EU FP7 CogX
ICT-215181
May 1 2008 (52months)

DR 4.3:
Planning for Knowledge Changes

Michael Brenner, Richard Dearden, Moritz Göbelbecker, Charles Gretton, Patrick Eyerich, Thomas Keller, Bernhard Nebel

Albert-Ludwigs-Universität Freiburg, University of Birmingham
⟨brenner@informatik.uni-freiburg.de⟩

Due date of deliverable:	31 July 2011
Actual submission date:	29 July 2011
Lead partner:	ALU
Revision:	v2
Dissemination level:	PU

In the CogX project, we want to build autonomous robots that can act in incompletely known dynamic environments. The goal of WP 4 is to enable a robot to plan its actions despite initial gaps in its knowledge and despite uncertainty about action outcomes and about the validity of its models. In this report, we describe the final version of our *switching planner*, a system that toggles between using a fast continual planner to guide the robot's general behaviour and a more computationally expensive decision-theoretic planner to verify assumptions of the continual planner, taking into account the full, complex sensing model. In addition, we describe several specific methods, both in the deterministic and the decision-theoretic setting, to plan for information gathering under uncertainty.

Executive Summary

In the reporting period, we have published two articles describing the so-called *switching planner*, the main promised result of Task 4.1. In these articles we detail the theoretical foundations as well as its practical realisation in the CogX demonstrator "Dora". With the present report, Task 4.1 can therefore be considered complete (although we will continue studying the topic).

We have also continued to investigate planning for information gathering (Task 4.2). The techniques developed in the last year, described in a number of publications in the annexes of this report, permit planning despite knowledge gaps and planning for closing them. In particular, by enabling deterministic planners to reason about uncertainty and sensing actions, the gap between the "classical" and decision-theoretic approaches used in the switching planner is diminished.

Role of Planning in CogX

Planning is a crucial capability for any cognitive agent, because it enables it to act autonomously: rather than just executing pre-defined scripts, a planning agent can devise its own solutions for the goals it is given or which it develops. In the CogX project, planning has an additional important role in detecting and filling gaps in knowledge: information-gathering actions, e.g. active visual search or asking a question, are planned whenever there is a knowledge gap crucial for achieving a goal.

Contribution to the CogX scenarios and prototypes

The planners developed in this workpackage are responsible for all behaviour-related decisions in the CogX prototypes Dora and George.

1 Tasks, objectives, results

1.1 Planned work

WP4 provides the CogX robots with capabilities for decision making under incomplete and uncertain knowledge. Although this is a very complex reasoning task it must be performed in real time to be usable on an interactive robot. This dichotomy puts serious resource constraints on planning. To address them, in the proposed work we identified a switching symbolic/decision-theoretic planner as a desirable goal for the project. This was expressed as Task 4.1.

> Task 4.1: A switching symbolic/decision-theoretic planner. *In this task we will look at how to combine these two approaches (symbolic and decision theoretic) into the switching planner discussed above. The result should be a planner that can do limited reasoning about belief states (the representation in terms of epistemic operators is much less expressive than a full probabilistic belief-state representation) but still make good decisions, and that will operate in close-to real-time.*

While in Task 4.1. the architecture of such a hybrid planning system is investigated, the goal of Task 4.2. is to study representations and methods for reasoning about information gathering in incompletely known and partially observable environments.

> Task 4.2: General planning of information gathering and dialogue actions. *The symbolic planner developed in Task 4.1 is limited in that it uses epistemic operators to represent beliefs, so it can only represent that a fact is known to be true, known to be false, or unknown. Better plans can be achieved by representing a much richer set of beliefs, for example by using probabilistic belief states. ... In Task 4.2 we will extend the planning system to allow arbitrary belief states to be reasoned about. The aim is to produce a planner capable of planning over arbitrary belief states, but specialised for the requirements of our domain.*

1.2 Actual work performed

1.2.1 The Switching Planner

In the reporting period, we have finished building the switching planner, evaluated it in simulation [1] and fully integrated in into the Dora robot demonstrator [2] .

At its core, the switching planner is a continual planner, interleaving planning and execution. The system *switches* in the sense that planning and plan execution proceed in interleaved sessions in which the *base planner* is either *sequential* or *decision-theoretic (DT)*. (The base planners have been

described in DR.4.2.) The planner is activated when a description of the current problem state and domain are posted to the sytem in the DTPDDL language (cf. DR.4.2). The initial planning session is sequential, thus a serial plan is computed that corresponds to one execution-*trace* in the underlying decision-process. That trace is a reward-giving sequence of process actions and *assumptive* actions. Each *assumptive* action corresponds to an assertion about some facts that are unknown at plan time – e.g. that a box of cornflakes is located on the corner bench in the kitchen. The trace specifies a plan and characterises a *deterministic approximation* (see [3]) of the underlying process in which that plan is valuable. Traces are computed by the cost-optimising classical planner which trades off action costs, goal rewards, and determinacy. Execution of a trace proceeds according to the process actions in the order that they appear in the trace. If, according to the underlying belief-state, the outcome of the next action scheduled for execution is not predetermined above a threshold (here 95%), then the system switches to a DT session.

Because online DT planning is impractical for the size of problem we are interested in, DT sessions plan in a small abstract problem defined in terms of the trace from the proceeding sequential session. This abstract state-space is characterised by a limited number of propositions, chosen because they relate evidence about assumptions in the trace. To allow the DT planner to judge assumptions from the trace, we add *disconfirm* and *confirm* actions to the problem for each of them. Those yield a relatively small reward/penalty if the corresponding judgement is true/false. Once the DT planner either confirms or rejects a hypothesis, it returns control back to the continual planner, which treats the outcome of the DT session like the outcome of any other action.

Relevant annexes:

- Annex 2.1 consist of a paper (accepted for AAAI 2011) describing the switching planner in detail as well as the actual variant of DTPDDL used by the system. Here, the approach is evaluated in simulation, but on fairly large problems, the number of states ranging from 10^{21} to more than 10^{36}.
- Annex 2.2 (the paper was accepted for ECMR 2011) presents the switching planner integrated into an actual robot system corresponding to the CogX demonstrator *Dora*. The switching planner is used for a particular task here, so-called *Active Visual Search (AVS*, i.e. the problem of determing a series of actions to localise a specific object in an unknown environment.

1.2.2 Planning of Information Gathering and Dialogue Actions

In the reporting period, we have further investigated methods for planning of information gathering actions and, in general, planning under uncertain and incomplete knowledge.

As already discussed in the project proposal, the real-time constraints of planning on a real robot system make full-blown decision-theoretic planning hard. Two of the papers in this deliverable, attached in Annexes 2.4 and 2.3, therefore investigate how reasoning about information gathering and uncertainty can be efficiently performed in a deterministic planning setting.

Keller and Eyerich [4] describe a new method for determinising probabilistic planning operators, i.e. for turning a problem of planning under uncertainty into a deterministic one. Such a *determinisation* can be used in a wide variety of ways, ranging from continual replanning approaches such as the one at the core of our switching planner over heuristics in search algorithms like LAO$\star$ to initial guidance for, e.g., the UCT-algorithm as used in the PROST planner that is briefly described later. Previous determinization techniques suffered from potential exponential blowup due to parallel, conjunctive probabilistic effects. This is overcome by a technique based on *Forked Normal Form*, a novel normal form for probabilistic planning where parallel probabilistic effects are applied *sequentially* and the exponential blowup in the determinization is avoided.

This technique has also been incorporated into a novel planning system, the PROST planner by Thomas Keller and Patrick Eyerich, which has recently won the International Probabilistic Planning Competition (IPPC) held at ICAPS 2011 (we have not written a paper about the planner yet). PROST is based on a sampling strategy, more specifically on the UCT-algorithm, which is guided initially by a determinzation based heuristic to avoid the random walks that typically prevent UCT from converging to a good strategy fastly.

Somewhat orthogonally, Brenner and Nebel show how branching upon (and therefore possibly determinisation of) possible outcomes can be avoided by introducing the concept of *runtime variables* into the paradigm of Continual Planning. Essentially, outcomes of planned sensing actions that are naturally still unknown at planning time can often still be reasoned about in a sequential plan if properly represented. For example, if a robot plans to ask a human user for the position of a box of cornflakes, it can plan to move to *that place* next without having to branch in advance over all possible answers. The paper shows how Continual Planning with runtime variables, in combination with the concept of *assertions*, can generate *series of sequential plans* that solve planning tasks that in non-continual planning would necessitate plans with conditional branching or even loops.

Finally, Nunez-Varela et al. describe a POMDP approach to reduce uncertainty about the position of objects in table-top manipulation scenario.

While we previously have investigated the POMDP as a model for planning under observational uncertainty in WP4, this treated systems that have a single thread of execution. Here, robots are thought of as having multiple motor systems or multiple threads of execution. Our work extends that of Ballard and Sprague [5]. The problem domain is one in which the robot has several motor systems that can alter the physical state of the world (e.g. manipulators that can pick up and drop objects) and a single perceptual system. The task of the robot is to move objects from the centre of a table to containers on either side. Each arm has high and low level controllers that would enable the completion of the task if the positions of the objects were precisely known. These controllers are learned using a reinforcement learning algorithm. The precise position of the objects, is however, uncertain, and the robot can reduce uncertainty about the pose of an object by looking at it. The resulting observations are integrated using a Bayesian recursive filter (in this case a particle filter). At execution time the robot chooses actions for each arm based on its current estimate of the positions of the objects on the table. It then chooses where to direct its gaze based on the additional reward it can generate by reducing its uncertainty about the state of the environment. The gaze choices the robot has are to look at any of the objects in its visual memory, or at landmark locations in the scene, such as the centre of the table. When uncertainty in the position of a target, such as an object, is reduced an action such as picking that object up becomes more reliable and thus generates a higher level of reward. The method is computationally equivalent to performing one step lookahead in a POMDP.

Relevant annexes:

- Annex 2.4 describes (continual) planning for knowledge changes with runtime variables.
- Annex 2.3 presents the determinisation approach that is used in the PPDDL version of our PROST planning system.
- Annex 2.5 describes our POMDP approach to gaze control.
- Annex 2.3 in DR.6.3 is also relevant for this task. The paper (submitted to ESSLLI SS 2011) describes how ideas from continual planning, namely the concept of *assertions*, can be used in a dialogue system to model knowledge gaps and enforce their "filling" in later phases of the dialogue.
- Finally, Annex 2.4 in DR.5.4 is also relevant here. This work facilitates information gathering actions for the case of learning object models by providing the quantitative measures for model completeness and

information gain of a learning action that are required in order to plan for knowledge changes.

1.3 Relation to the state-of-the-art

There are still only few integrated robot systems based on domain-independent planning. Among those that do, the need for a high-level *continual planning* and execution monitoring subsystem is widely recognised [6, 7, 8]. Yet, only few principled approaches to CP have been described [9, 10]. Our own CP approach is based on the idea of *proactive knowledge-gathering*: instead of planning for all possible contingencies, agents try to learn more about the state of the world directly [11]. In order to enable agents to reason about how they can gather additional knowledge it is necessary to explicitly model the agents' beliefs as well as their sensing capabilities as part of their formal planning domain [12, 13, 10, 14]. The idea of using runtime variables for referring to future results of information gathering actions is not new: Etzioni and colleagues mentioned the use of runtime variables in their systems [12, 15, 10]. Similarly, Petrick and Bacchus described the use of 0-ary functional fluents as runtime variables [14, 16]. None of the previous work, however, clearly defined the semantics of planning with them. To our knowledge, the work by Brenner and Nebel presented in this report is thus the first to provide such a definition [17], in the context of the Functional STRIPS language developed by Geffner [18].

One important open research challenge is to combine continual planning with *probabilistic* models of noisy sensing. Addressing task and observation planning specifically, as our own applications of the switching planner [1, 2], there have been a number of recent developments where the underlying problem is modelled as a POMDP. For vision algorithm selection, Sridharan, Wyatt and Dearden [19] exploit an explicitly modelled hierarchical decomposition of the underlying POMDP. Doshi and Roy [20] represent a preference elicitation problem as a POMDP and take advantage of symmetry in the belief-space to exponentially shrink the state-space. Although we have been actively exploring the Doshi and Roy approach [20], those exploitable symmetries are not present in problems we consider due to the task planning requirement.

Our switching planner approach can be thought of as an online POMDP solver that uses a sequential plan to guide the search, rather than (e.g., Monte-Carlo) sampling. Also, compared to most online POMDP procedures, which replan at each step, our approach involves relatively little replanning. In a similar vein, recent online POMDP solution procedures have been developed which leverage highly approximate value functions – computed using an offline procedure – and heuristics in forward search [21]. These approaches are applicable in relatively small problems, and can require expensive *problem-specific* offline processing in order to yield good be-

haviours. A very recent and promising online approach for larger POMDPs employs Monte-Carlo sampling to break the curse of dimensionality in situations where goal reachability is *easily* determined [22].

In the direction of leveraging *classical* approaches for planning under uncertainty, the most highlighted system to date has been FF-Replan [23], the winning entry from the probabilistic track of the 2004 International Planning Competition. In the continual paradigm, FFR_a uses FF to compute sequential plans and execution traces. Also leveraging deterministic planners in problems that feature uncertainty, CONFORMANT-FF [24] and T_0 [25] demonstrate how conformant planning – i.e., sequential planning in unobservable worlds – can be modelled as a deterministic problem, and therefore solved using sequential systems. In this conformant setting, advances have been towards compact representations of beliefs amenable to existing best-first search planning procedures, and lazy evaluations of beliefs. We consider it an appealing future direction to pursue conformant reasoning during the sequential sessions we proposed. Most recently this research thread has been extended to contingent planning in fully observable non-deterministic environments [26].

In order to leverage classical planning for probabilistic domain, most of the above mentioned systems, in particular those that have successfully participated in the International Probabilistic Planning Competition, e.g. FF-Replan [23], FPG [27] or RFF-(BG/PG) [28], make use of a *determinisation* of the probabilistic planning task. Two classes of determinization strategies have been described: *Single outcome* determinizations choose one possible outcome for each probabilistic operator, accepting that solvable tasks might become unsolvable in the determinization, while *all outcome* determinizations preserve solvability by generating all potential outcomes. The only all outcome determinization used in practice generates one operator for each potential outcome, possibly leading to exponentially many operators in the determinization [29]. The paper by Keller and Eyerich in this report provides a *polynomial* determinisation [4].

References

[1] Moritz Göbelbecker, Charles Gretton, and Richard Dearden. A switching planner for combined task and observation planning. In *Twenty-Fifth Conference on Artificial Intelligence (AAAI-11)*, August 2011.

[2] Alper Aydemir, Moritz Göbelbecker, Kristoffer Sjöö, Andrzej Pronobis, and Patric Jensfelt. Plan-based object search and exploration using semantic spatial knowledge in the real world'. In *Fifth European Conference on Mobile Robots (EMCR'11)*, 2011.

[3] Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *AAAI*, pages 1010–1016, 2008.

[4] Thomas Keller and Patrick Eyerich. A polynomial all outcome determinization for probabilistic planning. In *Proceedings of the 21th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, june 2011.

[5] Nathan Sprague, Dana Ballard, and Al Robinson. Modeling embodied visual behaviors. *ACM Trans. Appl. Percept.*, 4, July 2007.

[6] Jeremy L. Wyatt, Alper Aydemir, Michael Brenner, Marc Hanheide, Nick Hawes, Patric Jensfelt, Matej Kristan, Geert-Jan M. Kruijff, Pierre Lison, Andrzej Pronobis, Kristoffer Sjöö, Danijel Skočaj, Alen Vrečko, Hendrik Zender, and Michael Zillich. Self-understanding and self-extension: A systems and representational approach. *IEEE Transactions on Autonomous Mental Development*, 2(4):282 – 303, December 2010.

[7] K. Talamadupula, J. Benton, S. Kambhampati, P. Schermerhorn, and M. Scheutz. Planning for human-robot teaming in open worlds. *ACM Trans. Intell. Syst. Technol.*, 1:14:1–14:24, December 2010.

[8] D. Kraft, E. Başeski, M. Popović, A. M. Batog, A. Kjær-Nielsen, N. Krüger, R. Petrick, C. Geib, N. Pugeault, M. Steedman, T. Asfour, R. Dillmann, S. Kalkan, F. Wörgötter, B. Hommel, R. Detry, and J. Piater. Exploration and planning in a three-level cognitive architecture. In *CogSys*, 2008.

[9] José A. Ambros-Ingerson and Sam Steel. Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference of the American Association for Artificial Intelligence (AAAI-88)*, pages 83–88, Saint Paul, MI, August 1988.

[10] K. Golden. Leap before you look: Information gathering in the PUCCINI planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages 70–77, 1998.

[11] Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems*, 19(3):297–331, 2009.

[12] Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, pages 115–125, Cambridge, MA, 1992. Morgan Kaufmann.

[13] Hector J. Levesque. What is planning in the presence of sensing? In *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence (AAAI-96)*, pages 1139–1146. MIT Press, 1996.

[14] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, Toulouse, France, 2002. AAAI Press.

[15] Oren Etzioni, Keith Golden, and Daniel S. Weld. Sound and efficient closed-world reasoning for planning. 89(1-2):113–148, 1997.

[16] Ronald P. A. Petrick and Fahiem Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7, 2004, Whistler, British Columbia, Canada*, pages 2–11. AAAI Press, 2004.

[17] Michael Brenner and Bernhard Nebel. On continual planning with runtime variables. In Gerhard Lakemeyer and Sheila A. McIlraith, editors, *Knowing, Reasoning, and Acting: Essays in Honour of Hector J. Levesque*. College Publications, 2011.

[18] Hctor Geffner. Functional Strips: A more flexible language for planning and problem solving. In Jack Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer, Dordrecht, Holland, 2000.

[19] Mohan Sridharan, Jeremy Wyatt, and Richard Dearden. Planning to see: Hierarchical POMDPs for planning visual actions on a robot. *Artif. Intell.*, 174(11):704–725, 2010.

[20] Finale Doshi and Nicholas Roy. The permutable POMDP: Fast solutions to POMDPs for preference elicitation. In *AAMAS*, 2008.

[21] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *J. Artif. Int. Res. (JAIR)*, 32:663–704, July 2008.

[22] D. Silver and J. Veness. Monte-carlo planning in large POMDPs. In *NIPS*, 2010.

[23] Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: A Baseline for Probabilistic Planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 352–359, 2007.

[24] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search: a new approach. *Artif. Intell.*, 170:507–541, May 2006.

[25] Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res. (JAIR)*, 35:623–675, August 2009.

[26] Alexandre Albore, Héctor Palacios, and Héctor Geffner. A translation-based approach to contingent planning. In *IJCAI*, pages 1623–1628, 2009.

[27] Olivier Buffet and Douglas Aberdeen. FF + FPG: Guiding a policy-gradient planner. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 42–48, 2007.

[28] Florent Teichteil-Königsbuch, Guillaume Infantes, and Ugur Kuter. RFF: A Robust, FF-Based MDP Planning Algorithm for Generating Policies with Low Probability of Failure, 2008. IPPC Planner Abstract.

[29] Jussi Rintanen. Expressive Equivalence of Formalisms for Planning with Sensing. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 185–194, 2003.

2 Annexes

2.1 Göbelbecker et al. "A Switching Planner for Combined Task and Observation Planning" (AAAI 2011)

Bibliography Moritz Göbelbecker, Charles Gretton and Richard Dearden. "A Switching Planner for Combined Task and Observation Planning" In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI-11)

Abstract From an automated planning perspective the problem of practical mobile robot control in realistic environments poses many important and contrary challenges. On the one hand, the planning process must be lightweight, robust, and timely. Over the lifetime of the robot it must always respond quickly with new plans that accommodate exogenous events, changing objectives, and the underlying unpredictability of the environment. On the other hand, in order to promote efficient behaviours the planning process must perform computationally expensive reasoning about contingencies and possible revisions of subjective beliefs according to quantitatively modelled uncertainty in acting and sensing.

Towards addressing these challenges, we develop a continual planning approach that switches between using a fast satisficing "classical" planner, to decide on the overall strategy, and decision-theoretic planning to solve small abstract subproblems where deeper consideration of the sensing model is both practical, and can significantly impact overall performance. We evaluate our approach in large problems from a realistic robot exploration domain.

Relation to WP The switching planner is the result of our efforts to make automated planning practically usable for real-time robotics. This paper thus constitutes the main theoretical result of Task 4.1.

2.2 Aydemir et al. "Plan-based Object Search and Exploration Using Semantic Spatial Knowledge in the Real World" (ECMR 2011)

Bibliography Alper Aydemir, Moritz Göbelbecker, Kristoffer Sjöö, Andrzej Pronobis and Patric Jensfelt. "Plan-based Object Search and Exploration Using Semantic Spatial Knowledge in the Real World" In Fifth European Conference on Mobile Robots (EMCR'11))

Abstract In this paper we present a principled planner based approach to the active visual object search problem in unknown environments. We make use of a hierarchical planner that combines the strength of decision

theory and heuristics. Furthermore, our object search approach leverages on the conceptual spatial knowledge in the form of object cooccurences and semantic place categorisation. A hierarchical model for representing object locations is presented with which the planner is able to perform indirect search. Finally we present real world experiments to show the feasibility of the approach.

Relation to WP This paper complements Annex 2.1 by showing how our switching planner is used in practice on a physical robot, namely in the Dora demonstrator scenario.

2.3 Keller and Eyerich "A Polynomial All Outcome Determinization for Probabilistic Planning" (ICAPS 2011)

Bibliography Thomas Keller and Patrick Eyerich "A Polynomial All Outcome Determinization for Probabilistic Planning " In Proceedings of the 21th International Conference on Automated Planning and Scheduling (ICAPS).

Abstract Most predominant approaches in probabilistic planning utilize techniques from the more thoroughly investigated field of classical planning by determinizing the problem at hand. In this paper, we present a method to map probabilistic operators to an equivalent set of probabilistic operators in a novel normal form, requiring polynomial time and space. From this, we directly derive a determinization which can be used for, e. g., replanning strategies incorporating a classical planning system. Unlike previously described all outcome determinizations, the number of deterministic operators is not exponentially but polynomially bounded in the number of parallel probabilistic effects, enabling the use of more sophisticated determinization-based techniques in the future.

Relation to WP This work is part of our investigation of novel ways for planning under uncertainty in Task 4.2, but also provides groundwork for probabilistic extensions of our continual planning approach that may be used in future versions of the switching planner.

2.4 Brenner and Nebel "On Continual Planning with Runtime Variables"

Bibliography Michael Brenner and Bernhard Nebel "On Continual Planning with Runtime Variables" In G. and S. A. McIlraith (Eds.) Knowing, Reasoning, and Acting: Essays in Honour of Hector J. Levesque. College Publications, 2011.

Abstract This article discusses the problem of planning and acting in partially observable environments. In many such domains conditional planning for all contingencies is prohibitively hard. Therefore we advocate a continual planning approach, where decisions can, by means of so-called *assertions*, be deferred until execution time when more information is available. Additionally, we formalize the notion of *runtime variables* as functional fluents, which can act as placeholders for sensing results unknown at planning time. Using runtime variables and assertions we show how a *series of sequential plans* can solve planning tasks that in non-continual planning would necessitate plans with conditional branching and loops.

Relation to WP The paper is part of our investigation of planning for information gathering (Task 4.2) in a *deterministic* planning setting. In such a setting, runtime variables can conveniently represent future knowledge changes, i.e. sensing results yet unknown at planning time, without having to resort to more computationally complex representation schemes.

2.5 Nunez-Varela et al. "Gaze Allocation During Visually Guided Manipulation"

Bibliography J. Nunez-Varela, P. Mani, B. Ravindran and J. Wyatt "Gaze Allocation During Visually Guided Manipulation" Technical Report, University of Birmingham, School of Computer Science.

Abstract In this work we present principled methods for the coordination of a robot's oculomotor system with the rest of its body motor systems. The problem is to decide which physical actions to perform next and where the robot's gaze should be directed in order to gain information that is relevant to the success of its physical actions. Previous work on this problem has shown that a reward-based coordination mechanism provides an efficient solution. However, that approach does not allow the robot to move its gaze to different parts of the scene, it considers the robot to have only one motor system, and assumes that the actions have the same duration. The main contributions of our work are to extend that previous reward-based approach by making decisions about where to fixate the robot's gaze, handling multiple motor systems, and handling actions of variable duration. We compare our approach against two common baselines, random and round robin gaze allocation. We show how our method provides a more effective strategy to allocate gaze where is needed the most.

Relation to WP This work was done in the context of Task 4.2 (planning for information gathering). While we have investigated the POMDP as a model for planning under observational uncertainty in WP4 previously, this treated systems that have a single thread of execution. In fact robots

can be thought of as having multiple motor systems or multiple threads of execution. This paper desribes a model that plans perceptual actions for robots with multiple motor systems, also based on the POMDP formalism.

A Switching Planner for Combined Task and Observation Planning

Moritz Göbelbecker
Albert-Ludwigs-Universität Freiburg, Germany
goebelbe@informatik.uni-freiburg.de

Charles Gretton, Richard Dearden
University of Birmingham, United Kingdom
{c.gretton,R.W.Dearden}@cs.bham.ac.uk

Abstract

From an automated planning perspective the problem of practical mobile robot control in realistic environments poses many important and contrary challenges. On the one hand, the planning process must be lightweight, robust, and timely. Over the lifetime of the robot it must always respond quickly with new plans that accommodate exogenous events, changing objectives, and the underlying unpredictability of the environment. On the other hand, in order to promote efficient behaviours the planning process must perform computationally expensive reasoning about contingencies and possible revisions of subjective beliefs according to quantitatively modelled uncertainty in acting and sensing. Towards addressing these challenges, we develop a *continual planning* approach that *switches* between using a fast satisficing "classical" planner, to decide on the overall strategy, and decision-theoretic planning to solve small abstract subproblems where deeper consideration of the sensing model is both practical, and can significantly impact overall performance. We evaluate our approach in large problems from a realistic robot exploration domain.

Introduction

A number of recent integrated robotic systems incorporate a high-level *continual planning* and execution monitoring subsystem (Wyatt et al. 2010; Talamadupula et al. 2010; Kraft et al. 2008). For the purpose of planning, sensing is modelled deterministically, and beliefs about the underlying state are modelled qualitatively. Both Talamadupula et al. and Wyatt et al. identify continual planning with probabilistic models of noisy sensing and state as an important challenge for future research. Motivating that sentiment, planning according to accurate stochastic models should yield more efficient and robust deliberations. In essence, the challenge is to develop a planner that exhibits speed and scalability similar to planners employed in existing robotic systems – e.g., Wyatt et al. use a satisficing *classical* procedure – and which is also able to synthesise relatively efficient deliberations according to detailed probabilistic models of the environment.

This paper describes a *switching* domain independent planning approach we have developed to address this challenge. Our planner is *continual* in the usual sense that plans are adapted and rebuilt online in reaction to changes to the model of the underlying problem and/or domain – e.g., when goals are modified, or when the topological map is altered by a door being closed. It is integrated on a mobile robot platform that continuously deliberates in a stochastic dynamic environment in order to achieve goals set by the user, and acquire knowledge about its surroundings. Our planner takes problem and domain descriptions expressed in a novel extension of PPDDL (Younes et al. 2005), called *Decision-Theoretic DTPDDL*, for modelling stochastic decision problems that feature partial observability. In this paper we restrict our attention to problem models that correspond to deterministic-action goal-oriented POMDPs in which all actions have non-zero cost, and where an optimal policy can be formatted as a finite horizon contingent plan. Moreover, we target problems of a size and complexity that is challenging to state-of-the-art sequential satisficing planners, and which are too large to be solved directly by decision-theoretic (DT) systems.

Our planner *switches*, in the sense that the base planning procedure changes depending on our robot's subjective degrees of belief, and on progress in plan execution. When the underlying planner is a fast (satisficing) *classical* planner, we say planning is in a *sequential* session, and otherwise it is in a DT session. A *sequential* session plans, and then pursues a high-level strategy – e.g., go to the kitchen bench, and then observe the cornflakes on it. A DT session proceeds in a practically sized *abstract* process, determined according to the current sequential strategy and underlying belief-state.

We evaluate our approach in simulation on problems posed by object search and room categorisation tasks that our indoor robot undertakes. Those feature a deterministic task planning aspect with an active sensing problem. The larger of these problems features 6 rooms, 25 topological places, and 21 active sensing actions. The corresponding decision process has a number of states exceeding 10^{36}, and high-quality plans require very long planning horizons. Although our approach is not optimal, particularly as it relies on the results of satisficing sequential planning directly, we find that it does nevertheless perform better than a purely sequential replanning baseline. Moreover, it is fast enough to be used for real-time decision making on a mobile robot.

Propositionally Factored Decision-Theoretic Planning

We describe the partially observable propositional probabilistic planning problem, with costs and rewards. We model a process state s as the set of propositions that are true of the state. Notationally, $\mathcal{P}$ is the set of propositions, p is an element from that set, and we have $s \subseteq \mathcal{P}$. The underlying process dynamics are modelled in terms of a finite set of *probabilistic STRIPS operators* (Boutilier and Dearden 1994) $\mathcal{A}$ over state-characterising propositions $\mathcal{P}$. We say an action $a \in \mathcal{A}$ is applicable if its precondition $\texttt{pre}(a)$, a set of propositions, are satisfied in the current state – i.e., $\texttt{pre}(a) \subseteq s$. We denote by $\mu_a(a_i^d)$ the probability that nature chooses a deterministic STRIPS effect a_i^d, and for all a we require $\sum_{a_i^d} \mu_a(a_i^d) = 1$.

We are concerned with problems that feature partial observability. Although we could invoke *extended probabilistic STRIPS operators* (Rintanen 2001) to model actions and observations propositionally, we find it convenient for presentation and computation to separate sensing and action. Therefore, we suppose a POMDP has a perceptual model given in terms of a finite set of stochastic *senses* K, deterministic sensing outcomes K^d, and perceptual propositions Π, called *percepts*. In detail, we take an observation o to be a set of percepts , i.e., $o \subseteq \Pi$, and denote by O the set of observations. The underlying state of the process cannot be observed directly, rather, senses $\kappa \in K$ effect an observation $o \in O$ that informs what should be believed about the state the process is in. If action a is applied effecting a transition to a successor state s', then an observation occurs according to the active senses $K(a, s') \subseteq K$. A sense κ is active, written $\kappa \in K(a, s')$, if the senses' action-precondition, $\texttt{pre}_{\mathcal{A}}(\kappa)$, is equal to a, and the state-precondition $\texttt{pre}_{\mathcal{S}}(\kappa) \subseteq \mathcal{P}$ is satisfied by the state s', i.e., $\texttt{pre}_{\mathcal{S}}(\kappa) \subseteq s'$. When a sense is active, nature must choose exactly one outcome amongst a small set of deterministic choices $K^d(\kappa) \equiv \{\kappa_1^d, \ldots, \kappa_k^d\}$, so that for each i we have $\kappa_i^d \subseteq \Pi$. The probability of the i^{th} element being chosen is given by $\psi_\kappa(\kappa_i^d)$, where $\sum_{\kappa_i^d \in K^d(\kappa)} \psi_\kappa(\kappa_i^d) = 1$. The observation received by the agent corresponds to the union of perceptual propositions from the chosen elements of active senses.

A POMDP has a starting configuration that corresponds to a Bayesian belief-state. Intuitively, this is the robot's subjective belief about its environment. Formally, a belief-state b is a probability distribution over process states. We write $b(s)$ to denote the probability that the process is in s according to b, and b_0 when discussing the starting configuration.

Costs, Rewards, and Belief Revision

Until now we have discussed the POMDP in terms of propositions and percepts. In order to address belief revision and utility it is convenient to consider the underlying decision process in a flat format. This is given by the tuple $\langle \mathcal{S}, b_0, \mathcal{A}, \mathsf{Pr}, \mathsf{R}, O, v \rangle$. Here, b_0 is the initial belief-state, $\mathcal{S}$ is the finite set of reachable propositional states, $\mathcal{A}$ is the finite set of actions, and O is the finite set of reachable observations. Where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, from μ we have a state transition function $\mathsf{Pr}(s, a, s')$ giving the probability of a transition from state s to s' if a is applied. For any s and a we have $\sum_{s' \in \mathcal{S}} \mathsf{Pr}(s, a, s') = 1$. The function $\mathsf{R} : \mathcal{S} \times \mathcal{A} \to \Re$ is a bounded real valued reward function. Therefore a finite positive constant c exists so that for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $|\mathsf{R}(s, a)| < c$. We model costs as negative rewards. From ψ we have that for each $s' \in \mathcal{S}$ and action $a \in \mathcal{A}$, an observation $o \in O$ is generated independently according to a probability distribution $v(s', a)$. We denote by $v_o(s', a)$ the probability of getting observation o in state s'. For s' and a we have $\sum_{o \in O} v_o(s', a) = 1$.

Successive state estimation is by application of Bayes' rule. Taking the current belief b as the *prior*, and supposing action a is executed with perceptive outcome o, the probability that we are in s' in the successive belief-state b' is:

$$b'(s') = \frac{v_o(s', a) \sum_{s \in \mathcal{S}} \mathsf{Pr}(s, a, s') b(s)}{\texttt{Pr}(o|a, b)} \qquad (1)$$

where $\texttt{Pr}(o|a, b)$ is a normalising factor, giving the probability of getting observation o if a is applied to b.

Plan Evaluation

An optimal solution to a finite-horizon POMDP is a contingent plan, and can be expressed as a mapping from observation histories to actions. Although suboptimal in general, useful plans can also take a classical sequential format. This is the case in *conformant* planning, where the objective is to find a sequence of actions that achieves a goal —i.e., reaches a state that satisfies a given Boolean condition— with probability 1. Generally, whatever the plan format, its value corresponds to the expected reward:

$$V_{\texttt{PLAN}}(b) = \mathsf{E}\left[\sum_{t=0}^{N-1} \mathsf{R}(b_t, \texttt{PLAN}_t) \mid \texttt{PLAN}, b_0 = b \right] \qquad (2)$$

Where b_t is the belief-state at step t, $\texttt{PLAN}_t$ is the action prescribed at step t, and

$$\mathsf{R}(b, a) = \sum_{s \in \mathcal{S}} b(s) \mathsf{R}(s, a).$$

Planning Language and Notations

We give an overview of the declarative first-order language DTPDDL, an extension of PPDDL that can express probabilistic models of the sensing consequences of acting, to quantitatively capture unreliability in perception. There are straightforward compilations from problems expressed in DTPDDL to flat state-based (and propositionally factored) representations of the underlying decision process. Although similar to the POND input language (Bryce, Kambhampati, and Smith 2008), DTPDDL distinguishes itself by explicitly treating state and perceptual symbols separately, and by providing distinct declarations for operators (i.e, state model) and senses (i.e., observation model). In this last respect, DTPDDL admits more compact domain descriptions where sensing effects are common across multiple operators. In detail, DTPDDL has perceptual analogues of fluent

and predicate symbols. For example, a simple *object search* domain would have:

```
(:functions
 (is-in ?v - visual-object) - location )
(:perceptual-functions
 (o-is-in ?v - visual-object) - location )
```

Where the first fluent symbol models the actual location of objects, and the second the instantaneous sensing of objects following application of an action with sensing consequences. To model sensing capabilities, we have operator-like "sense" declarations, with preconditions expressed using state and action symbols, and uniformly positive effects over perceptual symbols. For example, where *look-for-object* is the operator that applies an object detection algorithm at a specific place, an *object search* task will have:

```
(:sense vision :parameters
 (?r -robot ?v -visual-object ?l -location)
 :execution (look-for-object ?r ?v ?l)
 :precondition (and (= (is-in ?r) ?l) )
 :effect (and
  (when (= (is-in ?v) ?l)
   (probabilistic .8 (= (o-is-in ?v) ?l)))
  (when (not (= (is-in ?v) ?l))
   (probabilistic .1 (= (o-is-in ?v) ?l)))))
```

I.e., there is a 10% *false positive* rate, and 20% probability of a *false negative*. This representation allows us to represent actions that have multiple independent observational effects.

The DTPDDL syntax for describing an initial state distribution is taken verbatim from PPDDL. That distribution is expressed in a tree-like structure of terms. Each term is either: (1) atomic, e.g., a state proposition such as $(= \texttt{(is-in box)office})$, (2) probabilistic, e.g., $(\texttt{probabilistic}\ \rho_1(T_1)..\rho_n(T_n))$ where T_i are conjunctive, or (3) a conjunct over probabilistic and atomic terms. The root term is always conjunctive, and the leaves are atomic. For example, a simplified object search could have:[1]

```
(:init (= (is-in Robot) kitchen)
 (probabilistic .8 (= (is-in box) kitchen)
                .2 (= (is-in box) office))
 (probabilistic .3 (= (is-in cup) office)
                .7 (= (is-in cup) kitchen)))
```

The interpretation is given by a *visitation* of terms: An atom is *visited* iff its conjunctive parent is visited, and a conjunctive term is visited iff all its immediate subterms are visited. A probabilistic term is visited iff its conjunctive parent is visited, and exactly one of its subterms, T_i, is visited. Each visitation of the root term according to this recursive definition defines a starting state, along with the probability that it occurs. The former corresponds to the union of all visited atoms, and the latter corresponds to the product of ρ_i entries on the visited subterms of probabilistic elements. Making this concrete, the above example yields the following flat distribution:

[1] In PDDL, $(\texttt{:init}\ T_1..T_n)$ expresses the conjunctive root of the tree – i.e., the root node $(\texttt{and}\ T_1..T_n)$. Also, we shall write p, rather than $(\texttt{and}\ p)$, for conjunctive terms that contain a single atomic subterm.

Probability	(is-in Robot)	(is-in box)	(is-in cup)
.24	kitchen	kitchen	office
.06	kitchen	office	office
.56	kitchen	kitchen	kitchen
.14	kitchen	office	kitchen

Switching Continual Planner

We now describe our *switching* planning system that operates according to the continual planning paradigm. The system *switches* in the sense that planning and plan execution proceed in interleaved sessions in which the *base planner* is either *sequential* or *decision-theoretic*. The first session is sequential, and begins when a DTPDDL description of the current problem and domain are posted to the system. During a sequential session a serial plan is computed that corresponds to one execution-*trace* in the underlying decision-process. That trace is a reward-giving sequence of process actions and *assumptive* actions. Each *assumptive* action corresponds to an assertion about some facts that are unknown at plan time – e.g. that a box of cornflakes is located on the corner bench in the kitchen. The trace specifies a plan and characterises a *deterministic approximation* (see (Yoon et al. 2008)) of the underlying process in which that plan is valuable. Traces are computed by a cost-optimising classical planner which trades off action costs, goal rewards, and determinacy. Execution of a trace proceeds according to the process actions in the order that they appear in the trace. If, according to the underlying belief-state, the outcome of the next action scheduled for execution is not predetermined above a threshold (here 95%), then the system switches to a DT session.

Because online DT planning is impractical for the size of problem we are interested in, DT sessions plan in a small abstract problem defined in terms of the trace from the proceeding sequential session. This abstract state-space is characterised by a limited number of propositions, chosen because they relate evidence about assumptions in the trace. To allow the DT planner to judge assumptions from the trace, we add *disconfirm* and *confirm* actions to the problem for each of them. Those yield a relatively small reward/penalty if the corresponding judgement is true/false. If a judgement action is scheduled for execution, then the DT session is terminated, and a new sequential session begins.

Whatever the session type, our continual planner maintains a factored representation of successive belief-states. As an internal representation of the $(\texttt{:init})$ declaration, we keep a tree-shaped Bayesian network which gets updated whenever an action is performed, or an observation received. That belief-state representation is used: (1) as the source of candidate determinisations for sequential planning, (2) in determining when to switch to a DT session, and (3) as a mechanism to guide construction of an abstract process for DT sessions.

Sequential Sessions

As we only consider deterministic-action POMDPs, all state uncertainty is expressed in the $(\texttt{:init})$ declaration. This declaration is used by our approach to define the starting state for sequential sessions, and the set of assumptive ac-

tions available to sequential planning. Without a loss of generality we also suppose that actions do not have negative preconditions. For a sequential session the starting state corresponds to the set of facts that are true with probability 1. Continuing our example, that starting state is the singleton:

$$s_0 \equiv \{(= (\texttt{is-in Robot})\ \texttt{kitchen})\}.$$

To represent state assumptions we augment the problem posed during a sequential session with an *assumptive action* $\mathcal{A}^{\circ}(\rho_i; T_i)$ for each element, $\rho_i(T_i)$, of each probabilistic term from (:init). Here, $\mathcal{A}^{\circ}(\rho_i; T_i)$ can be executed if no $\mathcal{A}^{\circ}(\rho_j; T_j)$, $j \neq i$, has been executed from the same probabilistic term, and, either (probabilistic ..ρ_i (T_i)..) is in the root conjunct, or it occurs in T_k for some executed $\mathcal{A}^{\circ}(\rho_k; T_k)$. We also add constraints that forbid scheduling of assumptions about facts after actions with preconditions or effects that mention those facts. For example, the robot cannot assume it is plugged into a power source immediately after it unplugs itself. Executing $\mathcal{A}^{\circ}(\rho_i; T_i)$ in a state s effects a transition to a successor state s^{T_i}, the union of s with atomic terms from T_i, and of course annotated with auxiliary variables that track the applicability of assumptive actions. For example, consider the following sequential plan:

$\mathcal{A}^{\circ}(.8; (= (\texttt{is-in box})\ \texttt{kitchen}))$;
$\mathcal{A}^{\circ}(.3; (= (\texttt{is-in cup})\ \texttt{office}))$;
$(\texttt{look box kitchen})$; $(\texttt{look cup office})$;
$(\texttt{report box kitchen})$; $(\texttt{report cup office})$

Applying the first action in s_0 yields a state in which the following facts are true:

$\{(= (\texttt{is-in Robot})\ \texttt{kitchen}), (= (\texttt{is-in box})\ \texttt{kitchen})\}$

In the underlying belief-state, this is true with probability 0.8. The assumed state before the scheduled execution of action $(\texttt{look box kitchen})$ is:

$\{(= (\texttt{is-in Robot})\ \texttt{kitchen}), (= (\texttt{is-in box})\ \texttt{kitchen}),$
$(= (\texttt{is-in cup})\ \texttt{office})\}$

Which is actually true with probability 0.24 according to the underlying belief.

To describe the optimisation criteria used during sequential sessions we model $\mathcal{A}^{\circ}(\rho_i; T_i)$ probabilistically, supposing that its application in state s effects a transition to s^{T_i} with probability ρ_i, and to $s^{\perp}$ with probability $1 - \rho_i$. State $s^{\perp}$ is an added sink. Taking ρ_i to be the probability that the i^{th} sequenced action, a_i, from a trace of state-action pairs $\langle s_0, a_0, s_1, a_1, .., s_N \rangle$ does not transition to $s^{\perp}$, then the optimal sequential plan has value:

$$V^* = \max_N \max_{s_0, a_0, .., s_N} \prod_{i=1..N-1} \rho_i \sum_{i=1..N-1} \mathsf{R}(s_i, a_i),$$

DT Sessions

When an action is scheduled whose outcome is uncertain according to the underlying belief-state, the planner switches to a DT session. That plans for *small* abstract processes defined according to the action that triggered the DT session, the assumptive actions in the proceeding trace, and the current belief-state. Targeted sensing is encouraged by augmenting the reward model to reflect a heuristic value of knowing the truth about assumptions. In detail, all rewards from the underlying problem are retained. Additionally, for each relevant assumptive action $\mathcal{A}^{\circ}(\rho_i; T_i)$ in the current trace, we have a *disconfirm action* $\mathcal{A}^{\bullet}(\rho_i; T_i)$ so that for all states s:

$$\mathsf{R}(s, \mathcal{A}^{\bullet}(\rho_i; T_i)) = \begin{cases} \$(T_i) & \texttt{if}\ \ T_i \not\subseteq s \\ \hat{\$}(T_i) & \texttt{otherwise} \end{cases}$$

where $\$(T_i)$ (resp. $\hat{\$}(T_i)$) is a small positive (negative) numeric quantity which captures the utility the agent receives for correctly (incorrectly) rejecting an assumption. In terms of action physics, a disconfirm action can only be executed once, and otherwise is modelled as a self-transformation. We only consider *relevant* assumptions when constructing the abstract model. If $\tilde{a}$ is the action that switched the system to a DT session, then an assumption $\mathcal{A}^{\circ}(\rho_i; T_i)$ is *relevant* if it is necessary for the outcome of $\tilde{a}$ to be determined. For example, taking the switching action $\tilde{a}$ to be $(\texttt{look box kitchen})$ from our earlier sequential plan example, we have that $\mathcal{A}^{\circ}(.3; (= (\texttt{is-in cup})\texttt{office}))$ is not relevant, and therefore we exclude the corresponding disconfirm action from the abstract decision process. Given $\tilde{a}$, we also include another once-only self-transition action $\mathcal{A}.\texttt{pre}(\tilde{a})$, a *confirmation action* with the reward property:

$$\mathsf{R}(s, \mathcal{A}.\texttt{pre}(\tilde{a})) = \begin{cases} \$(\texttt{pre}(\tilde{a})) & \texttt{if}\ \ \texttt{pre}(\tilde{a}) \subseteq s \\ \hat{\$}(\texttt{pre}(\tilde{a})) & \texttt{otherwise} \end{cases}$$

Execution of either a disconfirmation or the confirmation action returns control to a sequential session, which starts anew from the underlying belief-state.

Turning to the detail of (dis-)confirmation rewards, in our integrated system these are sourced from a motivational subsystem. In this paper, for $\mathcal{A}^{\bullet}(\rho_i; T_i)$ actions we set $\$(x)$ to be a small positive constant, and have $\hat{\$}(x) = -\$(x)(1 - \rho)/\rho$ where ρ is the probability that x is true. For $\mathcal{A}.\texttt{pre}(\tilde{a})$ actions we have $\hat{\$}(x) = -\$(x)\rho/(1 - \rho)$.

In order to guarantee fast DT sessions, those plan in an abstract process determined by the current trace and underlying belief-state. The abstract process posed to the DT planner is constructed by first constraining as statically false all propositions except those which are true with probability 1, or which are the subject of *relevant* assumptions. For example, taking the above trace with assumptive action probabilities changed to reflect the belief-state in Fig. 1B, given switching action "(look box kitchen)" the underlying belief in Fig. 1B would determine a fully constrained belief given by Fig. 1A. Next, static constraints are removed, one proposition at a time, until the number of states that can be true with non-zero probability in the initial belief of the abstract process reaches a given threshold. In detail, for each statically-false proposition we compute the *entropy* of the relevant assumptions of the current trace *conditional* on that proposition. Let X be a set of propositions and 2^X the powerset of X, then taking

$$\chi = \{ \bigwedge_{x \in X' \cap X} x \ \wedge \bigwedge_{x \in X \setminus X'} \neg x \mid X' \in 2^X \},$$

we have that χ is a set of conjunctions each of which corresponds to one truth assignment to elements in X. Where

(A) Fully constrained belief

```
(:init (=(is-in Robot)kitchen)
 (.6(=(is-in box)kitchen)))
```

(B) Underlying DTPDDL belief

```
(:init (=(is-in Robot)kitchen)
 (.6(and(=(is-in box)kitchen)
      (.9(=(is-in milk)kitchen))
       .1(=(is-in milk)office))
  .4(and(=(is-in box)office)
      (.1(=(is-in milk)kitchen))
       .9(=(is-in milk)office)))
 (.6(=(is-in cup)office)
  .4(=(is-in cup)kitchen)))
```

(C) Partially constrained belief

```
(:init (=(is-in Robot)kitchen)
 (.6(and(=(is-in box)kitchen)
      (.9(=(is-in milk) kitchen))
       .1(=(is-in milk)office))
  .4(and(=(is-in box)office)
      (.1(=(is-in milk)kitchen))
       .9(=(is-in milk)office)))
```

Figure 1: Simplified examples of belief-states from DT sessions.

$p(\phi)$ gives the probability that a conjunction ϕ holds in the belief-state of the DTPDDL process, the entropy of X *conditional* on a proposition y, written $\mathrm{H}(X|y)$, is given by Eq. 3.

$$\mathrm{H}(X|y) = \sum_{x \in X, y' \in \{y, \neg y\}} p(x \wedge y') \log_2 \frac{p(y')}{p(x \wedge y')} \qquad (3)$$

A low $\mathrm{H}(X|y)$ value suggests that knowing the truth value of y is useful for determining whether or not some assumptions X hold. When removing a static constraint on propositions during the abstract process construction, y_i is considered before y_j if $\mathrm{H}(X|y_i) < \mathrm{H}(X|y_j)$. For example, if the serial plan assumes the box is in the kitchen, then propositions about the contents of kitchens containing a box, e.g. $(=$ `(is-in milk)kitchen`$)$, are added to characterise the abstract process' states. Taking a relevant assumption X to be $(=$ `(is-in box)kitchen`$)$, in relaxing static constraints the following entropies are calculated:

```
.47 = H(X|(=(is-in milk)office))
    = H(X|(=(is-in milk)kitchen))
.97 = H(X|(=(is-in cup)office))
    = H(X|(=(is-in cup)kitchen))
```

Therefore, the first static constraint to be relaxed is for `(=(is-in milk)office)`, or equivalently `(=(is-in milk)kitchen)`, giving a refined abstract belief state depicted in Fig. 1C. Summarising, if for Fig.1B the DT session is restricted to belief-states with fewer than 8 elements, then the starting belief-state of the DT session does not mention a "cup".

Evaluation

We have implemented our switching approach in the MAPSIM environment (Brenner and Nebel 2009), using DLIB-ML (King 2009) for belief revision. Sequential sessions use a modified version of Fast Downward (Helmert 2006), and DT sessions use our own contingent procedure. Since most of the problems we consider are much larger than any available DT planner can solve directly, for comparison purposes we also implemented a simple dual-mode replanning baseline approach. Here, when a switching action is scheduled for execution the DT session applies a single entropy reduction action, whose execution can provide evidence regarding the truth value of a relevant assumption. Control is then immediately returned to a new sequential session.

We evaluate our approaches in robot exploration tasks from home and office environments. Spatially, these consist of *rooms* (office/kitchen/etc), and an underlying topological map over smaller areas of space, called *places*, and connectivity between those. The mobile *robot* and *visual objects* inhabit the topological places. Objects indicate the category of space they inhabit – e.g., spoons are likely to be in kitchens. By examining view cones at places for particular objects, the robot is able to: (1) categorise space at high (room) and low (place) levels, and (2) find objects for the user, exploiting information about object co-occurrence and room categories for efficiency. Also, in the presence of a person, the robot can ask about the category of the current room.

We compare switching to the baseline in several realistic tasks, with the number of rooms ranging from 3 (12-places, 16-objects, $|\text{states}| > 10^{21}$) to 6 (26-places, 21-objects, $|\text{states}| > 10^{36}$). We also compare those systems with near optimal policies computed using Smith's ZMDP for small 2 room problems (4-places, 3-objects, $|\text{states}| \simeq 5000$). Our evaluation considers 3 levels of reliability in sensing: *reliable* sensors have a .1 probability of a false negative, *semi-reliable* have a chance of 0.3 of false negative and 0.1 of false positive, and *noisy* sensors with probabilities of 0.5 and 0.2 respectively. Each object class is assigned one sensor model – e.g. cornflakes may be harder to detect than refrigerators. We performed several experiments with different levels of reliability for sensing the target object(s), while keeping sensing models for non-target objects constant.

Our evaluation examines DT sessions with initial belief-states admitting between 20 and 100 abstract states with non-zero probability. We run 50 simulations in each configuration, and have a timeout on each simulation of 30 minutes (1800 seconds)[2]. The continual planning times are reported in Fig. 2, and the quality data in Fig. 3. For each task, the goal is to find one or more objects and report their position to a user. Usually there is a non-zero probability that no plan exists, as the desired object might not be present in the environment. In these experiments we only allocate reward on the achievement of all goals, therefore we find it intuitive to report average plan costs and the success rates in problems that admit a complete solution (i.e., positive reward scaled by a constant factor). The exception occurs for items f and g of Fig. 3, where we report expected discounted rewards (not plan costs).

We find that if sensing is reliable, then little is gained using DT sessions, as the greedy approach of the baseline is sufficient. As sensing degrades DT sessions prove more useful. Here, time spent on DT planning increases steeply as the abstraction becomes more refined, which is compensated for by fewer planning sessions overall. More detailed abstractions lead to a better overall success rate, particularly for tasks d and e. Speaking to the effectiveness of our entropy heuristic for abstraction refinement, we see relatively

[2]All experiments were conducted on a 2.66GHz Intel Xeon X5355 using one CPU core.

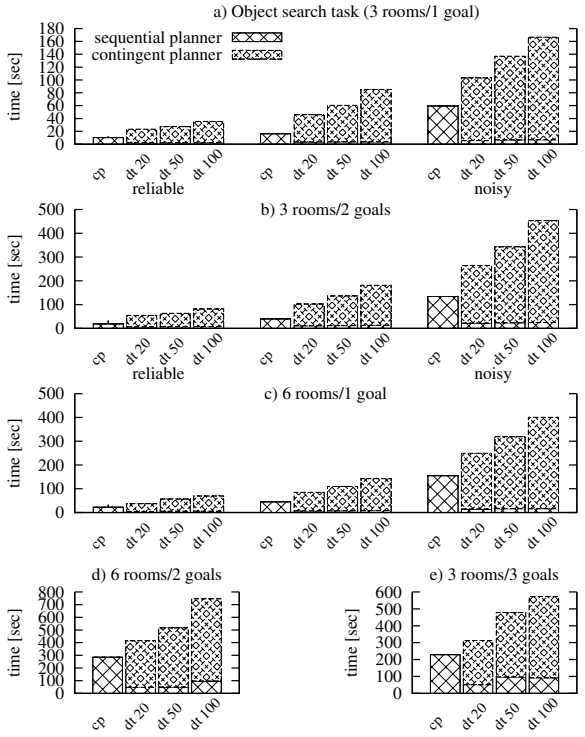


Figure 2: Average runtime

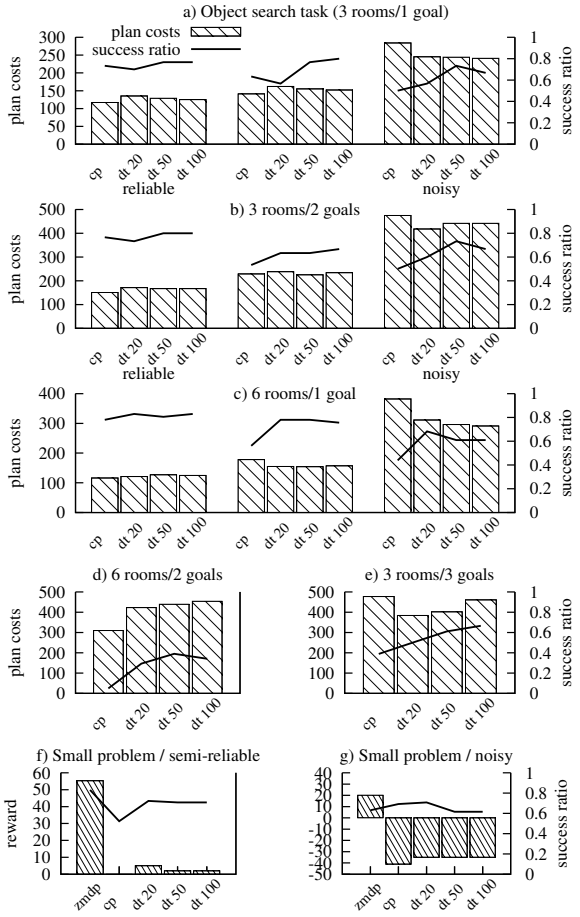


Figure 3: Average plan costs and number of successful runs.

high success rates irrespective of the level of refinement. Comparing finally to the best ZMDP policy, although producing relatively costly plans, the continual planners performed quite well, especially in terms success rate. A key source of inefficiency here, is due to sequential sessions always being optimistic, and refusing to abandon the search.

Related Work

Addressing task and observation planning specifically, there have been a number of recent developments where the underlying problem is modelled as a POMDP. For vision algorithm selection, Sridharan, Wyatt, and Dearden (2010) exploit an explicitly modelled hierarchical decomposition of the underlying POMDP. Doshi and Roy (2008) represent a preference elicitation problem as a POMDP and take advantage of symmetry in the belief-space to exponentially shrink the state-space. Although we have been actively exploring the Doshi and Roy approach, those exploitable symmetries are not present in problems we consider due to the task planning requirement. Also, our approach is in a similar vein to *dual-mode* control (Cassandra, Kaelbling, and Kurien 1996), where planning switches between entropy and utility focuses.

There has also been much recent work on scaling offline approximate POMDP solution procedures to medium-sized instances. Recent contributions propose more efficient belief-point sampling schemes (Kurniawati et al. 2010; Shani, Brafman, and Shimony 2008), and factored representations with procedures that can efficiently exploit structures in those representations (Brunskill and Russell 2010; Shani et al. 2008). Offline domain independent systems scale to *logistics* problems with 2^{22} states (Shani et al. 2008), taking over an hour to converge, and around 10 seconds on average to perform each Bellman backup. Brunskill and Russell are able to solve problems with approximately 10^{30} states, by further exploiting certain problem features – e.g., problems where no actions have negative effects. Moving someway towards supporting real-time decision making, recent online POMDP solution procedures have been developed which leverage highly approximate value functions – computed using an offline procedure – and heuristics in forward search (Ross et al. 2008). These approaches are applicable in relatively small problems, and can require expensive *problem-specific* offline processing in order to yield good behaviours. A *very* recent and promising online approach for larger POMDPs employs Monte-Carlo sampling to break the curse of dimensionality in situations where goal reachability is *easily* determined (Silver and Veness 2010). Our approach can also be thought of as an online POMDP solver that uses a sequential plan to guide the search, rather

than (e.g., Monte-Carlo) sampling. Also, compared to most online POMDP procedures, which replan at each step, our approach involves relatively little replanning.

In the direction of leveraging *classical* approaches for planning under uncertainty, the most highlighted system to date has been FFR$_a$ (Yoon, Fern, and Givan 2007); The winning entry from the probabilistic track of the 2004 International Planning Competition. In the continual paradigm, FFR$_a$ uses FF to compute sequential plans and execution traces. More computationally expensive approaches in this vein combine sampling strategies on valuations over *runtime variables* with deterministic planning procedures (Yoon et al. 2008).

Also leveraging deterministic planners in problems that feature uncertainty, CONFORMANT-FF (Hoffmann and Brafman 2006) and T_0 (Palacios and Geffner 2009) demonstrate how conformant planning – i.e., sequential planning in unobservable worlds – can be modelled as a deterministic problem, and therefore solved using sequential systems. In this conformant setting, advances have been towards compact representations of beliefs amenable to existing best-first search planning procedures, and lazy evaluations of beliefs. We consider it an appealing future direction to pursue conformant reasoning during the sequential sessions we proposed. Most recently this research thread has been extended to contingent planning in fully observable non-deterministic environments (Albore, Palacios, and Geffner 2009).

Concluding Remarks

We have addressed a key challenge, specifically that of high-level continual planning for efficient deliberations according to rich probabilistic models afforded by recent integrated robotic systems. We developed a system that can plan quickly given large realistic probabilistic models, by switching between: (a) fast sequential planning, and (b) expensive DT planning in small abstractions of the problem at hand. Sequential and DT planning is interleaved, the former identifying a rewarding sequential plan for the underlying process, and the latter solving small sensing problems posed during sequential plan execution. We have evaluated our system in large real-world task and observation planning problems, finding that it performs quickly and relatively efficiently.

Acknowledgements: The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement No. 215181, CogX.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *IJCAI*, 1623–1628.

Boutilier, C., and Dearden, R. 1994. Using abstractions for decision-theoretic planning with time constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1016–1022.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems* 19(3):297–331.

Brunskill, E., and Russell, S. 2010. RAPID: A reachable anytime planner for imprecisely-sensed domains. In *UAI*.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2008. Sequential monte carlo in reachability heuristics for probabilistic planning. *Artif. Intell.* 172:685–715.

Cassandra, A. R.; Kaelbling, L. P.; and Kurien, J. A. 1996. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *IROS*, 963–972.

Doshi, F., and Roy, N. 2008. The permutable POMDP: Fast solutions to POMDPs for preference elicitation. In *AAMAS*.

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

Hoffmann, J., and Brafman, R. I. 2006. Conformant planning via heuristic forward search: a new approach. *Artif. Intell.* 170:507–541.

King, D. E. 2009. Dlib-ml: A machine learning toolkit. *J. Mach. Learn. Res. (JMLR)* 10:1755–1758.

Kraft, D.; Başeski, E.; Popović, M.; Batog, A. M.; Kjær-Nielsen, A.; Krüger, N.; Petrick, R.; Geib, C.; Pugeault, N.; Steedman, M.; Asfour, T.; Dillmann, R.; Kalkan, S.; Wörgötter, F.; Hommel, B.; Detry, R.; and Piater, J. 2008. Exploration and planning in a three-level cognitive architecture. In *CogSys*.

Kurniawati, H.; Du, Y.; Hsu, D.; and Lee, W. S. 2010. Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research*.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Intell. Res. (JAIR)* 35:623–675.

Rintanen, J. 2001. Complexity of probabilistic planning under average rewards. In *IJCAI*, 503–508.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for POMDPs. *J. Artif. Int. Res. (JAIR)* 32:663–704.

Shani, G.; Poupart, P.; Brafman, R.; and Shimony, S. E. 2008. Efficient add operations for point-based algorithms. In *ICAPS*, 330–337.

Shani, G.; Brafman, R. I.; and Shimony, S. E. 2008. Prioritizing point-based pomdp solvers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* 38(6):1592–1605.

Silver, D., and Veness, J. 2010. Monte-carlo planning in large POMDPs. In *NIPS*.

Sridharan, M.; Wyatt, J.; and Dearden, R. 2010. Planning to see: Hierarchical POMDPs for planning visual actions on a robot. *Artif. Intell.* 174(11):704–725.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for human-robot teaming in open worlds. *ACM Trans. Intell. Syst. Technol.* 1:14:1–14:24.

Wyatt, J. L.; Aydemir, A.; Brenner, M.; Hanheide, M.; Hawes, N.; Jensfelt, P.; Kristan, M.; Kruijff, G.-J. M.; Lison, P.; Pronobis, A.; Sjöö, K.; Skočaj, D.; Vrečko, A.; Zender, H.; and Zillich, M. 2010. Self-understanding and self-extension: A systems and representational approach. *IEEE Transactions on Autonomous Mental Development* 2(4):282 – 303.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*, 1010–1016.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *ICAPS*, 352–.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *J. Artif. Intell. Res. (JAIR)* 24:851–887.

Plan-based Object Search and Exploration Using Semantic Spatial Knowledge in the Real World

Alper Aydemir* Moritz Göbelbecker† Andrzej Pronobis* Kristoffer Sjöö * Patric Jensfelt*
*Centre for Autonomous Systems, Royal Institute of Technology, Stockholm, Sweden
†Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Germany

***Abstract*— In this paper we present a principled planner based approach to the active visual object search problem in unknown environments. We make use of a hierarchical planner that combines the strength of decision theory and heuristics. Furthermore, our object search approach leverages on the conceptual spatial knowledge in the form of object cooccurences and semantic place categorisation. A hierarchical model for representing object locations is presented with which the planner is able to perform indirect search. Finally we present real world experiments to show the feasibility of the approach.**

***Index Terms*— Active Sensing, Object Search, Semantic Mapping, Planning**

I. Introduction

Objects play an important role when building a semantic representation and an understanding of the function of space [14]. Key tasks for service robots, such as fetch-and-carry, require a robot to successfully find objects. It is evident that such a system cannot rely on the assumption that all object relevant to the current task are already present in its sensory range. It has to actively change its sensor parameters to bring the target object in its field of view. We call this problem *active visual search* (AVS).

Although researchers began working on the problem of visually finding a relatively small sized object in a large environment as early as 1976 at SRI [4], the issue is often overlooked in the field. A common stated reason for this is that the underlying problems such as reliable object recognition and mapping are posing hard enough challenges. However as the field furthers in its aim to build robots acting in realistic environments, this assumption need to be relaxed. The main contribution of this work a method to relinquish the above mentioned assumption.

A. Problem Statement

We define the active visual object search problem as an agent localizing an object in a known or unknown 3D environment by executing a series of actions with the lowest total cost. The cost function is often defined as the time it takes to complete the task or distance traveled.

Let the environment be Ω and Ψ being the search space with $\Psi \subseteq \Omega$. Also let $P_o(\Psi)$ be the probability distribution for the position of the center of the target object o defined as a function over Ψ. The agent can execute a sensing action s in the reachable space of Ψ. In the case of a camera as the sensor, s is characterised by the camera position, (x_c, y_c, z_c), pan-tilt angles (p, t), focal length f and a recognition algorithm a; $s = s(x_c, y_c, z_c, p, t, f, a)$. The part of Ψ covered by s is called a *viewcone*. In practice, a has an effective region in which reliable recognition or detection is achieved. For the i^{th} viewcone we call this region V_i.

Depending on the agent's level of a priori knowledge of Ψ and $P(\Psi)$ there are three extreme cases of the AVS problem. If both Ψ and $P(\Psi)$ is fully known then the problem is that of sensor placement and coverage maximization given limited field of view and cost constraints.

If both Ψ and $P(\Psi)$ is unknown then the agent has an additional *explore* action as well. An exhaustive exploration strategy is not always optimal, i.e. the agent needs to select which parts of the environment to explore first depending on the target object's properties. Furthermore the agent needs to trade-off between executing a sensing action and exploration at any given point. That is, should the robot search for the object o in the partially known Ψ or explore further. This is classically known as the exploration vs. exploitation problem.

When $P(\Psi)$ is unknown (i.e. uniformly distributed) but Ψ is known (i.e. acquired a priori), the agent needs to gather information about the environment similar to the above case. However in this case, the exploration is for learning about the target object specific characteristics of the environment. Knowing Ψ also means that the robot can reason whether or not to execute a costly search action at the current position, or move to another more promising region of space. The rare case where $P(\Psi)$ is fully known but Ψ is unknown is not practically interesting to the scope of this paper.

So far, we have examined the case where the target object is an instance. The implication of this is that $P(\Psi) + P(\Omega \setminus \Psi) = 1$, therefore observing V_i has an effect on $P(\Psi \setminus V_i)$. However this is not necessarily true if instead the agent is searching for any member of an object category and the number of them is not known in advance. Therefore knowing whether the target object is a unique instance or a member of an object category is an important factor in search behavior.

Recently there's an increasing amount of work on acquiring *semantic maps*. Semantic maps have parts of the environment labeled representing various high level concepts and functions of space. Exploring and building a semantic map while performing AVS contributes to the estimation of $P(\Psi)$. The semantic map provides information that can be exploited by leveraging on common-sense conceptual knowledge about

This work was supported by the SSF through its Centre for Autonomous Systems (CAS), and by the EU FP7 project CogX.

indoor environments. This knowledge describes, for example, how likely it is that plates are found in kitchens, that a mouse and a computer keyboard occur in the same scene and that corridors typically connect multiple rooms. Such information offers valuable information in limiting the search space. The sources for those can be from online common-sense databases or world wide web among others. Acknowledging the need to limit the search space and integrate various cues to guide the search, [4] proposed *indirect search*. Indirect search as a search strategy is a simple and powerful idea: it's to find another object first and then use it to facilitate finding the target object, e.g. finding a table first while looking for a landline phone. Tsotsos [13] approached the problem by analyzing the complexity of the AVS problem and showed that it is NP-hard. Therefore we must adhere to a heuristics based solution. Ye [15] formulated the problem in probabilistic framework.

In this work we consider the case where Ψ and $P(\Psi)$ are both unknown. However, the robot is given probabilistic default knowledge about the relation betweeen objects and the occurences of objects in difference room category following our previous work [1, 6].

B. Contributions

The contributions of this work are four fold. First we provide the domain adaptation of a hierarchical planner to address the AVS problem. Second we show how to combine semantic cues to guide the object search process in a more complex and larger environment then found in previous work. Third, we start with an unknown map of the environment and provide an exploration strategy which takes into account the object search task. Four, we present real world experiments searching for multiple objects in a large office environment, and show how the planner adapts the search behavior depending of the current conditions.

C. Outline

The outline of this paper is as follows. First we present how the AVS problem can be formulated in a principled way using a planning approach (Section II). Section III provides the motivation for and structure of various aspects of our spatial representation. Finally we showcase the feasibility of our approach in real worl experiments (Section IV) and .

II. Planning

For a problem like AVS which entails probabilistic action outcomes and world state, the robot needs to employ a planner to generate flexible and intelligent search behavior that trade off exploitation versus exploration. In order to guarantee optimality a POMDP planner can be used in, i.e. a decision theoretic planner that can accurately trade different costs against each other and generate the optimal policy. However, this is only tractable when a complex problem like AVS is applied to very small environments. Another type of planner are the classical AI planners which requires perfect knowledge about the environment. This is not the case since both Ψ and $P(\Psi)$ are unknown.

A variation of the classical planners are the so called continual planners that interleave planning and plan monitoring in order to deal with uncertain or dynamic environments[3]. The basic idea behind the approach is to create an plan that *might* reach the goal and to start executing that plan. This initial plan takes into account success probabilities and action costs however it is optimistic in nature. A monitoring component keeps track of the execution outcome and notified the planner in the event of the current plan becoming invalid (either because the preconditions of an action are no longer satisfied or the plan does not reach the goal anymore). In this case, a new plan is created with the updated current state as the initial state and execution starts again. This will continue until either the monitoring component detects that the goal has been reached or no plan can be found anymore.

In this paper we will make use of a so called switching panner. It combines two different domain independent planners for different parts of the task: A *classical continual planner* to decide the overall strategy of the search (for which objects to search in which location) and a *decision theoretic planner* to schedule the low level observation actions using a probabilistic sensing model. Both planners use the same planning model and are tightly integrated.

We first give a brief description of the switching planner. We focus on the use of the planner in this paper and instead refer the reader to [5] for a more details description. We will also present the domain modeling for the planner, and give further details on various aspects of knowledge that planner makes use of.

A. Switching Planner

1) Continual Planner (CP): We build our planning framework on an extended SAS^+[2] formalism. As a base for the continual planner, we use Fast Downward[7]. Because our knowledge of the world and the effects of our actions are uncertain we associate a *success probability* $\mathrm{p}(a)$ with every action a. In contrast to more expressive models like MDPs or even POMDPs, actions don't have multiple possible outcomes, they just can succeed with probability $\mathrm{p}(a)$ or fail with probability of $1 - \mathrm{p}(a)$.

The goal of the planner is then to find a plan π that reaches the goal with a low cost. In classical planning the cost function is usually either the number of actions in a plan or the sum of all action's costs. Here we chose a function that resembles the expected reward adjusted to our restricted planning model. With $\mathrm{p}(\pi) = \prod_{a \in \pi} \mathrm{p}(a)$ as the plans total success probability and $\mathrm{cost}(\pi) = \sum_{a \in \pi} \mathrm{cost}(a)$ as the total costs, we get for the optimal plan π^*:

$$\pi^* = \operatorname*{argmin}_{\pi} \mathrm{cost}(\pi) + R(1 - \mathrm{p}(\pi))$$

where a is an action and the constant R is the reward the planner is given for achieving the goal. For small values of R the planner will prefer cheaper but more unlikely plans, for larger values more expensive plans will be considered.

Assumptions The defining feature of an exploration problem is that the world's state is uncertain. Some planning frameworks such as MDPs allow the specification of an initial state distribution. We choose not to do this for two different reasons: a) having state distributions would be a too strong departure from the classical planning model and b) the typical exploration problems we deal with have too many possible

states to express explicitly. We therefore use an approach we call *assumptive actions* that allow the planner to construct parts of the initial state on the fly, and which allows us to map the spatial concepts to the planning problem in an easy way.

2) Decision Theoretic (DT) Planner: When the continual planner reaches a sensing action (e.g. *search location1 for a object2*), we create a POMDP that only contains the parts of the state that are relevant for that subproblem with. This planner can only use MOVE and PROCESSVIEWCONE actions explained in Section II-B.2. The DT planner operates in a closed-loop manner, sending actions to be executed and receiving observations from the system. Once the DT planner either confirms or rejects a hypothesis, it returns control back to the continual planner, which treats the outcome of the DT session like the outcome of any other action.

B. Domain Modeling

We need to discretize the involved spaces (object location, spatial model and actions) to make a planner approach applicable to the AVS problem. Most methods make use of discretizations as a way to handle the NP-hard nature of the problem.

1) Representing space: For the purposes of obstacle avoidance, navigation and sensing action calculation, Ψ is represented as a 3D metric map. Ψ discretised into i volumetric cells so that $\Psi = c_0...c_i$. Each cell represents the occupancy with the attributes OCCUPIED, FREE or UNKOWN as well as the probability of target object's center being in that cell.

However, further abstraction is needed to achieve reliable and fast plan calculation as the number of cells can be high. For this purpose we employ a topological representation of Ψ called *place map*, see Fig 1(a). In the place map, the world is represented by a finite number of basic spatial entities called *places* created at equal intervals as the robot moves. Places are connected using paths which are discovered by traversing the space between places. Together, places and paths represent the topology of the environment. This abstraction is also useful for a planner since metric space would result in a largely intractable planning state space.

The places in the place map are futher segmented into rooms. In the case of indoor environments, rooms are usually separated by doors or other narrow openings. Thus, we propose to use a door detector and perform reasoning about the segmentation of space into rooms based on the doorway hypotheses. We use a template-based door detection algorithm which matches a door template to each acquired laser scan. This creates door hypotheses which are further verified by the robot passing through a narrow opening.

In addition, unexplored space is represented in the place map using hypothetical places called *placeholders* defined in the boundary between free and unknown space in the metric map.

We represent object locations not in metric coordinates but in relation to other known objects or rooms to achieve further abstraction. The search space is considered to be divided into *locations* $\mathcal{L}$. A location is either a *room* $\mathcal{R}$ or a *related space*. Related spaces are regions connected with a *landmark object* o, either *in* or *on* the landmark (see [1] for more details). The related space "in" o is termed $\mathcal{I}_o$ and the space "on" o $\mathcal{O}_o$.

2) Modeling actions: The planner has access to three physical actions: MOVE can be used to move to a place or placeholder, CREATEVIEWCONES creates sensing actions for an *object label* in *relation* to a specified *location*, PROCESSVIEWCONE executes a sensing action. Finally, the virtual SEARCHFOROBJECT action that triggers the decision theoretic planner.

3) Virtual objects: There are two aspects of exploration in the planning task: we're searching for an (at that moment) unknown object, which may include the search for support objects as an intermediate step. But the planner may also need to consider the utility of exploring its environment in order to find new rooms in which finding the goal object is more likely.

Because the planners we use employ the closed world assumption, adding new objects as part of the plan is impossible. We therefore add a set of *virtual objects* to the planning problem that can be instantiated by the planner as required by the plan. This approach will fail for plans that require finding more objects than pre-allocated, but this is not a problem in practice. The monitoring component tries to match new (real) objects to virtual objects that occur in the plan. This allows us to deliver the correct observations to the DT planner and avoid unnecessary replanning.

4) Probabilitic spatial knowledge: The planner makes use of the following probabilistic spatial knowledge in order to generate sensible plans:

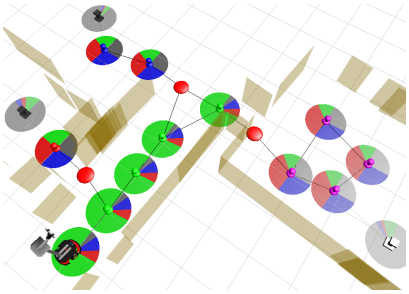
- $P_{category}(room_i)$ defines the distribution over room categories that the robot has a model for, for a given room integrated over places that belongs to $room_i$. The planner uses this information to decide whether to plan for a SEARCHFOROBJECT action or explore the remaining placeholders.
- $P_{category}(placeholder_i)$ represents the probability distribution of a placeholder turning into a new room of a certain category upon exploration. Using this distribution, the planner can choose to explore a placeholder instead of another, or plan to launch search altogether.
- $P(ObjectAt\mathcal{L})$ gives the probability of an object o being at location $\mathcal{L}$.

More details about calculation of these probabilities are further explained in Section III.

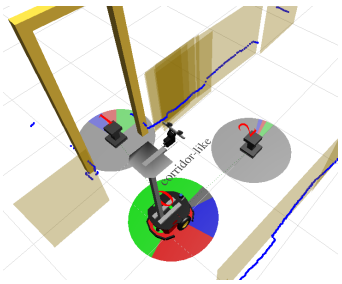
III. SPATIAL REPRESENTATION

5) Conceptual Map: All higher level inference is performed in the so called conceptual map which is represented by a graphical model. It integrates the conceptual knowledge (food items are typically found in kitchens) with instance knowledge (the rice package is in room4). We model this in a *chain graph* [8], whose structure is adapted online according to the state of underlying topological map. Chain graphs provide a natural generalisation of directed (Bayesian Networks) and undirected (Markov Random Fields) graphical models, allowing us to model both "directed" causal as well as "undirected" symmetric or associative relations.

The structure of the chain graph model is presented in Fig. 2. Each discrete place is represented by a set of random variables connected to variables representing semantic category of a room. Moreover, the room category variables are connected



(a)



(b)

Fig. 1. (a) A place map with several places and 3 detected doors shown as red. (b) Shows two placeholders with different probabilities for turning into new rooms: one of them is behind a door hypothesis therefore having a higher probability of leading into a new room. Colors on circular discs indicates the probability of room categories as in a pie chart: i.e. the bigger the color is the higher the probability. Here green is *corridor*, red is *kitchen* and blue is *office*.

by undirected links to one another according to the topology of the environment. The potential functions $\phi_{rc}(\cdot,\cdot)$ represent the type knowledge about the connectivity of rooms of certain semantic categories.

To compute $P_{category}(room_i)$ each place is described by a set of properties such as size, shape and appearance of space. These are are based on sensory information as proposed in [12]. We extend this work by also including presence of a certain number of instances of objects as observed from each place as a properties (due to space limitations we refer to [11] for more details). This way object presence or absence in a room also affects affects room category. The property variables can be connected to observations of features extracted directly from the sensory input. Finally, the functions $p_s(\cdot|\cdot)$, $p_a(\cdot|\cdot)$, $p_{o_i}(\cdot|\cdot)$ utilise the common sense knowledge about object, spatial property and room category co-occurrence to allow for reasoning about other properties and room categories.

For planning, the chain graph is the sole source of belief-state information. In the chain graph, belief updates are event-driven. For example, if an appearance property, or object detection, alters the probability of a relation, inference proceeds to propagate the consequences throughout the graph. In our work, the underlying inference is approximate, and uses the fast Loopy Belief Propagation [9] procedure.

A. Object existence probabilities

To compute the $P(ObjectAt\mathcal{L})$ value used in active visual search in this paper, objects are considered to be occurring:

1) independently in different locations $\mathcal{L}$
2) independently of other objects in the same location

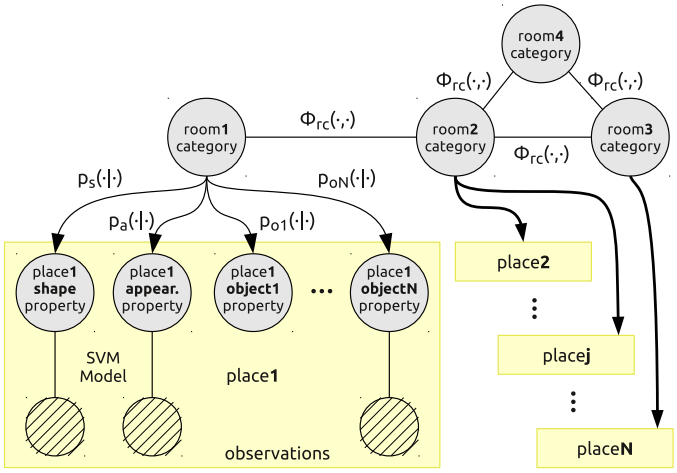


Fig. 2. Schematic image of chain graph

3) as Poisson processes over cells $c_0 ... c_i$ per location $\mathcal{L}$

In other words, each location has the possibility of containing, independently of all other locations, a number n_c of objects of a class c with probability

$$P(n_c = k) = \frac{\lambda^{\mathrm{k}}_{\mathcal{L},c} e^{-\lambda_{\mathcal{L},c}}}{\mathrm{k!}} \tag{1}$$

where $\lambda_{\mathcal{L},c}$ is the expected number of objects of class c in the location $\mathcal{L}$. The probability of *at least one* object in a location is

$$P(n_c > 0) = 1 - P(n_c = 0) = 1 - e^{-\lambda_{i,c}} \tag{2}$$

Because of the independence assumptions, the λ values for a location and all its subordinate locations can simply be added together to obtain the distribution of the number of objects of that class occurring in that whole hierarchy.

1) Exploration: In addition to making inferences about explored space, the conceptual map can provide predictions about unexplored space. To this end, we extend the graph by including the existence of placeholders. For each placeholder a set of probabilities is generated that the placeholder will lead to a room of a certain category.

This process is repeated for each placeholder and consists of three steps. In the first step, a set of hypotheses about the structure of the unexplored space is generated. In case of our implementation, we evaluate 6 hypotheses: (1) placeholder does not lead to new places, (2) placeholder leads to new places which do not lead to a new room, (3) placeholder leads to places that lead to a single new room (4) placeholder leads to places that lead a room which is further connected to another room, (5) placeholder leads to a single new room directly, and (6) placeholder leads to a new room directly which leads to another room. In the second step, the hypothesized rooms are added to the chain graph just like regular rooms and inference about their categories is performed. Then, the probability of any of the hypothesized rooms being of a certain category is obtained. Finally, this probability is multiplied by the likelihood of occurrence of each of the hypothesized worlds estimated based on the amount of open space behind the placeholder and the proximity of gateways. A simple example is shown in Fig. 1(b)

IV. EXPERIMENTS

Experiments were carried out on a Pioneer III wheeled robot, equipped with a Hokuyo URG laser scanner, and a camera mounted at 1.4 m above the floor. Experiments took place in 12x8 m environment with 3 different rooms, *kitchen*, *office1*, *office2* connected by a corridor. The robot had models of all objects it searches for before each search run. 3 different objects (*cerealbox*, *stapler* and *whiteboardmarkers*) were used during experiments. The BLORT framework was used to detect objects [10].

To highlight the flexibility of the planning framework evaluated the system with 6 different starting positions and tasked with finding different objects in an unknown environment. We refer the reader to `http://www.csc.kth.se/~aydemir/avs.html` for videos. Each sub-figure in Fig. 3 shows the trajectory of the robot. The color coded trajectory indicates the room category as perceived by the robot: red is kitchen, green is corridor and blue is office. The two green arrows denote the current position and the start position of the robot.

In the following we give a brief explanation for what happened in the different runs.

- Fig. 3(a) Starts: *corridor*, Target: *cerealbox* in *kitchen*
  The robot starts by exploring the *corridor*. The robot finds a doorway on its left and the placeholder behind it has a higher probability of yielding into a kitchen and the robot enters *office1*. As the robot acquires new observations the CP's kitchen assumption is violated. The robot returns to exploring the corridor until it finds the kitchen door. Here the CP's assumptions are validated and the robot searches this room. The DT planner plans a strategy of first finding a table and then the target object on it. After finding a table, the robot generates view cones for the $\mathcal{O}_{table,cornflakes}$ location. The cerealbox object is found.
- Fig. 3(b) Starts: *office2*, Target: *cerealbox* in *kitchen*
  Unsatisfied with the current room's category, the CP commits to the assumption that exploring placeholders in the corridor will result in a room with category kitchen. The rest proceeds as in Fig. 3(a).
- Fig. 3(c) Starts: *corridor* Target: *cerealbox* in *kitchen*
  The robot explores until it finds *office2*. Upon entry the robot categorises *office2* as kitchen but after further exploration, *office2* is categorised correctly. The robot switches back to exploration and since the kitchen door is closed, it passes kitchen and finds *office1*. Not satisfied with *office1*, the robot gives up since all possible plans success probability are smaller than a given threshold value.
- Fig. 3(d) Starts: *office1* Target:*stapler* in *office2*
  After failing to find the object in *office1* the robot notices the open door, but finding that it is kitchen-like decides not to search the kitchen room. This time the *stapler* object is found in *office2*
- Fig. 3(e) Starts: *kitchen* Target: *cerealbox* in *kitchen*
  As before it tries locating a table, but in this case all table objects have been eliminated beforehand; failing to detect a table the robot switches to looking for a counter. Finding no counter either, it finally goes out in the corridor to look for another kitchen and upon failing that, gives up.
- Fig. 3(f) Starts: *corridor* Target: *whiteboardmarker* in *office1*
  The robot is started in the corridor and driven to the kitchen by a joystick; thus in this case the environment is largely explored already when the planner is activated and asked to find a *whiteboardmarker* object. The part of the corridor leading to *office2* has been blocked. The robot immediately finds its way to *office1* and launches a search which results in a successful detection of the target object.

In the following, we describe the planning decisions in more detail for a run similar to the one described in Fig. 3(a), with the main difference being that the cereals could not be found in the end due to a false negative detection.

The first plan, with the robot starting out in the middle of the corridor, looks as follows:

ASSUME-LEADS-TO-ROOM place1 kitchen
ASSUME-OBJECT-EXISTS table IN new-room1 kitchen
ASSUME-OBJECT-EXISTS cerealbox ON new-object1 table kitchen
MOVE place1
CREATEVIEWCONES table IN new-room1
SEARCHFOROBJECT table IN new-room1 new-object1
CREATEVIEWCONES cerealbox ON new-object1
SEARCHFOROBJECT cerealbox ON new-object1 new-object2
REPORTPOSITION new-object2

Here we see several virtual objects being introduced: The first action assumes that *place1* leads to a new room *new-room1* with category kitchen. The next two assumptions hypothesize that a table exists in the room and that cornflakes exist on that table. The rest of the plan is rather straightforward: create view cones and search for the table, then create view cones and search for the cereal box.

Execution of that plan leads to frequent replanning, as the first assumption is usually too optimistic: most placeholders do not directly lead to a new room, but require a bit more exploration.

After following the corridor, the robot does find the office, and returns to the corridor to explore into the other direction. It finally finds a room which has a high likelihood of being a kitchen.

ASSUME-CATEGORY room3 kitchen
ASSUME-OBJECT-EXISTS table IN room3 kitchen
ASSUME-OBJECT-EXISTS cerealbox ON new-object1 table kitchen
MOVE place17
MOVE place18
MOVE place16
CREATEVIEWCONES table IN room3
SEARCHFOROBJECT table IN room3 new-object1
CREATEVIEWCONES cerealbox ON new-object1
SEARCHFOROBJECT cerealbox ON new-object1 new-object2

The new plan looks similar to the first one, except that we do not assume the existence of a new room but the

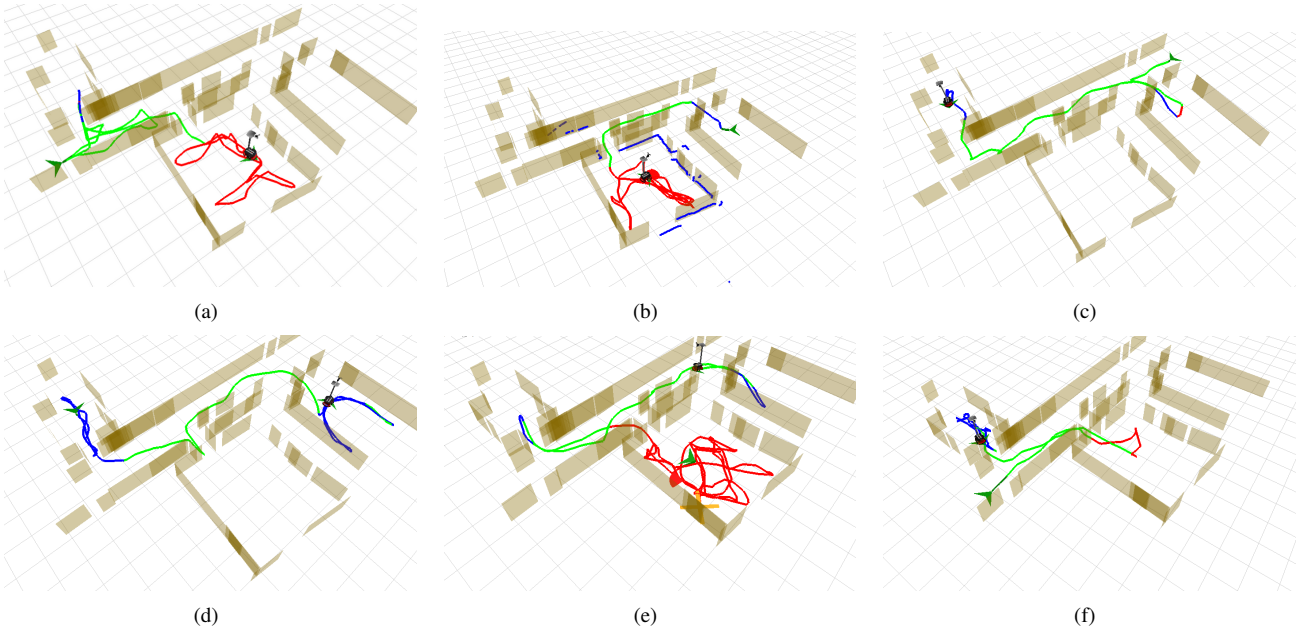


Fig. 3. Trajectories taken by the robot in multiple experiments

category of an existing one. Also, the robot cannot start creating view cones immediately because a precondition of the CREATEVIEWCONES action is that the room must be fully explored, which involves exploring all remaining placeholders in the room.

After view cones are created, the decision theoretic planner is invoked. We used a relatively simple sensing model, with a false negative probability of 0.2 and a false positive probability of 0.05 – these are educated guesses, though. The DT planner starts moving around and processing view cones until it eventually detects a table and returns to the continual planner. At this point the probability of the room being a kitchen is so high, that it considered to be certain by the planner. With lots of the initial uncertainty removed, the final plan is straightforward:

ASSUME-OBJECT-EXISTS cerealbox ON object1 table kitchen
CREATEVIEWCONES cerealbox ON object1
SEARCHFOROBJECT cerealbox ON object1 new-object2
REPORTPOSITION new-object2

During the run, the continual planner created 14 plans in total, taking 0.2 – 0.5 seconds per plan on average. The DT planner was called twice, and took about 0.5 – 2 seconds per action it executed.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a planning approach to the active object search. We made use of a switching planner, combing a classical continual planner with a decision theoretic planner. We provide a model for the planning domain appropriate for the planner and show by experimental results that the system is able to search for objects in a real world office environment making use of both low level sensor perceipt and high level conceptual and semantic information. Future work includes incorporating 3D shape cues to guide the search and a specialized planner for the AVS problem.

REFERENCES

[1] Alper Aydemir, Kristoffer Sjöö, John Folkesson, and Patric Jensfelt. Search in the real world: Active visual object search based on spatial relations. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[2] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Comp. Intell.*, 11(4):625–655, 1995.

[3] Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems*, 19(3):297–331, 2009.

[4] Thomas D. Garvey. Perceptual strategies for purposive vision. Technical Report 117, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Sep 1976.

[5] Moritz Göbelbecker, Charles Gretton, and Richard Dearden. A switching planner for combined task and observation planning. In *Twenty-Fifth Conference on Artificial Intelligence (AAAI-11)*, August 2011.

[6] Marc Hanheide, Charles Gretton, Richard W Dearden, Nick A Hawes, Jeremy L Wyatt, Andrzej Pronobis, Alper Aydemir, Moritz Göbelbecker, and Hendrik Zender. Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2011.

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

[8] S. L. Lauritzen and T. S. Richardson. Chain graph models and their causal interpretations. *J. Roy. Statistical Society, Series B*, 64(3):321–348, 2002.

[9] J. M. Mooij. libDAI: A free and open source C++ library for discrete approximate inference in graphical models. *J. Mach. Learn. Res.*, 11:2169–2173, August 2010.

[10] T. Mörwald, J. Prankl, A. Richtsfeld, M. Zillich, and M. Vincze. BLORT - The blocks world robotic vision toolbox. In *Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation at ICRA 2010*, 2010.

[11] Andrzej Pronobis and Patric Jensfelt. Hierarchical multi-modal place categorization. In *submitted to ECMR'11*, 2011.

[12] Andrzej Pronobis, Oscar M. Mozos, Barbara Caputo, and Patric Jensfelt. Multi-modal semantic place classification. *The International Journal of Robotics Research (IJRR), Special Issue on Robotic Vision*, 29(2-3):298–320, February 2010.

[13] J. K. Tsotsos. On the relative complexity of active vs. passive visual search. *International Journal of Computer Vision*, 7(2):127–141, 1992.

[14] S. Vasudevan and R. Siegwart. Bayesian space conceptualization and place classification for semantic maps in mobile robotics. *Robot. Auton. Syst.*, 56:522–537, June 2008.

[15] Yiming Ye and John K. Tsotsos. Sensor planning for 3d object search. *Comput. Vis. Image Underst.*, 73(2):145–168, 1999.

A Polynomial All Outcome Determinization for Probabilistic Planning

Thomas Keller and **Patrick Eyerich**
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
{tkeller,eyerich}@informatik.uni-freiburg.de

Abstract

Most predominant approaches in probabilistic planning utilize techniques from the more thoroughly investigated field of classical planning by determinizing the problem at hand. In this paper, we present a method to map probabilistic operators to an equivalent set of probabilistic operators in a novel normal form, requiring polynomial time and space. From this, we directly derive a determinization which can be used for, e. g., replanning strategies incorporating a classical planning system. Unlike previously described all outcome determinizations, the number of deterministic operators is not exponentially but polynomially bounded in the number of parallel probabilistic effects, enabling the use of more sophisticated determinization-based techniques in the future.

Introduction

Most probabilistic planning systems that have successfully participated in the International Probabilistic Planning Competition, e.g. FF-Replan (Yoon, Fern, and Givan 2007), FPG (Buffet and Aberdeen 2007) or RFF-(BG/PG) (Teichteil-Königsbuch, Infantes, and Kuter 2008), invoke a procedure called *determinization* of the probabilistic planning task.

Two classes of determinization strategies have been described: *Single outcome* determinizations choose one possible outcome for each probabilistic operator, accepting that solvable tasks might become unsolvable in the determinization, while *all outcome* determinizations preserve solvability by generating all potential outcomes. The only all outcome determinization used in practice generates one operator for each potential outcome, possibly leading to exponentially many operators in the determinization (Rintanen 2003). While Rintanen briefly mentions a solution to this problem, it has never been described in detail, a shortfall made up for with this paper by introducing the *forked normal form* (FNF) and a polynomial determinization based on FNF where operators are *split* into several deterministic ones that are applied *sequentially* to simulate one outcome.

After presenting our formal framework, we show how to split operators while preserving task equivalence, and how to generate a set of operators in FNF with size polynomial in the number of probabilistic effects. We continue by discussing the derived determinization, and finally point out the benefit with a short experiment.

Probabilistic Planning

Definition 1. A (fully-observable) probabilistic *planning task* T is a 4-tuple $\langle V, s_0, S_\star, O\rangle$, where V is a set of *finite-domain state variables* v with *domain* D_v; s_0, the *initial state*, is a valuation over V; $S_\star$, the *goal*, is a *condition* over V, a logical formula over atoms of the form $v{=}d$ ($v \in V, d \in D_v$); and O is a set of *operators*, 3-tuples $o = \langle \text{pre}(o), \text{eff}(o), c(o)\rangle$ with *precondition* $\text{pre}(o)$, a condition over V; *effect* $\text{eff}(o)$; and *cost* $c(o) \in \mathbb{N}_0$. Effects of operators $o \in O$ are recursively defined by:

- $\top$ is the *empty* effect.
- $v{\leftarrow}x$ with $v \in V, x \in D_v$ is an *atomic* effect.
- $e_1 \wedge e_2$ is an *conjunctive* effect if e_1 and e_2 are effects.
- $c \rhd e$ is a *conditional* effect if c is a condition and e is an effect.
- $p_1 e_1 | \ldots | p_n e_n$ is a *probabilistic* effect if $e_1, \ldots, e_n$ are effects, $0 < p_1, \ldots, p_n \leq 1$, and $\sum_{i=1}^{n} p_i = 1$.

An effect e without probabilistic effects is *deterministic*. An effect e is in *unary conditionality normal form* (UCNF) if e' is atomic for all $c \rhd e'$ in e; it is in *unary non-determinism normal form* (1ND) if e is of the form $p_1 e_1 | \ldots | p_n e_n$ with deterministic $e_1, \ldots, e_n$ (Rintanen 2003), the normal form the most common all outcome determinization is based on. We prohibit effects that assign multiple values to the same v as semantics might become ambiguous. To measure plan quality we use costs, but our results also hold for planning problems based on rewards (Condon 1992). In the following, we refer to valuations s over V as the *states* of T.

The *successor set* $\text{suc}(s, o)$ of applying an operator o in a state s is a set of state/probability pairs defined as in Rintanen's definition of operator application (2003).

Definition 2. A probabilistic *transition system* $\mathcal{T}$ is a 5-tuple $\langle S, O, \Delta, s_0, G\rangle$ where S is a set of *states*; O is a set of operators; for each $o \in O$ there is a $\delta_o \in \Delta$ that is a partial function mapping states to probability distributions over S; $s_0 \in S$ is the initial state; and $G \subseteq S$ is the set of goal states. States with more than one applicable operator are called *decision states* $\text{dec}(\mathcal{T})$, and $\text{suc}_{\text{dec}}(s) = (\text{suc}(s) \cap \text{dec}(\mathcal{T})) \cup \{\text{suc}_{\text{dec}}(s') | s' \in \text{suc}(s) \setminus \text{dec}(\mathcal{T})\} \subseteq \text{dec}(\mathcal{T})$ is the recursively defined set of *decision successors*.

A planning task $T = \langle V, s_0, S_\star, O\rangle$ induces a transition system $\mathcal{T} = \langle S, O, \Delta, s_0, G\rangle$, where S is the set of states of

V; for each $o \in O$ there is a $\delta_o \in \Delta$ defined for all states s where $s \models \text{pre}(o)$ mapping s to a probability distribution induced by $\text{suc}(s, o)$; and $G = \{s \in S | s \models S_\star\}$. A *policy* π is a mapping from states to operators. We refer to the set of all policies in a transition system $\mathcal{T}$ as $\Pi(\mathcal{T})$.

Since we compare planning tasks where the application of an operator in one task is matched to the application of a sequence of operators in the other (i.e. there is no one-to-one correspondence between transitions), we need our definition of equivalence to be more general than usual. We achieve this by exploiting the fact that a policy π naturally induces a probability distribution over costs $\alpha_\pi(s)$ for each state s:

Definition 3. Two transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ are *equivalent* ($\mathcal{T}_1 \equiv \mathcal{T}_2$) if there are two bijective mappings $\sigma : \text{dec}(\mathcal{T}_1) \to \text{dec}(\mathcal{T}_2)$ and $\theta : \Pi(\mathcal{T}_1) \to \Pi(\mathcal{T}_2)$ s.t. for all $\pi \in \Pi(\mathcal{T}_1)$ and $s \in \text{dec}(T_1)$ we have $\alpha_\pi(s) = \alpha_{\theta(\pi)}(\sigma(s))$.

Operator Splitting

A determinization based on 1ND leads to exponentially many operators if conjunctions of probabilistic effects are present. The main idea of our determinization is to split these, creating one operator per conjunctive element. In this section, we show how to split deterministic operators and enforce their sequential application, and how to prevent non-equivalent transformation due to conditional effects.

Operator splitting leads to *intermediate* states in the transformed task where only parts of the original operator's effects have been applied. To control correct operator application we introduce a variable v_{aux} and extend all preconditions of operators such that $v_{\text{aux}}{=}0$ is requested. In intermediate states we require different, unique values for v_{aux} and thereby enforce sequential application in an equivalence-preserving way for effects not containing conditional effects. Note that intermediate states are no decision states due to the uniqueness of the values for v_{aux}, a property that also holds in the rest of this paper.

Example 1. An operator $o = \langle v_0{=}2, v_1{\leftarrow}4 \wedge v_2{\leftarrow}0, c(o)\rangle$ can be split into $o'_1 = \langle v_0{=}2 \wedge v_{\text{aux}}{=}0, v_1{\leftarrow}4 \wedge v_{\text{aux}}{\leftarrow}1, c(o)\rangle$ and $o'_2 = \langle v_{\text{aux}}{=}1, v_2{\leftarrow}0 \wedge v_{\text{aux}}{\leftarrow}0, 0\rangle$.

This transformation can lead to a non-equivalent task if a variable v occurs both in an effect condition c and an atomic effect, and if that operator is split s.t. the assignment of v is applied before c is checked. To avoid this, we show how to transform a planning task $T = \langle V, s_0, S_\star, O\rangle$ into an equivalent planning task $T^\triangleright = \langle V \cup V^\triangleright \cup \{v_{\text{aux}}\}, \sigma(s_0), S_\star \cup \{v_{\text{aux}}{=}0\}, O^\triangleright \cup O^\star\rangle$, where no variable occurs both in effect conditions and atomic effects.

Theorem 1. *All planning tasks can be normalized in polynomial time and space to equivalent planning tasks where no variable occurs both in effect conditions and atomic effects.*

Proof: Let $\# : O \to \{1, \ldots, |O|\} \subset \mathbb{N}$ be bijective, $V^\triangleright$ be a set of variables with one variable $v_\triangleright$ for each $v \in V$, and the operators $o_\triangleright = \langle \text{pre}(o_\triangleright), \text{eff}(o_\triangleright), c(o)\rangle$ and $o_\star = \langle \text{pre}(o_\star), \text{eff}(o_\star), 0\rangle$ for each $o \in O$ be s.t.

$$\text{pre}(o_\triangleright) = \text{pre}(o) \wedge v_{\text{aux}}{=}0$$
$$\text{eff}(o_\triangleright) = \bigwedge_{v \in V} v_\triangleright{\leftarrow}\text{val}(v) \wedge v_{\text{aux}}{\leftarrow}\#(o)$$
$$\text{pre}(o_\star) = v_{\text{aux}}{=}\#(o)$$
$$\text{eff}(o_\star) = \text{eff}(o)[V/V^\triangleright] \wedge v_{\text{aux}}{\leftarrow}0$$

where $\text{val}(v) \in D_v$ is the value of v and $\text{eff}(o)[V/V^\triangleright]$ is equal to $\text{eff}(o)$ except that all occurrences of all $v \in V$ in effect conditions are replaced by their corresponding $v_\triangleright \in V^\triangleright$. As polynomial time and space transformation from T to $T^\triangleright$ is obvious we focus on the proof of equivalence with

$$\sigma(s) = s \cup \{(v, 0) | v \in V^\triangleright \cup \{v_{\text{aux}}\}\}, \text{ and}$$
$$\theta(\pi)(\sigma(s)) = o_\triangleright \Leftrightarrow \pi(s) = o$$

in the following. To show that σ is bijective, it is sufficient to show that $\sigma(s_1) \neq \sigma(s_2)$ unless $s_1 = s_2$ for $s_1, s_2 \in \text{dec}(\mathcal{T})$ and that $\text{dec}(\mathcal{T}^\triangleright) \subseteq \{\sigma(s) | s \in \text{dec}(\mathcal{T})\}$. The former follows directly from the definition of σ, while the latter is not as obvious: We prove it by showing that there is no $s_\triangleright \in \text{dec}(\mathcal{T}^\triangleright)$ for which there is no $s \in \text{dec}(\mathcal{T})$ with $\sigma(s) = s_\triangleright$. Following the definition of σ, we must thus show that $s_\triangleright(v) = 0$ for all $v \in V^\triangleright \cup \{v_{\text{aux}}\}$ and all $s_\triangleright \in \text{dec}(\mathcal{T}^\triangleright)$. Let $s_\triangleright \in S^\triangleright$ be a state with $s_\triangleright(v) \neq 0$ for some $v \in V^\triangleright \cup \{v_{\text{aux}}\}$. The definition of $\text{pre}(o_\triangleright)$ directly shows that $s_\triangleright \not\models \text{pre}(o_\triangleright)$ for all $o_\triangleright \in O^\triangleright$. Additionally, if $s_\triangleright \models \text{pre}(o_\star)$ it follows that $s_\triangleright \not\models \text{pre}(o'_\star)$ for $o_\star, o'_\star \in O_\star$ by definition of $\#(o)$ and $\text{pre}(o_\star)$. For this reason there is at most one operator o with $s_\triangleright \models o$, and thus $s_\triangleright \notin \text{dec}(\mathcal{T}^\triangleright)$.

The bijectivity of θ on $\text{dec}(T)$ and $\text{dec}(T^\triangleright)$ is given as $\pi(s_\triangleright) \notin O_\star$ for all $s_\triangleright \in \text{dec}(\mathcal{T}^\triangleright)$ because $s_\triangleright \not\models o_\star$ for all $o_\star \in O_\star$ for all policies π. To verify that $T \equiv T^\triangleright$, we first show that for each policy π in $\mathcal{T}$ it holds that $\alpha_\pi(s) = \alpha_{\theta(\pi)}(\sigma(s))$ for all states $s \in \text{dec}(\mathcal{T})$, which is given iff for all $s \in \text{dec}(\mathcal{T})$ and all sequences of operators leading from s to any $s' \in \text{suc}_{\text{dec}}(s)$, there is a sequence of operators from $\sigma(s)$ to $\sigma(s')$ that induces the same probability and cost (and vice versa). Let $o^1, \ldots, o^k \in O$ be the sequence of operators to reach an s' from s with probability p and cost c. Then the sequence $o^1_\triangleright, o^1_\star, \ldots, o^k_\triangleright, o^k_\star$ leads from $\sigma(s)$ to $\sigma(s')$ with probability p, as the $o_\triangleright$ are deterministic and the transition probabilities from the $o_\star$ and o are equal, and cost c, as $c(o) = c(o_\triangleright) + c(o_\star)$ by definition of the cost functions of $o_\triangleright$ and $o_\star$. As all sequences of operators in $T^\triangleright$ from any $s_\triangleright \in \text{dec}(\mathcal{T})$ to any $s'_\triangleright \in \text{suc}_{\text{dec}}(s_\triangleright)$ must be of the form $o^1_\triangleright, o^1_\star, \ldots, o^k_\triangleright, o^k_\star$ by definition of $\text{eff}(o_\triangleright)$, $\text{pre}(o_\star)$ and $\#(o)$, the opposite direction holds as well. ∎

Forked Normal Form

So far we restricted our analysis to deterministic operators, but the procedure to normalize operators with conditional effects is applicable to operators with probabilistic effects as well, resulting in a set of deterministic operators $O^\triangleright$ and a set of arbitrary operators $O^\star$. Without loss of generality, we also assume that $\text{eff}(o_\star)$ is in UCNF for all $o_\star \in O^\star$ (possible in polynomial time and space: Rintanen 2003), and that all instances of the left-hand side of the equivalence

$$p_1(p'_1 e'_1 | \ldots | p'_j e'_j) | \ldots | p_i e_i \equiv p_1 p'_1 e'_1 | \ldots | p_1 p'_j e'_j | \ldots | p_i e_i$$

are additionally replaced by the right-hand side (obviously possible in polynomial time and space), resulting in effects of the form

$$\bigwedge_i p_{i1} e_{i1} | \ldots | p_{in_i} e_{in_i} \wedge \bigwedge_j e_j,$$

where the e_{ik} are effects of that form themselves and the e_j are conditional effects $e_j = c \rhd e'$ with atomic effects e'.

Definition 4. An effect e is in *forked normal form* (FNF) if it is deterministic or of the form

$$(p_1 e_1 | \dots | p_i e_i | \dots | p_n e_n) \wedge \bigwedge_j e_j,$$

where all e_i and e_j are conditional effects $c \rhd e'$ with atomic effects e'. An operator o is in FNF if $\text{eff}(o)$ is in FNF.

While there are effects that cannot be transformed into a *single* effect in FNF, it is interesting nevertheless as there is a polynomial time algorithm to transform a planning task $T^{\rhd}$ (created from an arbitrary planning task T) into an equivalent planning task $T^{\Psi} = \langle V \cup V^{\rhd} \cup \{v_{\text{aux}}\}, \sigma(s_0), S_\star \cup \{v_{\text{aux}}{=}0\}, O^{\rhd} \cup O^{\Psi} \rangle$, where all operators are in FNF, and the number of operators is polynomially bounded in the number of probabilistic effects in $o_\star \in O^\star$.

All $o_\Psi \in O^\Psi$ that are created by Algorithm 1 from $o_\star \in O^\star$ have 3 important commonalities besides being in FNF:

- $\text{pre}(o_\Psi) = v_{\text{aux}}{=}\#(o_\Psi)$, where $\#(o_\Psi)$ is unique.
- $\text{eff}(o_\Psi) = \bigwedge_i e_i \wedge (p_1 v_{\text{aux}}{\leftarrow}x_1 | \dots | p_n v_{\text{aux}}{\leftarrow}x_n)$, where $x_1, \dots, x_n \in \mathbb{N}$ are pairwise distinct.
- $c(o) = 0$ (costs are covered by the procedure described earlier).

To create an operator o_Ψ we need to know its deterministic effects e_i, its index $\#(o_\Psi)$, the probabilities p_i, and the values x_i assigned to v_{aux}. While the first three pieces of information are known when an effect is met the first time (lines 2–4), the last is not available before all children have been created as their index corresponds to these values. Due to this, Algorithm 1 maintains a directed acyclic graph of FNF-nodes that correspond to the operators in FNF, as depicted in Figure 1. Once all children have been generated recursively (lines 6–8) they are appended to all leaves (lines 9–10) and o_Ψ is added (line 11).

Theorem 2. *Each planning task can be normalized in polynomial space and time to an equivalent planning task where all operators are in FNF.*

Proof sketch: Our procedure creates one operator for each probabilistic outcome, needing polynomial space, and is obviously computable in polynomial time. Equivalence of $T^{\rhd}$ and T^{Ψ} can be shown analogously to the proof of Theorem 1 by using the same mappings σ and θ and the same properties of decision states. Due to limited space, we sketch the algorithm's behavior in an example instead. ∎

Example 2. Consider the following $\text{eff}(o_\star)$ for $o_\star \in O^\star$:

$$\big(0.5(s{\leftarrow}0 \wedge (0.3\,t{\leftarrow}1 \mid 0.7\,u{\leftarrow}3) \wedge (0.6\,v{\leftarrow}0 \mid 0.4\,w{\leftarrow}4)) \mid 0.5(x{\leftarrow}1)\big) \wedge (0.5\,y{\leftarrow}2 \mid 0.5\,z{\leftarrow}1) \wedge q{\leftarrow}2$$

Let 1 be the first generated index. The effects in FNF, created from the directed acyclic graph in Figure 1, include:

$$\begin{aligned}
\text{eff}(o^1_\Psi) &= q{\leftarrow}2 \wedge (0.5 v_{\text{aux}}{\leftarrow}2 | 0.5 v_{\text{aux}}{\leftarrow}7) \\
\text{eff}(o^2_\Psi) &= s{\leftarrow}0 \wedge (0.3 v_{\text{aux}}{\leftarrow}3 | 0.7 v_{\text{aux}}{\leftarrow}4) \\
\text{eff}(o^3_\Psi) &= t{\leftarrow}1 \wedge (0.6 v_{\text{aux}}{\leftarrow}5 | 0.4 v_{\text{aux}}{\leftarrow}6) \\
\text{eff}(o^4_\Psi) &= u{\leftarrow}3 \wedge (0.6 v_{\text{aux}}{\leftarrow}5 | 0.4 v_{\text{aux}}{\leftarrow}6) \\
&\dots \\
\text{eff}(o^9_\Psi) &= z{\leftarrow}1 \wedge v_{\text{aux}}{\leftarrow}0
\end{aligned}$$

Algorithm 1: Algorithm creating a set of operators in FNF.

```
1  def buildFNF(effect)
2    if effect is conditional then
3        return FNFNode({effect},nextIndex())
4    result = FNFNode(effect.detEffs(),nextIndex())
5    for eff in effect.probEffs() do
6        children = {}
7        for out in eff.outcomes() do
8            children.add(⟨buildFNF(out), eff.prob(out)⟩)
9        for leaf in result.leaves() do
10           leaf.setChildren(children)
11           createOperator(leaf,children)
12   return result
```

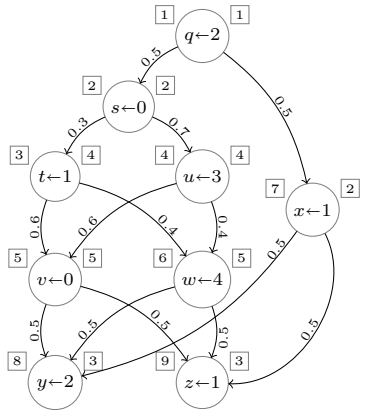


Figure 1: Directed acyclic graph created by Algorithm 1 for $\text{eff}(o_\star)$ in Example 2, each FNF-node corresponding to an operator in FNF and containing its deterministic effects. Numbers in the upper left corner denote the values of v_{aux} in preconditions, and labels give transition probabilities. Numbers in the upper right corner denote the values of v_{aux} in preconditions in the determinization.

Determinization

A deterministic planning task is a planning task in which all operators are deterministic. Our definition of a determinization, which transforms a probabilistic planning task to a deterministic one, captures the same semantics as that of a *compilation* by Little and Thiébaux (2007).

Definition 5. A sequence $t = \langle o_0, \dots, o_n \rangle$ of operators $o_i \in O$ is a *trajectory* in $\mathcal{T} = \langle S, O, \Delta, s_0, G \rangle$, if o_0 is applicable in s_0 and there are states $s_i \in S$ s.t. $\delta_{o_{i-1}}(s_{i-1})(s_i) > 0$ for $1 \le i \le n+1$ and $s_{n+1} \in G$ and o_i is applicable in s_i. The *cost* of t is $\text{cost}(t) = \sum_{i=0}^{n} c(o_i)$.

Definition 6. A planning task $T^d = \langle V^d, s^d_0, S^d_\star, O^d \rangle$ is a *determinization* of a planning task $T = \langle V, s_0, S_\star, O \rangle$ if it is deterministic and for each trajectory t^d in $\mathcal{T}^d$ there is a trajectory t in $\mathcal{T}$ s.t. $\text{cost}(t^d) = \text{cost}(t)$. If there also is a trajectory t^d in $\mathcal{T}^d$ for each trajectory t in $\mathcal{T}$ s.t. $\text{cost}(t) = \text{cost}(t^d)$, T^d is an *all outcome* determinization of T.

Given a planning task T^Ψ as created by Algorithm 1, we can directly generate an all outcome determinization T^d of T^Ψ. Due to limited space we only give a brief description

of the determinization process. Operators $o_{\triangleright} \in O^{\triangleright}$ are already deterministic, and all operators $o_{\Psi} \in O^{\Psi}$ are either deterministic or it holds that $\text{pre}(o_{\Psi}) = v_{\text{aux}}{=}\#(o_{\Psi})$ and $\text{eff}(o_{\Psi}) = \bigwedge_i e_i \wedge (p_1 v_{\text{aux}}{\leftarrow}x_1 | \ldots | p_n v_{\text{aux}}{\leftarrow}x_n)$. Note that assignments of the form $v_{\text{aux}}{\leftarrow}0$ only occur in deterministic operators which remain unchanged.

Let E_p be the set that contains the probabilistic part e_p of $\text{eff}(o_{\Psi})$ for all $o_{\Psi} \in O^{\Psi}$, i.e., assignments of the form $(p_1 v_{\text{aux}}{\leftarrow}x_1 | \ldots | p_n v_{\text{aux}}{\leftarrow}x_n)$ (note that e_p is equal for $o_{\Psi} \in O^{\Psi}$ that were created from FNF-nodes that share their children). Furthermore, let $\#_{\text{det}} : E_p \to \{1, \ldots, |E_p|\}$ be bijective. To determinize we simple replace the probabilistic part e_p of each operator $o_{\Psi} \in O^{\Psi}$ with the deterministic effect $v_{\text{aux}}{\leftarrow}\#_{\text{det}}(e_p)$, thereby creating a deterministic operator.

Example 3. In Figure 1, the numbers on the upper right corner of the nodes indicate the new values of v_{aux} when applying this procedure to Example 2, creating amongst others:

$$\text{eff}(o_d^1) = q{\leftarrow}2 \wedge v_{\text{aux}}{\leftarrow}2$$
$$\text{eff}(o_d^2) = s{\leftarrow}0 \wedge v_{\text{aux}}{\leftarrow}4$$
$$\text{eff}(o_d^3) = t{\leftarrow}1 \wedge v_{\text{aux}}{\leftarrow}5$$
$$\text{eff}(o_d^4) = u{\leftarrow}3 \wedge v_{\text{aux}}{\leftarrow}5$$

The resulting planning task T^d is an all outcome determinization of T^{Ψ} and thus, with our former results, of T.

Experiments

While the standard benchmarks of the latest IPPCs do not make intensive use of parallel probabilistic effects, we believe this feature to be significant in probabilistically interesting planning problems. One example is the Canadian Traveler's Problem (CTP), a path planning problem in an undirected graph where each edge has a weight and a probability of being traversable (Papadimitriou and Yannakakis 1991). Whether an edge is blocked or not is revealed to the agent only if it is located in an adjacent node, so the agent has to reason about the expected cost of paths. Recently, several high-quality domain-dependent strategies for solving the CTP have been proposed (Eyerich, Keller, and Helmert 2010). With the help of parallel probabilistic effects we can encode the CTP very concisely in PPDDL, requiring only one schematic operator changing both the agent's location and determining presence or absence of roads.

Table 1 compares the determinizations based on Rintanen's 1ND normal form (1NDD) and on the forked normal form proposed in this paper (FNFD). For three instances of the CTP with an increasing number of nodes and operators we denote the number of generated deterministic operators, the blowup factor relative to the number of probabilistic operators (indicated by ↑ in the table), and the average number of conditional or atomic effects in the generated operators.

It can be seen that FNFD generally generates far fewer operators than 1NDD. Since 1NDD is exponential in the number of parallel probabilistic effects it highly depends on average and maximal branching factor, which explains why the blowup factor in the problem with 50 nodes (standard deviation of 4.8) is larger than in the problem with 100 nodes (standard deviation of 3.7). In contrast to that, FNFD depends only linearly on the average number of parallel probabilistic effects.

Nodes/Ops	# Roads	Det. Ops	⌀Effects
20/98	4.9 ± 3.7	1122 ↑11.4	3.7
		6008 ↑61.3	21.9
50/278	5.56 ± 4.8	3550 ↑12.8	3.7
		29744 ↑107.0	25.3
100/568	5.68 ± 3.7	7348 ↑12.9	3.7
		55160 ↑97.0	23.7

Table 1: Determinizations based on 1ND (white) and FNF (light gray) on three instances of the CTP with different number of nodes, operators and adjacent roads (with the standard deviation). We state the number of generated deterministic operators (the blowup factor relative to the number of original operators is indicated by ↑) and the average number of effects of the generated operators.

Furthermore, the deterministic operators generated by 1NDD contain a lot of redundancy which is reflected in their average number of effects. In contrast to that, the operators generated by FNFD have a constant average number of effects (3.7) for all problems in the CTP.

Conclusion

In this paper, we have presented FNF, a normal form for probabilistic planning, and a polynomial time and space procedure to map planning tasks to equivalent tasks where all operators are in FNF. Each generated operator can directly be transformed into a single deterministic operator, thereby gaining an all outcome determinization without the exponential blowup resulting from previous determinizations. The results of our experiment emphasize the advantages of our FNF-based determinization.

We will further use FNF to implement a UCT-based domain-independent planning system, providing us with a second advantage: The structure of the graph depicted in Figure 1 can be used in the UCT rollouts to quickly generate nodes that correspond to intermediate states and where probabilistic outcomes are sampled sequentially, thereby avoiding the generation of the possibly exponentially many successor nodes when applying an operator that is not in FNF.

References

Buffet, O., and Aberdeen, D. 2007. FF + FPG: Guiding a policy-gradient planner. In *ICAPS*, 42–48.

Condon, A. 1992. The Complexity of Stochastic Games. *Information and Computation* 96(2):203–224.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-Quality Policies for the Canadian Traveler's Problem. In *AAAI*, 51–58.

Little, I., and Thiébaux, S. 2007. Probabilistic Planning vs. Replanning. In *ICAPS Workshop on IPC*.

Papadimitriou, C. H., and Yannakakis, M. 1991. Shortest Paths Without a Map. *TCS* 84(1):127–150.

Rintanen, J. 2003. Expressive Equivalence of Formalisms for Planning with Sensing. In *ICAPS*, 185–194.

Teichteil-Königsbuch, F.; Infantes, G.; and Kuter, U. 2008. RFF: A Robust, FF-Based MDP Planning Algorithm for Generating Policies with Low Probability of Failure. IPPC Planner Abstract.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS*, 352–359.

1

On Continual Planning with Runtime Variables

MICHAEL BRENNER AND BERNHARD NEBEL

ABSTRACT. This article discusses the problem of planning and acting in partially observable environments. In many such domains conditional planning for all contingencies is prohibitively hard. Therefore we advocate a continual planning approach, where decisions can, by means of so-called *assertions*, be deferred until execution time when more information is available. Additionally, we formalize the notion of *runtime variables* as functional fluents, which can act as placeholders for sensing results unknown at planning time. Using runtime variables and assertions we show how a *series of sequential plans* can solve planning tasks that in non-continual planning would necessitate plans with conditional branching and loops.

1 Introduction

In his 1996 article "What is Planning in the Presence of Sensing?" Hector Levesque discusses the *Airport* example, in which a traveler must find a way onto flight 123 at the local airport. To do this, she must find out the boarding gate for the flight and move there. In such a scenario execution-time sensing must be taken into account already during planning because the agent's behavior will be influenced by the outcome of the sensing action. It is widely accepted that solutions for this or similar problems must be conditional plans that branch over the possible sensor outcomes or even more general structures like policies or finite-state controllers. In 1996 Hector noted: "There clearly is no sequence of actions that can be shown to achieve the desired goal; which gate to go to depends on the (runtime) result of checking the departure screen".

The above statement is obviously true—if by "actions" we mean fully instantiated actions as typically used in classical propositional planning. However, consider how a human would probably represent her "recipe" for getting onto flight 123 in some arbitrary imperative programming language:

```
gate = read-departure-screen(flight123)
move-to(gate)
board(flight123)
```

This *is* a sequence of actions; however, it makes use of one the most basic concepts of imperative programming, namely a *runtime variable* `gate`. If our planning model allowed for such runtime variables, it would be possible to find a sequential plan to achieve the goal. It is true, however, that not all problems of planning with sensing can be solved using sequential plans and runtime variables—sometimes different sensing results will require different reactions. For instance, big international airports, such as Frankfurt or Toronto, usually have several terminals, so that getting to the gate for flight 123 may or may not

involve a transfer to another terminal. This surely cannot be expressed with a sequential plan, can it?

Our approach to Levesque's Airport Example is based on the observation that when people find their way around an airport they do act in a goal-directed, planful way, yet they do not seem to branch over all possible conditions in advance. This does not mean that they randomly explore the airport or make guesses about the departure gate that they must revoke when they find them violated. Rather, they rely on additional domain knowledge stating, e.g., that there will be signs to guide them to any gate—even if this includes transfer to another terminal. As a result, a typical passenger at Frankfurt airport with little information about the airport's terminals, gates, and layout may still confidently follow the following plan:

---

gate = read-departure-screen(flight123)
follow-signs(gate)
board(flight123)

This plan (or program) not only features the runtime variable `gate`, but also a call to a *subroutine* `follow-signs()`. Execution of the subroutine, i.e. actually following the signs will most likely involve moving around the airport and reading signs repeatedly. Interestingly, those details are usually not planned for in advance by humans and indeed often *cannot* be, because the planning domain is incompletely specified at planning time. In the Airport example, a traveler usually neither knows the number of possible gates nor the layout of an airport before getting there.

Instead, at planning time the human relies on the "contract"[1] of the subroutine, i.e. the yet unspecified subplan `follow-signs(gate)`: she knows that, once she is at the airport and has found out her gate, she will be able to find her way to that gate. In other words, although the subplan named `follow-signs(gate)` is not yet concretized, it is already characterized by a *Hoare triple* $\{P\}$`follow-signs(gate)`$\{Q\}$[Hoare 1969]: If certain preconditions P are guaranteed to hold before the subplan, it guarantees that certain postconditions Q can be achieved.

We will call such unspecified subplans *assertions*. The notion is inspired by the use of the same term in computer programming, where a programmer asserts that certain conditions must hold at a certain point in the program execution. An assertion is characterized by its pre- and postconditions, exactly like normal actions in the domain. However, like subroutines in a program or methods in Hierarchical Task Networks it will be replaced by a concretized plan before being executed. Note, however, that in both programming and hierarchical planning this concretization is assumed to be given in advance. In our approach, as in the Airport example, it is the agent itself that will fill in the details of the plan as soon as the missing information becomes available. The task is thus not decomposed hierarchically, but *temporally*: parts of the planning problem that cannot be solved *yet* are postponed and plan execution is started early in order to gather additional knowledge. Planning and execution are thus integrated and interleaved deliberately by the agent—a process that we call *Continual Planning*.

Interestingly, the assertion `follow-signs(gate)` is used like any normal action in the above plan, which therefore remains sequential. Likewise, when the assertion is *expanded*,

[1] This terminology is borrowed from the programming language Eiffel and its principle of "Design by Contract" [Meyer 1992].

i.e. replaced by a more concrete subplan[2] once the agent reads the departure screen, this new plan is again a sequential one. By then the agent will know the concrete departure gate and must only find *one* plan for how to reach it. It is the great advantage of this form of continual planning that branching over possible contingencies is thus avoided. Instead, a *series* of sequential plans is generated and executed.

The remainder of the paper is structured as follows. In the following section, we will discuss related approaches to planning with partial observability. Section 3 presents our continual planning formalism and algorithm. In Section 4 we show how an extended version of Levesque's Airport example can be solved by continual sequential planning, which otherwise would require plans with conditional branches and loops. We conclude with a brief discussion of future work.

2 Related Work

Planning with sensing actions has been extensively studied in the planning literature [Levesque 1996; Golden and Weld 1996; Weld, Anderson, and Smith 1998; Bonet and Geffner 2000; Petrick and Bacchus 2002; Petrick and Bacchus 2004]. Usually, this is done in a conditional planning setting. Unfortunately, conditional planning with partial observability is of prohibitively high computational complexity, even in relation to classical planning [Rintanen 2004].

Additionally, it is widely acknowledged that, as stated in the textbook by Russell and Norvig, "even the best-laid plans of mice, men and conditional planning agents frequently fail" [Russell and Norvig 2003]. Practical planning agents must therefore be extended with capabilities for execution monitoring and plan adaption or replanning in order to be able to react quickly to unexpected circumstances and events [Fritz and McIlraith 2007]. Such agents are sometimes called continual planning agents, because they continually switch between plan execution and replanning until they have reached their goal. In accordance with Russell and Norvig's textbook we will, however, refer to such agents as *replanning agents* and reserve the term *continual planning agent* to agents that can deliberately switch to plan execution during the planning process, i.e. even before a plan has fully been elaborated.

Continual planning agents try to evade the complexity of conditional planning by not only planning for information gathering, but by actually performing information gathering before planning for all future contingencies. Possibly the earliest approach so tightly integrating planning, monitoring, execution and information gathering was the IPEM system [Ambros-Ingerson and Steel 1988]. In this, as well as in later systems like Sage [Knoblock 1995], execution is integrated into a partial-order planning algorithm by treating unexecuted actions in the plan as a special kind of "flaw" that the planner has to resolve by executing the action at some point. It is not clear in these earlier approaches, however, if or how a planner can decide that it is necessary or helpful to start execution before a complete plan has been found.

In the present work, assertions are used to ensure the planner that it can safely switch to execution before having a detailed plan in order if necessary to proactively gather relevant information. The notion of assertions was developed in our previous work [Brenner and Nebel 2009]. However, there we used a different model for the symbolic effects of sensing actions which, in a nutshell, just stated that the value of a fluent would be *known* after a sensing action, similarly to corresponding models in epistemic logic [Fagin, Halpern,

[2] This plan may also contain assertions, but must have at least an executable prefix (cf. Section 3).

Moses, and Vardi 1995]. Assertions then used knowledge preconditions to make statements about the planning domain such as "If I knew my gate, I could find a plan to get on my flight". However, normal operators do not have knowledge preconditions, but refer to concrete facts in the current situation for testing applicability. When sensing operators are only modeled in terms of their epistemic effects, their results cannot be used to make these preconditions true. Therefore, in this work, we use a different model for sensing, based on runtime variables.

Etzioni and colleagues first used *runtime variables* to refer to sensing results in a plan that will only become known at execution time [Etzioni, Hanks, Weld, Draper, Lesh, and Williamson 1992; Etzioni, Golden, and Weld 1997; Golden 1998]. Similarly, Petrick and Bacchus used 0-ary functional fluents as runtime variables [Petrick and Bacchus 2002; Petrick and Bacchus 2004]. While these approaches successfully made use of runtime variables, they did not clearly define the semantics of planning with them. Here, we provide such a definition in the context of the Functional STRIPS language developed by Geffner [Geffner 2000].

Assertions are similar to schemata or "methods" in Hierarchical Task Networks (HTNs) [Yang 1997; Erol, Hendler, and Nau 1996; Nau, Cao, Lotem, and Munoz-Avila 1999; Nau, Au, Ilghami, Kuter, Murdock, Wu, and Yaman 2003] in that they will be decomposed into more concrete subplans until the goal can be reached. The purpose of both approaches is different, though: While HTNs essentially provide search guidance for a planner by explicitly decomposing a problem into (ideally) independent subtasks, assertions enable the planner to decompose the problem temporally into different planning and execution phases. As a result, where HTN planners generate abstraction hierarchies, continual planning with assertions generates a series of non-hierarchical plans.

While HTN domain designers need to explicitly provide method decompositions, no pre-defined abstraction hierarchies are required in continual planning. Rather, the designer only specifies conditions for the existence of (sub-)plans in a particular domain, much like a programmer specifying the "contract" of a procedure in programming languages like Eiffel [Meyer 1992]. The planner itself then finds expansions for assertions, i.e. it synthesizes concrete subprograms once the preconditions of the "contract" are satisfied.

Our approach is also related to work on integrating planning into Golog-like action languages [Giacomo, Lesperance, and Levesque 2000; De Giacomo, Lesprance, Levesque, and Sardiña 2002], in particular to our previous work on integrating a planner with IndiGolog [Classen, Eyerich, Lakemeyer, and Nebel 2007]. IndiGolog programs, like continual planning agents, can interleave planning, acting and information gathering, so that the results of sensing actions can be reacted to immediately. Previously, we have shown how planning subproblems can be extracted from an IndiGolog program at runtime and solved by a planning system [Classen, Eyerich, Lakemeyer, and Nebel 2007]. These calls to an external planner replace the generic forward-search operator *achieve* (or *search*), which is used to refer to some yet unspecified solution to a subproblem in an IndiGolog program—which is exactly the role of an assertion in a continual plan. Thus, where *achieve* enables an IndiGolog programmer to hand over some control to an autonomous planner, assertions can be said to enable the domain designer of a planning domain to specify some additional domain information, namely about the solvability of subproblems, to the planner.

3 Planning Model

Our planning language is a variation of Geffner's Functional STRIPS [Geffner 2000], a typed first-order language with *name*, *type* and *function* symbols without quantifiers or variables. Relations are not explicit in the language, but can be expressed by using functions with a Boolean codomain. To simplify the presentation, we will use standard relational notation in our examples anyway, i.e. we will write, e.g., $\neg$*connected(pos(), dest)* instead of *connected(pos(), dest)* $= \bot$ when comparing or *connected(pos(), dest)* $:= \bot$ when assigning.

Name symbols are assumed to be non-fluent, i.e. they are fixed names of objects interpreted *under the unique name assumption*; therefore we will simply equate name symbols with their denotations in the following. Function symbols are *fluent*, i.e. their denotation depends on the state they are interpreted in. Terms and formulas must obey the usual formation rules ("well-formedness"). For a term t and formula f, we write t^s and $f^s(t^s)$ to denote their interpretations in state s (but omit the superscript s if it is irrelevant or obvious from context).

3.1 Tasks

We can now define the tasks a continual planning agent is supposed to solve.

DEFINITION 1. A **continual planning task** is a tuple $T = \langle \mathcal{T}, F, \mathcal{O}, s_0, s_* \rangle$ where

- $\mathcal{T}$ is a set of **types**, where each type t is associated with a set of names, called its **domain** $\mathcal{D}^t$. $\mathcal{C} = \cup_{t \in \mathcal{T}} \mathcal{D}^t$ is the set of all **names**.
- $\mathcal{F}$ is a set of **function symbols**. Each $f \in \mathcal{F}$ has an associated signature $f : \mathcal{D}^{t_1} \times \cdots \times \mathcal{D}^{t_n} \rightarrow \mathcal{D}^{t_{n+1}}$ (with $n \geq 0$ and $t_i \in \mathcal{T}$). $\mathcal{D}^f = \mathcal{D}^{t_1} \times \cdots \times \mathcal{D}^{t_n}$ is called the domain of f, and $\mathcal{D}^{t_{n+1}}$ is called the codomain of f and referred to by *codomain*(f).
- A **state** s is an interpretation of the language that is defined by the names $\mathcal{C}$ and function symbols $\mathcal{F}$ over $\mathcal{C}$. Since, as stated above, we treat names as their own interpretations, the state s is represented by the interpretations of each function symbol f over its domain $\mathcal{D}^f$, i.e. the values $w \in$ *codomain*(f).
- A **belief state** b is a set of states representing the states the agent assumes to be possible. A set of belief states B will be called **belief set** and represents the possible beliefs the agent can have - given the sensing actions the agent performed.
- $\mathcal{O}$ is a set of **operators** of the form $\langle param, pre, \mathit{eff}, sense \rangle$.
  - $param$ is a list of typed schema variables. They can be used in the rest of the operator definition as place holders for fluents.
  - In analogy to Functional STRIPS [Geffner 2000], the precondition pre is a conjunction of conditions of the form $t = w$. With t and w be well-formed terms[3] .
  - The effect eff is a list of conditional updates $(c \rightarrow e)$ where c is an effect condition with the same restrictions as a precondition and e is an atomic update of the form $t := w$, and t and w are terms of the same type.

[3] *Terms* are built in the usual manner using function and name symbols of compatible types and arities [Geffner 2000].

  - $sense$ is the sensed fluent in a sensing action, otherwise it is unspecified.

  Every operator o is declared to belong to one of the following three disjoint sets:

  - the set of **sensing operators** $\mathcal{O}^s$($o \in \mathcal{O}^s$ iff $sense$ is specified for o)
  - the set of **assertions** $\mathcal{O}^a$,
  - the set of **standard operators** $\mathcal{O}^o$,

  The induced set of actions $\mathcal{A}$ is the set of parameter-free operators generated by substituting all possible functional fluent expressions for the scheme variables in the operators. This set is potentially infinite. The sets $\mathcal{A}^o$, $\mathcal{A}^a$, and $\mathcal{A}^s$ are defined analogously to the corresponding sets of operators.

- s_0 is a singleton belief set, called the **initial belief set**. It is specified by a conjunction of expressions of the kind $t = c$, with t being some term of our language and c a name. Since s_0 is singleton, it denotes the unique *belief state*, where for each fluent either the value is known or it can be any value of the domain of the fluent.

- s_* is a formula describing the **goal condition**, which we will assume to have a form like a precondition of an action, i.e., a conjunction of fluent equations.

Since continual planning alternates between planning and plan execution, we will need to distinguish between the semantics of the physical and the symbolic execution of an action, the latter describing the state transitions reasoned about in the planning process under incomplete knowledge, the former modeling the actual results of executing the action in a particular world state.

Furthermore, for the symbolic execution, we will distinguish between execution in the *full belief set space* and a *simplified model*, which is defined below.

In order to do so, we have to define what it means that a condition φ of an action a is satisfied by a belief set B, a belief state b and a classical state s. A state s satisfies φ, in symbols $s \models \varphi$, if s satisfies φ classically. A belief state b satisfies φ, symbolically $b||\!\models\varphi$, if for all $s \in b$ we have $s \models \varphi$. That is, regardless of what we consider as possible, the condition is true. Finally, a condition ϕ is **satisfied by the current belief set** B, symbolically $B||\!\models\varphi$, iff all $b \in B$ satisfy φ.

DEFINITION 2. The **symbolic execution over the full belief set space** is defined as follows:

- Standard actions and assertions $a \in \mathcal{A}^o \cup \mathcal{A}^a$ can only be executed if the preconditions are satisfied by B. The next belief set consists of all belief states that result from executing a in each of the belief states. Executing an action a in a belief state b is simply the execution on each state $s \in b$, which means the application of all conditional effects of a in s, i.e., if the effect condition is satisfied in s, then the effect is made true in the resulting state.

- A sensing action follows the same execution model except that right in the beginning, before the effects of a are applied, the sensing of the fluent f of type t in $sense$ takes place. This results in splitting each belief state into $\|\mathcal{D}^t\|$ belief states, such that for each $c_i \in \mathcal{D}^t$ that is possible for f in the belief state b, there is a belief state b_i that satisfies $f = c_i$ and is otherwise identical to the original belief state b. After that the

effects of the sensing action are applied. The resulting belief set contains all belief states generated in this way.

Based on these definitions, we say that a plan P is a successful plan for a planning task $T = \langle \mathcal{T}, F, \mathcal{O}, s_0, s_* \rangle$ if the actions in P sequentially executed on the belief set s_0 lead to a belief set B that satisfies s_*.

Domains:	*Egg: egg0, egg1* *Pen: pen1, pen2, pen3* *Object; egg0, pen1, sample* *Color: red, blue, green*
Fluents:	*color : Object* $\rightarrow$ *Color* *sampleColor :* $\emptyset$ $\rightarrow$ *Color*
Action:	***paint-egg****(egg:Egg, pen:Pen)*
Prec:	$\emptyset$
Post:	*color(egg) := color(pen)*
Sensor:	***sense-color****(obj:Object, col():Color)*
Sense:	*color(obj)*
Prec:	$\emptyset$
Post:	*col() := color(obj)*
Init:	*color(pen1)=green, color(pen2)=blue, color(pen3)=red*
Goal:	*color(egg0)=color(sample), color(egg1)=color(sample)*

Figure 1. *Easter eggs* continual planning domain

An example planning domain and task is shown in Figure 1 (syntactically, we follow Geffner's Functional Strips [Geffner 2000]). In this task, easter eggs must be painted in some sample color which must be determined by sensing first. In our framework, the goal is achieved without branching by the following plan:

```
sense-color(sample, sampleColor())
paint-egg(egg0, sampleColor())
paint-egg(egg1, sampleColor())
```

A more elaborate example will be discussed in Section 4.

3.2 A Simplified Execution Model

Our planning language gives us only limited ways of expressing uncertainty and knowledge gathering. In particular, because we stick to linear plans, we cannot branch on sensed values in the plan (we will later discuss how assertions can mitigate this restriction). On the positive side, a more compact representation of belief sets seems possible in our framework.

DEFINITION 3. The special value u does not belong to a type and denotes the **unknown** value. For any type t, $\mathcal{D}^t_{\mathsf{u}} = \mathcal{D}^t \cup \{\mathsf{u}\}$ and $\mathcal{C}_{\mathsf{u}} = \cup_{t \in \mathcal{T}} \mathcal{D}^t_{\mathsf{u}}$

A **simplified belief state** is a singleton set containing one state that is an interpretation of all the fluents over $\mathcal{C}_{\mathsf{u}}$ such that for all fluents that are interpreted as u, all values of the codomain of the fluent are possible.

A condition φ is satisfied by a simplified belief state $b = \{s\}$ iff none of the subterms in φ are interpreted as u in s and φ is classically satisfied by the state s.

Using this notion of simplified belief set, we can express the initial belief set that contains only one belief state as one interpretation. All fluents that have a value assigned have that value, all others have the value u.

Without sensing and without conditional effects, such a simple execution model would be actually equivalent to one that is executed over full belief sets. This can be shown by a simple compilation to standard basic STRIPS [Nebel 2000].

Domains:	*Type1: 1, 2*
Fluents:	*right, wrong, do, done:* $\emptyset \rightarrow$ *Bool* *a:* $\emptyset \rightarrow$ *Type1*
Action:	***case()***
Prec:	$\emptyset$
Post:	$a()=1 \rightarrow right(), a()=2 \rightarrow right(), a()=1 \rightarrow \neg wrong(),$ $\top \rightarrow do()$
Action:	***unsound()***
Prec:	*wrong(), do()*
Post:	*done()*
Init:	$\emptyset$
Goal:	*right()*

Figure 2. Small example demonstrating incompleteness and unsoundness

When the planning language contains conditional effects, planning with respect to simplified belief sets becomes incomplete with respect to planning over the full belief set space, as is demonstrated by the artificial planning task in Figure 2. In order to achieve the goal *right()*, the action **case** is sufficient in the full belief set space. Under the simplified model, however, the conditional effects are not activated, because the conditions are not satisfied.

Furthermore, one has to be careful to avoid the traps of being unsound with respect to the belief set model. For example, if we consider the initial description $\{wrong\}$ and the goal $\{done\}$, then there is no plan for the full belief set space. However, the plan $\langle$**case**, **unsound**$\rangle$ seems to achieve the goal under the simplified belief set space because none of the conditional effects in the action **case** are executed.

In order to avoid unsoundness, uncertainty has to be propagated over conditional effects. So, **execution over simplified belief sets** differs as follows: if a fluent has the value u and this fluent is part of an effect condition, then the effect fluent will become unknown as well.

This leads in general to the fact that less actions can be applied and that less fluents will have a known value.

PROPOSITION 4 (Soundness of the simplified model). *Any successful plan that can be executed over the simplified belief set space is a successful plan over the full belief set space.*

While this simplification reduces the number of states we have to store exponentially, the belief set still contains a number of states that is exponential in the number of sensing actions. However, do we really need to track all of these states? Wouldn't it be enough to simply remember that the value of a fluent is *known* after a sensing action has been executed on this fluent? For instance, we could introduce another special domain element k that could be (implicitly) assigned when a fluent is sensed and tested afterwards, e.g., with an expression such as *departure(flight123)* $=$ k. In fact, this was the intended semantics of the approach described in an earlier paper of ours [Brenner and Nebel 2009] (where we did not deal with functional fluents, though).

It is to be expected that this move, again, introduces incompleteness. However, even worse, it does not even allow to execute our examples, because they rely on the equality between two fluents, e.g., *gate()* equaling *departure(flight123)*, of which the concrete value is unknown at planning time. Similarly, our plan for the easter eggs example of Figure 1 would not ensure that eggs and sample object share the same color in the final state.

Another idea might be to introduce a new anonymous object for each value sensed. Such a move would make our examples executable. Again, such a model would not be complete with respect to the simplified belief model. For instance, in some task there may be a number of different persons, with each person's age being known. If we sense a particular person, we should know the age of the person as well. This is true for the semantics of Definition 2. However, if during the simplified symbolic execution a new anonymous object is introduced for the person seen, the value for her age will be unknown.

Things are even worse. Namely, introducing anonymous objects can make planning unsound with respect to the full belief set model, provided we can introduce unlimited many new anonymous objects. As sketched in Figure 3, we could simulate a Turing machine and solve the Halting Problem on the empty tape by planning in the execution model with anonymous objects.

Of course, the number of new anonymous objects could be restricted, but it is not clear whether this will lead to soundness. For this reason, as a compromise between efficiency and completeness, we adapted the simplified belief set space as described above. While it seems obvious that this simplified model is easier to implement, the question of whether there is indeed a reduction in computational complexity for planning has still to be resolved.

3.3 Continual Planning with Assertions

We can now describe how planning, i.e. reasoning about the symbolic execution of actions, can lead to physically achieving a goal.

In the following, we will refer to the **symbolic execution** of an action a or a plan P in the simplified belief set state B as $project(B, a)$ respectively as $project(B, P)$. This is to be contrasted with the *physical execution* of actions. These always take place in a single simplified belief state b and consist of checking the preconditions and executing the (conditional) effects. In case of *sensing* actions, one particular value is nondeterministically chosen. We denote the **physical execution** of an action a in a simplified belief state b by $execute(s, a)$ and, respectively, the physical execution of a plan P by $execute(s, P)$.

Domains:	*Cell: cell0*
	*Char: 1, 0, **
	State: q0, q1, q2, . . . , qf
Fluents:	*next : Cell → Cell*
	prev : Cell → Cell
	cont : Cell → Char
	head: ∅ → Cell
	state: ∅ → State
	init: ∅ → Bool
Sensor:	***fresh****(cell:Cell)*
Sense:	*next(cell)*
Prec:	*init()*
Post:	*prev(next(cell)) := cell, char(next(cell)) := **
Action:	***switch-to-computation****()*
Prec:	*init()*
Post:	*¬ init()*
Action:	$\boldsymbol{rule}_1()$
Prec:	*cont(head()) = 1, state() = qi*
Post:	*cont(head()) := 0, head() := next(head()), state() = qk*
⋮	
Init:	*cont(cell0) = *, init(), head() = cell0, state() = q0*
Goal:	*state() = qf*

Figure 3. Turing machine encoding

Assertions *cannot* be executed physically, i.e. $execute(s, a)$ is undefined for any assertion a, even if its preconditions are satisfied in s. This crucial semantic difference between $project()$ and $execute$ will lead to the intended "expansion" of assertions during continual planning: CP algorithms must take into account that assertions can never be selected for physical execution, therefore they must make sure that assertions never appear first in a plan (in the case of partially ordered plans, they must never be ordered immediately after the *init* action). Additionally, after an action is physically executed and removed from the plan *monitoring* must check if this constraint is violated in the updated plan. In that case, a new planning phase is triggered in which the assertion is not allowed to be used at the beginning of the plan any more.

Algorithm 3.3 shows the basic continual planning algorithm. Here, PLANNER is a generic planning algorithm computing valid, i.e. symbolically executable, plans and ensuring that every plan P returned by PLANNER$(\{b\}, G)$ does not start with an expandable

Algorithm 1 Generic continual planning algorithm.

```
P ← ⟨ ⟩
while b ⊭ s_* do
  P ← PLANNER({b}, G)
  if P = failure then
    return "No solution from b"
  while PLANISVALID(b, P) do
    a ← POP(P)
    b ← EXECUTEACTION(a, b)
return "Goal reached"
```

assertion a, i.e. $\{b\}$ does not satisfy $pre(a)$. Similarly, PLANISVALID is a generic plan monitoring procedure [Russell and Norvig 2003]. It checks if the plan P updated after its first action has been executed does not begin with an expandable assertion and, if that is not the case, computes $project(\{b\}, P)$ to verify whether the state resulting from symbolic execution of the updated plan P still satisfies the goal.

Algorithm 3.3 will produce a sequence of belief states and plans that we call a *CP trace*.

DEFINITION 5 (CP trace). Let $\langle \mathcal{T}, F, \mathcal{O}, s_0, s_* \rangle$ be a continual planning task. $T = \langle b_0, P_0, \ldots, P_{n-1}, b_n \rangle$ is a sequence alternating belief states and plans and is a called a **CP trace** if

- $project(\{b_i\}, P_i) \| \models s_*$, i.e. P_i symbolically executed in $\{b_i\}$ achieves s_*
- $b_{i+1} = execute(s_i, a)$ where $a = P[0]$, i.e. b_{i+1} results from physically executing the first action of P in b_i.

While it would be desirable that all CP traces for a goal G generated by Algorithm 3.3 always end in a goal state, in general this can be guaranteed only if assertions are expanded in a side-effect free manner.

PROPOSITION 6 (Soundness). *Let $\langle \mathcal{T}, F, \mathcal{O}, s_0, s_* \rangle$ be a continual planning task and b the simple belief state representing s_0. Then Algorithm 3.3 generates a CP trace $T = \langle b_0, P_0, \ldots, P_{n-1}, b_n \rangle$ such that b_n satisfies s_*, provided for each assertion it is always possible to find a plan that makes the effects of the assertions in all plan P_i true and does not change anything else.*

Obviously, enforcing complete absence of side effects is a rather strong limitation of applicability. However, in practice we can mitigate the restriction: We can determine those potential side effects that would be harmful to the plan suffix by regressing from the goal, and then force the planner to not achieve these when expanding assertions. How this affects soundness is a topic of future work. In particular, we are interested in describing structural properties of *domains* that guarantee soundness of planning with assertions.

4 Worked Example

Let's step through Hector Levesque's *Airport* example once more, modeled as a continual planning problem, as shown in Figure 4. In the beginning, the agent is situated at *gate0*, not knowing at which gate her flight, *flight123*, departs. At *gate0* there is a big screen indicating the departing gates for all flights, as indicated by the precondition of action

Domains:	*Gate : gate0, gate1, gate2, gate3, ...* *Flight: flight123, flight456, flight789, ...*
Fluents:	*departure : Flight → Gate* *pos : ∅ → Gate* *connected : Gate × Gate → Bool* *direction : Gate × Gate → Gate* *gateA, viaA : ∅ → Gate*
Action:	***move-to****(dest:Gate)*
Prec:	*connected(pos(), dest)*
Post:	*pos() := dest*
Action:	***board****(flight:Flight)*
Prec:	*pos() = departure(flight)*
Post:	*pos() := flight*
Assertion:	***follow-signs****(dest:Gate, via:Gate)*
Prec:	*connected(pos(), via), direction(pos(), dest) = via*
Post:	*pos() := dest*
Sensor:	***read-departure-screen****(flight:Flight,gate():Gate)*
Sense:	*departure(flight)*
Prec:	*pos() = gate0*
Post:	*gate() := departure(flight)*
Sensor:	***read-sign****(dest:Gate,via():Gate)*
Sense:	*direction(pos(), dest)*
Prec:	∅
Post:	*via() := direction(pos(), dest),* *connected(pos(), via()), connected(via(), pos())*
Init:	*pos() := gate0*
Goal:	*pos() = flight123*

Figure 4. *Airport* continual planning domain

read-departure-screen. This situation is described by the *Init:* statement of Figure 4, which defines the initial belief set $\{b_0\}$.

Using the PLANNER of Algorithm 3.3 the agent comes up with the following plan P_0:

```
read-departure-screen(flight123, gateA())
read-sign(gateA(), viaA())
follow-signs(gateA(), viaA())
board(flight123, gateA())
```

The first two actions in this plan are sensing actions, with fluents *gateA()* and *viaA()* acting as runtime variables that represent the values sensed in the remainder of the plan. The first action, *read-departure-screen(flight123, gate())* equates *gate()* with the departing gate for flight 123. Next, the agent will sense where to go next in direction of *gate()*. To enable this, the planning domain models "signs" with the function fluent *direction*. It is assumed to map pairs *(src,dest)* to the position closest to *src* on the way to *dest*. Thus, the second action, *read-sign(gateA(), viaA())*, determines the gate *via()* closest to the current position on the path to *gateA()*.

After the two sensing actions, *follow-signs(gateA(), viaA())* asserts that the agent will be able to find a path to *gateA()*, i.e. a subplan that will be continually developed after the agent has found out her concrete gate and while she is following the signs around the airport. Based on the postcondition of this assertion, the agent can safely plan to *board(flight123, gateA())* in the last step.

Having produced an initial plan that "hides" all contingencies and future plan variants, the agent can follow Algorithm 3.3 and switch to plan execution. When she physically executes *read-departure-screen(flight123)* she has reached belief state b_1 in the CP trace, in which she will know the actual departure gate. Let's assume this is *gate37*. The fluents *departure(flight123)* and *gateA()* both have this value in b_1. Likewise, when *follow-signs(gateA(), viaA())* is executed, *via()* is assigned a value, e.g. *gate1*, that corresponds to the one of *direction(gate0,gate37)*.

After their physical execution both sensing actions have been removed from the plan. We have reached belief state b_2 in the CP trace and the current plan looks like this (to clarify the presentation, the runtime variable fluents have been replaced with their values):

---

follow-signs(gate37, gate1)
board(flight123, gate37)

This plan, however, is no longer accepted by PLANISVALID, because it starts with an assertion. Thus, the agent enters a new planning phase, which produces the plan P_2:

---

move-to(gate1)
read-sign(gate37, viaA())
follow-signs(gate37, viaA())
board(flight123, gate37)

As can be seen, the original assertion *follow-signs(gate37, gate1)* has been expanded into a new subplan that starts with executable actions again. Note also that by deferring choices until the value of *departure(flight123)* was known, the continual planning agent avoided branching and can immediately commit to one specific more detailed plan now, namely the one having *gate37* as the boarding gate.

Since P_2 contains an assertion again, it can obviously still not be the final, fully executable plan in the CP trace. Most interestingly, P_2 uses the same assertion, *follow-signs*, as the previous plans *again*. It is obvious that the continual planner will *loop* through a cycle of producing similar plans and executing their prefixes until the agent has moved to a gate adjacent to *gate37*. This loop, however, is not explicit in the plan, as would be the case, e.g., in the approaches to planning with loops developed by Hector Levesque [Levesque 2005; Hu and Levesque 2009]. Instead, it is continually created by the continual planning

algorithm through the interleaving of planning with acting. Again, this phenomenon can be described with an analogy from computer programming: The continual planning algorithm creates looping behavior through *recursion*.

5 Discussion and Future Work

The conventional approach to incomplete knowledge and sensing is to employ conditional plans (or policies) in order to deal with contingencies after sensing actions. Since this is computationally infeasible in many applications, we instead proposed to rely on linear plans and defer decisions as much as possible to the execution time when more information is available [Brenner and Nebel 2009]. The tool to delay such planning time decisions are *assertions*, which are basically abstract actions that can be expanded into subplans at execution time. On top of that, we propose here to employ *runtime variables* to deal with values that only become known at runtime. This devices permits us to deal with a number of scenarios introduced in the literature dealing with incomplete knowledge and sensing.

In order to formalize the relationship between plans interpreted on the symbolic level and plans that are executed and replanned while being executed, we developed formal execution models for the symbolic and physical execution and showed that the planning process is sound under some severe restrictions.

A number of questions remain unanswered at this point. In particular,

- we would like to determine the precise computational complexity for verifying and generating plans under the different symbolic execution models;
- we are interested in devising efficient data structures for representing the simplified belief set space;
- we intend to identify more properties of domain structures that guarantee soundness of abstract plans;
- and finally, we want to evaluate the proposed approach empirically..

Acknowledgements

This work has been supported by the European Commission as part of the Integrated Project *CogX* (FP7-ICT-2xo15181-CogX), by the German Research Foundation (DFG) as part of the collaborative research center SFB/TR-14 *AVACS*, and by the German Space Agency (DLR) as part of the project *KontiPlan*.

References

Ambros-Ingerson, J. A. and S. Steel [1988, August). Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference of the American Association for Artificial Intelligence (AAAI-88)*, Saint Paul, MI, pp. 83–88.

Bonet, B. and H. Geffner [2000]. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, pp. 52–61. AAAI Press, Menlo Park.

Brenner, M. and B. Nebel [2009]. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems 19*(3), 297–331.

Classen, J., P. Eyerich, G. Lakemeyer, and B. Nebel [2007]. Towards an integration of Golog and planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 1846–1851. AAAI Press.

De Giacomo, G., Y. Lesprance, H. Levesque, and S. Sardiña [2002, April). On the semantics of deliberation in indigolog – from theory to implementation. In D. Fensel, F. Giunchiglia, D. McGuinness, and M. A. Williams (Eds.), *Proceedings of Eighth International Conference in Principles of Knowledge Representation and Reasoning (KR-2002)*, Toulouse, France, pp. 603–614. Morgan Kaufmann.

Erol, K., J. Hendler, and D. Nau [1996]. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence 18*, 69–93.

Etzioni, O., K. Golden, and D. S. Weld [1997]. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence 89*(1-2), 113–148.

Etzioni, O., S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson [1992]. An approach to planning with incomplete information. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference (KR-92)*, Cambridge, MA, pp. 115–125. Morgan Kaufmann.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi [1995]. *Reasoning About Knowledge*. MIT Press.

Fritz, C. and S. A. McIlraith [2007]. Monitoring plan optimality during execution. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, Providence, Rhode Island, USA. Morgan Kaufmann.

Geffner, H. [2000]. Functional Strips: A more flexible language for planning and problem solving. In J. Minker (Ed.), *Logic-Based Artificial Intelligence*. Dordrecht, Holland: Kluwer.

Giacomo, G. D., Y. Lesperance, and H. J. Levesque [2000]. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence 121*(1-2), 109–169.

Golden, K. [1998]. Leap before you look: Information gathering in the PUCCINI planner. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pp. 70–77.

Golden, K. and D. Weld [1996]. Representing sensing actions: The middle ground revisited. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 5th International Conference (KR-96)*. Morgan Kaufmann.

Hoare, C. A. R. [1969, October). An axiomatic basis for computer programming. *Communications of the ACM 12*, 576–580.

Hu, Y. and H. J. Levesque [2009]. Planning with loops: Some new results. In *Proceedings of the ICAPS 2009 Workshop on Generalized Planning: Macros, Loops, Domain Control. September 20th, 2009, Thessaloniki, Greece*.

Knoblock, C. A. [1995]. Planning, executing, sensing, and replanning for information gathering. In C. Mellish (Ed.), *Proc. the Fourteenth International Joint Conference on Artificial Intelligence*, San Francisco, pp. 1686–1693. Morgan Kaufmann.

Levesque, H. J. [1996]. What is planning in the presence of sensing? In *Proceedings of the 13th National Conference of the American Association for Artificial Intelligence (AAAI-96)*, pp. 1139–1146. MIT Press.

Levesque, H. J. [2005]. Planning with loops. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, Edinburgh, Scotland, UK, pp. 509–515. Professional Book Center.

Meyer, B. [1992, October). Applying "Design by Contract". *Computer 25*, 40–51.

Nau, D., Y. Cao, A. Lotem, and H. Munoz-Avila [1999, August). SHOP: Simple hierarchical ordered planner. In T. Dean (Ed.), *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, Sweden. Morgan Kaufmann.

Nau, D. S., T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman [2003]. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research 20*, 379–404.

Nebel, B. [2000]. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research 12*, 271–315.

Petrick, R. and F. Bacchus [2002]. A knowledge-based approach to planning with incomplete information and sensing. In M. Ghallab, J. Hertzberg, and P. Traverso (Eds.), *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, Toulouse, France. AAAI Press.

Petrick, R. P. A. and F. Bacchus [2004]. Extending the knowledge-based approach to planning with incomplete information and sensing. In S. Zilberstein, J. Koehler, and S. Koenig (Eds.), *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7, 2004, Whistler, British Columbia, Canada*, pp. 2–11. AAAI Press.

Rintanen, J. [2004]. Complexity of planning with partial observability. In S. Zilberstein, J. Koehler, and S. Koenig (Eds.), *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7, 2004, Whistler, British Columbia, Canada*, pp. 345–354. AAAI Press.

Russell, S. and P. Norvig [2003]. *Artificial Intelligence: A Modern Approach* (Second ed.). Englewood Cliffs, NJ: Prentice-Hall.

Weld, D. S., C. R. Anderson, and D. E. Smith [1998]. Extending Graphplan to handle uncertainty and sensing actions. In *Proceedings of the 15th National Conference of the American Association for Artificial Intelligence (AAAI-98)*, Madison, WI, pp. 897–904. MIT Press.

Yang, Q. [1997]. *Intelligent Planning: A decomposition and abstraction based approach*. Springer-Verlag.

Gaze Allocation During Visually Guided Manipulation

Technical Report

Jose Nunez-Varela[1], Priya A. Mani[2], B. Ravindran[2], Jeremy L.Wyatt[1]

[1]School of Computer Science
University of Birmingham

[2]Department of Computer Science and Engineering
IIT Madras

April 2011

Abstract

In this work we present principled methods for the coordination of a robot's oculomotor system with the rest of its body motor systems. The problem is to decide which physical actions to perform next and where the robot's gaze should be directed in order to gain information that is relevant to the success of its physical actions. Previous work on this problem has shown that a reward-based coordination mechanism provides an efficient solution. However, that approach does not allow the robot to move its gaze to different parts of the scene, it considers the robot to have only one motor system, and assumes that the actions have the same duration. The main contributions of our work are to extend that previous reward-based approach by making decisions about where to fixate the robot's gaze, handling multiple motor systems, and handling actions of variable duration. We compare our approach against two common baselines, random and round robin gaze allocation. We show how our method provides a more effective strategy to allocate gaze where is needed the most.

Contents

1 Introduction

1.1 Problem Definition

Real-world tasks require robots to act under uncertain and incomplete information. In this work we show how a robot should act to reduce that uncertainty by controlling its gaze in a principled way. Here, we consider a robot that has an oculomotor system and multiple motor systems. By oculomotor system we mean a system capable of moving the robot's cameras to specific fixation points (e.g. an object in the scene). These movements represent perceptual actions. Once a fixation is made, it is then possible to process the visual input to extract information. The robot's motor systems are for example, its arms, hands, legs, etc. Each motor system is capable of performing physical actions (e.g. grasping an object with the hand). The main problem is that all motor systems are running simultaneously and each of them might require the use of the oculomotor system in order to gather information relevant to the success of its physical actions. Therefore, a coordination mechanism must be defined in order to allocate the control of gaze to the motor system which most needs it.

The robot also needs to deal with the fact that cameras are limited and noisy. They are limited because it is not possible to get a full view of the world, but just a part of it, thus the robot needs to decide where to look. They are noisy due to a number of factors, e.g. resolution, illumination, quality of the camera, etc. This noise adds uncertainty to the information the robot is trying to extract.

Our proposed coordination framework is implemented using the iCub simulator [Metta et al., 2008] (Fig. 1). The iCub is a humanoid robot with multiple motor systems and an oculomotor system. To test our approach we have defined a task that consists in picking up objects from the table top and then placing them inside one of the containers, as shown in Fig. 1. Objects reachable by the right arm are placed inside the right container, and objects reachable by the left arm are placed inside the left container. Once an object is put inside a container it disappears and another object will appear on the table. Also, every 60 seconds a new object will appear on the table. The robot does not know the locations of objects and containers. The robot needs to fixate on an object/container in order to get an estimate of its true location. The only known location to the robot is the table's centre.

1.2 Related Work

Current research on visual perception has focused on active vision and attentional systems [Frintrop, 2006], where image processing techniques identify regions of interest so that the

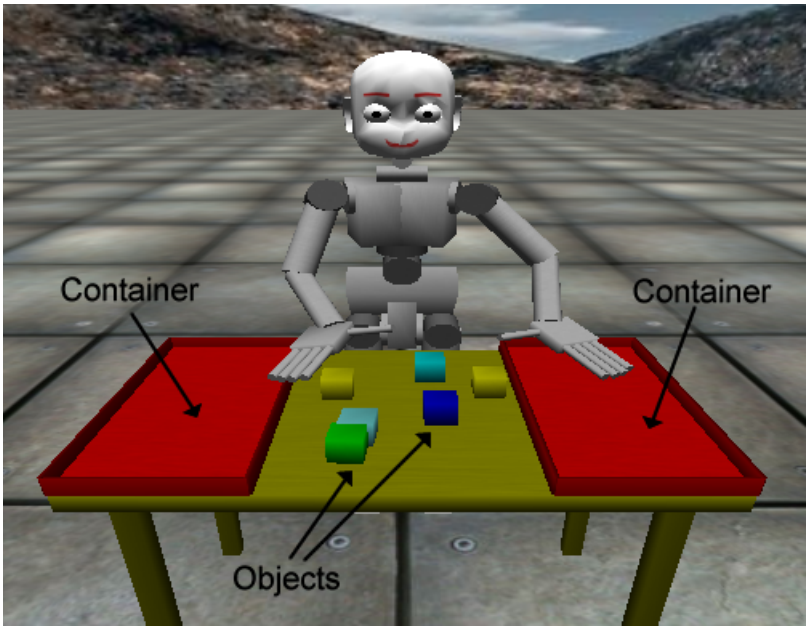


Figure 1: Snapshot of the iCub Simulator. The task is to pick up and place objects from the table top into the containers.

robot can move the camera to get different images of one object. Although these systems provide a good way to decide where to direct the gaze, they normally fail to incorporate information about the task being performed.

In [Sprague et al., 2007], Sprague and Ballard developed a reward-based perceptual coordination mechanism for a simulated human-agent. The agent is capable of performing a set of behaviours simultaneously thanks to the fact that all behaviours share the same set of actions. Each behaviour has an associated visual routine that updates the information needed by that behaviour. However, only one visual routine can be executed every time step, thus the agent has to decide which is the best visual routine to execute at any step. The key idea is to select the visual routine which will minimise the uncertainty associated to the behaviour that is likely to lose more reward. Their results show that a reward-based approach provides an effective way to coordinate perception and actions. However, there are several restrictions in their approach. First, the agent's eyes remain fixed, they do not consider the fact that eyes can move. Second, the agent is restricted to have only one motor system that works across all behaviours. Third, they assume that physical actions have the same duration.

1.3 Contributions

We follow Sprague and Ballard's idea of a reward-based approach, but we extend their approach by:

- Considering an oculomotor system capable of moving the robot's cameras.
- Considering multiple motor systems.
- Considering actions with variable duration.

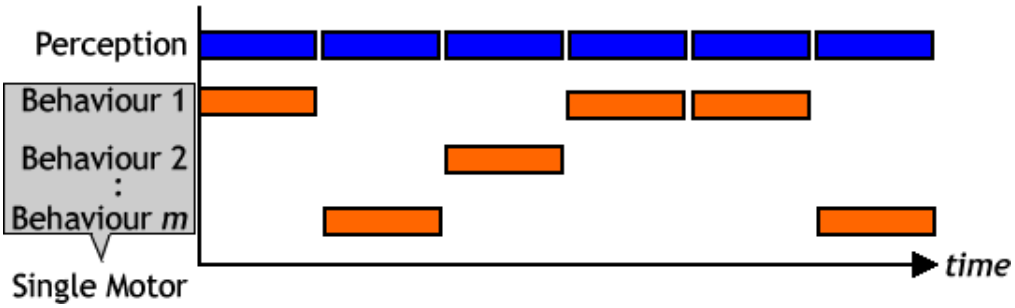


Figure 2: Sprague and Ballard's approach handles several behaviours but only one motor system with actions of same duration.

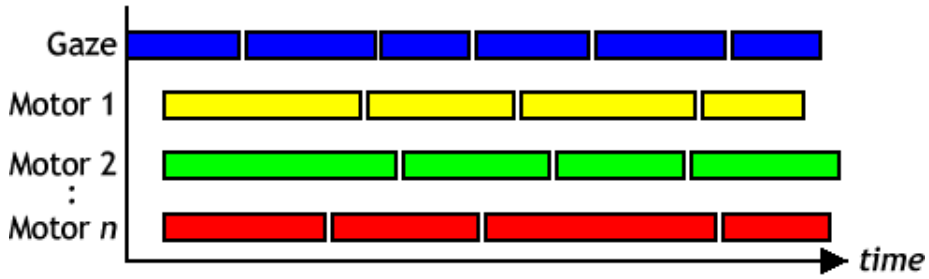


Figure 3: Our approach handles multiple motor systems with actions of variable duration.

Fig. 2 shows a graphical representation of Sprague and Ballard's system having multiple behaviours but with only one motor system and actions with the same duration. In contrast our proposed system, shown in Fig. 3, has multiple motor systems and allows actions with variable duration. Each block in the figure represents an individual action.

To summarise, our system lets the robot make three different kinds of decisions:

- Decide which motor system should take control over the oculomotor system (gaze allocation).
- Given the selected motor system, decide which perceptual action is to be executed (i.e. where to look).
- Decide which physical action to perform for each motor system.

1.4 Modelling Decision Problems

A common approach for modelling decision making problems is to use Markov decision processes (MDPs) [Puterman, 1994]. Sprague and Ballard's modelled each behaviour as an MDPs, and their task is learnt via reinforcement learning (RL) [Sutton and Barto, 1998]. During execution time, their perceptual coordination mechanism is formalised as a partially observable MDP (POMDP) [Kaelbling et al., 1998], in order to handle uncertainty. Nevertheless, a full POMDP solver is not required, only a one-step look-ahead is needed. In order to keep track of the uncertainty they use Kalman filters [Thrun et al., 2008].

The issues with MDPs are that they assume that actions have the same duration and are executed sequentially. As stated above, we want to consider actions that have variable duration to represent what really happens in real-world tasks. For instance, the time it takes to reach for an object is different to the time it takes to grasp it. Thus, instead of using MDPs we model our problems using semi-MDP (SMDP) [Puterman, 1994], which allow us to model actions with variable duration. These temporally extended actions are modelled as *options* [Sutton et al., 1999]. Options are high-level actions where it is possible to "look" inside them, since each option is composed of single-step actions.

Also, by having multiple motor systems, physical actions can be executed in parallel instead of sequentially. In order to exploit the parallelism provided by the multiple motor systems, we decompose the task and model each motor system separately, then each motor system learns its corresponding task via RL.

To handle uncertainty we also formalise the problem as a POMDP and use a one-step look-ahead algorithm. This uncertainty is represented as probability distributions. However, instead of using Kalman filters, we make use of particle filters [Thrun et al., 2008]. Kalman filters require the definition of a number of parameters that sometimes might be difficult to calculate. For instance, they require the definition of a model to predict the state of the variable being tracked. On the other hand, particle filters are non-parameterised methods that are more flexible at the moment of tracking uncertainty.

In the next section (Section 2), we describe the theory behind our proposed framework. Section 3 presents the experiments and the results obtained, and Section 4 provides some final remarks and future work.

2 Coordinating Gaze and Actions

2.1 How the System Works

Fig. 4 illustrates the interaction between the different components in our system. The system works in the following way:

1. **Learning phase:** The robot first learns via RL how to perform a particular task, keeping the learnt policy in memory. Learning occurs in a completely observable manner, i.e. perceptual actions are not involved.

2. **Execution phase:** During execution uncertainty is taken into account, thus the robot should now keep an estimate of the true state of some variable, where this estimate is represented by probability distributions. In our task, the location of objects and containers is the uncertain state variable. This uncertainty is tracked using particle filters for each object/container in the world. The information about objects/containers and its corresponding particle filter are kept in a *visual memory*.

   (a) **Physical action selection:** Is in charge of selecting physical actions for each motor system. The selected actions are sent to the corresponding motor system and are executed in parallel. Actions will likely change the state of the world and that of the robot.

   (b) **Perceptual coordination:** Allocates gaze to the motor system that most needs perception, i.e. the motor system that is more likely to lose reward if it is not given access to perception. Then it selects the best perceptual action for that motor system, where each perceptual action represents a fixation point. The perceptual action makes the oculomotor system move the cameras towards a specific part of the world.

   (c) **Image processing:** Once gaze is fixated on a particular point (e.g. an object), this module extracts information from the visual input to update the current state kept in visual memory. In our task, objects are detected and an estimate of their location is calculated, this new estimate updates its corresponding particle filter.

2.2 Learning Phase

The robot first learns the task in a completely observable manner. For our task this means that all objects' locations are known to the robot. Learning the task is achieved via reinforcement learning [Sutton and Barto, 1998]. In particular, we make use of the

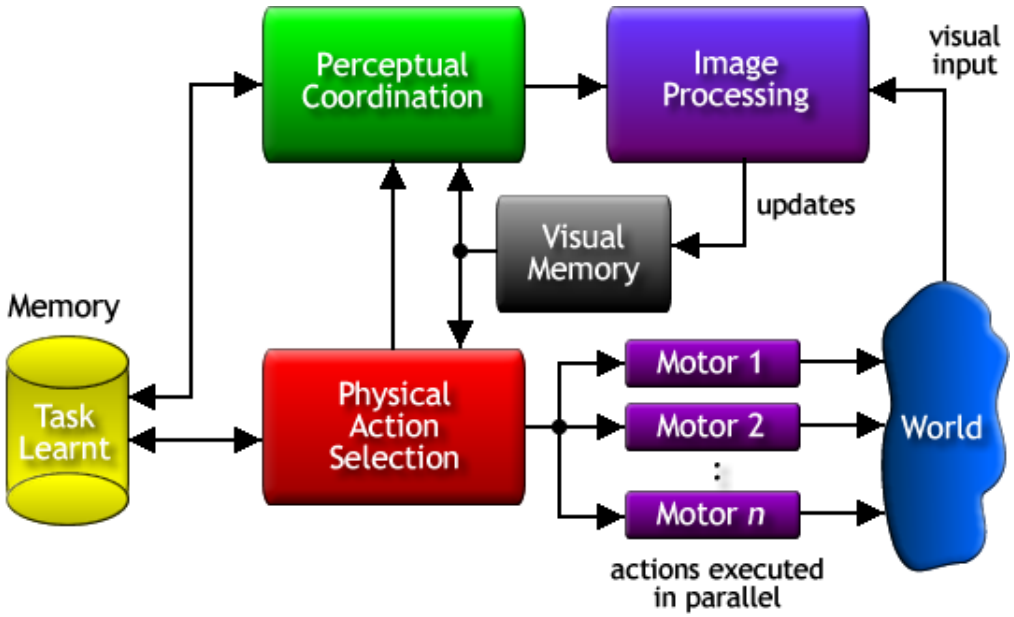


Figure 4: Interaction of the system's components.

SMDP Q-learning algorithm [Bradtke and Duff, 1995]. Recall that we model our problem as an SMDP because we are considering temporally extended actions (from now on these extended actions will be called *options* as in [Sutton et al., 1999]). Each motor system learns an independent policy that can be executed in parallel.

Each motor system $ms \in MS$, where MS is the set of motor systems, is modelled as a tuple $\langle \mathcal{S}_{ms}, \mathcal{O}_{ms}, \mathcal{T}_{ms}, \mathcal{R}_{ms} \rangle$, where $\mathcal{S}_{ms}$ is the set of states, $\mathcal{O}_{ms}$ is the set of options, $\mathcal{T}_{ms} : \mathcal{S}_{ms} \times \mathcal{O}_{ms} \times \mathcal{S}_{ms} \times \mathbb{N} \to [0, 1]$ is the transition probability distribution, where $\mathbb{N}$ is the set of natural numbers representing the time it takes to execute each option, and $\mathcal{R}_{ms} : \mathcal{S}_{ms} \times \mathcal{O}_{ms} \to \mathbb{R}_{ms}$ is the reward function. The goal of the learning phase is to learn a policy $\pi_{ms} : \mathcal{S}_{ms} \to \mathcal{O}_{ms}$, that defines a mapping from states to options.

Following [Sutton et al., 1999], each option is modelled as $\mathcal{O} = \langle \mathcal{M}, \mathcal{I}, \beta \rangle$, where $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ is an MDP, $\mathcal{I} \subseteq \mathcal{S}$ is the initiation set where the option can start, and $\beta : \mathcal{S} \to [0, 1]$ defines a termination condition. Options can be learnt or planned. However, in our case, options are defined as commands provided by the motor controllers available to the iCub [Pattacini et al., 2010].

2.3 Execution Phase

Once the task has been learnt and its policy is in the robot's memory, we incorporate uncertainty to the problem by not letting the robot know the locations of objects and containers in advance. For a given object, its location is represented as a probability distribution over the possible locations the object might be in. This probability distribution is tracked using a particle filter, thus each object/container has an associated particle

filter.

2.3.1 Physical Action Selection

During the execution phase, the robot needs to select options for each one of its motor systems. This selection has to take place based on uncertain information. In our case, options are selected based on the current belief the robot has about the locations of objects and the containers. We make use of the Q-MDP algorithm, which was formulated to find approximate solutions to POMDPs [Cassandra, 1998]. As mentioned above, we do not solve a full POMDP, we are just interested in using a one-step look-ahead strategy to decide which option to select next. For each motor system, the robot selects an option according to:

$$o_{ms} = \arg\max_{o \in \mathcal{O}_{ms}} \sum_{s \in \mathcal{S}_{ms}} bel(s) Q_{ms}(s, o), \tag{2.1}$$

where o_{ms} represents the option with the highest expected reward for motor system ms, according to the belief $bel(s)$ of being in state s, and $Q_{ms}(s, o)$ is the Q-value taken from the policy π_{ms}. Because we assume that motor systems are independent of each other, options for different motor systems can be executed in parallel without any problem.

In our case, particle filters are used to represent the belief $bel(s)$ of being in state s. They approximate this posterior via a finite number of values, each corresponding to an instance of the state space. The following equation redefines (2.1) in terms of particle filters:

$$o_{ms} = \arg\max_{o \in \mathcal{O}_{ms}} \frac{1}{M} \sum_{g} Q(g, o) weight(g), \tag{2.2}$$

where M is the number of particles, g refers to an individual particle, and $weight(g)$ defines the weight given to that particle. Each particle represents a particular location where the object might be in. The weight associated with each particle represents the belief about the location defined by the particle. The higher the weight, the more certain we are about being close to the true location.

2.3.2 Perceptual Coordination

The selection of good options depends almost completely on having the correct state information. By gathering information with sensors we can reduce this state uncertainty. But only some information needs to be known to complete each step of the task, and the robot can choose which information to gather by pointing its cameras. In our task, the

more the robot looks at an object, the more certain the robot will be about the object's location.

Since there is only one oculomotor system, then it must be shared amongst all the motor systems. This is why it is essential to have a good coordination mechanism for the selection of perceptual actions. Our coordination works in the following way:

1. We assume some visual memory about where objects and containers are relative to the robot that explicitly quantifies the state uncertainty.

2. Each object/container listed in visual memory represents a fixation point, i.e. a perceptual action p, that can be chosen by the coordination mechanism. Thus, a set $\mathcal{P}$ of perceptual actions is created.

3. For each motor system we work out the benefit for that motor system of each fixation point. We allocate the cameras to the motor system ms_E that will benefit the most. This is done by calculating:

$$ms_E = \arg \max_{ms \in MS} \{gain_{ms}\}, \tag{2.3}$$

where MS is the set of motor systems, and $gain_{ms}$ represents the gain that results if motor system ms is given access to perception. The gain of each motor system is computed as:

$$gain_{ms} = \max_{p_j \in \mathcal{P}} \{V_{ms}^{p_j}\} - \max_{o_i \in O_{ms}} \{V_{ms}^{o_i}\}, \tag{2.4}$$

where p_j is a particular perceptual action, and $\mathcal{P}$ is the set of all possible perceptual actions. The left hand side calculates the expected values $V_{ms}^{p_j}$ for motor system ms assuming perceptual action p is taken. The right hand side calculates the expected values $V_{ms}^{o_i}$ of each option when no perceptual action is executed, i.e. it calculates the value of an option with the current uncertainty. The difference between the maximum of these two values tells us how much we gain if we let this motor system to control the oculomotor system.

To calculate $V_{ms}^{o_i}$ we use the equation:

$$V_{ms}^{o_i} = \sum_{s \in \mathcal{S}_{ms}} bel(s) Q_{ms}(s, o_i), \tag{2.5}$$

which is very similar to (2.1), except that here we are calculating the value of an specific option o_i. To calculate $V_{ms}^{p_j}$ we follow:

$$V_{ms}^{p_j} = \sum_{\omega} \left[P(\omega \mid bel, p_j) \max_{o \in O_{ms}} \sum_{s \in \mathcal{S}_{ms}} bel^{\omega, p_j}(s) Q_{ms}(s, o) \right], \tag{2.6}$$

where $P(\omega \mid bel, p_j)$ is the probability of making an observation ω given the current belief bel and perceptual action p_j. $bel^{\omega,p_j}(s)$ is the belief that results assuming we have taken perceptual action p_j and ω is the observation that results from this action. The main idea is that this equation performs a one-step look-ahead to determine what the value would be if a particular perceptual action is taken.

4. Once a motor system ms_E is selected using (2.3), we choose to execute a particular perceptual action p_E such that:

$$p_E = \arg\max_{p_j \in \mathcal{P}} \left\{ V_{ms_E}^{p_j} \right\} \tag{2.7}$$

Again, since we are considering particle filters to track the uncertainty, we need to redefine (2.5) and (2.6). To calculate $V_{ms}^{o_i}$ using particle filters we follow the equation:

$$V_{ms}^{o_i} = \frac{1}{M} \sum_{g} Q_{ms}(g, o_i) weight(g), \tag{2.8}$$

again M is the number of particles, g is a particle, and $weight(g)$ is the weight of that particle. To calculate $V_{ms}^{p_j}$ we use the equation:

$$V_{ms}^{p_j} = \frac{1}{L} \sum_{l} \max_{o \in O_{ms}} \left(\frac{1}{M} \sum_{g} Q_{ms}(g, o) weight(g, l) \right), \tag{2.9}$$

where L is the number of observations, and l is a particular observation (similarly to ω in (2.6)); $weight(g, l)$ represents the weight of the particle g after having observed l. To generate these observations we randomly select a subgroup of particles from the current particle filter, and then we sample a number of observations from each of these selected particles. All these observations will form a set of size L.

Because the location of an object is distributed amongst the particles, the fixation point is calculated as the mean of the particles. Gaze movements are performed using the iCub oculomotor controller.

2.3.3 Image Processing

Once a perceptual action is chosen and executed, the robot's gaze will be fixating at some object/container. This is when visual input is read from the cameras and processed in order to make a new observation about the state of the world. At the moment we do

not perform an image processing algorithm per se; thanks to the simulator we can use a "fake" image processing routine to know which objects are currently being seen by the robot's cameras. However, it is important to note that even though the true object's location could be given by the simulator, the robot receives a noisy location to simulate the uncertainty present in the cameras and the image processing algorithms.

This noise is given by an observation model represented as a bivariate Gaussian distribution. This model was learnt offline in the following way:

- An object was placed in 15 different location on the table.
- On each location the robot performed a gaze pattern of 20 camera movements. Thus we had 20 images per location, i.e. 300 images. The gaze pattern consisted in moving the gaze across the table following a "mowing the lawn" strategy.
- Each image was processed with an object detection algorithm in order to locate the object within the image. At the end we obtained a file with 261 image coordinates[1]. Whenever the object was in the image it was detected.
- Using these image coordinates we used triangulation to determine the 3D coordinates with respect to the robot's frame.
- Figure 5 shows a plot of these 261 points after triangulation. It shows the 3D coordinates as seen from above with respect to the object's centre located at (0,0). Units are in centimetres, and the object size is 2 x 2 x 2 cm.
- Finally, these points were fit into a bivariate Gaussian distribution. Figure 6 shows the distribution, and figure 7 shows the top view. The mean is located in (0,0), where the object's centre is, and the covariance matrix is given by:

$$\Sigma = \begin{bmatrix} 0.6896 & -0.0557 \\ -0.0557 & 1.3596 \end{bmatrix}$$

In principle, objects appearing on the table are static. Nevertheless, sometimes whilst manipulating one object other objects can be hit and moved out of their original place. Therefore, the robot cannot assume that an object will remain in the same position all the time. Thus if an object that we have previously seen, is not seen again for a determined amount of time (3 seconds in our case), Gaussian noise is added to its current estimate. The same happen to the container's location, the robot needs to look the container in order to place an object.

[1]It's less than 300 because in some images the object was not present.

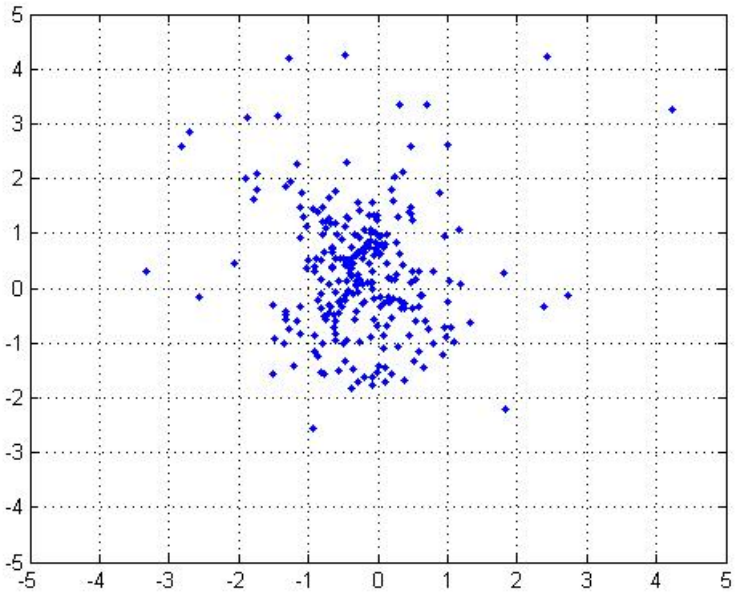


Figure 5: Raw data obtained just after image processing and triangulation. Coordinate (0,0) is where the object's centre is. Units are in centimetres.

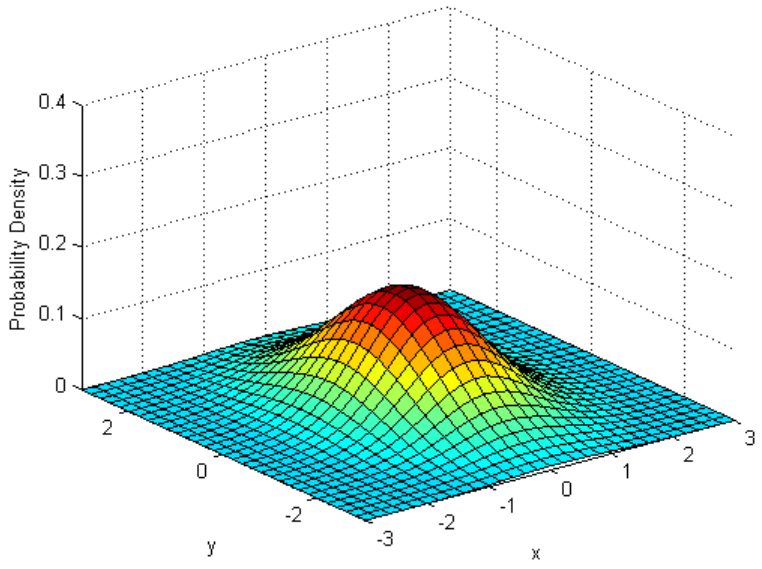


Figure 6: Bivariate Gaussian distribution obtained by fitting the points after triangulation. Units are in centimetres.

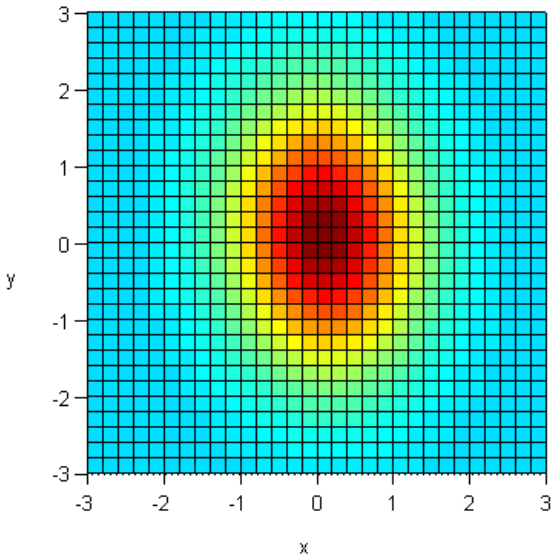


Figure 7: Top view of the same observation model. Units are in centimetres.

To illustrate how the particle filters work, figure 8 shows a top view of an object with the initial set of particles assigned to its location. Then, figure 9 shows what happens when the robot fixates on the object and a new observation occurs. Here the new observation updates the position of the particles so that the uncertainty about the object's location is reduced.

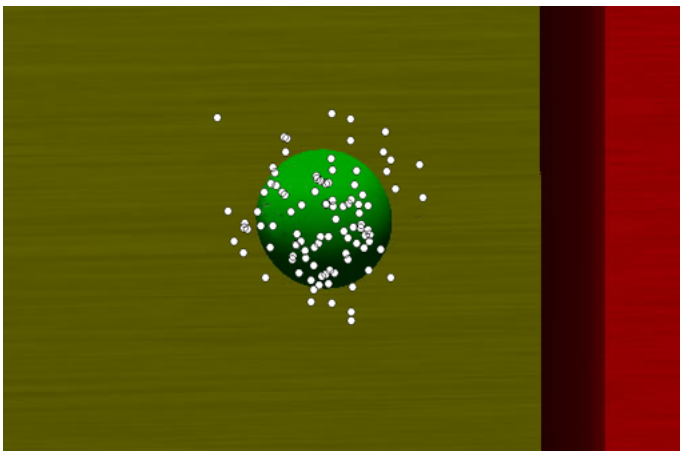


Figure 8: Top view of an object with the initial distribution of particles.

3 Experiments

The previous section formalised the problem and described the theory behind our coordination mechanism. In this section we will present some experiments and results from the implementation of our coordination mechanism. As mentioned above, the system was

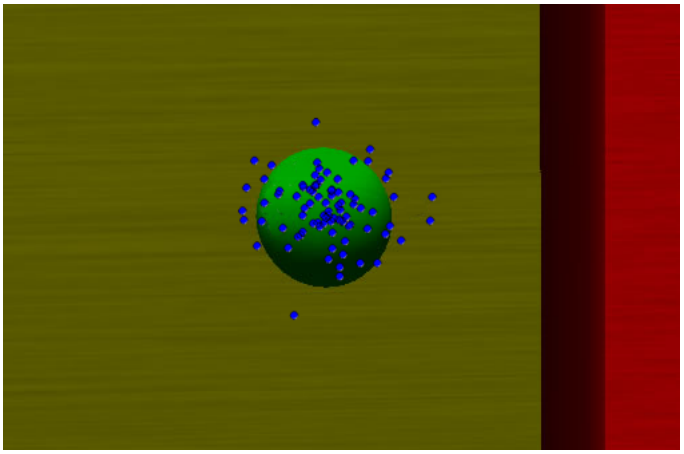


Figure 9: Top view of an object with the distribution of particles after an observation has occurred.

implemented and experiments were carried out using the iCub simulator [Metta et al., 2008][2]. The iCub simulator (see Fig. 1) is an open source research platform specifically designed for the iCub robot.

The robot has the task of picking up objects from the table top and then placing them inside one of the containers. Objects can have different features, such as type (cubes, cylinders and spheres), colour, orientation, location and size. However, we are only concerned with the location at the moment. Objects reachable by the right arm are placed inside the right container, and objects reachable by the left arm are placed inside the left container. Once an object is put inside a container it disappears and another object will appear on the table. Also, every 60 seconds a new object will appear on the table. The only known location to the robot is the table's centre.

3.1 Learning Phase

We start by modelling and learning the task. Two motor systems are considered: the right and left arm/hand. Thus we define the set of motor systems as $MS = \{right\_arm, left\_arm\}$. We define a factorised state space (which is the same for both arms: $\mathcal{S}_{right\_arm}$ and $\mathcal{S}_{left\_arm}$) as shown in Table 1.

Table 2 defines the set of options available for both arms ($\mathcal{O}_{right\_arm}$ and $\mathcal{O}_{left\_arm}$). Next to each option is the average time (in seconds) it takes to complete. These times were obtained by executing the option for 100 times and then averaging the time it took to complete. It is important to note that for option *graspObject* we use a special command, defined in the simulator, that makes the hand act like a magnet. By doing this we avoid, at least for now, the problem of controlling the individual fingers. As mentioned before, all

[2]The next step is to test our system in the real robot as well.

Table 1: Factorised state space for right and left arm

State Variable	*State Value*
armPosition	{onObject, onTable, onContainer, outside}
handStatus	{grasping, empty}
tableStatus	{objectsOnTable, empty}

Table 2: Options for right and left arms and their average time

Options	*Average time (secs)*
moveToObject	8.7
moveToTable	7.5
moveToContainer	7.3
graspObject	6.3
releaseObject	1.0
noAction	0.0

these options are performed using the controllers developed for the iCub robot [Pattacini et al., 2010].

In order to minimise the task's completion time, the reward function is defined such that the robot receives a reward of -1 unit after every decision epoch, and -3 units for every wrong option taken. Since both arms are independent, the high level policy learnt for one arm can be used for the other as well.

3.2 Results of the Execution Phase

Once a policy is learnt we can start the execution phase. To test how effective our approach is, we compared it against two perceptual strategies:

- **Random gaze allocation:** It creates a set of fixation points based on the list of objects/containers and the table's centre. Then it randomly selects one of those fixation points.
- **Round robin gaze allocation:** It creates a set of fixation points in the same way as the random strategy. The robot will go through the list fixating on each object, container and table one by one.

For each approach we performed 30 trials of 10 minutes each. At the end of each trial we counted the number of objects correctly placed inside the containers, and the number of options executed for the right and left arm.

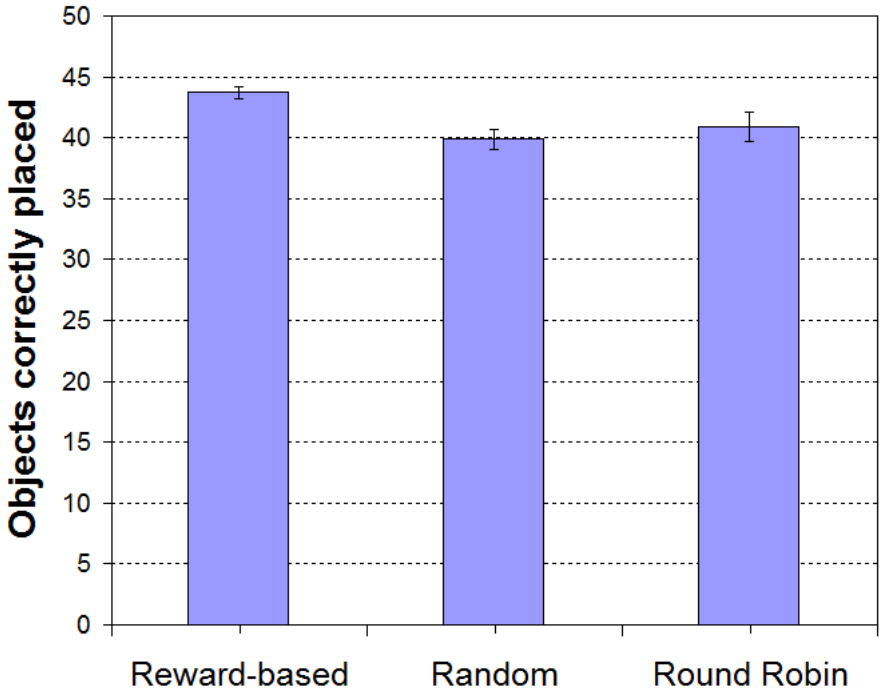


Figure 10: Comparison between the three gaze allocation strategies in terms of the average number of objects correctly placed. The error bars represent 95% confidence intervals.

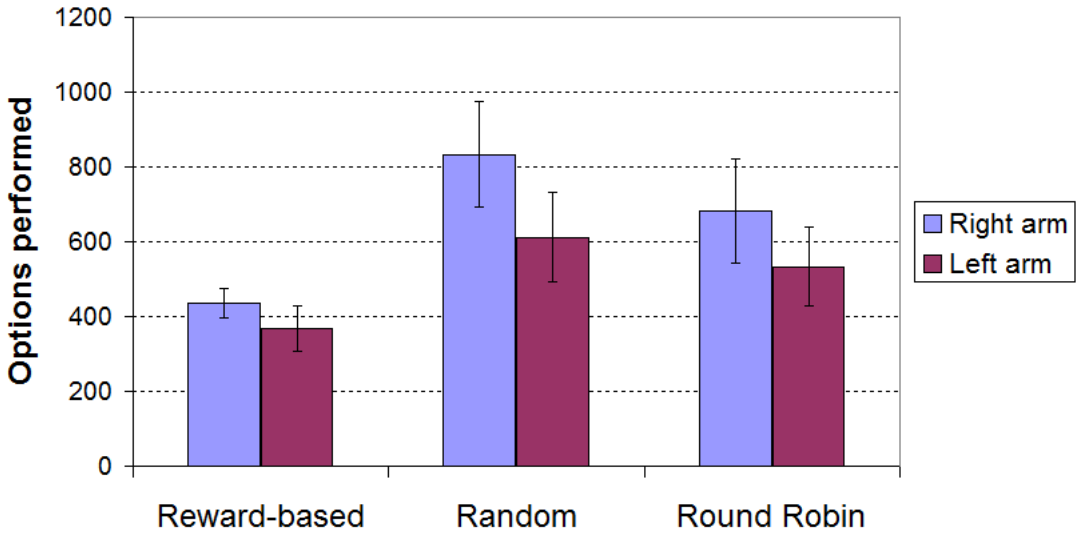


Figure 11: Comparison between the three gaze allocation strategies in terms of the average number of options performed by the right and left arms. The error bars represent 95% confidence intervals.

Fig. 10 compares the three strategies in terms of the average number of objects correctly placed inside the containers at the end of 10 minutes. The error bars in the graph represent 95% confidence intervals. Our reward-based approach is capable of placing in average almost five objects more than random, and around three objects more than round robin. It is important to note that the error bars show that our approach is fairly constant, it does not vary as much as the other two methods. This high variation results because in some runs random and round robin performed very poorly and in other runs performed well. The problem is that most of the times gaze is directed where no relevant information can be gathered[3].

Fig. 11 shows precisely the consequences of not having correct information when it is needed. The graph compares the three strategies in terms of the average number of options performed by the right and left arms during the 10 minute trials. The error bars in the graph represent 95% confidence intervals. Here there is a big difference not clearly seen in the previous graph. Random and round robin perform far more options than our approach. For example, if the arm is reaching for an object but its location is not certain, the option will fail and will keep trying until more information is gathered or it will try to reach for another object.

Furthermore, we also compared the outcomes using the unpaired T-test and the Mann-Whitney U-test. Both tests show that the results are statistically significant at 0.01 level of significance.

4 Conclusions and Future Work

The problem of how to coordinate gaze and actions in a robot was posed in terms of a reward-based decision making framework, where camera movements are selected such that they increase the expected return on the task achieved by the motor systems.

The results show that our coordination mechanism outperforms two gaze allocation strategies: random and round robin gaze allocation. Our approach is capable of placing more objects on average. It also makes the arms to execute fewer options by helping in gathering information that is relevant to the completion of the task.

Currently, we are working on several extensions to the system presented here:

- Our system assumes the motor systems to be independent of each other. We are now working on a *concurrent* coordination mechanism where motor systems interact with each other. For instance, when switching an object from one hand to the other.

[3] A video illustrating these problems can be found here: http://www.cs.bham.ac.uk/~jin803/videos/jnunez_gaze_control.wmv

- Instead of allocating the gaze to a specific motor system, in some cases gaze could also be allocated in such a way that it is useful for a subset of motor systems.
- We plan to add uncertainty to other object features, not just its location. For instance, a correct grasp depends on having information about location, orientation and type of object.
- We are also interested in implementing real visual routines during execution time. This visual processing module should be capable of extracting object's features and handling occlusions.

These extensions will enable our approach cover a broader class of problems.

5 Acknowledgements

We gratefully acknowledge support for this research under the EU FP7 IP project CogX: Cognitive Systems that Self-Understand and Self- Extend (ICT-215181), UKIERI grant SA08-031 and CONACYT studentship 179604.

References

S. Bradtke and M. Duff. Reinforcement learning methods for continuous-time markov decision problems. *Advances in Neural Information Processing Systems*, 8:393–400, 1995.

A. R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown Univ., Rhode Island, May 1998.

S. Frintrop. *VOCUS: A Visual Attention System for Object Detection and Goal-Directed Search*. Springer-Verlag, New York, NY, 2006.

L. P. Kaelbling et al. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

G. Metta et al. The icub humanoid robot: An open platform for research in embodied cognition. In *Proc. ACM 8th Workshop on Performance Metrics for Intelligent Systems*, pages 50–56, Gaithersburg, MD, USA, August 2008.

U. Pattacini et al. An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1668–1674, Taipei, Taiwan, October 2010.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, New York, NY, 1994.

N. Sprague, D. Ballard, and A. Robinson. Modeling embodied visual behaviors. *ACM Transactions on Applied Perception*, 4:–, 2007.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press Cambridge, Cambridge, MA, 1998.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press Cambridge, Cambridge, MA, 2008.