The term $l(a, s)$ is intended to denote the likelihood of action a in situation s. For example, suppose $sonar(z)$ is the action of reading the value z from a sensor that measures the distance to the wall, h.[3] We might assume that this action is characterized by a Gaussian error model:[4]

$$l(sonar(z), s) = u \equiv \\ (z \geq 0 \wedge u = \mathcal{N}(z - h(s); \mu, \sigma^2)) \vee (z < 0 \wedge u = 0)$$

which stipulates that the difference between a nonnegative reading of z and the true value h is normally distributed with a variance of σ^2 and mean of μ. In general, the action theory $\mathcal{D}$ is assumed to contain for each action type A an additional *action likelihood axiom* of the form

$$l(A(\vec{x}), s) = u \equiv \phi_A(\vec{x}, u, s)$$

where ϕ_A is a formula that characterizes the conditions under which action $A(\vec{x})$ has likelihood u in s. (Actions that have no sensing aspect should be given a likelihood of 1.)

Next, the p fluent determines a probability distribution on situations. The term $p(s', s)$ denotes the relative *weight* accorded to situation s' when the agent happens to be in situation s. The properties of p in initial states, which vary from domain to domain, are specified by axioms as part of $\mathcal{D}_0$. The following nonnegative constraint is also included in $\mathcal{D}_0$:

$$\forall \iota, s.\ p(s, \iota) \geq 0 \ \wedge \ (p(s, \iota) > 0 \supset Init(s)) \qquad \text{(P1)}$$

While this is a stipulation about initial states ι only, BHL provide a successor state axiom for p, and show that with an appropriate action likelihood axiom, the nonnegative constraint then continues to hold everywhere:

$$\begin{aligned} p(s', do(a, s)) = u \ \equiv \\ \exists s''\ [s' = do(a, s'') \wedge Poss(a, s'') \wedge \\ u = p(s'', s) \times l(a, s'')] \\ \vee\ \neg\exists s''\ [s' = do(a, s'') \wedge Poss(a, s'') \wedge u = 0] \end{aligned} \qquad \text{(P2)}$$

Now if ϕ is a formula with a single free variable of sort situation,[5] then the *degree of belief* in ϕ is simply defined as the following abbreviation:

$$Bel(\phi, s) \doteq \frac{1}{\gamma} \sum_{\{s' : \phi[s']\}} p(s', s) \qquad \text{(B)}$$

where γ, the normalization factor, is understood throughout as the same expression as the numerator but with ϕ replaced by *true*. For example, here γ is $\sum_{s'} p(s', s)$. We do not have to insist that s' and s share histories since $p(s', s)$ will be 0 otherwise. BHL show how summations can be expressed using second-order logic, see the appendix. That is, neither *Bel*'s definition nor summations are special axioms of $\mathcal{D}$, but simply convenient abbreviations for logical terms. To summarize, in the BHL scheme, an action theory consists of:

1. $\mathcal{D}_0$ as before, but now also including (P1);
2. precondition axioms as before;
3. successor state axioms as before, but now also including one for p *viz.* (P2);
4. foundational domain-independent axioms as before; and
5. action likelihood axioms.

From sums to integrals While the definition of belief in BHL has many desirable properties, it is defined in terms of a *summation* over situations, and therefore precludes fluents whose values range over the reals. The continuous analogue of (B) then requires *integrating* over some suitable space of values.

As it turns out, a suitable space can be found. First, some notation. We use a form of conditional *if-then-else* expressions, by taking some liberties with notation and the scope of variables as follows. We write $f = \textsc{If}\ \exists x.\ \phi\ \textsc{Then}\ t_1\ \textsc{Else}\ t_2$ to mean the logical formula

$$f = u \equiv \exists x.\ [\phi \wedge (u = t_1)] \vee [(u = t_2) \wedge \neg\exists x.\ \phi]$$

Now, assume that there are n fluents $f_1, \ldots, f_n$ in $\mathcal{L}$, and that these take no arguments other than a situation.[6] Next, suppose that there is exactly one initial situation for any vector of fluent values (Levesque, Pirri, and Reiter 1998):

$$[\forall \vec{x} \exists \iota \bigwedge f_i(\iota) = x_i] \ \wedge \ [\forall \iota, \iota'. \bigwedge f_i(\iota) = f_i(\iota') \supset \iota = \iota'] \quad (*)$$

Under these assumptions, it can be shown that the summation over all situations in (B) can be recast as a summation over all possible initial values $x_1, \ldots, x_n$ for the fluents:

$$Bel(\phi, s) \doteq \frac{1}{\gamma} \sum_{\vec{x}} P(\vec{x}, \phi, s) \qquad \text{(B}')$$

where $P(\vec{t}, \phi, s)$ is the (unnormalized) weight accorded to the *successor* of an initial world where f_i equals t_i:

$$\begin{aligned} P(\vec{x}, \phi, do(\alpha, S_0)) \doteq \\ \textsc{If}\ \ \exists \iota.\ \bigwedge f_i(\iota) = x_i \wedge \phi[do(\alpha, \iota)] \\ \textsc{Then}\ \ p(do(\alpha, \iota), do(\alpha, S_0)) \\ \textsc{Else}\ \ 0 \end{aligned}$$

[2] Alternatively, one could specify axioms for characterizing the field of real numbers in $\mathcal{D}$. Whether or not reals with exponentiation is *first-order* axiomatizable remains a major open question.

[3] Naturally, we assume that the value z being read is not under the agent's control. See BHL for a precise rendering of this nondeterminism in terms of Golog operators (Reiter 2001).

[4] Note that $\mathcal{N}$ is a continuous distribution involving π, e, exponentiation, and so on. Therefore, BHL always consider discrete probability distributions that *approximate* the continuous ones.

[5] The ϕ is usually written either with the situation variable suppressed or with a distinguished variable *now*. Either way, $\phi[t]$ is used to denote the formula with that variable replaced by t.

[6] Basically, if we were to assume that the arguments of all fluents, even k-ary ones, are taken from finite sets then this would allow us to enumerate the n random variables of the domain (for some large n). Note that, from the point of view of situation calculus basic action theories, fluents are typically allowed to take *arguments* from any set, including infinite ones. In probabilistic terms, this would this would correspond to having a joint probability distribution over infinitely many, perhaps uncountably many, random variables. We know of no existing work of this sort, and we have as yet no good ideas about how to deal with it.

where α is an action sequence. In a nutshell, because every situation has an initial situation as an ancestor, and because there is a bijection between initial situations and possible fluent values, it is sufficient to sum over fluent values to obtain the belief even for non-initial situations. Note that unlike (B), this one expects the final situation term $do(\alpha, S_0)$ mentioning what actions and observations took place to be explicitly specified, but that is just what one expects when the agent reasons about its belief after doing things, and for the projection problem in particular (Reiter 2001).[7]

The generalization to the continuous case then proceeds as follows. First, we observe that some (though possibly not all) fluents will be real-valued, and that $p(s', s)$ will now be a measure of *density* not weight. Similarly, the P term above now measures (unnormalized) density rather than weight.

Now suppose fluents are partitioned into two groups: the first k take their values $x_1, \ldots, x_k$ from $\mathbb{R}$, while the rest take their values $y_{k+1}, \ldots, y_n$ from countable domains, then the *degree of belief* in ϕ is an abbreviation for:

$$Bel(\phi, s) \doteq \frac{1}{\gamma} \int_{\vec{x}} \sum_{\vec{y}} P(\vec{x} \cdot \vec{y}, \phi, s)$$

The belief in ϕ is obtained by ranging over all possible fluent values, and integrating and summing the densities of situations where ϕ holds.[8] In the appendix, we show how integrals can be formulated using second-order quantification. That is, as before, *Bel*, *P*, integrals and sums are simply convenient abbreviations, and do not involve special axioms in $\mathcal{D}$. More precisely, the continuous extension to BHL has the same components from earlier, with a single revision:

1. $\mathcal{D}_0$ additionally includes $(*)$.

Note that likelihood axioms are specified as before, but we will no longer have to approximate Gaussian error models (or any other continuous models) as would BHL.

Location Estimation

We build a basic action theory $\mathcal{D}$ for a robot in a 2-dimensional grid. We imagine two fluents h and v in addition to *Poss*, l and p. The fluent h gives the distance to the wall and v gives the position of the robot along the vertical axis. We consider two physical actions *left*(z) and *up*(z), and two sensing actions *sonar*(x) and *gps*(x, y).

$\mathcal{D}_0$ includes the following domain-independent axioms: $(*)$ and (P1). Specific to the domain, imagine that $\mathcal{D}_0$ includes the following for p:[9]

$$p(\iota, S_0) = \begin{cases} .1 \times \mathcal{N}(v(\iota); 0, 16) & \text{if } 2 \le h(\iota) \le 12 \\ 0 & \text{otherwise} \end{cases}$$

This says that the value of v is normally distributed about the horizontal axis with variance 16, and independently, that the value of h is uniformly distributed between 2 and 12.[10] No other sentence is included in $\mathcal{D}_0$.

For simplicity, we assume that actions are always executable. $\mathcal{D}$'s successor state axioms are the following. There is a fixed one for p, which is (P2). For h suppose:

$$\begin{aligned} h(do(a, s)) = u \equiv \\ \neg\exists z(a = left(z)) \wedge u = h(s) \ \vee \\ \exists z(a = left(z) \wedge u = \max(0, h(s) - z)). \end{aligned} \tag{1}$$

This says an action *left*(z) moves the robot z units to the left (towards the wall), but that the motion stops if the robot hits the wall. It is also assumed that *left*(z) is the only action that affects h. Of course, to move away from the wall, z can be any negative value. Similarly, for v stipulate:

$$\begin{aligned} v(do(a, s)) = u \equiv \\ \neg\exists z(a = up(z)) \wedge u = v(s) \vee \\ \exists z(a = up(z) \wedge u = v(s) + z). \end{aligned} \tag{2}$$

This captures the upward motion of the robot, while assuming that *up*(z) is the only action affecting v.

Finally, we specify the likelihood axioms in $\mathcal{D}$. We will suppose that the sonar unit, which senses h, is quite accurate:

$$\begin{aligned} l(sonar(z), s) = u \equiv \\ (z \ge 0 \wedge u = \mathcal{N}(h(s) - z; 0, .25)) \\ \vee (z < 0 \wedge u = 0) \end{aligned} \tag{3}$$

which stipulates that the difference between a nonnegative reading of z and the true value h is normally distributed with a variance of .25 and mean of 0. (A mean of 0 indicates that there is no systematic bias in the reading.) For the GPS device, assuming that its absolute readings of latitude and longitude have been converted to relative readings (Hightower and Borriello 2001) for h and v, imagine a bivariate Gaussian error model:

$$l(gps(x, y), s) = \begin{cases} \mathcal{N}(h(s) - z, v(s) - y; \mu_1, \Sigma) & \text{if } h(s) \ge 2 \\ \mathcal{N}(h(s) - z, v(s) - y; \mu_2, \Sigma) & \text{otherwise} \end{cases}$$

where Σ is the 2×2 identity matrix, $\mu_1 = [0 \ \ 0]^T$ and $\mu_2 = [0 \ \ 2]^T$. This says that the components of the Gaussian are independent, and that there is systematic bias in the reading for v when the robot is close to the wall (due to a signal obstructions).

As mentioned earlier, physical actions such as *left*(z) and *up*(z) are assumed to be deterministic for this paper, so they are given trivial likelihoods, for example:

$$\begin{aligned} l(left(z), s) = 1, \\ l(up(z), s) = 1. \end{aligned}$$

This completes the specification of $\mathcal{D}$.

[7] In fact, we can use regression on the ϕ and the p to reduce the belief formula to one involving the initial situation only. More on this in the final section.

[8] We are assuming here that the density function is (Riemann) integrable. If it is not, belief is clearly not defined, nor should it be. Similarly, if the normalization factor is 0, which corresponds to the case of conditioning on an event that has 0 probability, belief should not be (and is not) defined.

[9] Initial beliefs can also be specified for $\mathcal{D}_0$ using *Bel* directly.

[10] We model a simple distribution for illustrative purposes. In general, neither do the variables have to be independent, nor does the specification need to be complete in the sense of mentioning all the variables.

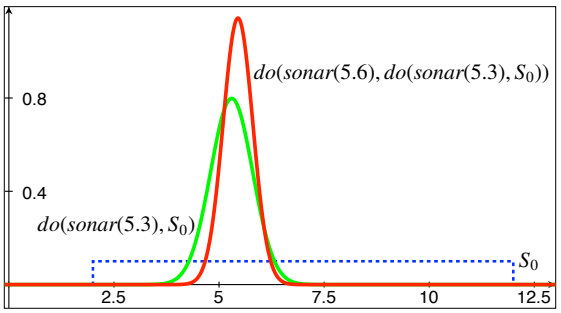


Figure 2: Belief density change for h at S_0 (in blue), after sensing 5.3 (in green), and after finally reading 5.6 (in red).

Theorem 1: *The following are logical entailments of $\mathcal{D}$:*

Initial beliefs

1. $Bel(true, S_0) = 1$.
2. $Bel(h = 2 \vee h = 3 \vee h = 4, S_0) = 0$
   Although we are integrating a density function $q(x_1, x_2)$ over all real values, $q(x_1, x_2) = 0$ unless $x_1 \in \{2, 3, 4\}$.
3. $Bel(5 \leq h \leq 5.5, S_0) = .05$
   Here we are integrating a function that is 0 except when $5 \leq x_1 \leq 5.5$. This is $\int_{\mathbb{R}} \int_5^{5.5} .1 \times \mathcal{N}(x_2; 0, 16)\, \mathrm{d}x_1\, \mathrm{d}x_2 = .05$.

Sensing by sonar

4. $Bel(5 \leq h \leq 5.5, do(sonar(5.3), S_0)) \approx .38$
   Compared to item 3, belief is sharpened significantly by obtaining a reading of 5.3 on the highly sensitive sonar. This is because the p function incorporates the likelihood of a $sonar(5.3)$ action. Starting with the density function in item 3, the sensor reading multiplies the expression to be integrated by $\mathcal{N}(x_1 - 5.3; 0, .25)$, as given by (3). This amounts to evaluating the expression
   $$\int_{\mathbb{R}} \int_A .1 \times \mathcal{N}(x_1 - 5.3; 0, .25) \times \mathcal{N}(x_2; 0, 16)\, \mathrm{d}x_1\, \mathrm{d}x_2$$
   with $A = [5, 5.5]$ for the numerator, and $A = [2, 12]$ for the denominator.
5. $Bel(4.5 \leq h \leq 6.5, do[sonar(5.3), sonar(5.6)], S_0) \approx .99$
   Two successive readings around 5.5 sharpen belief within 1 unit of 5.5 to almost certainty. Compared to item 4, the density function is further multiplied by $\mathcal{N}(x_1 - 5.6; 0, .25)$, and integrated over $[2, 12]$ for the denominator as usual but over $[4.5, 6.5]$ for the numerator. These changing densities are shown in Figure 2.

Physical actions

6. $Bel(h = 0, do(left(4), S_0)) = .2$
   Here a *continuous* distribution evolves into a *mixed* one. By (1), $h = 0$ holds after the action iff $h \leq 4$ held before. This results in $\int_{\mathbb{R}} \int_2^4 .1 \times \mathcal{N}(x_2; 0, 16)\, \mathrm{d}x_1\, \mathrm{d}x_2 = .2$.
7. $Bel(h \leq 5, do(left(4), S_0)) = .7$
   Bel's definition is amenable to a set of h values, where one value has a weight of .2, and all the other real values have a uniformly distributed density of .1. This change in weights is shown in Figure 3.
8. $Bel(h = 4, do([left(4), left(-4)], S_0)) = .2$
   $Bel(h = 4, do([left(-4), left(4)], S_0)) = 0$
   The point $h = 4$ has 0 weight initially (like in item 2). Moving leftwards *first* means many points "collapse", and so this point (now having h value 0) gets .2 weight which is retained on moving away. But not vice versa.
9. $Bel(-1 \leq v \leq 1, do(left(6), S_0))$
   $= Bel(-1 \leq v \leq 1, S_0)$
   $= \int_{-1}^{1} \mathcal{N}(x_2; 0, 16)\mathrm{d}x_2$
   Owing to Reiter's solution to the frame problem, belief in v is unaffected by a lateral motion. For $v \in [-1, 1]$ it is the area between $[-1, 1]$ bounded by the specified Gaussian.
10. $Bel(v \leq 1.5, do(up(3.5), S_0)) = Bel(v \leq -2, S_0)$
    After the action $up(3.5)$, the Gaussian for v's value has its mean "shifted" by 3.5 because the density associated with $v = x_2$ initially is now associated with $v = x_2 + 3.5$.

Sensing by GPS

11. $Bel(-1 \leq v \leq 1, do(gps(5, .1), S_0)) \approx .27$
    Compared to item 9, which evaluates to $\approx .19$, a GPS reading of .1 increases the posterior belief for $v \in [-1, 1]$ to $\approx .27$. Using the error model, this is a result of
    $$\int_2^{12} \int_A .1 \cdot \mathcal{N}(x_2; 0, 16) \cdot \mathcal{N}(x_1 - 5, x_2 - .1; \mu_1, \Sigma)\, \mathrm{d}x_2\, \mathrm{d}x_1$$
    with $A = [-1, 1]$ for the numerator and $A = [-\infty, \infty]$ for the denominator.[11]

Competing sensors

12. $Bel(5 \leq h \leq 5.5, do([gps(5, .1), gps(5.3, .1)], S_0)) \approx .27$
    $Bel(5 \leq h \leq 5.5, do([sonar(5.3), gps(5, .1)], S_0)) \approx .42$
    The sonar is more sensitive than the GPS, and so its reading is far more effective. Relating this to item 4, a GPS reading of 5 for h only slightly redistributes the density.

Systematic bias

13. $h(S_0) \leq 4 \supset$
    $Bel(-1 \leq v \leq 1, do([left(4), gps(1, 0)], S_0)) \approx 0$
    After moving left by 4 units, v's reading from the GPS has a systematic bias of 2. Among other things, this entails that the belief in $v \leq 1$ is almost 0 which is much weaker than its prior from item 9.

Nonstandard properties

14. $Bel(h > 7v, S_0) \approx .6$
    Beliefs about any mathematical expression involving the random variables, even when that does not correspond to

[11] This is a simple instance of Kalman filtering (Dean and Wellman 1991) where the value being sensed is static. Gaussian distributions enjoy the conjugate property: multiplying Gaussians results in another Gaussian (Box and Tiao 1973), and is easily computed.

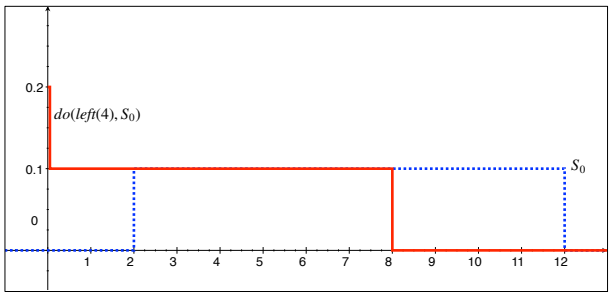


Figure 3: Belief update for h after physical actions. Initial belief at S_0 (in blue) and after a leftward move by 4 (in red).

well known density functions, are entailed. In this case, we are basically evaluating:

$$\int_2^{12} \int_{-\infty}^{x_1/7} .1 \times \mathcal{N}(x_2; 0, 16)\, \mathrm{d}x_2\, \mathrm{d}x_1.$$

15. $Bel([\exists a, s.\ now = do(a, s) \wedge h(s) > 1], do(left(4), S_0)) = 1.$
It is possible to refer to earlier or later situations using *now* as the current situation. This says that after moving, there is full belief that ($h > 1$) held before the action.

Related Work

Sensor fusion has been a primary concern in state estimation approaches (Thrun, Burgard, and Fox 2005). Popular models include variants of Kalman filtering (Fox et al. 2003), where priors and likelihoods are assumed to be Gaussian. We already pointed out that entailment item 11 is a simple instance of Kalman filtering. But in general, our approach does not make any assumptions about the nature of distributions, nor about how distributions and dependencies may evolve after actions, and allows for strict uncertainty. This distinguishes the current method from numerous probabilistic formalisms (Lerner et al. 2002; Dean and Wellman 1991; Fox et al. 2003), including those that handle explicit actions (Darwiche and Goldszmidt 1994; Hajishirzi and Amir 2010). To the best of our knowledge, none of these formalisms have treated cases where state variables change in the manner indicated in the paper.

Probabilistic logical formalisms such as (Halpern 1990; Bacchus 1990) are equipped to handle features such as disjunctions and quantifiers, but they do not explicitly address actions. Relational probabilistic languages and Markov logics (Ng and Subrahmanian 1992; Richardson and Domingos 2006) also do not model actions. Recent temporal extensions, such as (Choi, Guzman-Rivera, and Amir 2011), specifically treat Kalman filtering, but not complex actions. In this regard, action logics such as dynamic and process logics are closely related. Recent proposals, for example (Van Benthem, Gerbrandy, and Kooi 2009), treat sensor fusion. However, these and related frameworks (Halpern and Tuttle 1993), are mostly propositional. In the last years, there have been extensions to the PDDL planning language, so as to account for probabilistic effects and partial observability (Younes and Littman 2004; Sanner 2011). The focus in this literature, as well other probabilistic planning approaches (Kushmerick, Hanks, and Weld 1995), is on certain sorts of initial databases rather than a specification that allows for full first-order expressivity. Be that as it may, a closer examination of PDDL generalizations, and how they relate to situation calculus dialects might nevertheless be useful.

Finally, proposals based on the situation and fluent calculi are first-order (Bacchus, Halpern, and Levesque 1999; Poole 1998; Boutilier et al. 2000; Mateus et al. 2001; Shapiro 2005; Gabaldon and Lakemeyer 2007; Fritz and McIlraith 2009; Belle and Lakemeyer 2011; Thielscher 2001), but none of them deal with continuous sensor noise, and nor do the extensions for continuous processes (Reiter 2001; Herrmann and Thielscher 1996; Fox and Long 2006). For instance, in (Fritz and McIlraith 2009), state variables are assumed to be continuous in nature, but neither probabilistic sensing nor belief change over such sensing results is addressed.

Conclusions and Future Work

This paper illustrates location estimation for a robot operating in an incompletely known world, equipped with noisy sensors. In contrast to a number of competing formalisms, where the modeler is left with the difficult task of deciding how the dependencies and distributions of state variables might evolve, here one need only specify the initial beliefs and the physical laws. Suitable posteriors are then entailed. The framework of the situation calculus, and a recent generalization to the BHL scheme, allows us to additionally specify situation-specific biases and realistic continuous error models. Our example demonstrates that belief changes appropriately even when one is interested in nonstandard properties, such as logical relationships of state variables, all of which emerges as a side-effect of the general specification.

There are a number of avenues for future research. On the representation side, features such as continuous time, exogenous actions, decision theory and durative actions have been proposed in the situation calculus (Reiter 2001), which could be imported to our formalism. From a more computational side, this paper restricted itself to the logical specification. Nevertheless, our reformulation of belief seems amenable to *regression* (Reiter 2001), and a proposal to extend regression over degrees of belief is ongoing work. (See (Belle and Levesque 2013b) for steps in this direction.) Many planning approaches in the situation calculus are based on regression (Fritz and McIlraith 2009; 2007; Levesque 2005), and so it would be useful to explore reasoning about probabilistic belief in dynamic domains with this methodology. More broadly, we are interested in the *achievability* of plans (Levesque 1996), that is, the question of *when* can a plan be found and executed, given noisy effector and sensor specifications.

Appendix: Sums and Integrals in Logic

Logical formulas can be used to characterize sums and a variety of sorts of integrals. Here we show the simplest possible cases: the summing of a one variable function from 1

to n, and the definite integral from $-\infty$ to ∞ of a continuous real-valued function of one variable. Other complications are treated in a longer version of the paper.

First, sums. For any logical term t and variable i, we introduce the following notation to characterize summations:

$$\sum_{i=1}^{n} t = z \doteq \exists f\ [f(1) = t_1^i \wedge f(n) = z \wedge \forall j\ (1 \leq j < n \supset f(j+1) = f(j) + t_{(j+1)}^i\ )]$$

where f is assumed to not appear in t, and j is understood to be chosen not to conflict with any of the variables in t and i.

Now, integrals. We begin by introducing a notation for limits to positive infinity. For any logical term t and variable x, we let $\lim_{x\to\infty} t$ stand for a term characterized by:

$$\lim_{x\to\infty} t = z \ \doteq\ \forall u(u > 0 \supset \exists m\, \forall n(n > m \supset \left|z - t_n^x\right| < u)).$$

The variables u, m, and n are understood to be chosen here not to conflict with any of the variables in x, t, and z.

Then, for any variable x and terms a, b, and t, we introduce a term $\mathrm{INT}[x, a, b, t]$ to stand for the definite integral of t over x from a to b:

$$\mathrm{INT}[x, a, b, t] \ \doteq\ \lim_{n\to\infty} h \cdot \sum_{i=1}^{n} t_{(a+h\cdot i)}^x$$

where h stands for $(b-a)/n$. The variable n is chosen not to conflict with any of the other variables.

Finally, we define the definite integral of t over all real values of x by the following:

$$\int_x t \ \doteq\ \lim_{u\to\infty} \lim_{v\to\infty} \mathrm{INT}[x, -u, v, t].$$

The main result for this logical abbreviation is the following:

Theorem 2: *Let g be a function symbol of $\mathcal{L}$ standing for a function from $\mathbb{R}$ to $\mathbb{R}$, and let c be a constant symbol of $\mathcal{L}$. Let M be any logical interpretation of $\mathcal{L}$ such that the function g^M is continuous everywhere. Then we have the following:*

$$\textit{If } \int_{-\infty}^{\infty} g^M(x)\,.\,dx \ =\ c^M \ \textit{ then } \ M \models (c = \int_x g(x)).$$

References

Bacchus, F.; Halpern, J. Y.; and Levesque, H. J. 1999. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence* 111(1–2):171 – 208.

Bacchus, F. 1990. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press.

Belle, V., and Lakemeyer, G. 2011. A semantical account of progression in the presence of uncertainty. In *Proc. AAAI*, 165–170.

Belle, V., and Levesque, H. J. 2013a. Reasoning about continuous uncertainty in the situation calculus. In *Proc. IJCAI*.

Belle, V., and Levesque, H. J. 2013b. Reasoning about probabilities in dynamic systems using goal regression. In *Proc. UAI*.

Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI*, 355–362.

Box, G. E. P., and Tiao, G. C. 1973. *Bayesian inference in statistical analysis*. Addison-Wesley.

Choi, J.; Guzman-Rivera, A.; and Amir, E. 2011. Lifted relational kalman filtering. In *Proc. IJCAI*, 2092–2099.

Claßen, J.; Eyerich, P.; Lakemeyer, G.; and Nebel, B. 2007. Towards an integration of golog and planning. In *Proc. IJCAI*, 1846–1851.

Claßen, J.; Engelmann, V.; Lakemeyer, G.; and Röger, G. 2008. Integrating Golog and planning: An empirical evaluation. In *NMR Workshop*, 10–18.

Claßen, J.; Hu, Y.; and Lakemeyer, G. 2007. A situation-calculus semantics for an expressive fragment of PDDL. In *Proc. AAAI*, 956–961.

Darwiche, A., and Goldszmidt, M. 1994. Action networks: A framework for reasoning about actions and change under uncertainty. In *Proc. UAI*, 136–144.

Dean, T., and Kanazawa, K. 1988. Probabilistic temporal reasoning. In *Proc. AAAI*, 524–529.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational intelligence* 5(2):142–150.

Dean, T., and Wellman, M. 1991. *Planning and control*. Morgan Kaufmann Publishers Inc.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. MIT Press.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)* 27:235–297.

Fox, D.; Hightower, J.; Liao, L.; Schulz, D.; and Borriello, G. 2003. Bayesian filtering for location estimation. *Pervasive Computing, IEEE* 2(3):24–33.

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *ICAPS*, 144–151.

Fritz, C., and McIlraith, S. A. 2009. Computing robust plans in continuous domains. In *Proc. ICAPS*, 346–349.

Fritz, C.; Baier, J. A.; and McIlraith, S. A. 2008. Congolog, sin trans: Compiling congolog into basic action theories for planning and beyond. In *KR*, 600–610.

Gabaldon, A., and Lakemeyer, G. 2007. ESP: A logic of only-knowing, noisy sensing and acting. In *Proc. AAAI*, 974–979.

Gabaldon, A. 2006. Formalizing complex task libraries in golog. In *ECAI*, 755–756.

Hajishirzi, H., and Amir, E. 2010. Reasoning about deterministic actions with probabilistic prior and application to stochastic filtering. In *Proc. KR*.

Halpern, J. Y., and Tuttle, M. R. 1993. Knowledge, probability, and adversaries. *J. ACM* 40:917–960.

Halpern, J. 1990. An analysis of first-order logics of probability. *Artificial Intelligence* 46(3):311–350.

Herrmann, C. S., and Thielscher, M. 1996. Reasoning about continuous processes. In *AAAI/IAAI, Vol. 1*, 639–644.

Hightower, J., and Borriello, G. 2001. Location systems for ubiquitous computing. *Computer* 34(8):57–66.

Hintikka, J. 1962. *Knowledge and belief: an introduction to the logic of the two notions*. Cornell University Press.

Kripke, S. 1963. Semantical considerations on modal logic. *Acta Philosophica Fennica* 16:83–94.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1):239–286.

Lerner, U.; Moses, B.; Scott, M.; McIlraith, S.; and Koller, D. 2002. Monitoring a complex physical system using a hybrid dynamic bayes net. In *Proc. UAI*, 301–310.

Levesque, H., and Reiter, R. 1998. High-level robotic control: Beyond planning. Position paper at AAAI Spring Symposium on Integrating Robotics Research.

Levesque, H. J.; Pirri, F.; and Reiter, R. 1998. Foundations for the situation calculus. *Electron. Trans. Artif. Intell.* 2:159–178.

Levesque, H. J. 1996. What is planning in the presence of sensing? In *Proc. AAAI / IAAI*, 1139–1146.

Levesque, H. 2005. Planning with loops. In *Proc. IJCAI*, 509–515.

Lin, F., and Reiter, R. 1997. How to progress a database. *Artificial Intelligence* 92(1-2):131–167.

Mateus, P.; Pacheco, A.; Pinto, J.; Sernadas, A.; and Sernadas, C. 2001. Probabilistic situation calculus. *Annals of Math. and Artif. Intell.* 32(1-4):393–431.

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, 463–502.

Ng, R., and Subrahmanian, V. 1992. Probabilistic logic programming. *Information and Computation* 101(2):150–201.

Poole, D. 1998. Decision theory, the situation calculus and conditional plans. *Electron. Trans. Artif. Intell.* 2:105–158.

Reiter, R. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine learning* 62(1):107–136.

Sanner, S. 2011. Relational dynamic influence diagram language (rddl): Language description. Technical report, Australian National University.

Scherl, R. B., and Levesque, H. J. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence* 144(1-2):1–39.

Shapiro, S. 2005. Belief change with noisy sensing and introspection. In *NRAC Workshop*, 84–89.

Thielscher, M. 2001. Planning with noisy actions (preliminary report). In *Proc. Australian Joint Conference on Artificial Intelligence*, 27–45.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics*. MIT Press.

Van Benthem, J.; Gerbrandy, J.; and Kooi, B. 2009. Dynamic update with probabilities. *Studia Logica* 93(1):67–96.

Younes, H., and Littman, M. 2004. PPDDL 1. 0: An extension to pddl for expressing planning domains with probabilistic effects. Technical report, Carnegie Mellon University.