HYCAS 2009
1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control —

Preface

High-level control for Autonomous Systems (e.g. robots) is concerned with selecting the next action the system should perform. In particular this means that the system must be endowed with algorithms or schemes to take the next step towards its mission goal. The known paradigms for this action selection problem are learning, deliberation, reactive control schemes or combinations of these schemes, that is, hybrid approaches. Learning has been applied successfully to many robotics tasks. Most of the work is related to learning certain basic behaviors or skills. Examples where the high-level control strategy of robots (or agents) were successfully learned are rare. The deliberative approach for decision making of autonomous systems was successfully treated in research on Artificial Intelligence, following a top-down approach, which has severe limitations in real applications. In the reactive control paradigm the idea is that, with a combination of purely reactive action selection schemes, intelligent and goal-directed behaviors emerge, which can be seen as a bottom-up approach. These different paradigms have been known for over two decades, and in fact, in today's applications, often combinations of learning, deliberation and reactive control are deployed. Usually these combinations are used in an ad-hoc or even unconscious fashion. Although there is a number of proposed architectures and a huge body of literature, the issue of combining learning, reactive and deliberative control never has been intensively investigated. With this workshop we wish to bring together researchers from different areas who concentrate on combinations of learning, planning, and/or reactive schemes for decision making and the control of autonomous systems. The workshop is open to all members of the AI and Robotics community. We would specifically like to encourage students to participate. The questions to be addressed in this workshop are:

- How can learning, deliberation and/or reactive control be combined in a beneficial way?
- What are common representations for learning, deliberation, and/or reactive control such that the methods can benefit from each other?
- What are the challenging domains demanding for hybrid control?
- What are its successful applications?

The interesting workshop submissions fall, in general, into three different categories: (1) frameworks and architectures, (2) planning and execution, and (3) heuristic reinforcement learning; and we devoted a separate session to each of these categories at the workshop. The original works submitted to this workshop approach the particular research problem in a hybrid way. This comprises to combine deliberation with reactive control, deliberation with learning, or a combination of all three. As the number of interesting submissions exceeded the format of a 1-day workshop, we decided for a poster session to give more papers a chance for intensive discussion on the topic of hybrid high-level control for robots or agents. In the following we give a brief overview of the submissions.

Frameworks and Architectures

(Potgieter et al.) raise the question, what is needed to design a self-aware robot in terms of deliberation, reactive control, or learning techniques. While in this paper the authors position self-aware robots as a problem for hybrid control and express their understanding of self-awareness and consciousness, the paper by Bhatt (Bhatt) describes a framework which aims at bringing together logic-based robot controllers with cognitively-driven agent-control approaches. On the technical side, the aim is to combine a BDI architecture with a STRIPS planner and an IndiGolog Interpreter. Bhatt describes first experimental results with a simulated mobile humanoid gripper. (Hawes et al.) describe the PECAS architecture. In PECAS, components are structured by their function. Each of these components, called subarchitectures (SAs), run in parallel and combine deliberative and reactive control schemes. As an example the authors presents the different parallel SAs that are effective, when the robot is demanded to fetch a particular book. (Berger et al.) describe with their DAInamite Framwork a successful integration of learning, deliberation, and reactive control for a simulated robotic soccer agent. In their framework, it is possible to integrate on-line and off-line learning into the tactics of the agent without much effort. (Aaron and Admoni) combine learning, deliberation, and reactive control within their framework for hybrid dynamical cognitive agents, exploiting a common representation of dynamical intentions, and show how hebbian learning and belief-intention learning can be integrated with their approach.

Planning and Execution

(Powers and Balch) describe their hybrid robot architecture. They propose to use supervised machine learning techniques for improving the cost model which is used for the deliberation for the navigation task of a mobile robot. A reactive control layer is used for executing drive commands like *avoid obstacle* or *slow for turns*. Experimental results showing the improvement made by learning are gathered from simulations. (Guitton and Farges) address the problem, how a path planner can communicate with a task planner. They make use of a known interaction language, which also can be used to express spatial constraints as imposed by the path planner. These restrictions are communicated to the task planner, which in turn integrates these information into the planning process. (McGann et al.) present an approach to combine deliberation and reactive control for autonomous underwater vehicles. They make use of constraint-based temporal planning techniques which are combined with state estimation techniques based on hidden Markov models in order to plan and conduct marine experiments autonomously. (Müller) addresses the problem of action selection and introduces an action activation algorithm that is based on P. Maes' activation energy spreading algorithm and some of its existing extensions, while (Shafran et al.) investigates a robot's coverage problem under dead-reckoning errors. Making use of a hybrid coverage algorithm, the maximal dead-reckoning error can be bound. (Phillipsen et al.) describe a hybrid approach for a humanoid robot. Hierarchical tasks nets on the high level are combined with whole-body control mechanisms on the low level. (Rabinovich and Jennings) present with the Ensemble Action EMT algorithm an approach for on-line model calibration for their Dynamics Based Control framework.

Heuristic Reinforcement Learning

(Noglik and Pauli) propose heuristic functions for speeding up the learning behavior of a rational agent deploying reinforcement learning techniques. They show that for simple navigation problems like the Mountain Car Task the learning was sped up with a simple heuristic function making use of the Euclidean distance to the target. These results could be manifested even for a more complex 3D Mountain Car Task. (Bianchi et al.) also addresses heuristically accelerated reinforcement learning. They propose a novel class of distributed heuristically accelerated reinforcement learning algorithms. The heuristic function is an action heuristic function which indicates the importance of performing an action in a particular world state. They show that the Ant Colony Optimization Problem is a problem instance of this class of reinforcement learning problems for which a distributed heuristically accelerated approach can be used.

A. Ferrein, J. Pauli, N.T. Siebel and G. Steinbauer

July 2009

Organizing Committee

Programme Committee

Contents

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Self-Aware Robots – What do we need from Learning, Deliberation, and Reactive Control? *

A Ferrein, A Potgieter
Robotics and Agents Lab
University of Cape Town
alexander.ferrein@uct.ac.za
anet.potgieter@uct.ac.za

G Steinbauer
Inst. for Software Technology
Graz University of Technology
steinbauer@ist.tugraz.at

Abstract

Self-awareness is an important property of intelligent autonomous robots. There are many different understandings of what self-awareness means to an autonomous system. The knowledge representation community understands self-awareness as having the property to do introspection and reasoning about oneself, while the computational intelligence community understands the ability of being situation-aware and adaptive as self-awareness. In this position paper we want to discuss first ideas how both worlds can be brought together in order to design a self-aware mobile robot system. We propose a self-aware system architecture, which covers the classical tasks such as plan generation, plan monitoring, and plan execution. In our proposal, we deploy reasoning techniques for plan generation, model-based diagnosis for the execution monitoring, and complex adaptive systems theory for the execution component. Our interest in the workshop is to discuss what techniques needs to be applied to design a self-aware system.

1 Introduction

Self-awareness is an important property of intelligent autonomous robots. Intelligent means here, that the robot employs some form of reasonable behavior to fulfill its given goals. Why does the robot furthermore need to be self-aware? If the robot understands its own abilities, it could judge the appropriateness of its own actions. Consider a domestic robot that is interacting with its human masters. The robot should be aware that it has to drive more carefully and slowly, when it is delivering a cup of coffee and the cup is in the gripper of the robot. Self-awareness is the ability of judging the appropriateness of own actions and their effects in a given context or situation, and seems inherently connected with acting in an intelligent way.

The term self-awareness, on the one hand, is strongly connected with the term consciousness. As the topic of consciousness (even of machines) is topic for ongoing research in neuroscience and philosophy, a coherent definition still needs to be found. Besides these classical fields, also the Knowledge Representation and Reasoning (KR&R) community has a strong interest in self-aware robots. In 2004, a DARPA workshop addressed Self-aware Computer Systems [McCarthy and Chaundhri, 2004]. Amir et al. report on the result of this workshop in [Amir *et al.*, 2006], where they distinguish between self-awareness in a social-agent and in a monitoring-executive sense. In [Schubert, 2005] Schubert discusses some requirements for KR&R connected with self-awareness such as the requirement for a logical framework, which is able to represent events and situations and has some means for auto-epistemic inference.

On the other hand, in the analysis of complex adaptive systems, self-awareness is an inherent property of an emergent system. Here, self-awareness of an agent includes the situation-awareness of the robot, which comes into play, when the robot integrates sensor values into its internal model. All complex adaptive systems exhibit adaptive self-awareness, which is described in [Mitchell, 2005] as having information about the global state of the system, which feeds back to adaptively control the actions of the system's low-level components. According to Mitchell, the information about the global state is distributed and statistical in nature, and thus is difficult for an observer to understand. In a complex adaptive system, information about the global system state are emergent phenomena, which are instances of some emergent higher-order structure that may be explained by the lower-level dynamics generating the collective behavior [Baas and Emmeche, 1997]. Baas refer to these structures as hyperstructures or "emergent explanations", which constitute the internal model of the complex adaptive system. These hyperstructures are used for explanation and understanding.

The former definition of self-awareness can be seen as a high-level top-down view, while the latter represents a low-level bottom-up approach of self-awareness. From the high-level view, the missing part is the ability to adapt its model through learning, the low-level view is lacking the ability for full introspection, as the hyperstructures do not offer any means for reasoning. Our goal for achieving self-awareness is to marry both views and provide a coherent adaptive self-model, which can be used by a deliberation mechanism and that can evolve to accommodate emergent knowledge provided by the adaptive low-level behavior engine. We propose

*A. Ferrein is currently funded by a grant of the Alexander von Humboldt foundation. We would like to thank the reviewers for their helpful comments.

a system architecture which integrates both, the possibility to do high-level reasoning as well as employing complex adaptive systems on the behavior side. For checking discrepancies between the current model and the feed-back from the environment, we want to follow a model-based diagnosis approach. Hence, a diagnosis engine is the third building block in our architecture.

We are interested in discussing possible techniques for combining both worlds at the HYCAS workshop. What kind of techniques from the field of computational learning, KR&R, and reactive control are needed to design a self-aware robot system? In the next section, we review several different understandings of self-awareness and substantiate our understanding of the term. We derive the implications for a robot system architecture, which we present in Section 3. In Section 4 we propose which method we are going to use to implement the self-aware system. We conclude with Section 5.

2 Disambiguation: Consciousness and Self-awareness

We follow two separate lines to approach our understanding of self-awareness. One is a KR&R view and describes which properties a logic-based self-aware robot needs to meet, while the other illuminates the term self-awareness from the Complexity Theory side.

In [Holland, 2003], Holland collects a number of articles about the topic of machine consciousness and gives an overview of different streams of research in that field. For example, [Aleksander and Dunmall, 2003] define five axioms, or better, properties, which a conscious system has to fulfill. These are *depiction*, *imagination*, *attention*, *planning*, and *emotion*. While we are aiming to have the properties depiction, imagination, attention, and planning, we are —with our understanding of self-awareness— aiming at a more restricted form of consciousness. In contrast, for Holland and Goodman [Holland and Goodman, 2003] it is sufficient to have a control system with a sophisticated internal model. In the context of self-awareness, [Amir *et al.*, 2006] mentions three different notions: (1) explicit self-awareness, (2) self-monitoring, and (3) self-explanation. The first form is the strongest and presupposes a complete self-model that represents knowledge about itself including knowledge about the own abilities, the own knowledge and intentions, but also about the current situation. Self-monitoring means to watchdog the internal processes, for instance just like an operating system does and integrating new regularities into the self-model. Finally there is the notion of self-explanation. Here, the agent should be able to recount or justify its actions and inferences. [Amir *et al.*, 2006] moreover distinguish between two different flavors of self-awareness. The one sees the robot as a social subject in an agent community and self-awareness then means that the robot is aware of being one subject in a community, while the second one sees the agent in a monitoring-executive sense. Here, the robot or agent is regarded as an isolated entity with the ability to perform introspection. In [Gamez, 2008] Gamez gives a comprehensive overview of the field of machine consciousness in the various facets. For a more detailed overview, we refer to [Gamez, 2008]. Another concise overview of the topic can be found in [van Zon, 2006].

In the bottom-up approach to self-awareness, consciousness and self-awareness can be described by using complexity theory. Complexity theory is about complex adaptive systems, all sharing some fundamental principles. Examples in nature include the brain, the immune system, swarms and crowds. Each of these systems is comprised of many "agents" all acting in parallel, constantly interacting with each other and the environment. As the system increases in complexity, new behaviors emerge that are not evident from local interactions amongst agents —leading to the phenomenon of the whole being more than the sum of the parts. Complexity theory can be described as the science of emergence. Collectively agents in a complex adaptive system learn from experience in order to adapt to the dynamically changing environment. In the brain, the agents are nerve cells, and consciousness and self-awareness are emergent phenomena that cannot be described by static knowledge representation structures, as is currently used in top-down approaches. All complex adaptive systems exhibit adaptive self-awareness, which is described in [Mitchell, 2005] as having information about the global state of the system, which feeds back to adaptively control the actions of the system's low-level components. This information about the global state is distributed and statistical in nature, and thus is difficult for observers to interpret. However, the system's components are able, collectively, to use this information in such a way that the entire system appears to have a coherent and useful sense of its own state.

Emergent phenomena are instances of some emergent higher-order structure that may be explained by the lower-level dynamics generating the collective behavior [Baas and Emmeche, 1997]. The set of hyperstructures in a complex adaptive system constitute the internal model of such a system. All complex adaptive systems maintain internal models [Holland, 1995]. Emergence occurs as soon as the regularities identified in the input stream deviate from what is expected from the internal model maintained by the complex adaptive system. Emergence can usually be explained completely, once the interactions between the system components are taken into account [Minsky, 1988]. The internal model, consisting of hyperstructures, facilitates the explanation of emergence, and thus an understanding of cause-effect relationships between local situations and emergent global behaviors.

3 A Rough Architecture Sketch

From the above disambiguation and the different facets of self-awareness, we derive our understanding of self-awareness. For an autonomous robot, we see self-awareness in a *monitoring-executive* sense, rather than in the social-agent sense as described in [Amir *et al.*, 2006]. Moreover, we are not going towards consciousness in a broad philosophical sense, but want to build a *self-sustainable system* that is able to detect flaws in the behavior execution and is able to take appropriate counter measures. Additionally, we want to have an adaptable system. Hence for now, we do not aim at build-

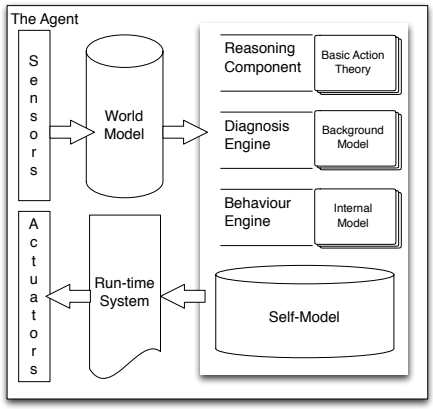


Figure 1: Architecture Overview

ing a robot system which is able to converse with humans in a smart way.

The implications for the high-level cognitive architecture according to [Amir *et al.*, 2006] are that "*a self-aware system must have sensors, effectors, memory (including representation of state), conflict detection and handling, reasoning, learning, goal setting, and an explicit awareness of any assumption. The system should be reactive, deliberative, and reflective*". The requirements for a self-sustainable, self-aware system imply in a way the different system components. Figure 1 sketches our target system architecture. It breaks down into a plan-generation, a plan-execution, a plan-monitoring component, and the self model. All these components are equipped with their own background models, common for them, though, is a self model which holds facts about the robot itself. From the sensory information, a world model is constructed which stores and derives all the relevant information needed by the action selection components.

The action selection is three-fold. On the one hand, there is a deliberative component which will make plans for future courses of actions. It makes use of a so-called basic action theory, a formal theory which describes the effects of performing a particular action in the world. These plans and actions will be passed to the behavior engine, which in turn will send it to the run-time system, which is executing them.

The monitoring tasks is done by an explicit diagnosis engine equipped with model-based reasoning techniques. The diagnosis engine uses models about the robot itself, its task and the world around it, to do a kind of self reflecting. The models and among them the self-model describe what the agent expects from itself, from actions, from plans and about the world. Moreover, the models specify the desired behaviors of sub-systems of the robot. We use consistency-based reasoning techniques. The diagnosis engine uses these models together with observations to derive if there is a discrepancy between the actual world and the expected world. This is similar to how a human reflects about a situation and detects if things evolved not as expected. The so generated feedback can be used in other components of the proposed architecture.

Next, there is a behavior engine, that is a complex adaptive system that mines regularities from the input stream, updates the hyperstructures in the internal model and the self model with emergent patterns that will be used by the diagnosis system and deliberative system to feedback the appropriate commands to the low-level behavior components. The behavior engine is based on the BaBe Adaptive Agent Architecture proposed and developed by Potgieter [Potgieter, 2004]. Collector agents feeds the input stream to the learning agents that mines regularities from the input stream and updates the hyperstructures in the internal model of the behavior engine with emergent patterns. The reasoning engine queries the reasoning agencies to determine the best actions to take given the global state of the robot.

Finally, there is the self model. It shall store all the information regarding the robot itself. We feel the need to separate knowledge about the robots inner workings from it operational knowledge, for example how to pickup the coffee cup. The action, how to grab the cup should be stored in the basic action theory, while, for example, the fact that lifting the cup will consume battery power should go to the self model. It should also contain that the robot has to drive more carefully. In that sense the self model interacts with the high-level decision making. There are also intersections between the knowledge of the diagnosis engine and the deliberation. The diagnosis engine will have a rather complete knowledge of the workings of the robot, though it needs to maintain some generic knowledge of the world, which it stores in its background knowledge. Both, the diagnosis engine and the deliberation will make use of the common knowledge which is stored in the self model. One of the open challenges will be how the adaptive behavior engine can share the common knowledge about the robot itself. We will sketch our ideas how this could be done in the next section.

4 Learning, Deliberation, or Reactive Control?

In the following, we sketch our first ideas, what might be needed from *learning, deliberation, or reactive control*. It is in a way driven from the scientific background of the authors, and we want to discuss our view on self-aware robots with researchers with different backgrounds at the HYCAS workshop.

4.1 The Self Model

The robot should be enabled to derive facts about itself. As an example, consider that the robot is to deliver the cup of coffee. A fact it needs to know is that driving from A to B consumes battery power. Holding the cup in the gripper costs additional power. It also should be aware that it should drive more carefully than without holding a cup. This distinguishes between the basic action theory, the diagnosis background model, and the self-model. The delivery-action's effects on the world will be kept in the basic action theory, while the effects of this action on the robot itself go to the self-model, e.g. that this action consumes power. In the same way, the differences between the background diagnosis model can be separated from the self model. How the robot works internally, i.e. its component model, is stored in the self model while diagnosis-related knowledge about the world is kept in

the background diagnosis model. Similarly, in the internal model the Behavior Engine will update its hyperstructures, for instance Bayesian nets, based on new sensor readings. At certain time instances, parts of the self model are updated based on the updated internal model of the Behavior Engine. The challenge here is to ground the symbols of the internal model and the self model appropriately. The world model, on the other side, stores an actual snapshot of the world based on the available sensors. Moreover, it provides aggregated information which were derived from the sensor readings.

Given the envisioned models, the self model should be encoded in a logical language. As we yet are not sure about the needed expressiveness, assume a first order representation for the time being. One of the challenges we have to cope with for future work is how the robot can make use of this kind of information to, for instance, improve or restrict the planning component. With relocating robot-specific information to the self model, the action theories of the robot could be designed in a more general way, yielding higher portability for the reasoning component. Moreover, the diagnosis engine can use such a self model and will benefit from it. The self model or parts of it can be seen as a description of the proper behavior of parts of the robot in the sense of model-based reasoning. These parts may comprise hardware or software or both, e.g., the locomotion system of the robot. Moreover, it can be a description about the expectations the robot has about the outcome of action, plans or goals.

As an example how the self-model connects to the behavior engine, consider that the robot might have learned over time the probabilistic cause-effect relationships between the amount of battery power that is consumed by different actions, for instance, holding the cup in the gripper. The learning agencies furthermore could have mined the cause-effect relationships between the speed of driving an the spilling of coffee. The reasoning engine will determine the state of the robot from the self-model, and determine that the robot is holding a cup of coffee in its gripper. It will then pose a What-If query to the reasoning agencies in the form of: "What is the type of driving that should be followed given that the robot is holding a cup of coffee?" From the internal model, the reasoning agencies, through inference, will determine that the best action to follow is careful driving. The action models of careful driving will be fed back to the competence agents that will deliver it to the actuators and that will interact with the diagnosis engine to make sure that careful driving is achieved.

4.2 The Reasoning Component

The reasoning component will be implemented as a high-level control program in the agent programming language Readylog [Ferrein and Lakemeyer, 2008]. Readylog is a language from the Golog language family, and the semantics of its constructs is based on situation calculus formulas [McCarthy, 1963; Reiter, 2001]. Readylog was designed to especially fit the needs of real-time dynamic application domains by integrating many different language constructs in one coherent language framework which comes with an effective run-time system. Planning in Readylog can done by decision-theoretic planning, though external planning systems can be interfaced. Before, Readylog was used to design the high-level control of agents and robots in robotic soccer competitions. The high-level Readylog controller is equipped with a model of all the basic behaviors the robot can perform. These are called primitive actions. For each of these primitive actions, a situation calculus formalization exists describing the effects of the particular action to the world. These form, together with precondition axioms as well as further background axioms, the basic action theory. Note, that the basic action theory contains knowledge of the world, not about the internal workings of the robot. Knowledge about the robot itself is stored in the self model and is conceptually kept separated from the basic action theory.

4.3 Diagnosis Engine

The diagnosis engine uses methods from model-based reasoning. It is based on the diagnosis ideas of Reiter [Reiter, 1987]. It is an consistency-based approach. The idea is that there is a model which describes the desired behavior of a component or system or the expected outcome of an action or plan. The engine uses this model together with actual observations to detect discrepancies between the expected and the real output or outcome. Such a detected discrepancy is a sign that something went wrong in the robot or around it. Regarding the application area of the diagnosis one can use qualitative, quantitative or combined model. The choice depends on the nature of the monitored system. We used Horn clauses as qualitative models to describe and diagnosis control software for autonomous robots [Steinbauer and Wotawa, 2005]. We used hybrid automata as combined models to handle faults in the drive hardware of a mobile robot [Hofbaur *et al.*, 2007]. Based on the ideas of Reiter the diagnosis engine is not only able to detect discrepancies. It is also able to deliver an explanation for the detected discrepancy, e.g. a broken part in the robot's hardware. Such information about the root cause was already used in [Hofbaur *et al.*, 2007] for a online reconfiguration of a drive controller in order to cope with faults in the drive hardware of a robot. In the proposed architecture the diagnosis engine is used as critic mechanism for different parts of the robot. That parts can be more physical like the hardware as pointed out above, the actions executed by the behavior engine [Steinbauer and Wotawa, 2008] and also parts of the deliberative decision making component. Once the architecture detects a discrepancy between the world and its expectations such a feedback can be used to adapt or optimize parts of the system.

4.4 Behavior Engine

The behavior engine will be a complex adaptive system to be implemented using the BaBe adaptive agent architecture. The BaBe agent architecture is a commercial agent architecture described in [Potgieter, 2004]. The BaBe agents are simple, resource-constrained agents, interacting locally through the environment, and coordinating their behavior through the environment. It consists of four types of simple agents, namely collector agents, learning agents, reasoning agents and competence agents. The learning agents are grouped into agencies according to data-groups provided by the collector agents. These agencies incrementally learn by mining patterns from the data groups. These patterns are stored in an internal

model, which is used by the reasoning agencies to reason about the current context of the environment, of which the outcome is used by the competence agencies to activate the appropriate behaviors suited to the current context of the environment. The BaBe agents exchange information through a shared environment. The collector agents extract data groups from different sensors and data sources, distributed through the environment. The learning agents collectively mine patterns from these data groups using distributed learning algorithms such as Bayesian learning, genetic algorithms, neural networks and leave patterns in the internal model that will be used by the deliberation and diagnosis engines to determine their subsequent actions or behaviors of the competence agents. The reasoning agents collectively reason about the environmental context provided by the deliberation engine using Bayesian belief propagation.

4.5 Research Questions

One of the major research challenges here is the needed unification of models from different parts of the system. The idea is that we have a unique description language for the self model which can be used by the reasoning component, the diagnosis engine and the behavior engine. That would be the ultimate outcome if we are able to describe the different aspects of the robot like actions, behaviors, controls in one self model usable by all components. So far we have very different languages like the situation calculus for the deliberation, horn clauses or hybrid automata for the diagnosis on the one hand, and Bayes nets for the execution, on the other hand. Another important question is what we have to model and on what level of abstraction in order to be able to form a self-ware system. Such questions frequently arise if one deals with knowledge-based agent and as far as we know there is no formal methodology nor best practice examples. Finally, we have to investigate how such a self-aware system can be applied to an autonomous robot with very limited resources in terms of computational power and memory like the humanoid bi-ped robot platform Nao from the French company Aldebaran. For sure there are additional algorithms like knowledge compilation and caching necessary beside the classical reasoning techniques in order to run such a system.

5 Conclusion

In this paper we presented some of our thoughts regarding the topic of self-aware robot systems. We aim at designing a self-sustainable, self-aware system in a monitoring-execution sense. Hence, our self-aware robot need to be able to reason about itself, needs conflict detection and handling, as well as reasoning and learning abilities. Taking this together, we propose a system with a logic-based high-level reasoning component, a diagnosis engine working on models formalized in Horn Logic, and a behavior engine, which takes observation of the world to update and adapt the internal model of the robot. At the workshop, we want to discuss our ideas about the different components and the methods we want to deploy for implementing this architecture on real robots.

References

[Aleksander and Dunmall, 2003] Igor Aleksander and Barry Dunmall. Axioms and tests for the presence of minimal consciousness in agents. *Journal of Consciousness Studies*, 10(4), 2003.

[Amir *et al.*, 2006] Eyal Amir, Michael Anderson, and Vinay Chaudhri. Report on the 2004 darpa workshop on self-aware computer systems. Technical report, SRI International, 2006.

[Baas and Emmeche, 1997] N. A Baas and C. Emmeche. On emergence and explanation. *Intellectica*, 25:67–83, 1997.

[Ferrein and Lakemeyer, 2008] Alexander Ferrein and Gerhard Lakemeyer. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems*, 56(11):980–991, 2008.

[Gamez, 2008] David Gamez. Progress in machine consciousness. *Consciousness and Cognition*, 17(3):887 – 910, 2008.

[Hofbaur *et al.*, 2007] Mathias Brandstötter Michael Hofbaur, Gerald Steinbauer, and Franz Wotawa. Model-based fault diagnosis and reconfiguration of robot drives. In *Proc. IROS-07*, San Diego, CA, USA, 2007.

[Holland and Goodman, 2003] Owen Holland and Russell B. Goodman. Robots with internal models: A route to machine consciousness? *Journal of Consciousness Studies*, 10(4):77–109, 2003.

[Holland, 1995] J. H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Massachusetts :Addison-Wesley Publishing Company Inc, 1995.

[Holland, 2003] Owen Holland, editor. *Machine Consciousness*. Imprint Academic, 2003.

[McCarthy and Chaundhri, 2004] John McCarthy and Vinay Chaundhri. DARPA Workshop on Self Aware Computer Systems, 2004.

[McCarthy, 1963] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963.

[Minsky, 1988] M. Minsky. *The Society of Mind (First Touchstone ed.)*. New York: Simon and Schuster, 1988.

[Mitchell, 2005] Melanie Mitchell. Self-awareness and control in decentralized systems. In *Working Papers of the 2005 AAAI Spring Symposium on Metacognition in Computation*. AAAI Press, 2005.

[Potgieter, 2004] Anet Potgieter. *The Engineering of Emergence in a Complex Adaptive System*. PhD Thesis, University of Pretoria, South Africa, Supervised by Judith Bishop, 2004.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Reiter, 2001] R. Reiter. *Knowledge in Action*. MIT Press, 2001.

[Schubert, 2005] Lenhart K. Schubert. Some kr&r requirements for self-awareness. In *2005 AAAI Spring Symposium*, pages 106–113. AAAI Press, 2005.

[Steinbauer and Wotawa, 2005] Gerald Steinbauer and Franz Wotawa. Detecting and locating faults in the control software of autonomous mobile robots. In *Proc. IJCAI-05*, Edinburgh, UK, 2005.

[Steinbauer and Wotawa, 2008] Gerald Steinbauer and Franz Wotawa. Enhancing plan execution in dynamic domains using model-based reasoning. In *Proc. ICIRA*, pages 510–519, 2008.

[van Zon, 2006] Kees van Zon. An introduction to machine consciousness. In *Intelligent Algorithms in Ambient and Biomedical Computing*, pages 57–70. Springer Verlag, 2006.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

TOWARD AN EXPERIMENTAL COGNITIVE ROBOTICS FRAMEWORK

Mehul Bhatt

SFB/TR 8 Spatial Cognition
Universität Bremen, Germany
bhatt@informatik.uni-bremen.de

ABSTRACT

We position our experimental framework for cognitive robotics that is aimed at integrating logic-based and cognitively-driven agent-control approaches, qualitative models of space and the ability to apply these in the form of planning, explanation and simulation in a wide-range of robotic-control platforms and simulation environments. In addition to its primary experimental function, the research proposed herein also has a utility toward pedagogical purposes. We present the overall vision of the project, and discuss ongoing work and present capabilities.

KEYWORDS: reasoning about actions and change, qualitative spatial reasoning, dynamic spatial systems, control, experimental robotics, simulation

1 INTRODUCTION

Research in the field of Reasoning about Actions and Change (RAC), also and increasingly being referred to as Cognitive Robotics, has considerably matured [Levesque and Lakemeyer, 2007]. Over the last decade, some of the theoretical work and the resulting formalisms for representing and reasoning about dynamic domains have evolved into practically applicable high-level agent control languages, the most prominent examples here being the situation calculus based GOLOG [Levesque et al., 1997] family of languages, e.g., CCGOLOG [Grosskreutz and Lakemeyer, 2000], CONGOLOG [De Giacomo et al., 2000], INDIGOLOG [Giacomo and Levesque, 1999], which is an incremental deterministic version of CONGOLOG, and the fluent calculus based language FLUX [Thielscher, 2005]. Differences in the theoretical underpinnings notwithstanding, a common feature of all these languages is the availability of imperative programming style constructs for the domain of robotics/agent-control, i.e., statement in the program correspond to actions, events and properties of the world in which an agent is operating. Parallel to the development in the area of reasoning about actions and change, the field of Qualitative Spatial Reasoning (QSR) has emerged as a sub-division in its own right within knowledge representation [Cohn and Hazarika, 2001]. Research in QSR has focused on the construction of formal methods (i.e., qualitative spatial calculi) for spatial modelling and reasoning. The scope of QSR, at least in so far as the context of qualitative spatial calculi is concerned, has been restricted to representational modes for spatial abstraction and reasoning. Major developments in this regard include: (a) the development of spatial calculi that are representative of distinct spatial domains, (b) constraint-based techniques for ensuring the global consistency of spatial information and (c) the application of conceptual-neighborhoods [Freksa, 1991] for dealing with continuous change and time. Similarly, there have also been considerable advances in the benchmarking of the computational aspects of the planning domain and de facto standardization of domain description languages in the form of the PLANNING DOMAIN DEFINITION LANGUAGE (PDDL) [McDermott et al., 1998] and related initiatives. Recent work even indicates a cross-over of results from the planning domain to the cognitive robotics area. For instance, the work by Claßen et al. [2007b,a] combines reasoning using the GOLOG language with modern PDDL planners by the embedding of state-of-art planning systems within the former. The main objective of this line of approach is that the power of modern efficient planners be exploited whilst preserving the overalll representational semantics of the situation calculus formalism that underlies the GOLOG language.

In this paper, we position our ongoing work toward the development of a framework for cognitive robotics that brings together logic-based and cognitively-driven agent-control approaches in an experimental manner. The proposed framework is designed to integrate diverse control calculi based on mathematical logic, qualitative models of space and the ability to apply these – in the form of spatial planning, explanation and simulation – for dynamic spatial modelling with a wide-range of robotic-control platforms and simulation environments. Indeed, the framework is driven by the need for a workbench that seamlessly brings together different control techniques, both logic-based and otherwise, and a generic (based) domain-description language, and qualitative spatial calculi under one unifying, experimental framework. The main objective here being that it should be possible for a domain-modeller to specify the physics of a particular domain once and exploit more than one control approach thereafter, without the need to dwell on the details of any of the available control approaches or qualitative spatial calculi. In addition, it is also required that the envisaged framework provide easy integration with existing low-level control apparatus such as robot control and simulation interfaces that

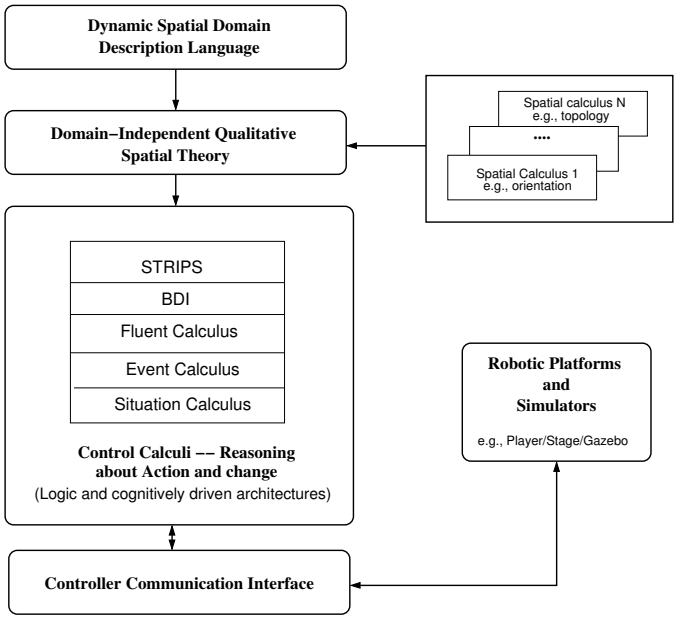


Figure 1: Overview of the Framework

exist in the open-source domain, a prime example of such an environment being the PLAYER/STAGE/GAZEBO project. We envisage that such a framework that supports easy experimentation with different control techniques, provides general modes of spatial information representation and reasoning, and additionally seamlessly integrates low-level control apparatus would, in addition to serving it primary experimental function, also be useful for pedagogical purposes at the tertiary-level.

Section 2 provides a brief overview of the proposed framework and section 3 positions the work in progress in terms of its implementation and other technical details. In section 4, we briefly discuss the immediate directions of the ongoing work.

2 OVERVIEW OF THE FRAMEWORK

The primary aim of the framework is to provide a workbench of different control approaches that may be used '*independently*' for experiments in cognitive robotics. Toward this end, the framework consists of four key components. In (C1–C4) in the following, a conceptual overview of these components, as they are presently envisioned, and the main motivations thereof are presented with reference to the schematic in Fig. 1:

C1. UNIFORM DOMAIN DESCRIPTION

A basic requirement within the framework is that it should be possible for modellers to specify the domain theory of their particular scenario, i.e., the underlying physics of the domain, using an uniform representation medium that is independent of the control apparatus that is being utilized. Such a medium requires a unified ontological view that transcends beyond any particular control calculus or non-logical control approach that is available within the framework. Among other things, what is required is that key ontological aspects pertaining to actions, events, effects, fluents and conditions need to be integrated in an ontology that may be utilized by the modeller of a domain. Such an ontology will facilitate the specification of a dynamic domain in a manner that is independent of the precise control mechanism (available within the framework) that is utilized as the basis of modelling and reasoning about change. Consequently, it is also implied that such a generic domain description is usable across all control (or reasoning) approaches that are available within the experimental framework. Although the issue of ontology construction is quite orthogonal to the issue of the precise language or mechanism to be used for instantiating it whilst modelling a domain, we consider the semantics of the basic versions of the PDDL language to be rich enough to cover most general scenarios. However, the use of PDDL as a means for uniform domain description within our framework is a topic of ongoing investigation.

C2. MULTIPLE CONTROL APPROACHES

As aforementioned, the primary aim of the framework is to provide a suite of different control approaches that may be used for representing and reasoning about dynamic environments. The suite of control approaches available with the framework also constitutes the most important (functional) component of the overalll experimental framework. It consists of a collection of several different formal techniques, both logic-based and cognitively-driven models, that can be used as control mechanisms in robotic domains or be used to reason about changing spatial environments in general. Control approaches based on the following formalisms will be available for use in an *independent* manner within the proposed framework:

1. A basic STRIPS like planning system [Fikes and Nilsson, 1990]
2. Belief-Desire-Intention (BDI) Approach [Bratman, 1987]
3. Event calculus [Kowalski and Sergot, 1986]
4. Fluent calculus [Thielscher, 1998]
5. Situation calculus [McCarthy and Hayes, 1969]

Several high-level languages that are directly based on the above mentioned formal approaches are already available, e.g., the GOLOG family of languages based on the situation calculus [Levesque et al., 1997, Lakemeyer and Grosskreutz, 2001], FLUX – the fluent calculus based language [Thielscher, 2005] and a discrete event calculus based reasoner [Mueller, 2007]. The utility of the aforementioned control calculi and the high-level languages that are based on these calculi for the modelling of dynamics cannot be taken granted – rather fundamental problems (e.g., frame, ramification, qualification) relevant to modelling changing environments have been thoroughly investigated in the context of the class of formalisms aforementioned [Shanahan, 1997]. This has also resulted in several non-monotonic extensions to classical symbolic approaches that are better suited for modelling dynamically changing systems [Bhatt and Loke, 2008, Bhatt, 2008] and representing cognitively adequate (human-like) common-sense reasoning with incomplete information. By

including these diverse control approaches within the framework, the objective is to facilitate and promote the experimental, pedagogical and other potential uses of the framework. Finally, it should be noted that the proposed framework directly embeds such high-level languages in a manner that completely abstracts from the control approach specific details by the use of the generic domain description component in (C1).

C3. Underlying Qualitative Physics

Of key interest to this work is to operationalize the notion of a domain-independent qualitative spatial theory, which is representative of an underlying 'qualitative physics' that is applicable for a wide-range of dynamic spatial domains/systems. Here, by a dynamic spatial system, we refer to a specialization of the dynamic systems [Sandewall, 1994] concept for the case where a domain theory consists of changing qualitative spatial relationships pertaining to arbitrary spatial aspects such as the orientational [Freksa, 1992, Moratz et al., 2000], directional [Frank, 1996, Ligozat, 1998] and topological [Randell et al., 1992] spatial dimensions. Basically, what this implies is that spatial relationships are modelled as time-dependent properties (i.e., fluents) and the manner in which they change are strictly governed by the rules (conceptual neighborhoods, compositional consistency and so forth) that are intrinsic to a particular spatial calculus (e.g., topological or orientation calculi) that is being modelled as a part of the underlying qualitative physics. This notion of a domain-independent qualitative spatial theory within the framework is primarily used as a means to demonstrate the applicability of (existing) qualitative spatial models relevant to different aspects of space in realistic dynamic spatial scenarios. In addition, such a theory has the advantage of being general and re-usable in a wide-range of spatial domains. In [Bhatt and Loke, 2008], we have presented an in-depth study of realizing such a domain-independent spatial theory in the context of the situation calculus formalism and presently, work is is progress to extend the approach therein to event calculus and fluent calculus.

C4. Application Platform Independence

It is necessary that the framework be independent of any particular robotic system/platform or agent simulation environment thereby ensuring applicability in a wide-range of real or simulated environments. Basically, an adequate level of abstraction between the experimental framework and robotic hardware and simulated systems is necessary. Toward this end, the framework consists of a Controller Communication Interface' (CCI) that provides the necessary abstraction between robotic or simulation platforms and the experimental framework. This independence is achieved by the generic CCI by explicitly defining all possible modes of communication (e.g., by way of serializing control actions to the robot's actuators and the inflow of sensing information) between the framework and the external world, which the framework is being interfaced with. Other details are included in section 3 (T3).

3 Technical Overview and Progress

The discussion in section 2 is intended to provide an overview of the complete framework, as it is presently envisaged. In this section, we report the preliminary progress in that direction and highlight our working exemplar that implements parts of the proposed framework. Because of the work-in-progress nature of the proposal, we only discuss aspects where conclusive implementations have been realized.

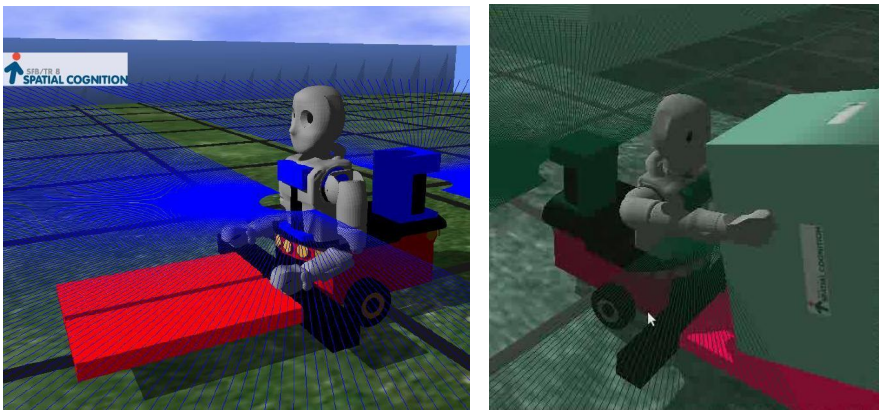
T1. The Control Apparatus – Reasoning about Action and Change

The framework presently embeds control approaches based on the STRIPS, BDI and an existing interpreter for the situation calculus based language INDIGOLOG [Giacomo and Levesque, 1999]. Without going into the details of any of these approaches, we would like to mention that the case of the STRIPS and BDI based control approaches is trivial in comparison to embedding the interpreter for INDIGOLOG. Indigolog supports the incremental execution of high-level agent control programs through the interleaving of planning, sensing and executing actions in the real/simulated world, i.e., sensing affects subsequent computation. The present communication controller interface (see T2) is minimal and has been designed to conform to the requirements of the INDIGOLOG interpreter, namely – serializing primitive actions execution commands to an arbitrary sink that is connected to the control module, reporting of exogenous events from the external world back to the control module and the capability to perform sensing actions to determine the state of certain properties of the world. The inclusion of the control approaches based on event calculus and fluent calculus is subject to further work and the completion of a complete working exemplar consisting of only STRIPS, BDI and INDIGOLOG.

T2. Controller Communication Interface

In the present exemplar, the controller communication interface has been designed to comply with an existing robotic hardware abstraction platform, namely the PLAYER/STAGE/GAZEBO project that available in the open-source domain [Gerkey et al., 2003]. With a network-centric client-server model, PLAYER provides an interface to a variety of robot and sensor hardware and allows for robot control programs to be written in any programming language and to run on any computer with a network connection to the robot. Since it is not an objective of this project to directly investigate the seamless integration of arbitrary real robotic or simulation platform, using the robot control abstractions provided by the PLAYER system within our exemplar is advantageous because of the following reasons:

1. PLAYER uses a generic API to control a wide range of robotic platforms thereby maximizing applicability in realistic applications
2. The accompanying STAGE and GAZEBO projects provide accurate physical simulators for the 2D and 3D case respectively that may be transparently used in conjunction with the PLAYER system, i.e., experiments may directly switch between real robotic and simulated modes



(a) Laser Guided Mobile Humanoid Gripper (b) Humanoid in Action

Figure 2: A Simulated Delivery Robot

without any change in the overall system architecture, and finally

3. The PLAYER/STAGE/GAZEBO project is open-source and continuously being enhanced and updated, which is clearly desirable from a long-term usability viewpoint.

Note that a full-integration of the PLAYER system being an open-ended task is considered beyond the scope of this project. However, working examples for a few PLAYER compatible robotic platforms in the context of the CCI have been demonstrated (see T3 and T4). To re-iterate, the main aim of this project is to develop a framework that can be utilized for high-level control and decision-making. As such, we only implement as much low-level motion control or a primitive skill set as is necessary for us to illustrate the utility of the framework for this purpose.

T3. A MOBILE HUMANOID ROBOT

The BANDIT model available within the GAZEBO simulator has been modified and extended to realize a laser-guided, mobile and grasp capable humanoid robot (see Fig. 2). The humanoid model consists of two simulated SICK lasers for simultaneous forward and backward alignment, a gripper to grasp and lift idealized objects, and a moving platform located at the intrinsic front of the bot on/from which objects may be loaded/unloaded. Indeed, the entire humanoid is mounted on a Pioneer 2DX that is capable of moving around using differential motors. When big objects that obstruct the robot's view are loaded on this platform, the backward laser is used for purposes of movement and alignment (see Fig. 2(b)). The model functions as one unit and is capable of moving around, performing turn actions at varying degrees and picking-up and dropping objects. This primitive skill set is sufficient for our exemplary purposes of realizing one complete exemplar that has high-level, logic-driven reasoning with qualities that is completely abstracted from the precise low-level motion control that occurs within the GAZEBO simulator.

T4. DEMONSTRATIVE SCENARIOS

Two exemplary scenarios have been implemented to realize a fully-functioning system consisting of high-level reasoning on the one hand and low-level motion control with the simulated humanoid robot on the other. One scenario consists of a delivery system that delivers objects from one location to another in an idealized 3D world. Another scenario involves the same simulated robot performing a room-clearing task; here, the objective is to re-arrange a set of objects in a room in a pre-specified manner. Indeed, both scenarios utilize the same set of primitive skills in so far as the movement and object manipulation are concerned. Note that given the outlook of this work (see section 4), we are intentionally focusing on problems that involve qualitative spatial reasoning abilities with orientation and topological information.

4 OUTLOOK

Present work is focused on developing a complete exemplar of the overall framework as proposed in section 2. Toward that end, of primary importance is incorporating a PDDL based domain description language and its mapping to the domain-theory and behavior specification constructs as required by the situation calculus based INDIGOLOG. Preliminary studies show that such a language subsumes similar requirements of the STRIPS and BDI control approaches. Secondarily, albeit purely in the context of the situation calculus, we are also integrating formal spatial (intrinsic orientational and topological) calculi in a way such that qualitative spatial reasoning in the form of consistency and conceptual neighborhood based dynamics may be utilized in arbitrary spatial scenarios. The development and illustration of a test-suite of problems in spatial control, primarily encompassing spatial planning and decision-making in real robotic-control and simulated environments is one of the main aims of this research. The test problems would be used to determine the feasibility of the implemented control approaches and also to perform empirical comparisons amongst them. In addition, they would also be extensively documented from an illustrative viewpoint so as to serve as examples for the utilization of

the experimental framework by other users or to be used for pedagogical purposes.

ACKNOWLEDGMENTS

The author acknowledges funding provided by the ALEXANDER VON HUMBOLDT STIFTUNG (GERMANY) toward this project. This work originated and benefited from my previous collaboration with Maurice Pagnucco (UNSW, AUSTRALIA). Practical contributions by Sanchit Sood (IIT BOMBAY, INDIA) and Hemant Agrawal (IIT ROORKEE, INDIA) in implementing the low-level control with PLAYER/GAZEBO are acknowledged.

REFERENCES

M. Bhatt. (Some) Default and non-monotonic aspects of qualitative spatial reasoning. In *AAAI-08 Technical Reports, Workshop on Spatial and Temporal Reasoning*, pages 1–6, 2008. ISBN 978-1-57735-379-9.

M. Bhatt and S. Loke. Modelling dynamic spatial systems in the situation calculus. *Spatial Cognition and Computation*, 8(1):86–130, 2008. ISSN 1387-5868.

M. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

J. Claßen, P. Eyerich, G. Lakemeyer, and B. Nebel. Towards an integration of golog and planning. In M. M. Veloso, editor, *IJCAI*, pages 1846–1851, 2007a.

J. Claßen, Y. Hu, and G. Lakemeyer. A situation-calculus semantics for an expressive fragment of pddl. In *AAAI*, pages 956–961. AAAI Press, 2007b. ISBN 978-1-57735-323-2.

A. Cohn and S. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundam. Inf.*, 46(1-2):1–29, 2001. ISSN 0169-2968.

G. De Giacomo, Y. Lésperance, and H. J. Levesque. ConGolog, A concurrent programming language based on situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 88–97. Kaufmann, San Mateo, CA, 1990.

A. U. Frank. Qualitative spatial reasoning: Cardinal directions as an example. *International Journal of Geographical Information Science*, 10(3):269–290, 1996.

C. Freksa. Conceptual neighborhood and its role in temporal and spatial reasoning. In M. Singh and L. Travé-Massuyès, editors, *Decision Support Systems and Qualitative Reasoning*, pages 181–187. North-Holland, Amsterdam, 1991.

C. Freksa. Using orientation information for qualitative spatial reasoning. In *Proceedings of the Intl. Conf. GIS, From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 162–178, London, UK, 1992. Springer-Verlag. ISBN 3-540-55966-3.

B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *ICAR 2003*, pages 317–323, Coimbra, Portugal, June 2003.

G. D. Giacomo and H. J. Levesque. An incremental interpreter for high-level programs with sensing. In H. J. Levesque and F. Pirri, editors, *Logical Foundation for cognitive agents: contributions in honor of Ray Reiter*, pages 86–102. Springer, Berlin, 1999.

H. Grosskreutz and G. Lakemeyer. cc-Golog: Towards More Realistic Logic-Based Robot Controllers. In *NMR-00*, 2000.

R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986. ISSN 0288-3635.

G. Lakemeyer and H. Grosskreutz. On-line execution of cc-golog plans. In *IJCAI*, pages 12–18, 2001.

H. Levesque and G. Lakemeyer. Chapter: Cognitive robotics. In V. Lifschitz, F. van Harmelen, and F. Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007.

H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *J. Log. Program.*, 31(1-3):59–83, 1997.

G. Ligozat. Reasoning about cardinal directions. *J. Vis. Lang. Comput.*, 9(1):23–44, 1998.

J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Language – version 1.2. In *Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control*, 1998.

R. Moratz, J. Renz, and D. Wolter. Qualitative spatial reasoning about line segments. In *ECAI*, pages 234–238, 2000.

E. T. Mueller. Discrete event calculus reasoner. In *System Documentation, IBM Thomas J. Watson Research Center*, 2007. URL `http://decreasoner.sourceforge.net/`.

D. A. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 165–176. Morgan Kaufmann, San Mateo, California, 1992.

E. Sandewall. *Features and Fluents (Vol. 1): The Representation of Knowledge about Dynamical Systems*. Oxford University Press, Inc., New York, NY, USA, 1994.

M. Shanahan. *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT Press, 1997. ISBN 0-262-19384-1.

M. Thielscher. Introduction to the fluent calculus. *Electron. Trans. Artif. Intell.*, 2:179–192, 1998.

M. Thielscher. Flux: A logic programming method for reasoning agents. *Theory Pract. Log. Program.*, 5(4-5):533–565, 2005. ISSN 1471-0684.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

An Incremental Approach to Adaptive Integration of Layers of a Hybrid Control Architecture

Matthew Powers and Tucker Balch
College of Computing,
Georgia Institute of Technology
mpowers@cc.gatech.edu, tucker@cc.gatech.edu

Abstract

Hybrid deliberative-reactive control architectures are a popular and effective approach to the control of robotic navigation applications. However, due to the fundamental differences in the design of the reactive and deliberative layers, the design of hybrid control architectures can pose significant difficulties. We propose a novel approach to improving system-level performance of hybrid control architectures, by incrementally improving the deliberative layer's model of the reactive layer's execution of its plans. Incremental supervised learning techniques are employed to learn the model. Quantitative and qualitative results from a physics-based simulator are presented.

1 Introduction and Related Work

Hybrid deliberative-reactive control architectures for robotic navigation have long been an active area of research. Despite their success, open questions remain how to best integrate the layers to maximize overall system performance. In this work, we propose a novel method to improve the integration of deliberative planning and reactive control in a robotic navigation system. In particular, we will use supervised machine learning techniques to improve the deliberative layer's model of the reactive layer's interpretation of its plans.

Arkin's AuRA architecture [Arkin and Balch, 1997] and Gat's Atlantis architecture [Gat, 1991] are early examples of hybrid deliberative-reactive architectures. In both, the reasoning done by the deliberative layer is fundamentally different from that done by the reactive layer. The deliberative layer works to achieve global goals based on world models. The reactive layer works to achieve local constraints based on current sensor input. Each architecture suggests methods for combining the globally-based deliberative input with the locally-based reactive reasoning.

Many modern systems use an implementation of a hybrid layered approach to robot control architecture, using decoupled layers of functionality to satisfy both the robot's immediate constraints and its longer-term objectives. In the case of robot navigation (especially in the area of field robotics), many modern architectures make use of a lower-fidelity global deliberative planner and a higher-fidelity local reactive controller [Albus, 2002], [Thrun *et al.*, 2006], [Urmson *et al.*, 2008].

Finding a compromise between global objectives and local constraints is not always easy, and often the tradeoffs have to be empirically "fine-tuned" by the robot software designer. Either the deliberative layer's model of the world and the robot, or the reactive layer's interpretation of the deliberative layer's input must be adjusted. This process can be time-consuming and is subject to human interpretation of the robot's performance. It can also simply be difficult for a human to make sense of how all the degrees of freedom that a complex software system may contain might affect the robot's system-level performance. Because of these difficulties, a body of work has evolved promoting learning across layer boundaries or across task decompositions.

Early work in reinforcement learning across task decomposition was done by Lin [Lin, 1993]. In Lin's work, the system designer decomposed the robot's task into low-level skills and high-level skills (which make explicit use of the low-level skills) that the robot will need to complete the task. Q-learning was used to first learn the low-level skills, then the high-level skills. Similar to Lins work, Stone [Stone, 1998] implemented an approach to task decomposition and learning within the context of robot soccer. Rather than using reinforcement learning at all layers, Stone relied on human insight to choose appropriate learning techniques at each layer. Higher-level layers were learned making explicit use of learned low-level layers. In [Balch, 1998], Balch demonstrated the use of reinforcement learning for robots to learn a sequential layer strategy in the form of a finite state automata (FSA), based on a designer-implemented reactive layer. Balch implemented a set of behavioral assemblages, defined states in an FSA mapping to each behavioral assemblage, and used Q-learning to learn transitions between the states in the FSA.

Beyond the difficulties of designing the appropriate models for use in a hybrid control architecture, it may be desirable to eliminate a priori models altogether. For example, a situation may call for a robot being placed in a new environment and learning how best to navigate within the new environment. Working within this scenario, a significant body of research has also been built around learning models of traversability by experience. In [Sun *et al.*, 2006], used short-range stereo vision as a supervisory signal to learn models based on longer-

range color vision. Note, however, that this work does not actually measure the robot's performance within environment, but rather finds a mapping from learned color-based models to a priori stereo-based models of traversability.

We propose an approach to improving system-level performance of hybrid control architectures by learning models of the reactive layer's execution of the deliberative layer's plans, based on measurements of actual executions. Our approach collects training data by measuring the performance of the execution of the plans. Supervised machine learning techniques (in particular, an implementation of the k-nearest neighbor algorithm), are used to abstract that performance to predict performance in other environments. Then, this learned model is fed back to the planner for use in creating plans with better overall performance.

2 Representation

Following the pattern of other hybrid control architectures, we divide the architecture into two distinct components, the reactive layer and the deliberative layer. The reactive layer is responsible for monitoring the robot's sensors, performing low-level navigation and decision making, and actuating the robot's motors. We model the robot's reactive layer as a continuous controlled dynamical system. The deliberative layer is responsible for integrating sensor input into maps, and planning routes and actions toward a given goal. We model the deliberative layer as a discrete process providing regular input to the reactive layer.

2.1 Reactive Layer

We begin by modeling the robot as a controlled dynamical system,

$$\dot{x} = f(x,u),\ x \in \mathbb{R}^a,\ u \in \mathbb{R}^b \tag{1}$$

which exists in a world $W \subset \mathbb{R}^d$.

We are given a measurement equation,

$$y = h_y(x) \tag{2}$$

that provides access to the state of the system. Additionally, we are given another measurement equation that gives sensory access to the state of the world, $h_s(x,w)$. We then define the array s as the collection of all observable sensory input:

$$s = \{h_s(x,w)\}_{w \in W} \tag{3}$$

We can then close the loop, defining the control input u as a function of the measurements of both the system state and the environment state. Thus,

$$u = g(y,s) \tag{4}$$

2.2 Deliberative Layer

We model the robot's deliberative layer as a regularly updating discrete-timed event system which updates at times $t_0, t_1, \ldots, t_{final}$, where $t_i - t_{i-1} = \Delta t$, $\Delta t > 0$. These intervals account for the practical requirements of the execution of complex algorithms and management of large data sets.

Given that the robot is using a map to guide its path planning algorithms, we define the map M as an integrated set of sensory input. In each update cycle, the most recent set of sensory input, s_t is integrated into the map, relative to the most recent measured state of the system, y_t, by the integration function m,

$$M_t = m(s_t, y_t, M_{t-1}) \tag{5}$$

We assume m is a non-invertable function. That is, given M_t we cannot directly recover $\{(s_0,y_0),\ (s_1,y_1),\ \ldots,\ (s_t,y_t)\}$. This is an important point, as given M_t, we cannot directly recover the reactive layer's control output, $g(y_t,s_t)$

Given that the world $W \subset \mathbb{R}^d$ is compact and connected, assume W is partitioned into a set of n regions,

$$R = \{r_i\}_{i=1}^n \tag{6}$$

such that

$$\bigcup_{r \in R} r = W \tag{7}$$

and

$$r_i^o \bigcap r_j^o = \phi,\ \forall i,j,\ i \neq j \tag{8}$$

where r^o denotes an interior region.

For each region, we are given a collection of m control laws,

$$G_{r^o} = \{g_i(y,s)\}_{i=0}^m,\ \forall r^o \in R \tag{9}$$

and a transition function $d(r^o, M, g)$ which provides a mapping from an interior region and a control law to the next region the control law will drive the robot toward. Intuitively, we can think of this is as the expected outcome of the control law. This mapping is important to the planning process as it provides a model of the outcome of the action of employing a particular control law. We are also given a cost model, $c(r^o, M, g)$ that provides an expected cost of traversing region r^o, using the control law g, given the map M.

Given a goal region, r_{goal}, and a starting region, r_{start}, this representation is easily mapped into a graph-based model (compatible with many planning algorithms), $\Gamma = (V, E, l, v_{start}, v_{goal})$, where:

- V is a set of vertices, directly corresponding to the set of interior regions, $\{r^o\}_{r^o \in R}$
- E is a set of directed edges, $E \subseteq V \times V$. This set of edges corresponds to the connectivity described by the transition model, $E = \{r^o \times d(r^o, M, g)\}_{r^o \in R,\ g \in G_{r^o}}$
- l is a cost function $l : E \rightarrow \mathbb{R}^+$ directly corresponding to the cost model $c(r^o, M, g)$, where the edge corresponding to the weight is given by the transition model, $e = (r^o \times d(r^o, M, g)),\ e \in E$.
- v_{start} and v_{goal} are the starting and goal vertices, respectively. These vertices correspond directly to the regions r^o_{start} and r^o_{goal}.

Within this graph-based representation, the path planning problem can be defined as selection a sequence of edges

$$Plan = \{e_0 = (v_{start}, v_1), \ldots, e_N = (v_{goal-1}, v_{goal})\} \tag{10}$$

to minimize the total cost

$$Cost = \sum_{e \in Plan} l(e) \tag{11}$$

In this case, the set of edges is provided by the transition model, $d(r^o, M, g)$, which is a function of the selection of of the control law, g. We can then more precisely define the planning problem as choosing a mapping $b \in B$ (where B is the set of all possible mappings), from each r^o to a $g \in G_{r^o}$,

$$\dot{x} = f(x, g(y,s)),\ \forall x \mid p(x) \in r^o,\ g = b(r^o) \tag{12}$$

(where $p(x) \in W$ is the position of the system) , such that

$$b = \arg\min_{b \in B} \sum_{i=0}^{goal} c(r_i^o, M, b(r_i^o)) \tag{13}$$

where

$$r_{i+1}^o = d(r_i^o, M, b(r_i^o)) \tag{14}$$

2.3 Learning

Two components of the deliberative layer's planning process rely primarily on a priori models of the reactive layer's execution of the provided plans. The transition model, $d(r^o, M, g)$ predicts the the next region the system will enter, given the region the robot is currently in and the control law the robot is currently executing. The cost model $c(r^o, M, g)$ predicts the cost incurred by the system until the next region is reached. It is the goal of this work to improve the integration of the deliberative and reactive layers by learning a cost model that better represents the cost actually incurred by the reactive execution of the plan. Improving performance by learning the transition model as well will be discussed further in the Future Work section.

We begin by defining a measurement function, $m_c(x, r^o)$ which measures the cost incurred by the execution of a control law in region r^o given map M. We define the learning problem as choosing a cost model that best predicts the measured cost, given all measurements up to time t,

$$c_t = \arg\min_{c \in C} E[\ |m_c(x, r^o) - c(r^o, M, g)|\ ]\ |\ \{m_c(x, r^o)\}^t \tag{15}$$

where C is the set of all possible cost functions. This optimization is over an expectation not only because of possible noise in the sensory information, but because, as noted earlier, the mapping function is non-invertable. Therefore, the cost model, which is a function of M, cannot directly access the reactive output measured by the measurement function. The best the cost model can do is a prediction of the reactive output. Our goal is to minimize the error in this prediction.

3 Implementation

To demonstrate the capabilities and performance of the proposed system, a simulated robot and hybrid control architecture was implemented. A car-like robot was implemented in the Gazebo simulation environment [Gerkey *et al.*, 2003]. The robot's physical state is represented as simply its 2-dimensional position and heading,

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{16}$$

The robot has control over its translational velocity, v and its steering angle, which is proportional to the curvature of its path, κ,

$$u = \begin{bmatrix} \kappa \\ v \end{bmatrix} \tag{17}$$

Thus, the dynamics of the robot are defined,

$$\dot{x} = \begin{bmatrix} v \cdot \sin\theta \\ v \cdot \cos\theta \\ v \cdot \kappa \end{bmatrix} \tag{18}$$

The simulated robot is equipped with sensors to measure its own state, and the state of the world. A simulated GPS module provides the robot with a measurement of its own state,

$$y = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{19}$$

Simulated laser range-finders provide measurements of the state of the world,

$$h_s(x, w) = \begin{cases} o(w) & \text{if } \|p(x) - w\| \le \Delta \\ \phi & \text{otherwise} \end{cases} \tag{20}$$

where Δ is the range of the measurement system, and $o(w)$ is the *occupancy* of the point $w \in W$. The occupancy of a point w is defined as:

$$o(w) = \begin{cases} 1 & \text{if the point } w \text{ is occupied} \\ 0 & \text{else} \end{cases} \tag{21}$$

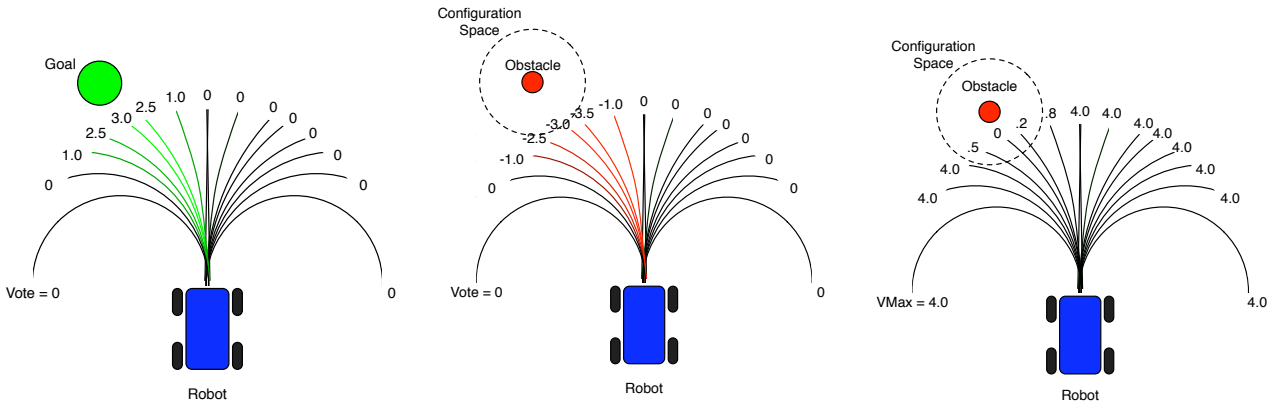
3.1 Reactive Layer

The reactive layer is implemented in a behavior-based voting design, explained in detail in [Wooden *et al.*, 2007]. In this design a number of behaviors evaluate candidate actions over a short temporal scale, each behavior representing a specific interest pertaining to the robot's objective.

In this implementation, as shown in Figure 1, the behaviors reason over constant curvature arcs. Each behavior distributes an allocation of votes over an array of potential arcs for the robot to navigate along. The behaviors can allocate votes for arcs that work to achieve its interests, or against arcs that are detrimental to its interests. In addition to distributing votes for or against arcs, behaviors assign a maximum allowable velocity, associated with each arc. Behaviors need not necessarily express an interest across both curvature and velocity. A behavior may vote for curvatures and leave the allowable velocities set to the robot's maximum velocity, it may cast no votes for or against curvatures and express its interest across the allowable velocities, or it may express its interest across both dimensions.

To choose a curvature and velocity for the robot to execute, an arbiter sums the votes cast by each behavior for each curvature arc, weighting the votes for each behavior according to a predetermined weighting scheme. It selects for execution the curvature arc with the highest total of votes. It then selects for execution the minimum of the maximum allowable velocities assigned by the respective behaviors to the selected curvature arc. The selected curvature and velocity are sent on to low-level controllers for execution.

Five behaviors were used in this implementation (three of which are diagrammed in Figure 1):



(a) The *Move to Waypoint* behavior. (b) The *Avoid Obstacles* behavior. (c) The *Maintain Headway* behavior.

Figure 1: Three of the behaviors used in the implementation of the reactive layer. In (a), the *Move to Waypoint* behavior votes in support of curvatures that take the robot closer to the provided waypoint. In (b), the *Avoid Obstacles* behavior votes against curvatures that take the robot toward sensed obstacles. In (c), the *Maintain Headway* behavior sets a maximum allowable translational velocity for each curvature, with respect to sensed obstacles. An arbiter tallies the weighted votes provided by the set of behaviors, and outputs the curvature with the most votes, and the minimum allowable translational velocity for that curvature.

- *Move to Waypoint* - shown in Figure 1(a), allocates positive votes to arcs according to a linear control law relating the local heading to the waypoint to a commanded curvature.
- *Avoid Obstacles* - shown in Figure 1(b), allocates negative votes to arcs according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. Arcs that do not cross into the configuration space of the obstacle are not voted against.
- *Maintain Headway* - shown in Figure 1(c), sets maximum allowable velocities for each arc according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. If the arc does not cross into the configuration space of the obstacle, the robot's maximum speed is assigned. If the arc crosses into the configuration space of the obstacle within a parameterized safety distance, the maximum allowable velocity is zero.
- *Slow for Congested Areas* - sets maximum allowable velocities for each arc according to the distance along the arc that the arc crosses into an intentionally large configuration space around a detected obstacle. If the arc does not cross into the the configuration space of the obstacle, the robot's maximum speed is assigned.
- *Slow for Turns* - sets a maximum allowable velocity for each arc according to a parameterized maximum allowable rotational velocity. If the calculated maximum allowable velocity is larger than the robot's top speed, the robot's top speed is assigned.

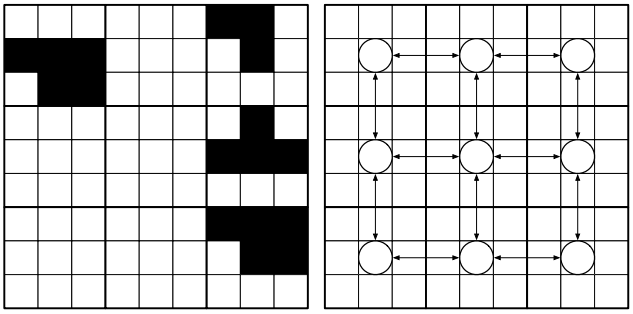
In this implementation, the set of control laws G_{r^o} is provided by parameterizing the given set of behaviors with a waypoint from each adjacent region. (i.e., each member of the set of control laws drive the robot toward one of the adjacent regions, using the full compliment of behaviors.) The transition model $d(r^o, M, g)$ is simply defined as mapping to the region associated with the waypoint parameterizing the control law g, regardless of the map.

3.2 Deliberative Layer

The deliberative layer is implemented as a global path planner over a relatively high-resolution occupancy grid. As sensory information is accumulated in the local frame it is integrated into the global map based on the robot's current global state measurement. Detected obstacles are placed into grid cells based on their discretized global position. Each grid cell can be marked as either occupied or unoccupied. Obstacles associated with unoccupied cells cause the cell to be marked as occupied. Obstacles associated with occupied cells have no effect on the cell; the cell remains marked occupied.

The grid cells are then grouped into regions, as depicted in Figure 2(b). A count of occupied grid cells is kept within each region. This count is used by the planning algorithm in evaluating the cost of traversing each region.

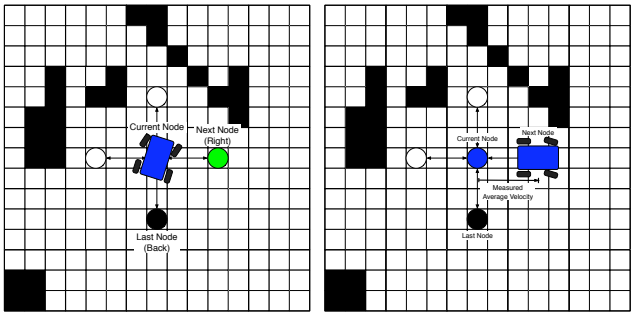
To represent the connectivity between the regions, a graph is overlaid on the map, as shown in Figure 2(b). One graph vertex is placed at the center of each region. Edges are added between contiguous nodes. Figure2(b) depicts a four-connected graph based on the structure of the occupancy grid. The cost of traversing each edge is proportional to the expected time to move between its source node and destination node. The time to move between nodes is modeled as the distance between the nodes divided by the expected average velocity of the robot over that distance. The naïve planner uses a binary model of the robot's velocity. If the count of occupied grid cells within the region associated with either node is larger than a parameterizable count, the expected velocity is zero (i.e. the edge is not traversable and is assigned an infinite cost). Otherwise, the expected velocity is the robot's



(a) The occupancy grid.

(b) The occupancy grid divided into regions, and overlaid with a graph.

Figure 2: The occupancy grid, regions and graph used by the deliberative layer. In (a), the occupancy grid is shown. The world is discretized into a grid. Each cell represents either occupied (black) or open (white) terrain. In (b), the occupancy grid is divided into regions (shown by thicker lines). A graph is overlaid on regions, describing the connectivity of the map. Planning is done over the graph, using the underlying occupancy grid as input for the cost model.



(a) Learning experience setup

(b) Learning experience measurement of average velocity between regions

Figure 3: A depiction of the encoding of the learning experience. In (a), the learning experience is setup. The robot is shown in the center of the diagram. The next planned waypoint is represented by the graph vertex to the right of the robot. The local map, out to the range of the robot's sensors is included under the graph. In (b), the robot has completed the experience. The supervisory signal (in this case, average velocity) is measured and provided to the learning algorithm, along with the experience representation.

top speed. This graph structure is a suitable data structure for many planning algorithms. In this implementation, an instance of the D*-Lite [Koenig and Likhachev, 2002] [Koenig and Likhachev, 2005] algorithm is employed.

3.3 Learning

The learning component of this approach is implemented within a supervised learning paradigm. To keep the learning problem tractable, it is important to create a compact, yet meaningful representation of the robot's experiences executing proposed planning segments. We define the length of a learning experience to be time to move from one region, r^o to the next region in the plan, $d(r^o, M, g)$, given the control law g provided by the planning algorithm. We use the following representation of a learning experience:

$$Exp = (g, M_{local}) \tag{22}$$

where M_{local} is a local representation of the map, M. To take advantage of symmetry in the problem, we orient the experience into the robot's local frame. That is, rather than encoding the segments of the plan as "move north" or "move east", it is more general to encode the robot's experiences as "move forward" or "move right". A more general encoding of experiences makes learning over these experiences more tractable, as it reduces the dimensionality of the problem. Figure 3 is a graphical representation of the robot's planning experience.

A supervisory signal is provided by the measurement function $m_c(x, r^o)$, which measures the reactive layer's interpretation of the commanded plan. As shown in Figure 3(b), the measurement function measures the average speed of the robot during the experience.

As experiences are collected, they are integrated into the learning algorithm. The learner uses the experiences to extrapolate expected results from new proposed experiences. In this implementation, the learned model of expected velocity is used by the global planner to plan subsequent navigation paths. The planner uses the model to evaluate the expected cost of traversing an edge, in terms of expected time to traverse the edge. To evaluate the cost of an edge, the edge is encoded in terms of a planning experience. The learned velocity model returns an expected velocity over the edge. The time-based cost model is obtained by dividing the distance between the source node and the destination node by the expected velocity. The planning algorithm plans over these costs to find the fastest route.

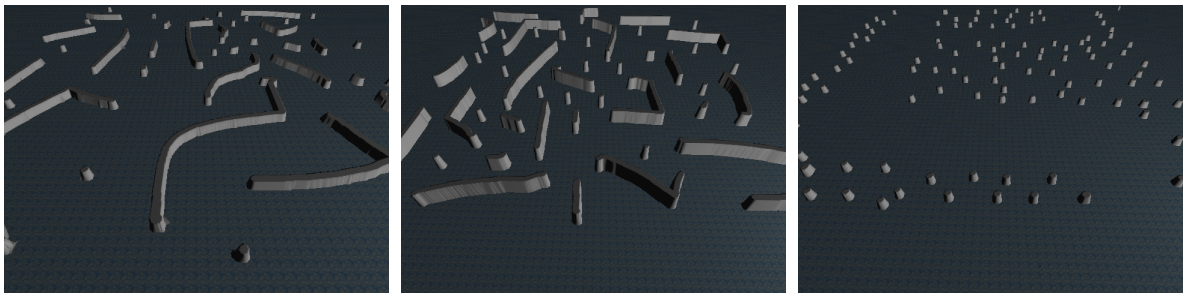
4 Experimental Setup and Results

4.1 Experimental Setup

Three complex environments were designed within the Gazebo simulation environment. The environments used are shown in Figure 4. Environment 4(a) was used for gathering training data. Environment 4(b) was used for running tests comparing different systems. Environment 4(c) was used to demonstrate the qualitative behavior of the system.

To effectively demonstrate differences in performance for different amounts of training data, data was collected a priori. Cost models were built using different amounts of training data (ranging from 100 to 5000 experiences). Each cost model was tested independently without incremental learning. The performance of each cost model was then compared, providing data on how performance improves with the number of training instances.

Data was gathered in the training environment by tasking the robot to achieve a series of randomly generated goals around the environment, using the naïve planner and the above described reactive layer. Every time the robot achieved



(a) The training environment (b) The quantitative testing environment (c) The qualitative testing environment

Figure 4: The environments used in training and testing. The environments were built in the Gazebo simulation environment. Environment (a) was used in gathering training data for the learning process. Environment (b) represents a slightly more complex environment than (a), and was used for quantitative testing. (c) represents a plausible environment, consisting of a path through a dense forest, and was used for qualitative testing.

a waypoint the robot's experience was recorded, including the local map, the robot's average velocity, the commanded waypoint, and the waypoint actually achieved. Approximately 5000 learning examples were collected.

Several different supervised learning algorithms were evaluated for use. The Weka machine learning environment [Frank *et al.*, 2005] provided a library of community-supported implementations of well known algorithms. For learning prediction of the robot's velocity (a real-valued signal), we evaluated the k-nearest neighbor algorithm for several values of k, a multi-layer perceptron network, linear regression, and the naïve strategy of always assuming the robot travels at its maximum speed.

Models were built from the training data, using each algorithm. Cross-validation tests on the training data were performed to evaluate the effectiveness of each algorithm. Table 1 displays the cross validation results for each algorithm on the velocity data. The k-nearest neighbor algorithm with $k = 5$ produced the highest correlation and nearly the lowest average relative error of the algorithms tested, and was chosen for use in testing. Table 2 displays cross validation results for instances of the k-nearest neighbor algorithm for various numbers of training examples. The results show a clear trend of improvement as the number of training examples increases.

The learned models were then incorporated into the cost function of the global planner. The naïve system was compared to the system using the learned model. Each system was tasked with achieving a sequence of goals criss-crossing the test environment. This sequence of goals totaled a piecewise straight-line distance of over 1500 simulated meters. Results were compiled comparing the average time to complete each goal between different systems.

In addition to quantitative experiments, a qualitative experiment was performed to demonstrate, in an intuitive way, the effect learning had on the overall system performance. An environment was constructed to resemble an open path through a wooded area, shown in Figure 4(c). The wooded area is sparse enough that the robot is capable of finding a path between the trees, but would travel that path slowly due to its tendency to drive slowly in tight spaces and slow down for the frequent required turns. The robot was tasked with navigating to a goal whose straight-line path would take the robot through the woods. The plans and resulting paths created by the naïve system and the system that had learned a cost model were compared qualitatively and quantitatively.

	Naïve	KNN $k = 50$	KNN $k = 15$	KNN $k = 5$	KNN $k = 2$
Average Relative Error	106%	80%	72%	64%	61
Correlation	-.44	.57	.62	.65	.63

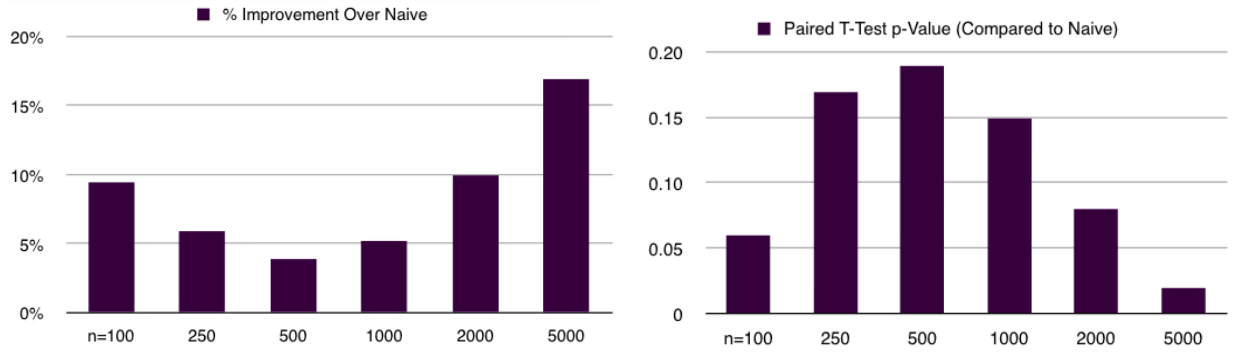
Table 1: Results of a 10-fold cross validation test on several learners predicting the robot's velocity given a proposed planning transition, using 5000 training examples. The k-nearest neighbor algorithm with the k-value set to 5 produced the best results, nearly cutting the average relative error in half, compared to the naïve approach.

	$n =$ 100	$n =$ 250	$n =$ 500	$n =$ 1000	$n =$ 2500	$n =$ 5000
Average Relative Error	112%	95%	87%	74%	71%	64%
Correlation	.33	.43	.49	.54	.56	.65

Table 2: Results of a 10-fold cross validation test on several k-nearest neighbor instantiations, using varying numbers of examples to train. The value of k was set to $k = 5$ throughout the tests.

4.2 Quantitative Results

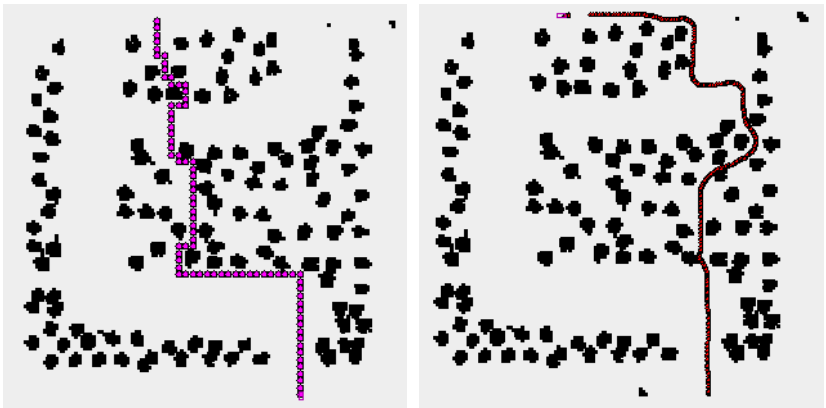
To demonstrate how the performance of the planner improved with learning, tests were run with different numbers of examples used in the learned model. Figure 5(a) shows the trend of performance improvement over the baseline naive planner. The learned planner improves significantly with just 100 examples, and starts a steady upward trend from 500 to 5000



(a) Performance as a function of number of training instances. Larger bars indicate better performance.

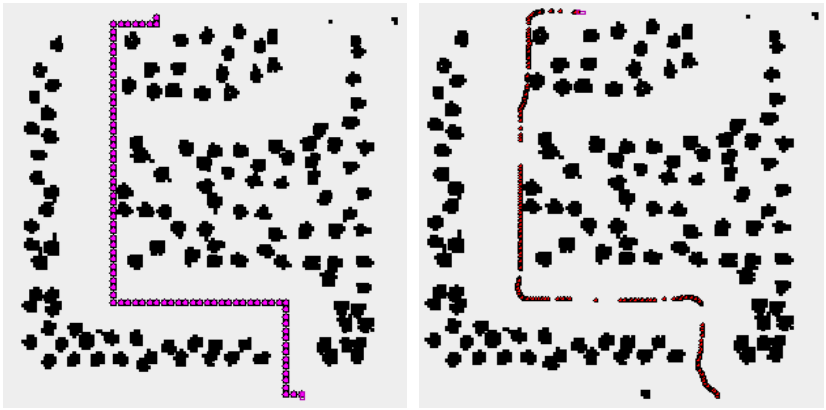
(b) Significance as a function of number of training instances. Smaller bars indicate statistically more significant performance.

Figure 5: System performance improvement over the baseline naive planner, as a function of the number of examples used in training a k-nearest neighbor regression model. (a) shows the improvement over the naive baseline system, while (b) shows the paired t-test p-value for each instantiation.



(a) Naïve system's planned route

(b) Naïve system's traveled trajectory



(c) Learned system's planned route

(d) Learned system's traveled trajectory

Figure 6: In (a), the planned path from the robot to the goal through the "path in the woods" environment using the naïve planner. The robot is at the bottom of the image. The goal is at the top of the image. The planned path is shown by pink waypoints. Obstacles in the map are shown in black. In (b), the actual trajectory taken by the system (trajectory in red). In (c), the planned path from the robot to the goal through the environment, using the learned cost model. The resulting plan is longer given a constant velocity model of the robot, but when used as input to the reactive layer, as shown in (d), reduces mission time by 25% over the plan shown in (a).

examples. Figure 5(b) shows the statistical significance of each test.

4.3 Qualitative Results

Figure 6 shows the results of the qualitative tests in the path through the woods environment. Figure 6(a) shows the naïve planner's planned route through the environment. Note how the planned route snakes through the dense obstacle field on its way to the goal. Figure 6(b) shows the trajectory actually taken by the robot following the planner's output. Note that it departs from the planned route early in the mission. The planner continues to suggest updated routes based on the robot's position, and the robot eventually achieves the goal.

Figure 6(c) shows the route provided by the planner using both the velocity and transition models. Note that it prefers a slightly longer (by distance) route that follows the wide path. Figure 6(d) shows the trajectory actually taken by the robot following this plan. In this trial, the robot completes the mission in 25% less time than the naïve planner. This demonstrates a clear qualitative and quantitative improvement in system-level performance in a plausible environment.

5 Conclusions and Future Work

In this paper, we propose a novel approach to the problem of improving system-level performance of a hybrid deliberative-reactive control architecture for robotic navigation. In particular, we propose using supervised machine learning techniques to improve the deliberative layer's cost model, based on measured performance of the reactive layer's execution of plans. The system was implemented in physics-based simulation environment. Quantitative and qualitative experimental results were compiled and presented.

Certainly more work in the area can be done. For example:

- It is not yet clear if the relatively good performance by the model using 100 examples is the result of over-fitting or the ordering of the examples used. In general, how does the order of training examples affect the learning process?
- It is not yet clear how map representation effects the performance of the learning component of the approach. Can representations be chosen to improve learning?
- While improving the planner's cost model certainly has the potential to improve overall system performance, judging from the trajectory in Figure 6(d), it is apparent that improving the transition model may also have a significant effect on system performance.

We look forward to exploring these questions in future work.

References

[Albus, 2002] James S. Albus. 4d/rcs: a reference model architecture for intelligent unmanned ground vehicles. In *In Proceedings of SPIE Aerosense Conference*, pages 1–5, 2002.

[Arkin and Balch, 1997] Ronald C. Arkin and Tucker Balch. Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189, 1997.

[Balch, 1998] Tucker Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, December 1998.

[Frank *et al.*, 2005] Eibe Frank, Mark A. Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H. Witten, and Leonhard Trigg. Weka - a machine learning workbench for data mining. In Oded Maimon and Lior Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005.

[Gat, 1991] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots. *SIGART Bulletin*, 2(4):70–74, 1991.

[Gerkey *et al.*, 2003] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *In Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

[Koenig and Likhachev, 2002] Sven Koenig and Maxim Likhachev. D*-lite. In *National Conference on Artificial Intelligence*, pages 476–483, 2002.

[Koenig and Likhachev, 2005] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, June 2005.

[Lin, 1993] Long-Ji Lin. Hierarchical learning of robot skills by reinforcement. In *International Conference on Neural Networks*, 1993.

[Stone, 1998] Peter Stone. *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1998.

[Sun *et al.*, 2006] Jie Sun, Tejas Mehta, David Wooden, Matthew Powers, James Rehg, Tucker Balch, and Magnus Egerstedt. Learning from examples in unstructured, outdoor environments. *Journal of Field Robotics*, 23(11/12):1019–1036, November/December 2006.

[Thrun *et al.*, 2006] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, et al. The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23:661–692, 2006.

[Urmson *et al.*, 2008] Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew Bagnell, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(1):425–466, June 2008.

[Wooden *et al.*, 2007] David Wooden, Matthew Powers, Magnus Egerstedt, Henrik Christensen, and Tucker Balch. A modular, hybrid system architecture for autonomous, urban driving. *Journal of Aerospace Computing, Information, and Communication*, 4(12):1047–1058, December 2007.

Towards a Hybridization of Task and Motion Planning for Robotic Architectures

Julien Guitton
ONERA - DCSD
Toulouse, France
julien.guitton@onera.fr

Jean-Loup Farges
ONERA - DCSD
Toulouse, France
jean-loup.farges@onera.fr

Abstract

Mission planning for a mobile robot involves symbolic and geometric reasonings. In conventional robotic architectures, modules in charge of these reasonings are commonly decoupled. In this paper, we propose a new framework for mission planning by bringing together a high-level planner and a low-level reasoner. This framework is based on the integration of geometric constraints into symbolic action descriptions and on an interaction protocol between the symbolic and geometric planners.

1 Introduction

Defining an architecture for autonomous robots implies defining several control levels from high-level mission management to low-level system controllers. Two approaches are commonly used: deliberative approach in which a high-level plan is computed and refined through different levels until obtaining sensor-motor functions, and reactive approach in which localization functions are directly linked with navigation functions. However, these methods have the following drawbacks: the deliberative approach does not allow a quick reaction to unforseen events. The reactive approach does not use global knowledge about the environment and thus the produced action sequence is globally sub-optimal.

Deliberative robotic architectures are commonly defined by three layers [Alami *et al.*, 1998]: decision level, control level, and functional level. Decision level includes capacities of producing a task plan and supervising its execution. Control level aims at controlling and coordinating execution of functions while respecting task definitions. Functional level contains built-in functions and perception capacities.

In this paper, we are interested by planning problems for a mobile robot, and more particularly, by task planning and path planning. In architectures designed for this kind of robots, the task planner is at the decision level and the motion planner at the control level. Using deliberative three-layered architectures for mobile robotics has well-identified problems [Estlin *et al.*, 2001]: they suffer from a high computation time, each level has its own description of the environment and actions, and translation of computed results between levels is not easy. In order to overcome these drawbacks, we investigate the formalization of a strong and clear link between task planning and motion planning. we adopt an experimental and pragmatic approach. We start by presenting experiments in section 2 which highlight the interest of delegating motion problems to a specialized planner. These results provide first hints as to the best way to define the coupling with a task planner. We also discuss these first properties on planners coupling and relate them to existing work. Finally, we propose both an abstract model in which motions are defined at the symbolic level and a protocol between the reasoning modules in order to formalize in a generic framework the properties of interleaved architectures.

2 Coupling reasoners: a bridge between robotic architecture levels

Mission planning is performed at the decision level. This mission planning implies motion management of one or several robots. However, motions are computed at the execution level. Thereby, the high-level planner uses an abstract world description in a form of a set of waypoints and a set of accessibility relations between them.

In this section, we present results of two experiments that aim at demonstrating that full delegation of motion planning to a specialized reasoner allows to obtain faster and better plans, and that an interleaved execution of both planners enables a faster and better management of events during the planning process.

2.1 Experimental framework

In these experiments, we link a high-level planner with a path planner. The high-level planner is a hierarchical task planner (HTN planner) and the path planner is based on a visibility graph for the environment description and on the Dijkstra algorithm for path searchs. The graph is composed of 70 vertices and 1126 edges. For the classical approach, the graph is translated into a set of logical predicates constituting the initial state of the planning problem. If there is an edge between vertices `v` and `v'`, a predicated `(visible v v' l)` is added to the initial state, where `l` is the length of the edge.

2.2 Classical vs. hybrid approach

Classical approach corresponds to an abstract mission plan search, *i.e.,* the high-level planner reasons only on logical predicates representing the environment. We call "hybrid

approach", the search for a plan in which motions are computed by solving a shortest path problem: given a weighted graph $\langle V, E, l : E \rightarrow R \rangle$, find a path β from v to v' so that $\sum l(e),\ e \in R$ is minimal.

The planning domain used in the first experiment is inspired by the rover domain proposed at the 3rd International Planning Competition[1]: a robot has to collect a set of samples in the environment while avoiding obstacles.

We compare three different methods. First, classical approach with blind search, *i.e.* first waypoint is choosen. Then, classical approach with nearest-first heuristic search, *i.e.* nearest waypoint of the current robot position is chosen. Finally, hybrid approach *i.e.* a specialized reasoner is used. We compare the computation time spent by each method to find a solution plan as well as the the solution quality in terms of traveled distance on 15 examples of increasing complexity.

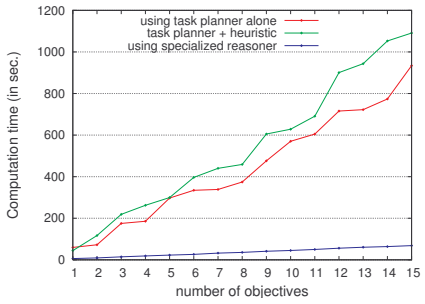


Figure 1: Comparison of the computation time

Figure 1 depicts the evolution of computation time according to the complexity (number of objectives) for the three methods. The computation time is significantly reduced when using the hybrid one. The evolution of computation time when the number of goals increases is slower. Indeed, adding a new goal increases the time by, in average, 50 seconds for classical approaches against 4 sec. for the hybrid approach.

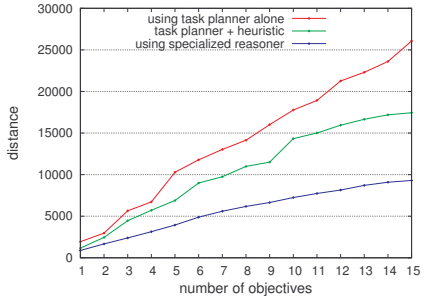


Figure 2: Comparison of the overall distance

Using a specialized reasoner allows to obtain plans with a higher quality in terms of path length than classical approaches (Figure 2) even when a heuristic to improve the overall traveled distance is used.

[1]http://planning.cis.strath.ac.uk/competition/

# errors	hierarchical	hierarchical with BT	interleaved
0	11	11	11
1	22	22	12
2	52	32	13
3	124	41	14
4	292	49	15
5	677	56	16
6	-	62	17
7	-	67	18
8	-	71	19

Table 1: Number of exchanged requests

Moreover, in this experiment, we used a simple path planner on a static problem. Mission planning for real-world problems involves geometric and kinematic constraints when computing robot motions. These constraints cannot be taken into account at the symbolic planning level and the resulting abstract path is not always achievable. In case of failures, replanning loops are necessary. The method for coupling the two planners will strongly impact the time to compute a feasible solution.

2.3 Linking planners

Using a specialized reasoner for some identified subproblems can reduce the computation time and increase the solution quality. The question we address is this section is how to efficiently link the high-level planner with the motion planner.

In three-level architectures, a hierarchy of planners is defined. First, an abstract plan is computed, then motions are refined while taking into account geometric constraints. Another method to bind the two planners is to call the motion planner each time a motion computation is needed.

In the second experiment, we investigate three coupling methods: hierarchical, hierarchical with backtrack and interleaved approaches.In the hierarchical approach with backtrack, in case of error, the planning process is restarted at the unfeasible action while keeping already computed motions. In hierarchical methods, motion requests are sent *after* obtaining a symbolic plan whereas in the interleaved approach, they are sent *during* the task planner execution.

The problem is as follows: a robot has to gather 10 samples in the environment. For each example, a motion error is added, *i.e.* the robot is not at a good location and has to try another action to achieve its mission.

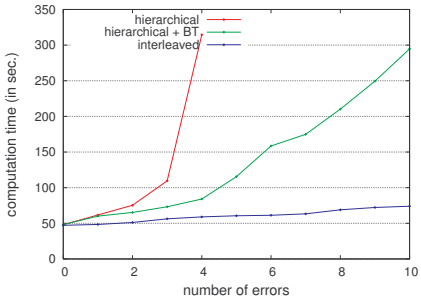


Figure 3: Comparison of the computation time

Figure 3 illustrates comparison of computation time between the three presented coupling methods. Using planning

information to backtrack to the first unfeasible action is more efficient than using a blind hierarchical approach. However, the interleaved method still requires a smaller computational effort. Indeed, if the current specified location is unreachable, the high-level planner can immediatly backtrack and try another action. Table 1 summarizes the number of requests sent by the high-level planner to the motion planner. This number of sent requests evolves linearly with the computation time. This correlation indicates a causal effect between the number of paths to be computed and the required computation time.

In conclusion, interleaved plan computation leads to lower computation time than hierarchical plan computation because of a lower number of path computation requests.

2.4 Related work

Different methods have been explored in order to take into account navigation and motion tasks at the mission planning level. For example, [Brumitt and Stenz, 1998] define a grammar allowing to reason about the mobile parts of the mission. But the most explored approach is to use domain-specific planners to assist a general one [Kambhampati *et al.*, 1993; Lamare and Ghallab, 1998]. These works have been extended in order to take into account geometric preconditions [Guere and Alami, 2001]. However, they do not present an efficient and general framework to bind the high-level mission planner with specialized reasoners: they are dedicated to specific problems or use interaction protocols too specialized.

3 Formalizing the coupling

In section 2, we have seen that a specific reasoner is better suited for managing navigation tasks and that better performances are obtained when calls to this reasoner are interleaved with the high-level planning process.

In this section, we introduce a new formalism in order to express motion conditions for an action achievement at the symbolic level, and an interaction protocol between the task planner and the motion planner.

3.1 Action description

Since robot motions are computed by a low-level planner, symbolic navigation actions are not necessary at the mission planning level. Instead, it can be assumed that motions are preconditions to an action achievement. For example, in order to collect a sample in a specific area, the robot must travel to this area. In this case, the action description can be augmented with the necessary geometric constraints.

Thus, an action description[2] includes two kinds of geometric preconditions called respectively *attitude preconditions* and *behaviour preconditions*. Attitude preconditions are used to express the attitude, *e.g.* its orientation and heading, that the robot must have in order to achieve an action. Behaviour preconditions define the robot behaviour during the action achievement.

Definition 1 (Action) *An action A is defined as a tuple* $\langle\ head(A), pre(A), att\_pre(A), behav\_pre(A),$ *eff(A)* $\rangle$

[2]or *planning operator*: see [Ghallab *et al.*, 2004] for a formal description of planning languages.

<precondition>	::	(<element>*)
<element>	::	<statement> | <rule>
<statement>	::	(<concept> <var_C>)
<var_C>	::	`variable`
<concept>	::	*agent* | *object*
<property>	::	<global_property> | <concept_property>
<global_property>	::	*user-defined global property*
<concept_property>	::	<var_C>.<parameter>
<parameter>	::	*user-defined robot parameter*
<rule>	::	(<function> | <elt_r> `comparator` <elt_r>)
<elt_r>	::	<function> | <property> | `constant`
<function>	::	f_ident(<arg>$_1$,...,<arg>$_n$)
<f_ident>	::	*user-defined function header*
<arg>	::	<elt_r>

Table 2: Formal description of the attitude and behaviour preconditions syntax.

```
;; Operator film_objective
  (Operator (!film_objective ?r ?o)
;; symbolic preconditions
   ((rover ?r)(objective ?o)(camera ?c)
    (has_camera ?r ?c)(is_calibrated ?c))
;; attitude preconditions
   ((agent ?r)(object ?o)
   ((distance(r.pos, o.pos)>= 10)
    (distance(r.pos, o.pos)<= 20)
    (rel_angle(r.pos, o.pos, r.heading) = 90)
    (r.speed = r.speed_max))
;; behaviour preconditions
   ((duration  = 10)
    (constant(r.heading))
    (constant(r.speed)))
;; symbolic effects
   ((has_film ?r ?o))
  )
```

Figure 4: Example of an action description

where $head(A)$ is the identifier of the action, $pre(A)$ is the set symbolic preconditions, *i.e.* high-level conditions allowing to decide if this action can be applied, $att\_pre(A)$ defines the attitude preconditions, $behav\_pre(A)$ is the behaviour preconditions, and *eff(A)* is high level effects that modify the current world state into a new resulting state. Figure 4 presents an example of an action description.

3.2 Expressing constraints

The motion planner has to satisfy a set of geometric constraints given by the attitude and behaviour preconditions $att\_pre(A)$ and $behav\_pre(A)$ whenever the corresponding high-level action is selected. In order to allow communication between the task and motion planners, these constraints must be expressed in a common standardized fashion. This permits message exchange between planners. Table 2 proposes a syntax for this encoding based on *concepts*, *global properties*, *concept properties* and *rules*.

Concepts identify agents and environment objects that will be handled by both planners. A set of properties called *concept properties* is associated with each concept. *Global properties* specify properties of a motion subproblem that have to be respected independently of concepts such as duration of a motion. *Rules* aim at defining robot kinematic constraints as well as constraints between agents and physical objects.

In the example the attitude and behaviour are constrained by 4 and 3 rules reciprocally. This specifies two continuous sets in the state space of the robot.

3.3 Interaction between planners

During the planning process, both planners will interact together. We define two classes of messages: planning messages and system messages. Planning messages contain planning information and are the essential elements of the proposed planning architecture: planning requests and advices. System messages aim at guiding the planning process: system requests, acknowledgement and error messages, and advice queries.

Planning requests are sent by the task planner to the motion planner and contain the geometric and kinematic constraints defining a requested motion.

Definition 2 (Planning request) *A planning request R is defined as a tuple* $\langle\, Type(R), Agent(R), Id_{action}(R), \mathcal{C}(\mathcal{R}) \,\rangle$

where $Type(R)$ is the type of the request. For example, *attitude* to define an attitude precondition request, or *behaviour* for a behaviour precondition request. $Agent(R)$ identifies the agent concerned by the request in the case of a multi-agent problem. $Id_{action}(R)$ is an action identifier allowing to maintain a consistancy between the task planner and the path planner processes. $\mathcal{C}(\mathcal{R})$ is the set of geometric constraints to be respected during robot motions.

Advices are sent by the motion planner to the high-level planner and aim at helping it in its choices. We define three different type of advice: heuristic, optimization and repair.

Definition 3 (Advice) *An advice A is defined as a tuple* $\langle\, Type(A), Content(A), Agent(A), set\{Id_{action}(A)\} \,\rangle$

$Type(A)$ is the type of the advice: heuristic, optimization or repair. $Content(A)$ is the content of the advice. $Agent(A)$ identifies which agent is concerned by the advice. $set\{Id_{action}(A)\}$ is a set action identifiers concerned by this advice.

System requests are sent by an interface defined between planners and aim at initializing and controlling the interaction between planners.

Acknowledgments and error messages are sent by the motion planner in response to a planning request. An acknowledgment is sent when a motion has been computed. Error messages are sent when a requested motion is unfeasible, and contain the failure cause. For example, for the problem of section 2, if there is a constraint on path length, the path planner would return an error message when the shortest path is longer than the maximal value.

Advice queries are sent by the high-level planner when information about the environment is needed in order to choose the next action to be undertaken. However, advices can be sent even without an advice query from the task planner.

3.4 Global overview of the planning architecture

Figure 5 presents the place of the planning module in the robotic architecture. At the initialization stage, the prediction module provides necessary information in order to solve a problem. These informations are a set of symbolic action descriptions, *i.e.* descriptions of actions the robot can perform, the environment description, and the mission to achieve. When a plan is found, it is sent to the execution module. This solution plan contains the actions and motions computed by both planners.

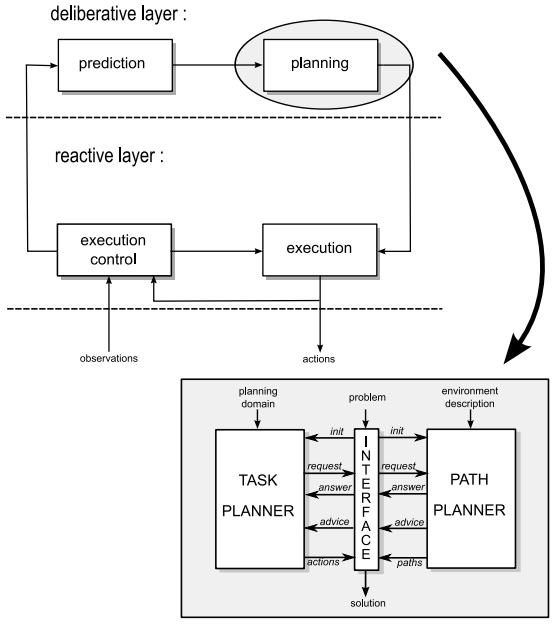


Figure 5: Description of the planning module and its place in a robotic architecture

4 Conclusion and perspectives

In this paper, we proposed a preliminary formalization of a general framework that aims at coupling a high-level mission planner with a low-level motion planner. This coupling tries to overcome drawbacks of classical robotic architectures.

Our contribution is based on a enhanced description of action that allows to express geometric and kinematics constraints, and on a structured interaction between both planners. The two main communication primitives are planning requests to transmit geometric constraints and advices to use the motion planner expertise.

In future work, this architecture will be tested on search-and-rescue missions for a team of autonomous aerial vehicles.

References

[Alami *et al.*, 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17:31–337, 1998.

[Brumitt and Stenz, 1998] B. L. Brumitt and A. Stenz. GRAMMPS: A generalized mission planner for multiple mobile robots in unstructured environments. In *IEEE International Conference on Robotics and Automation*, pages 1564–1571, 1998.

[Estlin *et al.*, 2001] T. Estlin, R. Volpe, I. Nesnas, D. Mutz, F. Fisher, B. Engelhardt, and S. Chien. Decision-making in a robotic architecture for autonomy. In *6th International Symposium on AI, Robotics and Automation in Space*, 2001.

[Ghallab *et al.*, 2004] M. Ghallab, D. Nau, and P. Traverso. *Automated planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Guere and Alami, 2001] E. Guere and R. Alami. Let's reduce the gap between task planning and motion planning. In *IEEE International Conference on Robotics and Automation*, pages 15–20, 2001.

[Kambhampati *et al.*, 1993] S. Kambhampati, M.R. Cutkosky, J.M. Tenenbaum, and S.H. Lee. Integrating general purpose planners and specialized reasoners: case study of a hybrid planning architecture. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 23, pages 1503–1518, 1993.

[Lamare and Ghallab, 1998] B. Lamare and M. Ghallab. Integrating a temporal planner with a path planner for a mobile robot. In *AIPS Workshop Integrating planning, scheduling and execution in dynamic and uncertain environments*, pages 144–151, 1998.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Planning and Acting with an Integrated Sense of Space

N. Hawes[1] and **H. Zender**[2] and **K. Sjöö**[3] and **M. Brenner**[4] and **G.J.M. Kruijff**[2] and **P. Jensfelt**[3]

[1]Intelligent Robotics Lab, School of Computer Science, University of Birmingham, UK

[2]Language Technology Lab, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

[3]Centre for Autonomous Systems, Royal Institute of Technology (KTH), Stockholm, Sweden

[4]Institute for Computer Science, Albert-Ludwigs-Universität, Freiburg, Germany

[1]nah@cs.bham.ac.uk, [2]{zender, gj}@dfki.de, [3]{krsj,patric}@csc.kth.se, [4]brenner@informatik.uni-freiburg.de

Abstract

The paper describes PECAS, an architecture for intelligent systems, and its application in the Explorer, an interactive mobile robot. PECAS is a new architectural combination of information fusion and continual planning. PECAS plans, integrates and monitors the asynchronous flow of information between multiple concurrent systems. Information fusion provides a suitable intermediary to robustly couple the various reactive and deliberative forms of processing used concurrently in the Explorer. The Explorer instantiates PECAS around a hybrid spatial model combining SLAM, visual search, and conceptual inference. This paper describes the elements of this model, and demonstrates on an implemented scenario how PECAS provides means for flexible control.

1 Introduction

Recently there has been an enormous increase in R&D for domestic robot assistants. Moving beyond the Roomba, more complex robot "gophers" are envisioned, to assist in performing more demanding tasks in human environments. To achieve this vision, the study of integrated robotic systems that fulfill many different requirements is necessary.

Research on individual aspects of such systems has yielded impressive robots, e.g. the museum guides Rhino [Burgard *et al.*, 2000] and Robox [Siegwart and et al., 2003], or the in-store assistant ShopBot [Gross *et al.*, 2008]. Other robots, like RoboVie [Ishiguro *et al.*, 2001], Mel [Sidner *et al.*, 2004], BIRON [Peltason *et al.*, 2009], or our CoSy systems [Hawes *et al.*, 2007; Kruijff *et al.*, 2007] provide capabilities for the robot to interact with a human using spoken dialogue. As impressive as they are, all these systems lack the wide range of capabilities needed by a versatile robotic assistant. Producing such a system by integrating the results of specialized subfields such as control, perception, reasoning, and dialogue remains a major challenge to AI and robotics.

If we wish to build a mobile robotic system that is able to act in a real environment and interact with human users we must overcome several challenges. From a system perspective, one of the major challenges lies in producing a single intelligent system from a combination of heterogeneous specialized modules, e.g. vision, natural language processing, hardware control etc. Ideally this must be done in a general-purpose, extensible and flexible way, with the absolute minimum of hardwired behaviors. This both allows solutions to be reused in different systems (allowing an understanding of the design trade-offs to be obtained), and for the same system to be altered over time as requirements change. Additionally, taking account of the "human in the loop" poses the challenge of relating robot-centric representations to human-centric conceptualizations, such as the understanding of large-scale space [Kuipers, 1977].

In this paper we present PECAS (see Section 2), our novel approach to integrating multiple competences into a single robotic system. PECAS allows us to address many of the previously described problems in an architectural way, providing an approach that is ultimately resuable in other robots and domains. For a general-purpose architecture to be deployed it must be *instantiated* with task-specific content. Section 3 presents the Explorer system, our instantiation of PECAS in an interactive mobile robot. Following this we use the Explorer instantiation to present examples of PECAS as a control system (in a general sense). Section 4 presents a complete system run from our implementation, demonstrating how the flow of information and control passes between low and high levels in our system. Section 5 discusses control in PECAS in general, and the strengths and weaknesses of our approach.

2 The PECAS Architecture

Our recent work on intelligent robotics has led to the development of the PlayMate/Explorer CoSy Architecture Sub-Schema (**PECAS**). PECAS is an information-processing architecture suitable for situated intelligent behavior [Hawes *et al.*, 2009]. The architecture is designed to meet the requirements of scenarios featuring situated dialogue coupled with table-top manipulation (the **P**layMate focus [Hawes *et al.*, 2007]) or mobility in large-scale space (the **E**xplorer focus [Zender *et al.*, 2008]). It is based on the CoSy Architecture Schema (**CAS**), which structures systems into *subarchitectures* (SAs) which cluster *processing components* around *shared working memories* [Hawes *et al.*, 2007]. In PECAS, SAs group components by function (e.g., vision, communication, or navigation). All these SAs are active in parallel, typically combining reactive and deliberative forms of processing, and all operating on SA-specific representations (as

is necessary for robust and efficient task-specific processing). These disparate representations are unified, or *bound*, by a *subarchitecture for binding* (binding SA), which performs abstraction and cross-modal information fusion on the information from the other SAs [Jacobsson *et al.*, 2008]. PECAS makes it possible to use the multiple capabilities provided by a system's SAs to perform many different user-specified tasks. In order to give the robots a generic and extensible way to deal with such tasks, we treat the computation and coordination of overall (intentional) system behavior as a *planning* problem. The use of planning gives the robot a high degree of autonomy: complex goal-driven behaviors need not be hard-coded, but can be flexibly planned and executed by the robot at run-time. The robot can autonomously adapt its plans to changing situations using *continual planning* and is therefore well suited to dynamic environments. Relying on automated planning means that tasks for the robot need to be posed as goals for a planner, and behavior to achieve these goals must be encoded as actions that the planner can process. The following sections expand upon these ideas.

2.1 Cross-Modal Binding

Cross-modal binding is an essential process in information-processing architectures which allow multiple task-specialized (i.e., *modal*) representations to exist in parallel. Although many behaviors can be supported within individual modalities, two cases require representations to be shared across the system via binding. First, the system requires a single, unified view of its knowledge in order to plan a behavior that involves more than one modality (e.g., following a command to do something relative to the object or area). Second, binding is required when a subsystem needs information from another one to help it solve a problem (e.g., using visual scene information to guide speech recognition [Lison and Kruijff, 2008]).

Our approach to binding underlies much of the design and implementation of our systems, and so we will reiterate it here (for more details see [Jacobsson *et al.*, 2008]). Each PECAS SA that wishes to contribute information to the shared knowledge of the system must implement a *binding monitor*. This is a specialized processing component which is able to translate from an arbitrary modal representation (e.g., one used for spatial modeling or language processing) into a fixed *amodal* (i.e., behavior neutral) representation. Across a PECAS system the binding monitors provide a parallel abstraction process mapping from multiple, different representations to a single, predicate logic-like representation. Binding monitors deliver their abstracted representations into the binding SA as binding *proxies* and *features*. Features describe the actual abstract content (e.g., color, category, or location) in our amodal language, whilst proxies group multiple features into a single description for a piece of content (such as an object, room, or person), or for relationships between two or more pieces of content. The binding SA collects proxies and then attempts to fuse them into binding *unions*, structures which group multiples proxies into a single, cross-system representation of the same thing. Groupings are determined by feature matching. Figure 1 illustrates this: the SA for navigation (nav SA) and the SA for conceptual mapping and reasoning (coma SA),

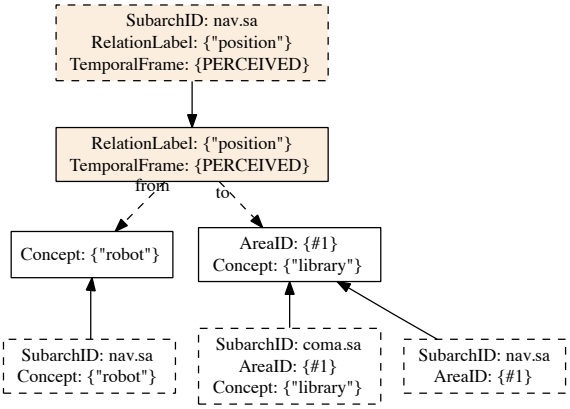


Figure 1: Binding localization and conceptual information: "the robot is in the library." Proxies have dashed borders, unions solid borders. Relation proxies, -unions are colored.

provide their information to the binding SA. Throughout this process links are maintained between all levels of this hierarchy: from modal content, to features and proxies, and then on to unions. These links act like pointers in a programming language, facilitating access to information content regardless of location. Binding thus supports the two identified cases for cross-modal binding: the collection of unions provide a single unified view of system knowledge, and cross-subsystem information exchange is facilitated by linking similarly referring proxies into single union.

2.2 Planning for Action and Processing

For PECAS we assume that we can treat the computation and coordination of overall system behavior as a planning problem. This places the following requirements on PECAS: it must be able to generate a state description to plan with; system-global tasks for the robot need to be posed as goals for a planner; and behavior to achieve these goals must be encoded as actions which can be processed by the planner. In our implementation we use the MAPSIM continual planner and its Multi-Agent Planning Language (MAPL) [Brenner and Nebel, 2009]. In MAPL, we can model beliefs and mutual beliefs of agents as well as operators affecting these, i.e., perceptual and communicative actions. The continual planner actively switches between planning, execution, and monitoring in order to gather missing goal-relevant information as early as possible.

To provide a planning state, the planning SA automatically translates from the unions in the binding SA into MAPL. The planner thus automatically receives a unified view of the system's current knowledge. As we maintain links from unions back to modal content, our planning state, and therefore our plans, remain grounded in representations close to sensors and effectors. In PECAS, planning goals arise as modal *intentional content* which is then abstracted via binding monitors and placed in the planning SA's working memory. From here we use the same translation method as is used on the planning state to produce MAPL goals for the planner.

While the traditional use of planning is achieving goals in the world using physical actions, such direct interpretations of behavior are the exception rather than the rule in cognitive robotics (cf. [Shanahan, 2002]). Here, where infor-

mation is incomplete, uncertain, and distributed throughout subsystems, much of the actions to be performed by the system are to do with processing or moving *information*. Whilst some information processing may be performed continually (e.g., SLAM), much of it is too costly to be performed routinely and should instead be performed only when relevant to the task at hand, i.e., it should be planned based on context.

Underlying our approach to information-processing is the functionally decomposed, concurrently active, structure of PECAS. As each SA is effectively a self-contained processing unit, our design leads naturally to an integration strategy: each SA is treated as a separate agent in a multi-agent planning problem. A crucial feature of this strategy is that each SA's knowledge is separate within the planning state, and can only be reasoned about using epistemic operators (i.e., operators concerned with knowledge). Likewise, goals are often epistemic in nature, e.g., when a human or a SA wants to query the navigation SA for the location of an object.

To realize internal and external information exchange each SA can use two epistemic actions, *tell-value* and *ask-value*, coupled with two facts about SAs, *produce* and *consume*. The actions provide and request information respectively. The facts describe which SAs can produce and consume which predicates (i.e., where certain types of information can come from and should go). For example, if a human teacher tells our robot that "this is the kitchen," this gives rise to the motivation that all SAs which consume room knowledge (e.g., coma SA described in the next section) should know the type of the room in question. This may lead to a plan in which the SA for situated dialogue (comsys SA) uses a tell-value action to give the coma SA this information.

Using this design, planning of information-processing becomes a matter of planning for epistemic goals in a multi-agent system. This gives the robot more autonomy in deciding on the task-specific information flow through its subsystems. But there is another assumption underlying this design: whilst the binding SA is used to share information throughout the architecture, not all information in the system can or should be shared this way. Some of it is unavailable because it is modality specific, and even cross-modal knowledge is often irrelevant to the task at hand. If all information was shared this would overwhelm the system with (currently) irrelevant information (e.g., lists of all the people, rooms, objects, object categories etc. that parts of the system know about). Thus, in order to restrict the knowledge the planner gives "attention" to without losing important information, it needs to be able to *extend* its planning state on-the-fly, i.e., during the continual planning process. In PECAS state extension can be done using ask-value and tell-value actions, and results in a process we call *task-driven state generation*.

3 The Explorer Instantiation

The binding and planning SAs described above are system and scenario independent. We now discuss the Explorer-specific SAs to describe concrete functionality and how this relates to system control. All SAs have been implemented in CAST (an open-source toolkit implementing the CAS schema) and tested on an ActivMedia PeopleBot. Figure 2 shows all SAs used in the Explorer PECAS instantiation. Most components used in the different SAs have been discussed in detail in earlier work (references provided below).

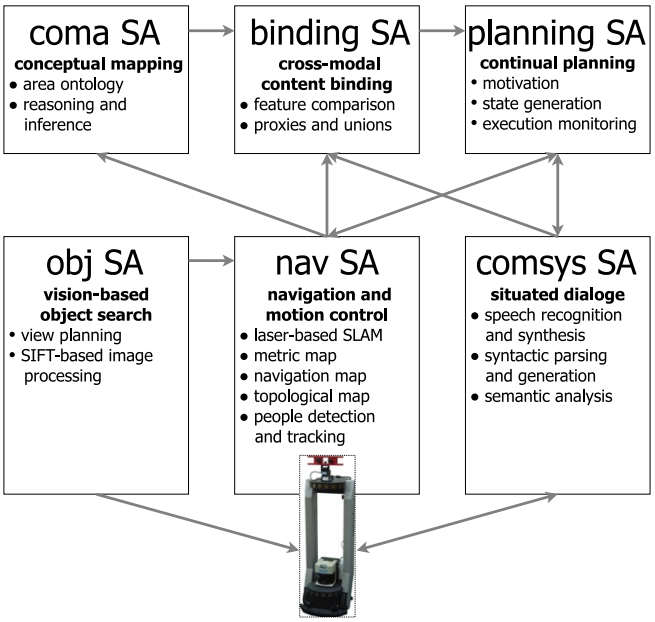


Figure 2: The Explorer Architecture

For a mobile robotic system that is supposed to act and interact in large-scale space, an appropriate spatial model is key. The Explorer maintains a multi-layered conceptual spatial map of its environment [Zender *et al.*, 2008]. It serves as a long-term spatial memory of large-scale space. Its individual layers represent large-scale space at different levels of abstraction, including low-level metric maps for robot motion control, a navigation graph and a topological abstraction used for high-level path planning, and a conceptual representation suitable for symbolic reasoning and situated dialogue with a human. In the Explorer, different SAs represent the individual map layers. For the details on human-augmented map acquisition see [Kruijff *et al.*, 2007].

nav SA The SA for navigation and spatial mapping hosts the three lowest levels of the spatial model (metric map, navigation map, and topological layer). For low-level, metric mapping and localization the nav SA contains a module for laser-based SLAM. The nodes and edges of the *navigation map* represent the connectivity of visited places, anchored in the metric map through x-y-coordinates. Topological areas, corresponding roughly to rooms in human terms, are sets of navigation nodes. This level of abstraction in turn feeds into the conceptual map layer that is part of the coma SA.

The nav SA contains a module for laser-based people detection and tracking [Zender *et al.*, 2007]. The nav SA binding monitor maintains the robot's current spatial position and all detected people, as proxies and relations on the binding SA. The smallest spatial units thus represented are areas. This provides the planner with a sufficiently stable and continuous description of the robot's state. The planning SA can pose *move* commands to the nav SA. The target location is defined based on the current task which might be to follow a person, move to a specific point in space, etc. Move commands are executed by a navigation control module, which

performs path planning on the level of the navigation graph, but automatically handles low-level obstacle avoidance and local motion control.

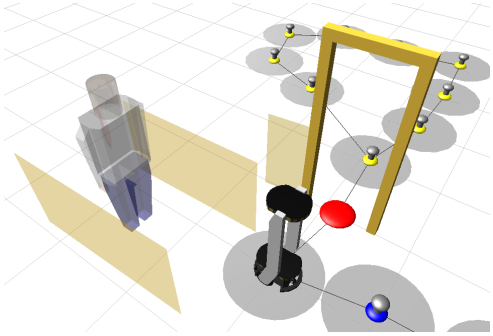
obj SA The SA for vision-based object search contains the components for finding objects using vision. It consists of a module for view planning and one for visual search. The view planning component creates a plan for which navigation nodes to visit, in what order and in what directions to look. Details of the process can be found in [Gálvez López *et al.*, 2008]. The visual search consists of SIFT feature matching directly on acquired images. Objects that are found are published on the obj SA working memory. The nav SA detects this and in turn extends the spatial model with the new objects. This then propagates the information to the coma SA and, if and when necessary, to the binding SA.

coma SA The SA for conceptual mapping and reasoning maintains an abstract symbolic representation of space suitable for situated action and interaction. It represents spatial areas (nav SA), objects in the environment (obj SA), and abstract properties of persons (e.g., ownership relations) in a combined A-Box and T-Box reasoning framework based on an OWL-DL reasoner, which can infer more specific concepts for the area instances [Zender *et al.*, 2008]. The coma SA makes its information available to the binding SA on demand, i.e., whenever planning SA sends an *ask-val* command to the coma SA, it will add its knowledge about spatial entities, especially their most specific concepts. In our system the explicit definitions of area concepts through occurrences of certain objects are also used to raise expectations about typical occurrences of certain objects. If the planning SA needs to know the location of an object that has not been encountered before, it can query the coma SA, which will then provide a *typical* location of the object in question. This is done via special T-Box queries involving the OWL-DL definitions of concepts. An example of this will be discussed in Section 4.

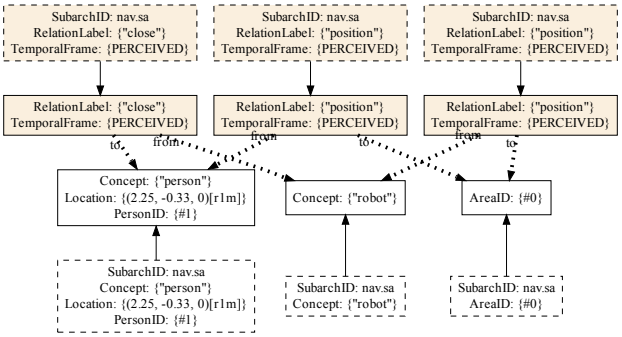
comsys SA The subarchitecture for situated dialogue processing has a number of components concerned with understanding and generation of natural language utterances [Kruijff *et al.*, 2009]. Speech recognition converts audio to possible text strings, which are subsequently parsed. Parsing produces a packed representation of logical forms (LFs) that correspond to possible semantic interpretations of an utterance. Finally, the semantics are interpreted against a model of the dialogue context. Content is connected to discourse referents, being objects and events talked about over the course of an interaction. In the dialogue context model, both the content of the utterance and its intent are modeled. All of this information is communicated to the planning SA and the binding SA through proxies representing the indexical and intentional content of the utterances. In rough terms the indexical content (information about entities in the world) is used by the binding SA to link with information from other modalities. Meanwhile the intentional content (information about the purpose of the utterance) is used by the planning SA to raise goals for activity elsewhere in the system [Kruijff *et al.*, 2009].

4 Example: Finding a book

This section presents a scenario in which a human asks the Explorer to perform a task. It shows how PECAS controls system behavior and information-processing. The example is taken directly from our implemented system, showing system visualizations (with minor post-processing).



(a) Screenshot of the visualization tool



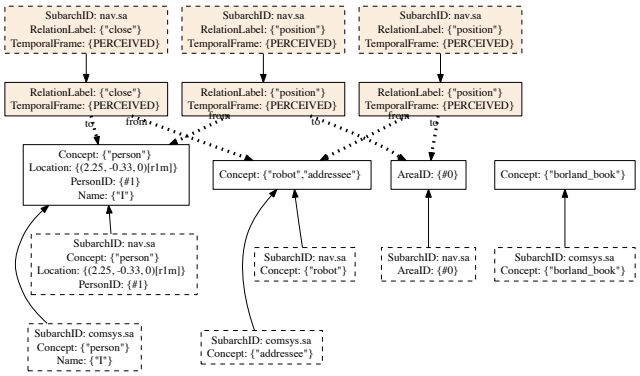
(b) Contents of binding working memory

Figure 3: Initial situation: the user approaches the robot

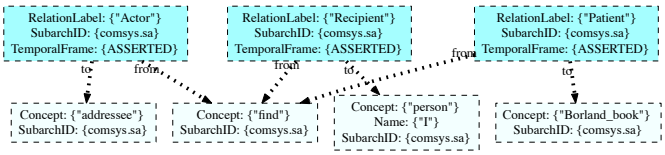
The system starts in the spatial context and binding state visualized in Figure 3: the robot and person are occupying the same area, and the person is close to the robot. The *robot proxy* is provided by the nav SA which it abstracts from its representation of the robot pose. The *person proxy* is provided by the nav SA because a person is being tracked. In addition to these, the nav SA makes available a proxy for the *area* in which one of these proxies occurs, linking them with a *position* relation proxy. Finally, the *close* relation proxy connects the robot proxy to the proxy of the person because the person is geometrically close to the robot. Note that no objects are present, nor are other areas except the current area.

Next, the human approaches the robot and says "find me the Borland book". The comsys SA interprets this utterance, presenting the elements of its interpretation to the rest of the system as proxies. Figure 4a shows the results. The Explorer itself (the recipient of the order) is represented by a proxy with Concept `addressee`, which binds to the robot proxy already present. The word "me" refers to the speaker, and generates a "person" proxy identified by the Name feature `I`. The expression referring to the book is given by a "Borland_book" proxy, not yet bound to any other proxies.

The comsys SA can determine the intention of this utterance, and separates the intentional elements of the interpretation from the aforementioned descriptive proxies. This intentional content is written to planning SA as a proxy structure with links back to the binder. The structure of this *motive* can be seen in Figure 4b. Planning SA, detecting a new mo-



(a) State of binding SA



(b) Representation of intentional content (as a *motive*)

```
Objects:                    Facts:
(area_id_0 - area-id)       (area-id gensym1 : area_id_0)
(gensym0 - robot)           (area-name area_id_0 : gensym1)
(gensym1 - area-name)       (perceived-pos gensym0 : area_id_0)
(gensym4 - person)          (perceived-pos gensym4 : area_id_0)
(gensym6 - movable)         (close gensym4 gensym0 : true)
```

(c) Planning state after processing the intentional content

Figure 4: State after the user has uttered the command "Find me the Borland Book."

tive, begins the process of creating a plan to fulfill it. First, it converts the information on the binder (Figure 4a) to the MAPL representation in Figure 4c. In this process unions become objects and predicates in the planning state. E.g., as the person union is related by a position relation union to an area union, this will be expressed to the planner as `(perceived-pos gensym4 : area_0)`, where `gensym4` is an auto-generated planning symbol referring to the person, and `area_0` refers to the area. The planner similarly converts the motive from Figure 4b into a MAPL goal `(K gensym4 (perceived-pos gensym6))`. This can be read as the Explorer having the goal of the the person knowing the position of the book. We use this interpretation of the command "Find me...", as the robot does not have the ability to grasp objects.

Given this state and goal, the planner creates a plan:

```
L1: (negotiate_plan gensym0 coma_sa)
L2: (tell_val_asserted-pos
        coma_sa gensym0 gensym6)
L3: (find_a gensym0 gensym6 gensym0)
L4: (tell_val_perceived-pos
        gensym0 gensym4 gensym6)
```

This plan states that the Explorer must find the location of the book (`L3`), then report this location to the person (`L4`). Before it does this it must negotiate with the coma SA (as each subarchitecture is treated as a separate agent) to provide a location where it might be able to find the book (`L1`,`L2`). The reasoning behind this plan is that Explorer must provide the person with a perceived location for the book (as is specified in the goal), and, having not seen it recently, the only way to obtain a perceived location is via its object search functionality. To perform an object search the system must have both an object to search for (the book in this case) and an area to search. *Typical* positions of objects (as opposed to their *perceived* positions) are stored in the ontology in coma SA. Rather than make all of this knowledge available via binding by default (a choice which would add many extra and redundant facts to the planning state), typical positions are offered by coma SA using a `produce` fact (see Section 2.2). This allows the planner to query coma SA for typical positions when it requires them. One advantage of this on-demand state generation is that the comsys SA could also be used to provide the same knowledge (and would be if the book was not found initially). In the above plan, the planner makes use of this by

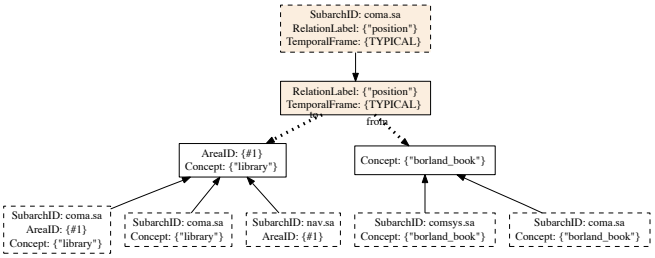


Figure 5: Hypothetical position of the Borland book.

getting the coma SA to `tell-val` the typical position of the Borland book to the binding SA. This yields the information that, as it is a book, it would typically be found in a library. Along with this, the coma SA also volunteers the specific information it has on libraries: an area exists in its map that is a library. This is illustrated in Figure 5.

Given this hypothesis for the book's location, MAPSIM uses a replanning step to expand the initial plan to include steps to move the robot to the library, search there for the book, then move back and report to the user. The updated plan and planning state are now as follows:

```
Objects:
(area_id_0 - area-id)   (area_id_1 - area-id)
(gensym0 - robot)       (gensym1 - area-name)
(gensym4 - person)   (gensym6 - borland_book)
(gensym6 - movable)     (gensym7 - area-name)

Facts:
(area-id gensym1 : area_id_0)
(area-id gensym6 : area_id_1)
(area-name area_id_0 : gensym1)
(area-name area_id_1 : gensym7)
(asserted-pos gensym6 : gensym7)
(perceived-pos gensym0 : area_id_0)
(remembered-pos gensym4 : gensym1)

Plan:
L1: (move gensym0 area_id_1 area_id_0)
L2: (object-search-in-room
        gensym0 gensym6 area_id_1)
L3: (approach-person
        gensym0 gensym4 area_id_0)
L4: (tell_val_perceived-pos gensym0
        gensym4 gensym6)
```

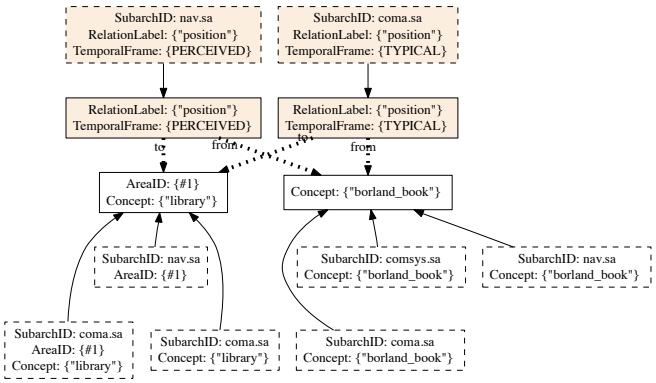


Figure 6: Perceived location of the book

In the above, `gensym7` is the binding union of the library. Using the **AreaID** feature from the this union, the planner issues a command to the nav SA which moves the robot to the library (fulfilling step `L1`). As with all other steps in the plan (including the information-processing ones), the results of this action are checked by MAPSIM to determine whether it has completed successfully or whether replanning is required. This check is performed by inspecting the planning state and comparing it to the expected state. This means that all actions must have effects that are visible on the binding SA (for subsequent translation). Once the check has passed for `L1` (confirming the robot has arrived in the library), the planner issues an object search command to the object SA. The Explorer searches the room as described previously. Once the object is found, the nav SA adds it to the navigation graph. Since it is part of the current spatial context, it is also exported to the binder in the form of an object proxy, which is connected to the room's proxy by a new position proxy. This position proxy has a `PERCEIVED` temporal frame.

The new proxies generated by object search bind to the existing complex (pictured in Figure 5), resulting in the structure in Figure 6. This binding provides the original comsys SA book proxy with a perceived position (in addition to its typical one). With this knowledge in the planning state (i.e., the effect of `L2` is verified, which satisfies one precondition of `L4`), the planner is able to trigger the remaining steps in the plan: moving to the user and reporting the perceived position. A move command is send to the nav SA referencing the **Location** feature from the person proxy. Once close to this location, a `tell-val` is sent to the comsys SA to communicate the book's location to the person. A content generation component in the comsys SA uses the contents of the binder (see Figure 6) to generate the utterance "the Borland book is in the library", thus completing the plan and satisfying the original goal (that the person knows the position of the book).

5 Discussion

The preceding sections illustrate how our architectural ideas come together to create a control system for intelligent behavior. Although the surface form of the scenario does not present much in the way of novel interactions, the PECAS architecture and the multi-level spatial representation provide a novel system-level approach with a number of important features. Including the binding SA in the architecture allows multiple modalities to collaborate on problems that a single modality in isolation would not be able to solve. E.g., in the Explorer the comsys SA initially provides a description of an object based on natural language input, conceptual mapping then extends this description, and vision finally completes it. Whilst other systems may include elements of cross-modal fusion, we have taken the additional, novel step of using the results of fusion to provide input to a continual planner. This allows the behavior of multiple modalities to be marshalled in pursuit of system goals in a general, extensible manner. Using continual planning PECAS achieves this in a way that is responsive to external change and certain types of failure. In PECAS all this is true both of actions that have physical effects, and of internal, information-processing actions.

Both the theory and implementation of the Explorer system and PECAS are works in progress, so we can identify many areas that need further study, or currently limit our approach. E.g., while the the use of MAPSIM provides many of the strengths of our work, planning occurs at quite a high level of abstraction. This consequently also applies to interactions between subarchitectures, and between the Explorer and the world. Whilst this has some advantages (e.g., subsystems are free to interpret commands in modality specific ways, something discussed in more detail below), it may be a hindrance to more closely coupled interactions between behaviors, such as positioning the robot to see an object that it is trying to pick up. Also, actions used in the PECAS architecture must have effects that are visible at the level of binding proxies, which may not hold for actions that have effects in a single SA. Our system also relies on many translations between formalisms. Whilst our structured support for this (via binding) is a clear strength, in practice the translations can become somewhat arbitrary and hard to maintain.

5.1 Approaches to Control

The HYCAS workshop aims to investigate issues of hybrid control in autonomous systems, so what lessons can we learn from the work presented here? In all PECAS systems we have a number of control patterns working in parallel. At the lowest level we can reasonably discuss in terms of architecture, components typically run in one of two modes. Either they perform continuous processing which provides a stream of data to working memory, or they wait for a particular event which triggers some processing (which may or may not result in a change to working memory). In this case events may either be external to the system (e.g., a sensor, such as a microphone, being triggered), or internal (where an event describes a change to working memory contents). Within a SA these types of processing behaviors happen concurrently (with SAs also working in parallel to each other). Control of SA-level processing is typically constrained at design-time, when components are set to listen to particular types of events. At run-time these events, and mediated access to the information they describe, provide implicit synchronization during processing. Thus PECAS does not provide explicit control strategies within SAs (although a few common control strategies tend to be reused).

As described in the preceding sections, the path to high-level control in PECAS comes via SAs exposing modal con-

tent to the rest of the system via the binding system. The process by which this occurs plays a major role in system control. Binding monitors provide abstracted representations of SA-local content. They typically do this based on three different triggers: SA-internal events, SA-external events, and on-demand. The first of these is the most basic case: the generation of a new local representation triggers the SA's binding monitor to generate a proxy. This happens in the Explorer for discourse referents in the comsys SA. The second case, SA-external events, typically provides a way for the existing binding state to influence the generation of further proxies (one of the limited, distributed, forms of attention in PECAS): the monitor listens for both SA-internal and -external events, then, when some particular events co-occur, it generates a proxy from some local content. This happens in the Explorer when the conceptual mapping SA provides proxies in response to proxies generated by other SAs (e.g., when the comsys SA generates a proxy for an object, the coma SA provides additional proxies to bind with it). The final case, on-demand monitor operation, occurs when a binding monitor is explicitly asked (rather than implicitly triggered) to provide information about a particular entity already represented in the binding SA. This approach is used by the system to deliberatively add information to the binding system. This is typically done during the planning process (as an element of on-demand state generation).

The first two of these binding monitor triggers represent additional design-time control decisions within PECAS systems. The designer explicitly chooses which SA and system events should cause information to be shared via binding (and thus added to the planning state for system control). The underlying assumption is that the system will need high-level access to this information regardless of context, and therefore this hard-wired approach is acceptable. The latter case, on-demand triggering, provides a system with explicit control over the information shared between all SAs and used for planning. We expect this approach, whether driven by planning or other mechanisms, to become the dominant approach in future PECAS systems. The alternative (implicit control over the contents of the binding SA) would place the system entirely at the mercy of reactive control, potentially flooding the binding SA with irrelevant or redundant information.

Binding monitors typically provide two types of abstraction: level-of-detail abstraction and temporal abstraction. The former has been taken for the implicit meaning of "abstraction" in the preceding sections: translation of a complex modal representation into a less complex amodal representation. Temporal abstraction is often implicit in level-of-detail abstraction, but it is important to make its presence explicit as it influences our control approach. Changes within SAs typically occur at a rate linked to the rate of change of sensors used for that SA's modality or the processing schemes used to interpret the results of those sensors. E.g., in the nav SA the pose of the robot is updated by SLAM at 5Hz, in the comsys SA elements of the discourse references are incrementally updated during an utterance interpretation (and across multiple utterances if they are reused), and in the obj SA object positions are updated as close to framerate as the system can manage. If the planner, or any other deliberative system, had to take control decisions using information at this level of description from multiple SAs, its decisions would only be valid for that length of time all of these representations remained unchanged (a number limited by the most volatile item of information). This would make system-wide control rather difficult. Binding monitors ameliorate this problem by only propagating *relevant* changes from the SA level to the binding SA. What constitutes a relevant change is both SA- and task-specific, but often relevance is coupled to the potential of the change to significantly alter the global state of the system. Temporal abstraction occurs because significant changes typically do not occur at the same rate as all changes; they often happen much less frequently. This highlights the close coupling between temporal abstraction and level-of-detail abstraction, as the latter defines our global state. This fact is often relied upon in systems which operate on multiple levels of abstraction. In this sense the role binding plays in PECAS can be meaningfully compared with the definition of an *interface layer* in the work of Wood (e.g., [Wood, 1994]). Interface layers are where a designer identifies critical points in the representations used by a system. These are points at which the representations become suitable for particular types of reasoning tasks. The identification of these layers is crucial for system control; they provide a way to match up representations with decision making approaches, e.g., detailed, dynamic representations for reactive control, and more abstract, stable representations for deliberative control. So, to reiterate an important point, unions and proxies (and to some extent the actions used by the planner) represent a stable point in the space of representation used by PECAS systems. Without them we would not be able to use planning (which requires such stability) to control system behavior.

From a control perspective there are two interesting aspects to our use of planning. First, as mentioned previously, we use continual planning: we integrate execution monitoring and replanning into our high-level control system. This provides a form of closed-loop high-level control for our system, where the effects of actions are monitored relative to expectations established by their definitions, and replanning is triggered if these expectations are violated. Second, the planner only has an opaque interface to the actions themselves. Rather than being concerned with how each action is implemented, PECAS only requires that the implementing SA abides by the contract provided by the action definition; otherwise planning and monitoring will fail. This is in contrast to other systems (e.g., 3T [Bonasso *et al.*, 1997]) where high-level control is used to schedule behaviors all the way down to the lowest-level (e.g., skills) too. By adopting a less exacting approach to action execution we allow each SA to interpret the action in a contextually appropriate way. SAs may choose to use one or many components to execute an action and may go through as many intermediate steps as required. This allows a single high-level control action to become a multiple step lower-level action, e.g., when an action results in a dialogue, or a visual search behavior. Of course, this means the planner is unable to directly influence the creation or scheduling of these lower level tasks. This is not a problem in our current domains where actions do not compete for resources across SAs, but in future this could become a problem. Possible

solutions include making the actions available to the planner less coarse but still not providing a one-to-one mapping to SA-internal actions (i.e., giving it tighter control over SA behavior), or annotating actions with resource constraints.

In summary, the overall behavior of a PECAS system, including the Explorer instantiation described in this paper, emerges from the interaction of reactive and deliberative control systems at multiple levels of abstraction. Multiple concurrent components within SAs are controlled implicitly by design-time event-subscription rules, and use CAST's event mechanisms and working memories to synchronism their processing at run-time. Across the system a collection of binding monitors provide an interface at which representations become abstract and stable. This allows a single deliberative control process to interact with the multiple concurrent SAs. It is this interface level which allows a PECAS instantiation to solve some problems with deliberative approaches (e.g., cross-SA coordination) and others with reactive approaches (e.g., within-SA coordination and sensor and effector control) whilst remaining contextually appropriate and responsive to its environment (i.e., no single control strategy ever exclusively takes charge of the entire system). However, this approach currently relies on an external designer fixing the representations either side of the interface level. Whilst this is not necessarily a problem in the short-term, in the future we would like to investigate what properties define a good interface level so that new system designers will not have to make uninformed design decisions.

6 Conclusion

We described PECAS, an architecture for intelligent systems. PECAS is a new architectural combination of information fusion and continual planning. Its purpose is to plan, integrate and monitor the asynchronous flow of information between multiple concurrent systems to achieve a task-specific system-wide goal. We used the Explorer instantiation to show how this works out in practice. The Explorer instantiates PECAS around a hybrid spatial model combining SLAM, visual search, and conceptual inference, with the possibility to use spoken dialogue to interact with a human user. We described the elements of this model, and demonstrated using a realistic (and implemented) scenario how PECAS provides a novel approach to control for autonomous systems.

Acknowledgements

Supported by the EU FP7 ICT Cognitive Systems Integrated Project "CogX" (FP7-ICT-215181-CogX) and in part by the Swedish Research Council, contract 621-2006-5420.
For more information see `http://cogx.eu`.

References

[Bonasso *et al.*, 1997] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack. Experiences with an architecture for intelligent, reactive agents. *J. of Experimental and Theoretical Artificial Intelligence*, 9(2-3):237–256, 1997.

[Brenner and Nebel, 2009] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Aut. Agents and Multi-Agent Sys.*, 2009. accepted for publication.

[Burgard *et al.*, 2000] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1–2), 2000.

[Gálvez López *et al.*, 2008] D. Gálvez López, K. Sjö, C. Paul, and P. Jensfelt. Hybrid laser and vision based object search and localization. In *ICRA '08*, pages 2636–2643, 2008.

[Gross *et al.*, 2008] H. M. Gross, H. J. Böhme, C. Schröder, S. Müller, A. König, C. Martin, M. Merten, and A. Bley. ShopBot: Progress in developing an interactive mobile shopping assistant for everyday use. In *SMC '08*, pages 3471–3478, 2008.

[Hawes *et al.*, 2007] N. Hawes, A. Sloman, J. Wyatt, M. Zillich, H. Jacobsson, G. J. Kruijff, M. Brenner, G. Berginc, and D. Skočaj. Towards an integrated robot with multiple cognitive functions. In *AAAI '07*, pages 1548–1553, 2007.

[Hawes *et al.*, 2009] N. Hawes, M. Brenner, and K. Sjöö. Planning as an architectural control mechanism. In *HRI '09*, pages 229–230, New York, NY, USA, 2009. ACM.

[Ishiguro *et al.*, 2001] H. Ishiguro, T. Ono, M. Imai, T. Maeda, T. Kanda, and R. Nakatsu. Robovie: an interactive humanoid robot. *Int. J. Industrial Robot*, 28(6):498–503, 2001.

[Jacobsson *et al.*, 2008] H. Jacobsson, N. Hawes, G. J. Kruijff, and J. Wyatt. Crossmodal content binding in information-processing architectures. In *HRI '08*, 2008.

[Kruijff *et al.*, 2007] G. J. Kruijff, H. Zender, P. Jensfelt, and H. I. Christensen. Situated dialogue and spatial organization: What, where... and why? *International Journal of Advanced Robotic Systems*, 4(1):125–138, 2007.

[Kruijff *et al.*, 2009] G. J. Kruijff, P. Lison, T. Benjamin, H. Jacobsson, H. Zender, I. Kruijff-Korbayová, and N. Hawes. Situated dialogue processing for human-robot interaction. In H. I. Christensen, G. J. Kruijff, and J. L. Wyatt, editors, *Cognitive Systems*. Springer Verlag, 2009. to appear.

[Kuipers, 1977] B. Kuipers. *Representing Knowledge of Large-scale Space*. PhD thesis, MIT, 1977.

[Lison and Kruijff, 2008] P. Lison and G. J. Kruijff. Salience-driven contextual priming of speech recognition for human-robot interaction. In *ECAI '08*, 2008.

[Peltason *et al.*, 2009] J. Peltason, F. H. K. Siepmann, T. P. Spexard, B. Wrede, M. Hanheide, and E. A. Topp. Mixed-initiative in human augmented mapping. In *ICRA '09*, 2009. to appear.

[Shanahan, 2002] M. Shanahan. A logical account of perception incorporating feedback and expectation. In *KR '02*, pages 3–13, 2002.

[Sidner *et al.*, 2004] C. L. Sidner, C. D. Kidd, C. H. Lee, and N. B. Lesh. Where to look: A study of human-robot engagement. In *IUI '04*, pages 78–84, 2004.

[Siegwart and et al., 2003] R. Siegwart and et al. Robox at expo.02: A large scale installation of personal robots. *Robotics and Autonomous Systems*, 42:203–222, 2003.

[Wood, 1994] S. Wood. *Planning and Decision Making in Dynamic Domains*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[Zender *et al.*, 2007] H. Zender, P. Jensfelt, and G. J. Kruijff. Human- and situation-aware people following. In *RO-MAN '07*, pages 1131–1136, 2007.

[Zender *et al.*, 2008] H. Zender, O. Martínez Mozos, P. Jensfelt, G. J. Kruijff, and W. Burgard. Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493–502, 2008.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Integrated Planning and Execution for Robotic Exploration

Conor McGann
Willow Garage,
Menlo Park, California
mcgann@willowgarage.com

Frédéric Py, Kanna Rajan
Monterey Bay Aquarium Research Institute
Moss Landing, California
{*fpy, kanna.rajan*}*@mbari.org*

Angel Garcia Olaya
Universidad Carlos III de Madrid
&
Monterey Bay Aquarium Research Institute
agolaya@mbari.org

Abstract

This paper uses Constraint-based Temporal Planning (CTP) techniques to integrate deliberation and reaction in a uniform representation for autonomous robot control. We do so by formulating a control structure that *partitions* an agent into a collection of coordinated control loops, with a recurring sense, plan, act cycle. Algorithms are presented for sharing state between controllers to ensure consistency during execution and enable compositional control. The partitioned structure makes it practical to apply CTP for both deliberative and reactive behavior and promises a scalable and robust approach for control of real-world autonomous robots operating in dynamic environments. The resulting framework is independant of the domain and provides a principled approach to building autonomous systems.

1 Introduction

The ocean plays a crucial role in the ecosystem of our planet. This vast, hostile, unstructured and unpredictable environment has proven resistant to extensive scientific study. Its depths are largely inaccessible to humans, and impervious to surface and space based observation techniques. Oceanographic ship-based measurements have recently been augmented by untethered robotic platforms such as Autonomous Underwater Vehicles (AUVs) [Yuh, 2000]. They carry sophisticated science payloads for measuring important water properties [Ryan *et al.*, 2005], as well as instruments for recording the morphology of the benthic environment with advanced sonar equipment [Thomas *et al.*, 2006]. The extensive payload capacity and operational versatility of these vehicles offer a cost-effective alternative to current methods of oceanographic measurements. Fig. 1 shows one such AUV, the *Dorado*, which can operate to depths of 1500m with its mid-section water sampler.

As in other domains, marine robots must be proactive in the pursuit of goals and reactive to evolving environmental conditions. These concerns must be balanced over short and long term horizons to consider timeliness, safety and efficiency. In the ocean for example, a variety of important upper-water column phenomenon such as Intermediate Nepheloid Layers or INLs (fluid sheets of suspended particulate matter that originate from the sea floor), blooms (patches of high biological activity) and ocean Fronts, can occur over a

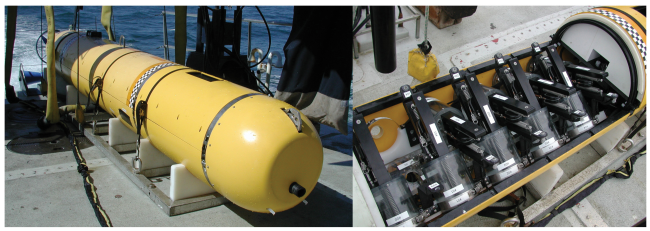


Fig. 1: The *Dorado* vehicle at sea (left) and its mid-body water sampler (right). The vehicle has a range of optical sensors in the nose section, a battery aft of the samplers and the CPU and drive electronics in the tail.

wide range of scales, from thousands of kilometers (fronts and blooms) down to tens/hundreds of meters, each characterized by strong spatio-temporal dependence. This dependence results in a high degree of unpredictability in the location and intensity of the phenomena. Further, when events are detected by surface observations, synoptic multi-disciplinary observations are necessary below the surface in order to understand the oceanographic dynamics influencing these processes. To understand the biological context of such dynamic processes, water samples also need to be returned to shore.

Such science drivers require our AUV to plan a number of survey transects in advance, while being proactive in triggering water samplers within feature hotspots. Additionally, variability of ocean currents and lack of localization underwater also requires the AUV to compensate its navigation for errors to stay within a survey area. To do so, the AUV may reasonably deliberate for many minutes if necessary; in contrast, instrument and vehicle control decisions require faster reaction times but can be taken with a more myopic view of implications to the plan. This suggests that the control responsibilities of the agent can be *partitioned* to exploit differing requirements in terms of which state variables need to be considered together, over what time frames, and at what reaction rates.

Current AUV control systems [Bellingham and Leonard, 1994] are a variant of the behavior-based Subsumption architecture [Brooks, 1986] which rely on manually scripted plans generated a priori. This prevents *in-situ adaptation* of mission structure essential to improving operation in a dynamic environment and to pursue unanticipated science opportunities.

We have developed and deployed an onboard adaptive control system that integrates Planning and Probabilistic State Estimation in a hybrid Executive [McGann *et al.*, 2008b; 2008a]. Probabilistic State Estimation integrates a number

of science observations to produce a likelihood that the vehicle sensors perceive a feature of interest. Onboard planning and execution enables adaptation of navigation and instrument control based on the probability of having detected such a phenomenon. It further enables goal-directed commanding within the context of projected mission state and allows for replanning for off-nominal situations and opportunistic science events. This paper focuses on the impact of partitioning to robust plan synthesis and execution; lack of space prohibits discussion of Probabilistic State Estimation within this framework with no loss of generality of the concepts described.

The novelty of this work is two-fold. We integrate deliberation and reaction systematically over different temporal and functional scopes within a single agent and a single model that covers the needs of high-level mission management, low-level navigation, instrument control, and detection of unstructured and poorly understood phenomena. Secondly, we break new ground in oceanography by allowing scientists to cost-effectively obtain samples precisely within a scientific feature of interest using an autonomous robot. Together our work has resulted in the only operational AUV anywhere being used for routine scientific surveys with onboard deliberation.

The remainder of this paper is organized as follows. Section 2 places this work in the context of other integrated planning and execution frameworks followed by section 3 that introduces key concepts and definitions. The core of the paper is in section 5 on partitioned control which presents algorithms for coordinated control and synchronization of state along with complexity analysis. Finally we conclude in section 6 with empirical data both on shore and at sea.

2 Related Work

The dominant approach for building agent control systems utilize a three-layered architecture [Gat, 1998], notable examples of which include IPEM [Ambros-Ingerson and Steel, 1988], ROGUE [Haigh and Veloso, 1998], the LAAS Architecture [Alami *et al.*, 1998], the Remote Agent Experiment [Muscettola *et al.*, 1998] and ASE [Chien *et al.*, 1999] (see [Knight *et al.*, 2001] for a survey). Scalability is of concern since the planning cycle in these approaches is monolithic often making fast reaction times impractical when necessary. Many of these systems also utilize very different techniques for specifying each layer in the architecture resulting in duplication of effort and a diffusion of knowledge. This work builds on the approach used by IDEA [Muscettola *et al.*, 2002] in utilizing a collection of controllers, each interleaving planning and execution in a common framework. IDEA however, provides no support for conflict resolution between controllers, nor does it provide an efficient algorithm for integrating current state within a controller, relying instead on a possibly exponential planning algorithm. Efficient synchronization of state in a partitioned structure is fundamental to making the approach effective in practice.

The contribution of this paper is in providing a formal basis for partitioning a complex control problem to achieve scalability and robustness and an approach for synchronization in polynomial time. The key ideas in this paper are independent of the domain and provide a principled approach to building autonomous systems.

3 System Overview

We formally define an agent control structure as a composition of coordinated control loops with a recurring sense, plan, act (SPA) cycle. Representational primitives in our framework are based on the semantics of Constraint-based Temporal Plans [Jónsson *et al.*, 2000; Frank and Jónsson, 2003] using unified declarative models.

Each control loop operates on its own partial plan. Feedback maps to new open conditions in the partial plan which are resolved through a synchronization algorithm. Time is discrete, and synchronization occurs when the discrete clock transitions. A planner also operates on this partial plan, resolving flaws over a time horizon specific to that control loop. Flaws for the planner arise from the model and from goals requested from external sources. The planner runs as necessary between discrete time transitions. Execution involves dispatching elements of a partial plan over a selected dispatch time window. In a system with multiple control loops, there will be multiple partial plans. Where information is shared between control loops, the intersection of partial plans (shared state variables over common time horizons) must be maintained. We manage the information flow within the partitioned structure to ensure consistency in order to direct the flow of goals and observations in a timely manner. The resulting control structure improves scalability since many details of each controller can be encapsulated within a single control loop. Furthermore, partitioning increases robustness since controller failure can be localized to enable graceful system degradation, making this an effective divide-and-conquer approach to the overall control problem.

The key idea underlying our approach is that the shared state between control loops will be substantially smaller than their combined state and that this will make operations on a partial plan more efficient, bound the impacts of plan failures and simplify planning by reducing the scope of any one planning problem.

4 Definitions

We formally define a situated agent in a world $\mathcal{W}$ by:

- A set of **state variables**: $\mathcal{S} = \{s_1, \ldots, s_n\}$
- A **lifetime**: $\mathcal{H} = [0, \Pi)$ defining the interval of time in which the agent will be active. $\mathcal{H} \subseteq \mathbb{N}$.
- The **execution frontier**: $\tau \in \mathcal{H}$ is the elapsed execution time as perceived by the agent. This value increases as time advances in $\mathcal{W}$. The unit of time is called a *tick*. The agent observes and assumes world evolution at the tick rate; changes between ticks are ignored.

During its lifetime an agent observes the evolution of the world through $\mathcal{S}$ via timelines as the execution frontier advances [Muscettola *et al.*, 2002].

Definition 1 *A timeline, L, is defined by:*

- *a* **state variable***: $s(L) \in \mathcal{S}$.*
- *a set of* **tokens** *assigned to this timeline: $\mathcal{T}(L)$. Each token $t \in \mathcal{T}(L)$ expresses a constraint on the value of s over some temporal extent. A token $p(start, end, \overrightarrow{x})$ indicates that the predicate p with its attributes $\overrightarrow{x}$ holds for the temporal domain $[start, end)$ where start and end are flexible temporal intervals.*
- *an ordered set $\mathcal{Q}(L) \subseteq \mathcal{T}(L)$ of tokens describing the evolution of the state variable over time.*

We use the notation, $L(t)$ to refer to the set of the tokens *ordered* in the timeline L that overlaps time t:

$$L(t) = \{a; a \in \mathcal{Q}(L) \wedge a.start \leq t < a.end\}$$

We introduce a primitive for control called a *reactor* with which the global control structure is composed. Coordination is based on an explicit division of authority for determining the value for each state variable among the set of reactors, *R*. A reactor may *own* or *use* one or more state variables. A state variable s is *owned* by one and only one reactor r ($owns(r, s)$) which has the unique authority to determine the value for that state variable as execution proceeds. Conversely, a reactor r *uses* a state variable s ($uses(r, s)$) if it needs to be notified of changes (*observations*) or it needs to request values for this state variable (*goals*). Coordination is also based on explicit information about the *latency* and temporal scope or *look-ahead* of deliberation of a reactor. This information is used to ensure information is shared as needed and no sooner.

Fig.2 shows an agent with four reactors. A Mission Manager provides high-level directives to satisfy the scientific and operational goals of the mission: its temporal scope is the entire mission and it can take minutes to deliberate; the Navigator and Science Operator manage the execution of sub-goals generated by the Mission Manager and could deliberate within a second. The temporal scope for both is in the order of a minute. The Executive encapsulates access to functional layer commands and state variables. It is approximated as having zero latency with no deliberation.

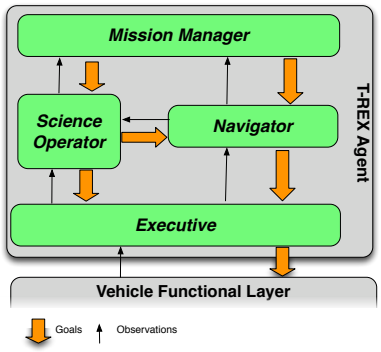


Fig. 2: An agent is composed of multiple reactors or control loops.

Definition 2 *Each reactor r is a controller defined by:*

- *a* **latency***: λ_r is the maximum amount of time the reactor r can use for deliberation.*
- *a* **look-ahead***: $\pi_r \in [\lambda_r, \Pi]$ defines the temporal duration over which reactor r deliberates. When deliberation starts for reactor r, its planning horizon is:*

$$h_r = [\tau + \lambda_r, \tau + \lambda_r + \pi_r) \quad (1)$$

- *a set of* **internal** *timelines: $\mathcal{I}_r = \{I_1, \ldots, I_k\}$. Timelines in $\mathcal{I}_r$ refer to state variables reactor r* owns.
- *a set of* **external** *timelines: $\mathcal{E}_r = \{E_1, \ldots, E_l\}$. Timelines in $\mathcal{E}_r$ refer to state variables reactor r* uses. *$\varepsilon(s)$ defines all external timelines referring to a given state variable s as $\varepsilon(s) = \{E : E \in \cup_{r \in \mathcal{R}} \mathcal{E}_r, s(E) = s\}$*
- *a set of* **goal tokens***: $\mathcal{G}_r$. Goal tokens express constraints on the future values of a state variable* owned *by this reactor.*
- *a set of* **observation tokens***: $\mathcal{O}_r$. Observation tokens express present and past values of a state variable* ***used*** *by this reactor. They must be identical to the corresponding values of this state variable as described by its* owner.

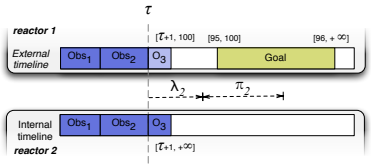


Fig. 3: Two reactors sharing one state variable. Observations from the past are consistent while projections in the future can differ to reflect each reactor's plan.

- *a* **model***: $\mathcal{M}_r$. The model defines the rules governing the interactions between values on and across timelines.*

Fig.3 illustrates the above definition.

We consider a special set of reactors, $\mathcal{R}_w$, which includes all reactors r that have no *external* dependency (i.e $\mathcal{E}_r = \emptyset$). In practice, such primitive reactors encapsulate the exogenous state variables of the agent. We further define the model for the agent, $\mathcal{M}_w$, as the union of all models of each individual reactor: $\mathcal{M}_w = \bigcup_{r \in \mathcal{R}} \mathcal{M}_r$

5 Partitioned Control

Partitioning exploits a divide-and-conquer problem solving principle where each reactor manages a component of the agent control problem thereby allowing deliberation and reaction at different rates, over different time horizons and on different state variables. More importantly, partitioning allows plan failures to be *localized* within a control loop without exposure to other parts of the system. In this section we present the overall control loop for coordinating these reactors and detail how synchronization of state is accomplished efficiently.

5.1 The Agent Control Loop

The agent coordinates the execution of reactors within a global control loop based on the SPA paradigm. The *uses* and *owns* relations provide the necessary detail to direct the flow of information between reactors. Observations flow from each reactor that *owns* a state variable to all reactors that *uses* it during a process called *synchronization*. Goals flow from reactors that *uses* a state variable to the reactor that *owns* it through a process called *dispatching*. Dispatching involves posting tokens from each external timeline e in a complete plan as goals for the reactor r where *owns(r, s(e))* over a horizon given by h_r in (1). *Deliberation* is the process of planning an evolution of state variables from current values to requested values. Synchronization, deliberation and dispatching are steps of the core agent control loop described in Algorithm 1.

The agent executes the set of reactors, R, concurrently. Execution of the loop occurs once per tick. Each step begins at the start of a tick by dispatching goals from users to owners. The agent then synchronizes state across all reactors in $\mathcal{R}$ at the execution frontier, τ. If there is deliberation to be done, it must be preemptible by a clock transition which occurs as an exogenous event.

5.2 Synchronization

In a partitioned control structure, opportunities for inconsistency exist since each reactor has its own representation for the values of a state variable. While we allow divergent views of future values of a timeline to persist, we require that all

Algorithm 1 The agent control loop

RUN$(\mathcal{R}, \tau, \Pi)$

1 **if** $\tau \geq \Pi$ **then return** ;*// If the mission is over, quit*
// Dispatch goals for all reactors for this tick
2 DISPATCH$(\mathcal{R}, \tau)$;
// Synchronize. If no reactor left afterwards, quit
3 $\mathcal{R}' \leftarrow$ SYNCHRONIZEAGENT$(\mathcal{R}, \tau)$;*// Alg. 2*
4 **if** $\mathcal{R}' = \emptyset$ **then return** ;
// Deliberate in steps until done or the clock transitions
5 $\delta \leftarrow \tau + 1$;
6 $done \leftarrow \bot$;
7 **while** $\delta > \tau \wedge \neg\, done$
8 **do** $done \leftarrow$ DELIBERATE$(\mathcal{R}', \tau)$;
9 **while** $\delta > \tau$ **do** SLEEP;*// Idle till the clock transitions*
// Tail recursive, with possibly reduced reactor set
10 RUN$(\mathcal{R}', \tau, \Pi)$;

timelines converge at the execution frontier after synchronization. For an agent to be complete, agent state variables must have a valid value at τ. It is the responsibility of the owner reactor to determine this value during synchronization and the responsibility of users of that state variable to reconcile their internal state with this observation. This explicit ownership specification enables conflict resolution.

Requirements

The concept of a flaw [Bedrax-Weiss *et al.*, 2003] i.e a potential inconsistency that must be resolved, is central to the definition of synchronization. We are concerned with flaws that may render the state of the reactor inconsistent at the execution frontier. More specifically, we are concerned with any token *t* that *necessarily* impacts any timeline *L* of a reactor *r* at the execution frontier τ, which has not been *inserted* in $\mathcal{Q}(L)$. Formally a flaw is a tuple $f = (t, L)$ that is resolved by *insertion* in $\mathcal{Q}(L)$.

Definition 3 *A reactor r is synchronized for τ, denoted $\textcircled{S}(r, \tau)$, when the following conditions are satisfied:*

- *All flaws are resolved at τ when $\mathcal{F}_\tau(r) = \emptyset$, where $\mathcal{F}_\tau(r)$ returns an arbitrarily ordered set of flaws of a reactor for synchronization at the execution frontier. A formal definition of $\mathcal{F}_\tau(r)$ follows in Definition 4.*
- *All internal and external timelines have a unique valid value at τ, i.e there are no "holes" or conflicts in the timeline:*

$$\forall L \in (\mathcal{I}_r \cup \mathcal{E}_r), \exists o \in \mathcal{T}(L) : L(\tau) = \{o\} \qquad (2)$$

- *All external timelines have the same value as the corresponding internal timeline of the owner reactor at τ. This ensures that every reactor shares a consistent view of the current state of the world.*

Synchronization of the agent means synchronization of its reactors: $\forall r \in \mathcal{R} : \textcircled{S}(r, \tau)$

Assumptions

We introduce a number of reasonable assumptions to reduce synchronization complexity to polynomial time. Our approach builds on the semantics of the partitioned structure to enable synchronization of the agent via incremental local synchronization of each reactor.

The agent percieves the world using the *Synchronous hypothesis* [Berry and Gonthier, 1991] and observes the world only at tick boundaries. From the perspective of a reactor, observations are taken as facts that are exogenous and monotonic: once an observation is published/received, it cannot be retracted. We call this the *Monotonicity Assumption (MA)*. It implies that a reactor r will publish an observation o starting at τ only if it *owns* the corresponding timeline and it has been fully synchronized for this tick ($\textcircled{S}(r, \tau) = \top$).

The last observation made on an *external* timeline is valid until a new observation is received. We call this the *Inertial Value Assumption (IVA)*. It implies that once a reactor has received all the observations for its external timelines at τ, all external timelines $E \in \mathcal{E}_r$ that do not have an associated observation o for this tick will consider that the previous state still holds :

$$(\nexists o \in \mathcal{O}_r \cap \mathcal{T}(E) : o.start = \tau) \Rightarrow E(\tau).end > \tau \qquad (3)$$

This assumption ensures that all values of an external timeline contain an observation up till τ. Consequently we require an initial observation at τ = 0. In the case of internal timelines, the model $\mathcal{M}_w$ must specify the value to assign in all cases. A corollary of the MA is that the IVA should not be applied for a reactor r_1 until:

$$\forall r_2 \in \mathcal{R}, r_1 \rhd r_2 : \textcircled{S}(r_2, \tau) \qquad (4)$$

where $\rhd$ is a relation defined as follows:

$$\forall r_1, r_2 \in \mathcal{R} :$$
$$\big(\exists s \in \mathcal{S}; uses(r_1, s) \wedge owns(r_2, s)\big) \Rightarrow r_1 \rhd r_2$$
$$(\exists r_3 \in \mathcal{R}; r_1 \rhd r_3 \wedge r_3 \rhd r_2) \Rightarrow r_1 \rhd r_2$$

(4) imposes a strong relation between the synchronization of a reactor and the structure in the dependency graph (defined by $\rhd$). Indeed, even though we could allow cyclic dependencies between reactors this would require iteration though this graph until we find a fixed point where all reactors are synchronized.

To avoid thus we ensure $\rhd$ is a partial order (ie $r_1 \rhd r_2 \Rightarrow \neg(r_2 \rhd r_1)$) Consequently, all reactors are distributed across a directed acyclic graph (DAG) where the root nodes are the reactors of $R_{\mathcal{W}}$. This *Acyclic Dependency Assumption* (ADA) ensures synchronization is achievable via iteration through a linear transformation of the partial order $\rhd$ (denoted by $R[i]$). Where state variables are cyclically dependent, they must be owned by the same reactor. Absent this assumption, global synchronization requires iterative local synchronization to a fixed point which cannot guarantee polynomial time convergence necessary for real-world systems.

Determining how to arrive at a suitable partition design is challenging; [Salido and Barber, 2006] offers a promising direction in using constraint cliques.

Synchronizing the Agent

Algorithm 2 shows how the agent is synchronized by synchronizing each reactor in the order defined by $R[i]$. It is possible that a reactor may fail to synchronize. Such a failure implies that no consistent and complete assignment of values was possible for its timelines, and usually indicates an error in the domain model. Under these conditions, the agent must remove the reactor as well as all its dependents from the control structure in order to satisfy the requirements for consistent and complete state. This failure mode offers the potential

for graceful degradation in agent performance with the possibility of continued operation of reactors implementing safety behaviors.

Algorithm 2 Agent synchronization

SYNCHRONIZEAGENT($\mathcal{R}, \tau$)

```
1  R_in ← ∅;
2  R_out ← ∅;
3  for i ← 1 to SIZE(R)
   // Get next reactor on dependency list
4      do r ← R[i];
5         if (∃ r_out ∈ R_out : r ▷ r_out)
              ∨¬SYNCHRONIZE(r, τ) // Alg. 3
          // If r cannot be synchronized exclude it
6            then R_out ← R_out + r;
7            else R_in ← R_in + r;
   // Return the reactors that are still valid
8  return R_in;
```

Synchronizing a Reactor

Algorithm 3 describes synchronization of a reactor. Synchronization begins by applying IVA to extend the current values of external timelines with no new observations according to (3). *Relaxation* is required if planning has taken longer than permitted ($> \lambda_r$) or if there is no complete and consistent refinement of the set of flaws at the execution frontier (see Algorithm 4). Relaxation decouples restrictions imposed by planning from entailments of the model and execution state by deleting the plan but retaining observations and committed values. Goals must be re-planned in a subsequent deliberation cycle. After relaxation, a second attempt is made to complete the execution frontier. If this fails, synchronization fails and the reactor will be taken off line by the agent. If this succeeds, the reactor will iterate over its internal timelines, publishing new values to its users. Finally, at every step of synchronization, a garbage collection algorithm is executed cleaning out tokens in the past with no impact to the present or future. Details of garbage collection and plan relaxation are outside the scope of this paper.

Resolving Flaws at the Execution Frontier

We now define $\mathcal{F}_\tau$ (used in Definition 3) and describe its application in the function RESOLVEFLAWS used to synchronize a reactor. The components of this definition are:

- the temporal scope of the execution frontier which we define to include the current state (i.e. tokens that contain τ) and the prior state (i.e. tokens that contain $\tau - 1$).
- an operator $\mathcal{F}_\pi$ which returns the set of flaws for deliberation. Flaws in $\mathcal{F}_\pi$ should not be in $\mathcal{F}_\tau$. The intuition is to check if a token is a goal, or if it there is a path from the token to a goal in the causal structure of the plan.
- a unit decision operator $\mathcal{U}$ for a flaw, f, that excludes flaws that can be placed at more than one location around τ in $\mathcal{Q}(f.L)$:

$$\mathcal{U}(f) \Rightarrow \left(\begin{array}{l} \forall q_1, q_2 \in (f.L(\tau-1) \cup f.L(\tau)) : \\ \quad (q_1 \otimes f.t) \wedge (q_2 \otimes f.t) \Rightarrow q_1 = q_2 \end{array} \right)$$

where $\otimes$ indicates that two tokens can be *merged*:

$$p_1(s_1, e_1, \overrightarrow{x_1}) \otimes p_2(s_2, e_2, \overrightarrow{x_2}) \Rightarrow$$

Algorithm 3 Single reactor synchronization

SYNCHRONIZE(r, τ)

```
   // Apply IVA to extend current observations
1  for each E ∈ E_r
2      do if (∄ o ∈ O_r ∩ T(E); o.start = τ)
3            then E(τ).end ← E(τ).end ∩ [τ + 1, ∞];
   // Complete execution frontier
4  if (τ ∈ h_r ∨ ¬COMPLETE(r, τ)) // Alg. 4
5     then if (¬RELAX(r, τ) ∧ ¬COMPLETE(r, τ)) // Alg. 4
6            then return ⊥
   // Publish new state values to users of internal timelines
7  for each I ∈ I_r
8      do State ← I(τ);
9         if State[0].start = τ
10           then for each E ∈ ε(s(I))
11                   do T(E) ← T(E) ∪ State;
12                      O(E) ← O(E) ∪ State;
   // Clean out tokens in the past that have no impact
13 GARBAGECOLLECT(r, τ);
14 return ⊤;
```

$$(p_1 = p_2) \wedge (s_1 \cap s_2 \neq \emptyset) \wedge (e_1 \cap e_2 \neq \emptyset) \wedge (\nexists x \in \overrightarrow{x_1} \cap \overrightarrow{x_2}; x = \emptyset)$$

Definition 4 *For a given reactor r, the set of synchronization flaws $\mathcal{F}_\tau(r)$ is defined by the set of flaws $f \notin \mathcal{F}_\pi(r)$ that overlaps the execution frontier τ and are unit decisions ($\mathcal{U}(f) = \top$).*

The call to the function RESOLVEFLAWS(r, τ) iteratively selects one of the flaws in $\mathcal{F}_\tau(r)$ and resolves it by insertion in its timeline until $\mathcal{F}_\tau(r)$ is empty. Insertion may be infeasible indicating that no complete and consistent refinement of the current execution frontier is possible. [Bernadini and Smith, 2007] describes token insertion in a partial plan.

Completion

Algorithm 4 utilizes RESOLVEFLAWS to complete synchronization. It begins by resolving all the available flaws. Resolution of the set of flaws is a necessary condition for completeness. However, it is not a sufficient condition for two reasons. First, it is possible that holes may exist in the internal timelines (application of IVA ensures that all external timelines are complete) that must be filled. Second, it is possible that the end time for the current token in an internal timeline I, is an interval. This must be restricted so that $I(\tau)$ returns a singleton after synchronization; see (2). To address this we define a policy to complete an internal timeline under these conditions. For the first case, MAKEDEFAULTVALUE will generate a default value according to the model which is inserted to fill the hole. In the second case, the end time of the current value will be extended. These modifications may generate more flaws in turn. For example, tokens previously excluded from synchronization by $\mathcal{U}$ may now become unit decisions. Furthermore, additional rules in the model may now apply. Consequently, we invoke RESOLVEFLAWS again.

5.3 Complexity Analysis

We now consider the complexity of synchronizing an agent. The key result is that synchronization is a polynomial time

Algorithm 4 Completion of the execution frontier

```
COMPLETE(r, τ)
1  if ¬RESOLVEFLAWS(r, τ)
2     then return ⊥
   // Complete Internal Timelines
3  for each I ∈ I_r
4        do v ← I(τ);
            // Fill with default value if empty
5           if v = ∅
6              then INSERT(I, MAKEDEFAULTVALUE(I, μ_r, τ));
7              else v[0].end ← v[0].end ∩ [τ + 1, ∞];
8  return RESOLVEFLAWS(r, τ);
```

multiple of the cost of primitive operations on a plan. We assume the following operators are executed in amortized constant time:

- $\mathcal{F}_\tau$ The operator to obtain the sequence of flaws in the execution frontier.
- INSERT(L, t) The procedure to insert a token t in timeline L.
- MAKEDEFAULTVALUE(L, μ_r, τ) The procedure to generate a default token for timeline L.

We further assume:

- GARBAGECOLLECT is linear in the number of tokens in the past that have not yet been removed.
- RELAX is linear in the number of tokens in all timelines.
- Insertion of a token by merging with an existing token generates no new flaws.
- The costs of $\otimes, \cap, \cup$ as used in synchronization are bounded and negligible.

Consider the procedure RESOLVEFLAWS. This procedure is linear in the number of flaws, since for each flaw encountered, it is resolved by an insertion operation within amortized constant time with no backtracking. Assume a reactor *r* has $\mathcal{N}_r$ timelines. In the worst case, every timeline in a reactor will require a new value for the current and prior tick. Assume that in the worst case, every new value generates a flaw for every other possible position in all timelines in the execution frontier (i.e. $2\mathcal{N}_r$ -1 flaws per new value). This gives a maximum complexity for RESOLVEFLAWS of $2\mathcal{N}_r \times (2\mathcal{N}_r$-1) or O($\mathcal{N}_r{}^2$). In the worst case, the procedure COMPLETE calls RESOLVEFLAWS twice. However, if simply refining the execution frontier, this does not change the cumulative number of flaws. Since iteration over the internal timelines is linear in a value $\leq \mathcal{N}_r$, we have a complexity of O($\mathcal{N}_r{}^2$) for Algorithm 4.

In the worst case, synchronization of a reactor (Algorithm 3) incurs the following costs:

- O($\mathcal{N}_r$) to complete external timelines
- O($\mathcal{N}_r{}^2$) to call COMPLETE the first time, which we assume will fail.
- O(P_r) to RELAX the plan where P is the number of tokens in the plan.
- O($\mathcal{N}_r{}^2$) to call COMPLETE the second time, which we assume will succeed.
- O($\mathcal{N}_r$) to publish observations
- O(H_r) to garbage collect where H is the number of tokens in the plan that have passed into history.

Since synchronization of the agent is accomplished by iteration over the set of reactors, without cycling, the worst case time complexity for synchronization is given by:

$$O(\text{Ⓢ}) = O(\sum_{r \in \mathcal{R}} \mathcal{N}_r^2 + P_r + H_r) \quad (5)$$

6 Experimental Results

In this section we evaluate the performance of synchronization in partitioned and non-partitioned control structures. When plan operations are constant time, we show that synchronization is $O(N^2)$ in the worst case. Moreover, we demonstrate that actual costs of synchronization are accrued based on what changes at the execution frontier, making it efficient in practice. We further demonstrate that since operations on a plan are typically not constant time, but vary in diverse and implementation dependent ways according to plan size, partitioning control loops can reduce the net cost of synchronization and deliberation.

The following results are based on our current implementation of the framework with each reactor using the same CTP based planner using chronological-backtracking refinement search to deliberate. The agent invokes each reactor as defined in the Algorithm 1. Our lab experiments were executed on a MacBook Pro running at 2 Ghz.

Fig. 4 shows results of a run with a single reactor configuration where each problem instance runs for 50 time ticks. A set of problem instances were generated by modifying the number of internal timelines (I) and the connectivity of the constraint graph (C). There is no deliberation involved. CPU usage for synchronizing the agent is measured at every tick, and averaged over all ticks. The figure shows that average synchronization cost increases linearly in I and C and quadratically as the product of I and C.

Fig. 5 shows the impact of a good partitioning scheme on problem solving, by varying the partitioned structure of an agent without changing the number of timelines being synchronized. We used 120 internal timelines spread evenly across all reactors. For each problem instance we distributed the timelines between a varying number of reactors between 1 and 120. This problem was designed to show the dependency between timelines exhibiting both abstraction and functional separation. With 120 reactors there is only 1 timeline per reactor and the agent control structure is maximally partitioned. Deliberation fills out the timeline with 1 token per tick for 10 ticks with no search required. For each problem, the cumulative synchronization and deliberation CPU usage was measured. As illustrated, a good partition design not only reduces

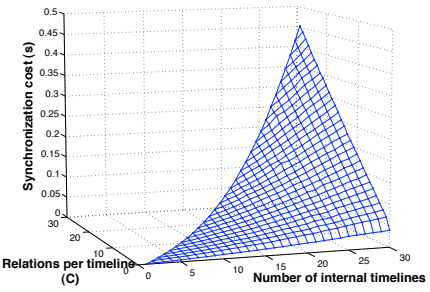


Fig. 4: The relationship between synchronization time, number of constraints and number of internal timelines.

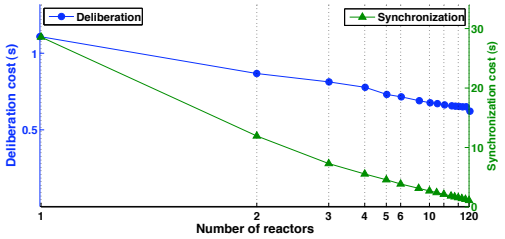


Fig. 5: Synchronization costs associated with partitioning.

deliberation cost per tick but can also potentially reduce synchronization costs resulting in a more efficient system design.

Using the same model we use in our sea trials, Fig. 6 compares system robustness between 3 and 4 reactors ($3r$ and $4r$ respectively); Fig. 2 shows the $4r$ configuration; $3r$ was a combination of the Science Operator and Navigator into a single reactor. We limited the number of deliberation steps allowed at each tick and did several runs in both configurations to measure the scalability of our approach. The steps/tick is a proxy for the computational power of the CPU; larger the number of steps/tick, the more computational power is available (for example, our AUVs embedded CPU at 357 MHz is capable of 60 steps/tick where a tick is 1 sec). The implication of such a proxy measure is in enabling system design evaluation where problem solving is weighed against available compute power of the platform as well as the partition design.

Fig. 6 shows that deliberation time is uniform between $3r$ and $4r$ especially when both configurations converge on a plan. Conversely with limited computation power in the embedded processor, this figure shows that an agent with more partitions would succeed where another with fewer reactors does not. While the $4r$ agent was able to complete missions with as few as 21 steps/tick, the $3r$ agent needed at least 24 steps/tick.

For experiments with 15 and 18 steps/tick, the $4r$ agent while rejecting the original mission goals, was able to safely terminate the mission by dispatching safety commands to the vehicle. In contrast the $3r$ agent with a merged reactor, is unable to complete the goal or send safety commands in time for graceful recovery (in the field such a configuration would have risked vehicle safety). In 20 steps/tick scenarios, the $4r$ agent fails; there are sufficient number of steps/tick for deliberation in more abstract reactors. But the lowest-level reactor is unable to cope with both the goals dispatched to it, as well as in dealing with dispatching safety commands, hence running out of time and leading to mission failure.

These experiments indicate a possible improvement of reactor scheduling; those reactors that are deadline limited should be granted a larger slice of the CPU with more steps/tick. Additionally, our intuition is that for each reactor configuration, there is also a likely phase transition [Cheeseman *et al.*, 1991] between successful execution and mission failure which is a smaller steps/tick for larger partitions. Finally, it is important to note that $4r$ is able to fail gracefully (between 15 and 18 steps/tick) in comparison to $3r$ below this transition limit. Exploring these transition phenomena is part of our future work.

Fig. 7 shows the performance gain between $4r$ and $3r$ configurations for a 7 hour mission. The y-axis ratio between $3r$ and $4r$ includes time to deliberate and synchronize. $4r$ is 1.14 times faster than $3r$ at peak and subsequently stabilizes around 1.04 showing that $4r$ is on average faster. Both configurations exhibit a similar plan with comparable vehicle path and reactivity. Lack of space precludes detailed descriptions of plan trajectories.

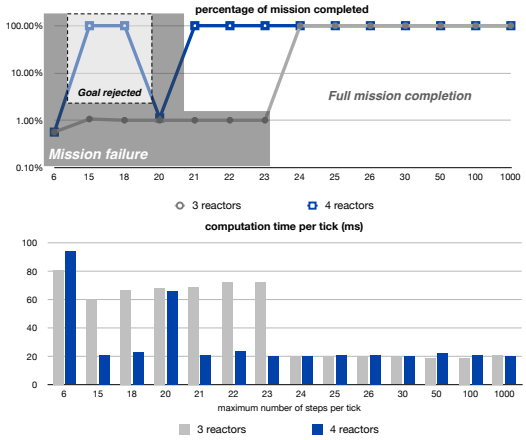


Fig. 6: Comparative analysis between $4r$ & $3r$ configurations. The figure on top shows likely phase transitions for partition design for $4r$ & $3r$.

Our framework has been integrated onboard our *Dorado* platform (Fig. 1) and deployed for scientific exploration in Monterey Bay, California [McGann *et al.*, 2008c]. The agent runs on a 367 MHz EPX-GX500 AMD Geode stack using Red Hat Linux, with the functional layer running on a separate processor on real-time QNX. In November 2008 for example, on a mission over the Monterey Canyon, our science objectives were to carry out a volume survey while estimating the presence of an INL. The model exhibited both long term planning as well as reactive execution in response to environmental changes impacting the full scope of the mission. During the uninterrupted 6 hours and 40 minutes run, our system was able to identify the INL features accurately (using Probabilistic State Estimation which is out of scope for this paper) to bring back targeted water samples from within biological hotspots. Fig. 8 shows the vehicles transect and the context of the INL in the water-column detected by the AUVs sensors. It also shows the vehicle changing its navigation for sampling resolution, starting with high and ending with low resolution transects where the INL is not visible to the AUV's sensors. Water samples are shown to be taken within the INL as required. Scientific results from this run in coastal larval ecology can be found in [Johnson *et al.*, 2008].

7 Conclusion

We introduce a formal framework for specifying an agent control structure as a collection of coordinated control loops

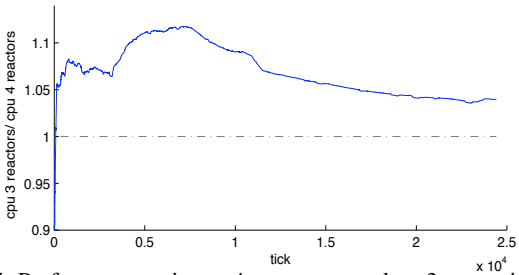


Fig. 7: Performance gain on $4r$ as compared to $3r$; x-axis shows time in ticks and y-axis indicates ratio between cumulative cpu time used by $3r/4r$.

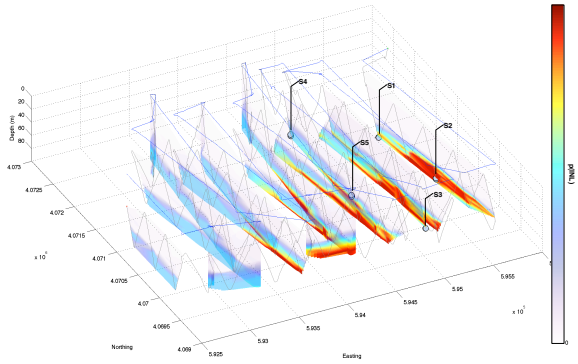


Fig. 8: Visualization of a mission over the Monterey Canyon in November 2008. Red indicates high probability of INL presence as detected by onboard sensors. S1-S5 indicate triggering of 10 water samplers two at a time.

while decoupling deliberation over goals from synchronization of agent state. We present algorithms for integrated agent control within this partitioned structure. For weakly coupled reactors partitioning offers performance improvements with the overhead of information sharing. Further partitioning allows plan failures to be localized within a control loop without exposure to other parts of the system, while ensuring graceful system degradation.

8 Acknowledgments

This research was supported by the David and Lucile Packard Foundation. We are grateful to MBARI oceanographer John Ryan in helping formulate and drive our technology and our engineering and operations colleagues for their help in integration and deployment. We thank NASA Ames Research Center for making the EUROPA Planner available, Willow Garage for supporting McGann's collaboration and the Spanish government for Olaya's fellowship at MBARI.

References

[Alami *et al.*, 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *The International Journal of Robotics Research*, Jan 1998.

[Ambros-Ingerson and Steel, 1988] J. Ambros-Ingerson and S. Steel. Integrating Planning, Execution and Monitoring. *Proc. 7th AAAI*, Jan 1988.

[Bedrax-Weiss *et al.*, 2003] T. Bedrax-Weiss, J. Frank, A.K. Jonsson, and C. McGann. Identifying Executable Plans. In *Workshop on Plan Execution, ICAPS*, 2003.

[Bellingham and Leonard, 1994] J.G. Bellingham and J.J. Leonard. Task Configuration with Layered Control. In *IARP 2nd Workshop on Mobile Robots for Subsea Environments*, May 1994.

[Bernadini and Smith, 2007] S. Bernadini and D. Smith. Developing Domain-Independent Search Control for EUROPA2. In *Proc. ICAPS-07 Workshop on Heuristics for Domain-independent Planning*, 2007.

[Berry and Gonthier, 1991] G Berry and G Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. Technical report, INRIA, 1991.

[Brooks, 1986] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W. Taylor. Where the really hard problems are. *Proceedings of the 12th IJCAI*, Jan 1991.

[Chien *et al.*, 1999] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Integrated Planning and Execution for Autonomous Spacecraft. *IEEE Aerospace*, 1:263–271 vol.1, 1999.

[Frank and Jónsson, 2003] J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003.

[Gat, 1998] E. Gat. On Three-Layer Architectures. In D. Kortenkamp, R. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. MIT Press, 1998.

[Haigh and Veloso, 1998] K. Haigh and M. Veloso. Interleaving Planning and Robot Execution for Asynchronous User Requests. *Autonomous Robots*, Jan 1998.

[Johnson *et al.*, 2008] S. B. Johnson, A. Sherman, R. Marin, J. Ryan, and R. C. Vrijenhoek. Detection of Marine Larvae using the AUV Gulper and Bench-top SHA. In *8th Larval Biology Symposium*, Lisbon, Portugal, July 2008.

[Jónsson *et al.*, 2000] A. Jónsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *AIPS*, 2000.

[Knight *et al.*, 2001] R Knight, F Fisher, T Estlin, and B Engelhardt. Balancing Deliberation and Reaction, Planning and Execution for Space Robotic Applications. *Proc. IROS*, Jan 2001.

[McGann *et al.*, 2008a] C. McGann, F. Py, K. Rajan, J. P. Ryan, and R. Henthorn. Adaptive Control for Autonomous Underwater Vehicles. In *AAAI*, Chicago, IL, 2008.

[McGann *et al.*, 2008b] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. A Deliberative Architecture for AUV Control. In *ICRA*, Pasadena, CA, May 2008.

[McGann *et al.*, 2008c] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. Preliminary Results for Model-Based Adaptive Control of an Autonomous Underwater Vehicle. In *Int. Symp. on Experimental Robotics*, Athens, Greece, 2008.

[Muscettola *et al.*, 1998] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. In *AI Journal*, volume 103, 1998.

[Muscettola *et al.*, 2002] N. Muscettola, G. Dorais, C. Fry, R. Levinson, and C. Plaunt. IDEA: Planning at the Core of Autonomous Reactive Agents. In *IWPSS*, October 2002.

[Ryan *et al.*, 2005] J. P. Ryan, F. P. Chavez, and J. G. Bellingham. Physical-Biological Coupling in Monterey Bay, California: Topographic influences on phytoplankton ecology. *Marine Ecology Progress Series*, 287:23–32, 2005.

[Salido and Barber, 2006] M. A. Salido and F. Barber. Distributed CSPs by Graph Partitioning. *Applied Mathematics and Computation*, 183:212–237, 2006.

[Thomas *et al.*, 2006] H. Thomas, D. Caress, D. Conlin, D. Clague, J. Paduan, D. Butterfield J, W. Chadwick, and P. Tucker. Mapping auv survey of axial seamount. *Eos Trans. AGU*, 87(52)(Fall Meet. Suppl., Abstract V23B-0615), 2006.

[Yuh, 2000] J. Yuh. Design and Control of Autonomous Underwater Robots: A Survey. *Autonomous Robots*, 8:7–24, 2000.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Application of a Heuristic Function in Reinforcement Learning of an Agent

Anastasia Noglik, Michael Müller, Josef Pauli
Universität Duisburg-Essen
Abteilung für Informatik und Angewandte Kognitionswissenschaft
{ anastasia.noglik, josef.pauli }@uni-due.de

Abstract

The present work examines the possibilities of the application of a-priori rudimental context knowledge. The aim is to accelerate the learning process of a rational agent. The properties of the environment are estimated by means of an estimator function. The approach is similar to the application of the heuristic function in the A* search algorithm. Two methods are proposed that integrate hidden, context knowledge in form of a heuristic function into the learning process. Numerous investigations are performed for the classic Mountain Car Task (MCT) to measure the impact of the context knowledge on the convergence progress in comparison to the standard method. Furthermore different types and methods of discretization of the state space are considered. The improvements of the learning process are confirmed for an extended 3D Mountain Agent Task.

1 Introduction

A rational agent has to learn as much as possible by perception. It can do this by means of paradigms of reinforcement [Russell and Norvig, 2003]. The improvement of learning processes is an important task in the theory of rational agents. One of the advantages of Reinforcement Learning algorithms (RL algorithms) is the possibility of learning without an accurate or even without any model of the environment [Bertsekas, 1995], [Sutton, 1998]. Another advantage is the application of past experience into the ongoing decision making process. This is known as online learning.
One of several issues of RL algorithms is the relatively slow convergence [Watkins and Dayan, 1992]. This can be avoided by using robot knowledge and knowledge about the environment or by applying a suitable discretization. There is only very few information about the environment available. The agent gets rudimental knowledge about the environment. For example the destination is specified. A suitable reward model tells the agent which behavior is good and which is bad.
In the present work two approaches are suggested and investigated which are based on heuristics, context knowledge (heuristic function). For example a heuristic function can evaluate the distance between the agent and the destination. This idea is similar to the application of a heuristic function in the A* search algorithm [Judea, 1984]. In the present work the influence of a heuristic function is investigated for a standard MCT (see [Sutton, 1998]) and a 3D Mountain Agent Task that is mentioned in [Taylor *et al.*, 2008].
This task has been chosen as a benchmark problem to prove possible advantages of the expansion of RL methods. The idea for that expansion did arise during the investigation of a navigation problem. However the primary goal has been to verify the better performance for the benchmark problem.
Suitable types and methods of discretization play an important role in the learning process. However the direct comparison of discretization methods is rarely described in the literature. The correlation between the type and the method of discretization of the state space is investigated as well as the influence of context information on the learning process. Furthermore the impact and interaction of all mentioned aspects on the learning progress is examined. Numerous test series are presented.
In RL it is searched for an optimum sequence of actions that achieves the maximum expected profit (target function). Here the target function is extended by a heuristic function. The heuristic function gives a certain measurement for some state. This can be regarded as a form of characterization of each state or of an important part of a state. This mostly results in a faster achievement of the target.
The term "important part of a state" is explained by means of a small example in the standard MCT. A state consists of the two elements, position and speed $(x, \dot{x})$. The aim is to achieve a certain position on the mountain (see the right picture in figure 1). The speed of the car at destination is not important. The significant part of the state is the position of the car. The distance is estimated analogously to the A* search algorithm. It is the beeline distance between the position of the car and the destination. It can be easily estimated without additional knowledge about the properties of the environment.
In case of a possibly difficult landscape the information about this distance can be helpful. The distance in the example corresponds to the introduced heuristic function.
Such an approximation is not always possible. But it is assumed that for many problems which can be solved with RL algorithms it is possible to define a heuristic function without a huge effort. The advantage of the RL algorithms still remains since a-priori knowledge about the environment is not

needed. The information can be integrated into the learning process of the afore mentioned example in form of a heuristic function already after the first arrival at the destination. Even if the destination coordinate is not available right from the start. Hence only scarce information is needed in the navigation task, quite comparable to standard MCT. One of the most significant advantages of RL algorithms is still valid.

2 Application of Heuristic Function

The suggested extensions are based on the SARSA algorithm (see equation below) but can be integrated without difficulties into other approaches what is showed below.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (1)$$

where s_t and s_{t+1} are the state and the subsequent state respectively. The symbols a_t and a_{t+1} stand for the action and subsequent action which is chosen based on the ϵ-greedy strategy. The reward is denoted by r_t. In each step the Q-function is recalculated for the current pair (s_t, a_t). α, γ and ϵ are constants used for learning.
An extended version of the SARSA-algorithm is used including eligibility traces which require the determination of the fourth learning constant λ.
In the first suggested method the context knowledge is embedded in the Q-function. The heuristic function is regarded as a continuous reward versus a discrete reward model in the standard method.
In the second approach it is suggested to include context knowledge in the decision making process that is the preparation of the policy and chooses the next action.

2.1 Extension of the Reward Model

The basic idea of the first approach is to learn the context knowledge directly in the Q-function, according to equation:

$$\begin{aligned} Q(s_t, a_t) \leftarrow \; & Q(s_t, a_t) + \alpha(r_t + \\ & + h(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \end{aligned} \quad (2)$$

Similar ideas that use a constant reward function which is adapted to the task are mentioned in [Papierok *et al.*, 2008], [Russell and Norvig, 2003]. In the present paper a thorough analysis of the method is described.
In the classic version of RL little attention is paid to the reward model. There is no tight continuous correlation between states and rewards.
The reward model can be regarded as a discrete and in many cases binary function. This function projects the state S on a finite number of subsets of the state space. The reward function maps each of the subsets to a constant value r_i. It is suggested to extend the reward function with a heuristic function. This heuristic function depends on the current state and the destination state of the agent. The current state is given depending on the task in a global or local (with respect to the starting point) coordinate system. It is also important to keep the correct balance between the discrete reward model and the values provided by the heuristic function. A necessary condition for the application of the heuristic function in the Q-function is

$$-1 \cdot |r_{max}| \leq inf_{s \in S} h(s) \leq sup_{s \in S} h(s) \leq 1 \cdot |r_{max}|$$

where $r_{max} = max_i |r_i|$. The correctly defined heuristic function will not affect the convergence of the algorithm. The basis for that proposal is explained in detail in chapter 3.2.

2.2 Heuristic Function in the Action Selection

The second suggested approach uses the context information in the action selection:

$$a^* = max_a(Q(s_t, a_t) + h(s_{t+1}(s_t, a_t))) \quad (3)$$

Thereby the values of the heuristic function are not learned contrary to the first approach mentioned, since otherwise the context knowledge would have a double influence on the learning process.
The adequate relation between the heuristic function and the learned Q-function is ensured by the choice of the limiting values of the heuristic function depending on the single rewards (see details in chapter 3.2).
If an action is not optimally chosen due to the influence of the heuristic function, negative effects on the learning process can occur. The proposal for the choice of the limits for the heuristic function is for each possible policy $\pi(s_t) = a_t := max_a Q^*(s_t, a)$ and each state:

$$h(s_t) - h(s_{t+1}(s_t, a_t)) \leq Q^*(s_{t+1}, a_{t+1}) - Q^*(s_t, a_t)$$

Then the Q-function converges to the *optimum function* Q^*, although it is possible that more iteration steps are needed. This method has its most influence in the beginning phase of the learning process in which many Q-values or parameters in the Q-function are equal (mostly zero). These arguments are confirmed by the experimental results shown below.

3 Heuristic Function

The use of a heuristic function has a solid basis. This has been proven in the A* search algorithm. The application of context knowledge is also borrowed from the field of evolutionary algorithms. The heuristic function can be compared to a kind of fitness function. The context knowledge is indirectly used in learning of a structure. This structure is presented in form of a neural network for example in [Kassahun, 2006] and [Siebel *et al.*, 2008].

3.1 Possible Definition of Heuristic Function

A possibility to show the heuristic function for the MCT is the horizontal blue line on the right side in figure 1. A state consists of the position and speed of the agent. The vertical coordinate on the mountain $f(x)$ is hidden in the model of the gravity. This information is communicated to the agent in form of a speed. Relevant however is only the information of the horizontal position of the agent and its horizontal distance to a certain position (target).
Alternatively, it is possible to define a heuristic function which evaluates the distance between the target position in the 2D world $(x^{target}, f(x^{target}))$ and the agent position in 2D $(x, f(x))$. For example the state in the valley is priced worse than the state at the opposite mountain. But in this case more knowledge about the environment is necessary. For a navigation problem (see left side in figure 1) the same context

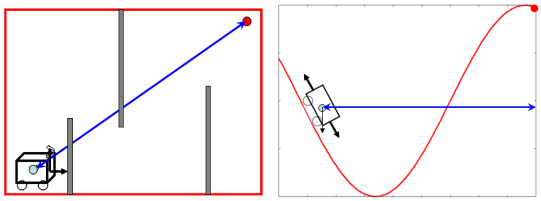


Figure 1: Heuristic function for two different environments, left: navigation problem, right: Mountain Car Task

knowledge as in the MCT can be used. The beeline is defined in the following equation:

$$h(s, s^{target}) = k \cdot \sqrt{(s^{target} - s)^T \cdot (s^{target} - s)} \quad (4)$$

where k is a parameter which should be chosen depending on the standard reward model. The parameter k controls the influence of the heuristic function on the learning process.
This heuristic function can be seen as a form of potential function. Each state contains a certain potential which increases with decreasing distance to the destination for $k < 0$. The heuristic function is a kind of distribution of the potential. For the computation of the heuristic function the same information is needed as in the standard method: the current position and the position of the destination which is derived from the context knowledge. The current position is calculated from the action chain. If the starting point remains always the same, the current position in the coordinate system of the target can be calculated by backward transformation of the action chain.
Alternatively, if the starting point of the agent is always changing, independent on the application of context information, the approximate position in the coordinate system of the global environment has to be known.

3.2 Discussion of Convergence Aspects

It has to be paid attention to some conditions when defining the heuristic function. The function has to be bounded above and below on the entire domain of the state space or sub state space: $c \leq inf_{s \in S} h(s) \leq sup_{s \in S} h(s) \leq C$.
The limits c, C should be in a certain relation to the single values of the ordinary rewards $(r_1, \ldots, r_N)$ of the model. If the values c and C are chosen too small $|c| < |C| << |r_i|$ for all i, the effect of the heuristic function will not be noticeable at all. In case of too large values a long time is necessary to compensate the negative effects. Or in case $|c| >> r_i$, it can not be ensured that the heuristic function will converge to the optimum value-function. Furthermore the heuristic function shall not represent the relation of one state to the other (some states are better than others) depending on the distance to the target.
The influence of the heuristic function in the learning process is investigated in the example 4.1 in [Sutton, 1998]. In the following the positive and negative influence of the heuristic function on the learning progress is described. In this example the iterative policy evaluation is used in a small grid world. Each learning step for the value-function is defined:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma V_k(s')] \quad (5)$$

This example has been chosen due to the simple environment, the fast convergence of the method to the optimum policy and a good visualization of the learning process. The number of steps required to achieve the optimum policy by using a sequence of approximations of the state-value function for the random policy (all actions equal) is a good measurement. In the standard method the optimum policy is achieved after 3 iterations. With this measurement the comparison of the standard method with the method extended by a heuristic function is made and the influence of the chosen limits for the heurisic function is investigated.
A simple example is introduced to confirm the statement, that wrongly chosen limits for the heuristic function results in a worse convergence or even in a divergence of the algorithm. The goal hereby is to confirm the suggested limits for the heuristic function.
A 4×4 small grid world with 14 states $S = \{1, \ldots, 14\}$ is considered. In figure 2 the environment (left part), the heuristic function (middle part) and the sequence of greedy policies corresponding to the value function estimates (right part) are shown. Suppose the agent follows the random policy in which all actions are equally likely, $\pi(s,a) = \frac{1}{4}$ for all $a \in A(s)$. The target states are shaded in the figure. In each state four actions are possible: $A = \{left, right, down, up\}$, which deterministically cause the corresponding state transitions. Exceptionally, the actions that would take the agent off the grid will keep the state unchanged. An example for the probability of the transition is: $P^{down}_{12,12} = 1$, $P^{left}_{12,1} = 0$, $P^{right}_{12,13} = 1$. The elementary reward is $r = -1$ on all transitions until the target state is reached. Thus the reward function is $R^a_{s,s'} = r$ for all states s, s' and actions a.
This example is extended by the proposed heuristic function which is defined in equation 4. The heuristic function is defined depending on the parameter k. Thereby the impact of the heuristic function on the learning process is investigated systematically. Positive impacts ($k < 0$) and negative impacts ($k > 0$) are determined. In case of $k > 0$ the heuristic function is in a conflict to the target of the agent. The reason is that according to the heuristic function the state with the larger distance to the target is better than a state with a shorter distance in case of a positive distance. The first method, where

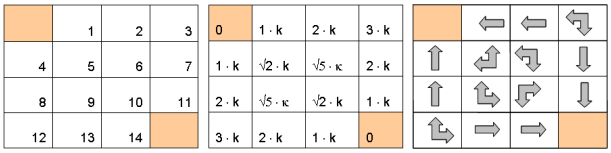


Figure 2: Example of the grid world and the context knowledge depending on the parameter k, the right image is the optimum policy; this is a greedy policy corresponding value function after the 3rd iteration, which was obtained using a standard method.

the heuristic function is learnt within the value function, is

based on the following consideration: According to the learning directive it is necessary that

$$V_{k+1}(s) \overset{(5)}{=} \sum_a \pi(s,a) \sum_{s'} [R^a_{s,s'} + \gamma V_k(s')] \overset{\pi(s,a)=\frac{1}{4}}{=} \sum_{s'} [\gamma V_k(s') + \frac{1}{4} \cdot (R^{left}_{s,s'} + h(s,left) + R^{right}_{s,s'} + h(s,right) + R^{up}_{s,s'} + h(s,up) + R^{down}_{s,s'} + h(s,down))] \quad (6)$$

for all $s \in S$. In order that the correct reward model can be effective, the following shall be valid:

$$\sum_a h(s,a) \overset{(6)}{<} \sum_a |R^a_{s,s'}|. \quad (7)$$

The largest absolute value of the heuristic function is in the states 12 and 3. Now the correlation between elementary rewards and the heuristic function for these states is described in detail. The following shall be valid: $(h(12,left) + h(12,right) + h(12,up) + h(12,down)) = = k \cdot (3+2+2+3) \overset{(7)}{<} 4 \cdot |-1| \Rightarrow k \leq \frac{4}{10}$. In other cases the iterative policy method will never converge to the optimum policy. The impact of an incorrectly defined heuristic function can not be compensated anymore by the correctly defined standard reward model. In the first method a necessary condition for the definition of the heuristic function for the learning of Q-function is $[c;C] \subset [-max_i|r_i|; max_i|r_i|]$ and for the learning of the V-function $\sum_a |h(s,a)| < \sum_a |R^a_{s,s'}|$ for all $s \in S$.

In figure 3 the number of iteration steps (to the achievement of the optimum policy) is shown for the first method depending on the parameter k. For each $k < 0$ the impact of the context knowledge is positive. Only one step is necessary to achieve the optimum policy. On the contrary the standard method requires 3 iteration steps. In case of $0 < k < 0.4$ a negative impact of an incorrectly defined heuristic function can be observed. In the second method the context knowledge is used

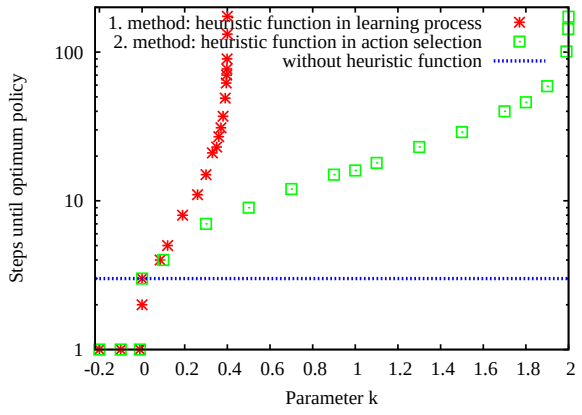


Figure 3: Positive respectively negative impact of the heuristic function and the number of steps until the optimum policy for the first method and positive respectively negative impact of the heuristic function in the action choice and the number of steps until the optimum policy for the second method.

in the action choice. Hereby certain rules for the choice of the limits for the heuristic function should be followed. As an example the limit case for state 12 is considered in detail. According to the learning rule it shall be valid:

$$V^*(12(12,left)) + h(12) < V^*(8(12,up)) + h(8). \quad (8)$$

In this case the method can still achieve the optimum policy. That is valid, if $V^*(12(12,left)) - V^*(8(12,up)) = (-22 - (-20)) < h(8) - h(12) := k \cdot (2-3)$, thus $k < 2$. The values of the heuristic function shall not exceed the values of the optimum V-function for each state. $h(s) - h(s'(s,a^{greedy})) \overset{(8)}{<} V^*(s'(s,a^{greedy})) - V^*(s)$ for all $s \in S$ and a^{greedy} on the basis of V^*.

In figure 3 the impact of the correct ($k < 0$) or incorrect ($0 < k < 2$) heuristic function on the learning process is shown. In case of $k < 0$ the positive impact of the context knowledge can be observed in both suggested methods. Also here only one learning step is required to achieve the optimum policy.

4 Evaluation Strategy

The problem of a relatively slow convergence can be avoided by using robot knowledge about the environment or through a suitable discretization of the state space. The performance of the convergence progress will be evaluated by checking it against the performance of the method with standard configuration, MCT with standard discretization of the state space (Tile Coding (TC)) and standard reward model.

Since the convergence process is dependent on the discretization of the state space, different configurations have been evaluated. In the presented paper Tile and Coarse Coding have been chosen as discretization methods. In the case of Coarse Coding (CC) the state space is represented by binary features whose receptive fields overlap. Depending on the position in the state space one state can be represented by several or one feature only. The respective Q-value is computed as the average of the parameter vector coefficients, which correspond to the currently involved binary features. In case of TC the receptive fields of features do not overlap. Every part of the state space is represented by one particular feature. For the computation of the respective Q-value only one coefficient of the parameter vector is considered.

According to the previously specified MCT the two axes of the state space correspond to the horizontal position and the speed of the car. The discretization of the state space significantly influences the performance of the learning process. The discretization $m \times n$ specifies the number of feature centers along the position and speed axes respectively. Since the domain of the position values is greater than that of the speed values the following discretization of the state space was chosen to be 21×5. To obtain some comparative value the method was also tested with the discretization of 9×9, according to [Sutton, 1998].

In the results presented in the next section different abbreviations are used to indicate the examined method. The first two letters define how the state space is descretized: CC - Coarse Coding, TC - Tile Coding. The next two letters of the abbreviation specify the reward model: SR - Standard Reward

Model (see equation 1); HR - Heuristic Reward in the Q-function (see equation 2); H - Heuristic function in the action selection (see equation 3).
The proposed methods of how the context information can be incorporated into the learning process have been evaluated according to the type of the discretization and the discretization of the state space. To obtain meaningful results the following four criteria are used.
The first criterion is the learning progress. In case of the standard mountain task it is defined as follows: $LearningProgress_{SMT} := \frac{1}{30}\sum_{i=1}^{30}\bar{s}_i$, where $\bar{s}_i$ is the average number of steps in the i-th episode. For every episode in total 100 tests have been conducted. Although 200 episodes have been realized, for the computation of the learning process value only the first 30 episodes are considered. As stated by [Sutton, 1998] in case of the standard MCT the computed values do not change a lot after the 30-th episode. According to the definition smaller values indicate a better learning process. These values are shown in the second column of the tables 1 and 2.
In case of the extended mountain agent task the value of the learning progress $LearningProgress_{EMA} = \frac{1}{500}\sum_{i=1}^{500}\bar{s}_i$ is computed over all performed 500 episodes. These values are shown in table 3.
The second criterion is defined as the episode number, in which the corresponding curve reaches the convergence range of the curve for the first time. The convergence range begins where the curve breaks through the threshold $t = \bar{s}_{min} \cdot 1.1$. This criterion shows how fast the agent learns. The smaller the value, the better is the performance. The values of this criterion are shown in the third column of the corresponding tables in the next section.
The third criterion is defined as the minimal average number of steps, $\bar{s}_{min}$. In case of the standard MCT the values of this criterion do not vary much when applied to the evaluated methods. The variance of the computed values lies around 5%. Therefore the convergence range for the standard MCT is defined depending on the value of this criterion. In case of the 3D Mountain Agent Task the differences are larger, so that the same convergence area is used. A smaller value of this criterion indicates a better performance. The values of this criterion are shown in the fourth column of the corresponding tables in the next section.
The fourth criterion is the average variance V. The value of this criterion indicates the stability of the examined method. Again the smaller the value, the better is the performance. The results of this criterion are shown in the fifth column of the tables in the next section.

5 Results

First the results for the standard MCT and afterwards the results for the extended 3D environment are presented and explained. The motivation for the extension of the task was the assumption of a larger effect of the context information for a more complex application. Basis for this assumption is the higher significance of spatial information. Furthermore the context information could help the agent since the number of degrees of freedom of the agent is higher.

5.1 Standard Mountain Car Task

The standard MCT is taken from [Sutton, 1998] and is used as a benchmark for the comparison of the learning process of methods with context knowledge and different discretization. According to that

$$x_{t+1} = bound(x_t + \dot{x}_{t+1})$$

$$\dot{x}_{t+1} = bound(\dot{x}_t + 0.001a_t - 0.0025\sin(\arctan(3\cos(3x))))$$

where $a_t \in \{-1.0, 0.0, 1.0\}$ is the control and the chosen action at the point of time t. $(x_t, \dot{x}_t) \in [-1.2; 0.6] \times [-0.07; 0.07]$ are the position and speed of the agent at the point of time t. The standard reward is -1 per time step. TC has one tiling 21×5 and 9×9 respectively. In CC ellipses are used for the discretization of the state space. The radii of the ellipses (see equation 9) depend on the number of tiles (21×5 and 9×9) in each direction and on the co-domain:

$$\sigma_\circ = \frac{\sqrt{log_4 10}}{2} \cdot \delta_\circ \text{ with } \circ \text{ for } x, \dot{x} \tag{9}$$

where $\delta_\circ$ is the box width which depends on the co-domain of each coordinate of the state space $[\circ_{mix}; \circ_{max}]$ and the number of tilings ($Number_{Tile}$) of each coordinate. There are two different forms of tilings: 21 tiles for the position part of the state and 5 tiles for the speed part. Thus the tiling is named as 21×5 tiling. The same number of tiles for each part of the state space is denoted as 9×9, $\delta_\circ = (\circ_{max} - \circ_{min})/Number_{Tile}$. The parameters are equal for each combination during the learning. $\epsilon = 0.0$, $\alpha = 0.5$, $\lambda = 0.9$, $\gamma = 1.0$.
The method and type of discretization depend on each other. The 9×9 type of discretization in combination with CC shows the best results for this task (see table 1 and table 2). The maximum (worst) value for the average number of steps is 730 using a 9×9 discretization (see second column in table 1) and is still lower (better) than the lowest (best) value 945 using a 21×5 discretization (see table 2). Furthermore the same is valid for the minimum achieved number of steps (see fourth column in both tables). The reason is a non-optimum discretization of the speed range which has only 5 tiles. This can be confirmed with a subsequent computation of the learning process with a tiling of 21×9 (see figure 4). Furthermore it is also interesting that CC turned out to be the better discretization method with the better 9×9 discretization. But it can not be concluded that CC is the better discretization method in general, since it shows worse results than TC if used with the worse discretization of 21×5. The influence of context knowledge on the convergence of the learning process is investigated for all four combinations of the two different methods and forms of discretization. The method with application of the context knowledge in the Q-function can achieve an improvement of up to 28% depending on the environment and discretization (see table 2, sixth column) in comparison to the method with a standard reward model. Each run shows an improvement of the convergence. Thus if the discretization is not optimally chosen a significant improvement due to application of context knowledge can be observed (see figure 5). Furthermore a stabilizing effect can be identified. However an enhancement regarding the learning progress due to

Table 1: Discretization with 9×9 tiling: Comparative learning progress values for the 6 methods, standard reward model as benchmark for the standard MCT, Sixth column: Improvement of the learning progress in comparison to the standard reward model using the same discretization method

Method	Learn progress	Episode no.	Min steps	Variance	Improvement
CC-SR	583	125	139	583	-
CC-HR	540	125	143	395	7%
CC-H	561	126	141	415	4%
TC-SR	709	87	186	595	-
TC-HR	681	75	190	516	4%
TC-H	730	81	182	565	-3%

Table 2: Discretization with 21×5 tiling: Comparative values for the shown methods, standard reward model as benchmark for the standard MCT, Sixth column: Improvement of the learning progress in comparison to the standard reward model using the same discretization method

Method	Learn progress	Episode no.	Min steps	Variance	Improvement
CC-SR	2290	144	257	3292	-
CC-HR	1657	76	254	2390	28%
CC-H	2180	130	259	3230	5%
TC-SR	1019	52	252	1123	-
TC-HR	945	72	241	955	7%
TC-H	1011	53	239	1151	1%

application of the heuristic function in the action selection is not clear by evidence. A decrease of the learning progress of up to 5% as well as an increase of up to 3% is measured (see table 1 and 2). But regarding the episode number firstly entering into the convergence range the method is not worse than the other two.
The difference in the minimum achieved number of steps (forth column in table 2) as average of 100 runs is marginal within each combination.
The average variance is similar for each method but is always lower for the method with application of context knowledge. This indicates a better stability (see table 1 and table 2, fifth column). The relation of the variance and the number of outliers in the learning process is shown in figure 6.
Learning curves for the standard MCT with different discretization models and a 9×9 tiling are shown in figure 7.

5.2 3D Mountain Agent Task

The 3D Mountain Agent Task is generated by expansion of the standard MCT with an additional position coordinate and a speed in this coordinate. The task is similar to the one which is described in [Taylor *et al.*, 2008]. The domain for x and y coordinates is expanded to create several mountains. The agent can now move in a 3D space. The third coordinate is determined by the landscape. The position is two-dimensional and consists of two coordinates: $(x, y) \in [-1.2 : 4.8] \times [-1.2 : 4.8]$. The velocity is also two-dimensional and consists of two speeds: $(\dot{x}, \dot{y}) \in [-0.07 : 0.07] \times [-0.07 : 0.07]$. Hence the dimension of the state space is four $(x, \dot{x}, y, \dot{y})$. The starting point is in the coordinates $x^{start} = -0.523, y^{start} = -0.523$. The target is

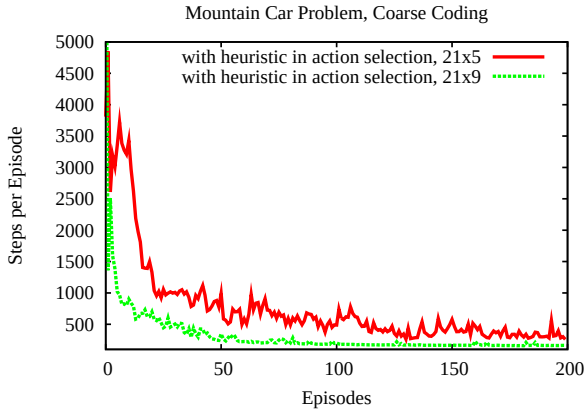


Figure 4: Comparison of the learning curve for a discretization of 21×5 and 21×9

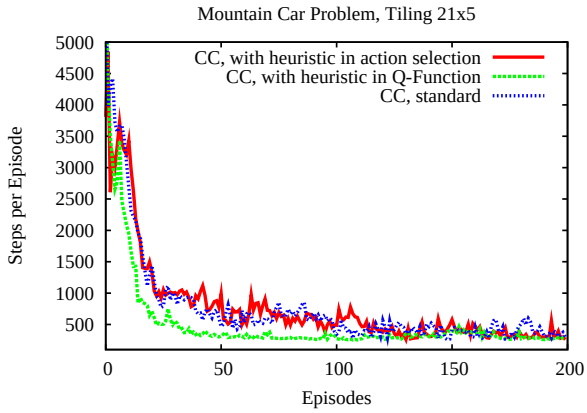


Figure 5: Untypical learning curves of the different methods (heuristic action selection, heuristic in Q-function, without heuristic) with 21×5 CC

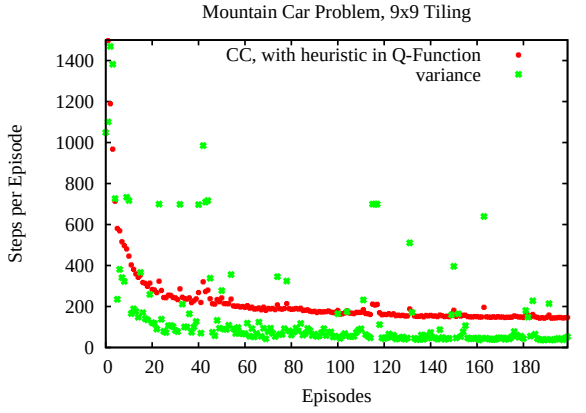


Figure 6: Relation between the learning curve and the variance

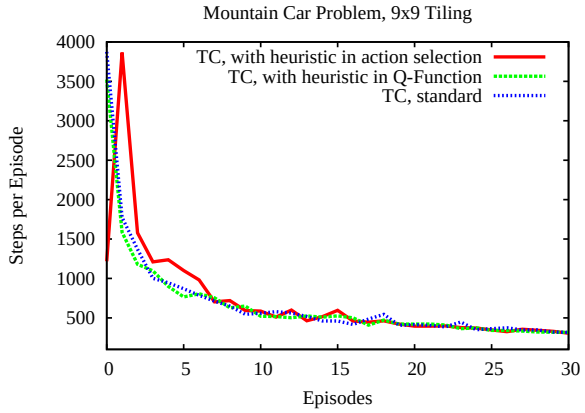


Figure 7: Learning curves of the different methods (heuristic action selectio, heuristic in Q-function, without heuristic) with 9×9 TC

achieved, if both coordinates are $x > 4.7; y > 4.7$.
The control is also a two-dimensional vector. Each action consists of a x and y part, $(a_x, a_y) \in \{(\pm -1, 0); (0, \pm 1); (\pm\frac{1}{\sqrt{2}}, \pm\frac{1}{\sqrt{2}}); (0, 0)\}$. There exist nine actions (on the unit circle). The new state is calculated with the following formulas:

$$x_{t+1} = bound(x_t + \dot{x}_t),\ y_{t+1} = bound(y_t + \dot{y}_t)$$
$$\dot{x}_{t+1} = bound(\dot{x}_t + 0.001 \cdot a_{x_t} - \cos(3x_t) \cdot 0.0025)$$
$$\dot{y}_{t+1} = bound(\dot{y}_t + 0.001 \cdot a_{y_t} - \cos(3y_t) \cdot 0.0025)$$

In table 3 more results are shown for the extended task.
The results confirm an improvement with application of the heuristic rewards in the Q-function of up to 8 % (see table 3). Furthermore the minimum average step number is clearly lower (up to 26%) for the method with application of the context knowledge in the Q-function than for the standard method (see seventh column in table 3).
In figure 9 four different learning trajectories are shown (24th, 48th, 72nd and 98th episode). These trajectories were generated during the learning processes with application of the CC discretization and the heuristic reward function. In all four example episodes the agent needs the first 300 steps in its policy to leave the first valley. Already after the 24th episode (see figure 9(a)) the approach using context knowledge has a good solution to achieve the target. Although the agent needs a low step number to arrive at the destination the chosen trajectory is very sensitive to mistakes and inaccuracies. The agent moves over the hillside. Such states are relatively instable. Due to a wrong move the agent could fall into the next valley which is like a trap. After the 48th episode the agent chooses a stable trajectory (see figure 9(b)). The decision to move at the limit of the environment seems to be logical. The agent is supported by the boundaries of its world in the final part of the green trajectory. A new state is always converted into a valid state. The action set is bounded by this range since the actions which would bring the agent outside the environment are ignored. Hence the agent can make less wrong at the boundary. This can also be observed in the further development of the learning process (see figure 9(c)). In the 96th episode the agent moves on the mountain ridge which is more stable (see figure 9(d)).
The learning curves of the best learning methods for each discretization method are plotted in figure 8. A discretization with CC has a clear advantage to TC. Both best methods are using the context knowledge in the Q-function.

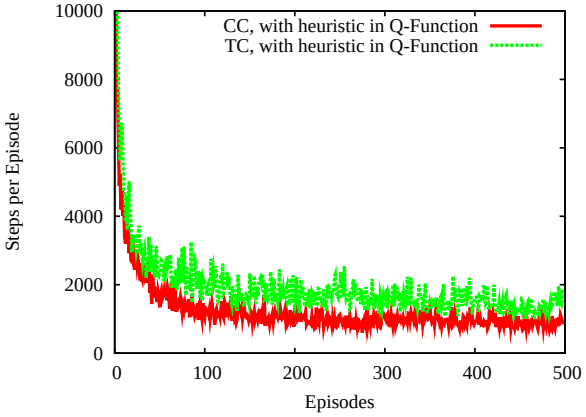


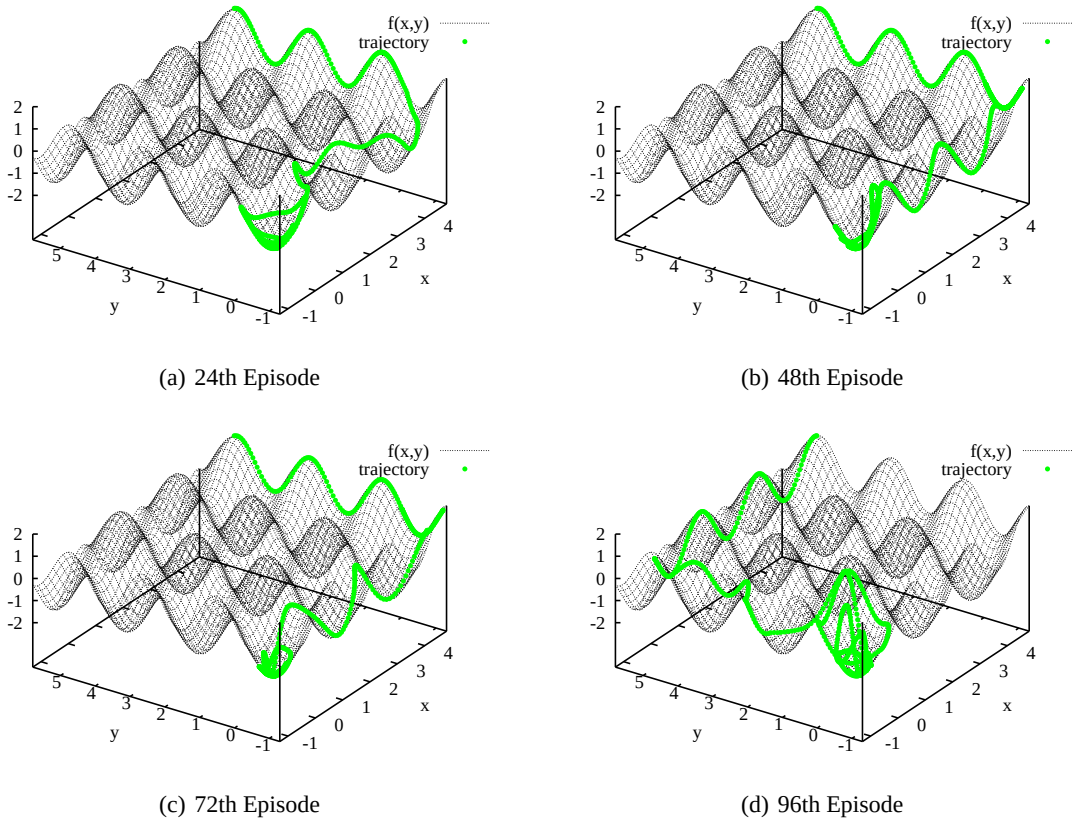
Figure 8: Learning curves for the 3D Mountain Agent Task

Table 3: Comparative values for the suggested methods and the benchmark method with TC for 3D Mountain Agent Task. Improvement A: Learning progress using heuristic in comparison to the standard reward model using the same discretization method. Improvement B: Minimum number of steps using heuristic in comparison to the standard reward model using the same discretization method.

Method	Learn prog.	Epis. no.	Min steps	Variance	Impr. A	Impr. B
CC-SR	1413	101	821	1324	-	-
CC-HR	1297	111	605	1209	8%	26%
CC-H	1413	134	838	1352	0%	-2%
TC-SR	2144	113	1343	2066	-	-
TC-HR	2006	130	1003	1933	6%	25%
TC-H	2117	126	1244	2008	1%	7%

6 Discussion

Two methods are proposed which use context knowledge and integrate it into the learning process. The result is an acceleration of the learning convergence in comparison to the standard method, which does not use the context knowledge. The advantages of the application of the heuristic function are proven for the standard MCT as well as for an extended 3D Mountain Agent Task. For the 3D Mountain Agent Task also the minimum number of steps for the best found solution is significantly lower which results in a shorter way to the target. Furthermore the used method and type for the discretization of the state space have a high impact on the learning process. Learning is faster using a discretization with 9×9 tiles than with 21×5 tiles for the standard MCT. The performed investigation on two example tasks shows promising results. The



(a) 24th Episode

(b) 48th Episode

(c) 72th Episode

(d) 96th Episode

Figure 9: Development of the learning process, trajectories in the 24th, 48th, 72th and 96th episode, heuristic in Q-function, 3D Mountain Car Problem, $21 \times 5 \times 21 \times 5$ Tiling, CC.

application of heuristic functions should be further investigated for other agent problems like the navigation problem in real environments.

In the present work only an example for a heuristic function is investigated. The application of other context knowledge from the environment in form of a heuristic function is possible. Such context knowledge could be for example the information if the agent is moving upwards or downwards the hill. Another heuristic function could evaluate if the target is on the right or the left side of the agent. This and other possible heuristic functions will be investigated in future.

References

[Bertsekas, 1995] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[Judea, 1984] Pearl Judea. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[Kassahun, 2006] Yohannes Kassahun. Evolutionary learning of neural structures for visuo-motor control, 2006.

[Papierok *et al.*, 2008] S. Papierok, A. Noglik, and J. Pauli. Application of Reinforcement Learning in a Real Environment using RBF Networks. In *Proceedings of the International Workshop on Evolutionary Learning for Autonomous Robot Systems*, pages 17–22, Juli 2008.

[Russell and Norvig, 2003] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[Siebel *et al.*, 2008] Nils T. Siebel, Gerald Sommer, and Yohannes Kassahun. Evolutionary learning of neural structures for visuo-motor control. In *Computational Intelligence in Medical Informatics*, pages 93–115. Springer, 2008.

[Sutton, 1998] Richard S. Sutton. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, 1998.

[Taylor *et al.*, 2008] Matthew E. Taylor, Nicholas K. Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, volume 5212 of *Lecture Notes in Artificial Intelligence*, pages 488–505, September 2008.

[Watkins and Dayan, 1992] Christopher J. C. H. Watkins and Peter Dayan. Q-learning2. *Machine Learning*, 8(3-4):279–292, 1992.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

On the relation between Ant Colony Optimization and Heuristically Accelerated Reinforcement Learning *

Reinaldo A. C. Bianchi
Centro Universitário FEI
São Bernardo do Campo, Brazil.
rbianchi@fei.edu.br

Carlos H. C. Ribeiro
Technological Institute of Aeronautics
Computer Science Division
São José dos Campos, Brazil.
carlos@comp.ita.br

Anna H. R. Costa
Escola Politécnica
University of São Paulo
São Paulo, Brazil.
anna.reali@poli.usp.br

Abstract

This paper has two main goals: the first is to propose a new class of Heuristically Accelerated Reinforcement Learning algorithms (HARL), the Distributed HARLs, describing one algorithm of this class, the Heuristically Accelerated Distributed Q-Learning (HADQL); and the second is to show that Ant Colony Optimization (ACO) algorithms can be seen as instances of Distributed HARLs algorithms. In particular, this paper shows that the Ant Colony System (ACS) algorithm can be interpreted as a particular case of the HADQL algorithm. This interpretation is very attractive, as many of the conclusions obtained for RL algorithms remain valid for Distributed HARL algorithms, such as the guarantee of convergence to equilibrium. In order to better evaluate the proposal, we compared the performances of the Distributed Q-Learning, the HADQL and the ACS algorithms in the Traveling Salesman Problem domain. The results show that HADQL and the ACS algorithm have similar performances, as it would be expected from the hypothesis that they are, in fact, instances of the same class of algorithms.

1 Introduction

In the last few years several researchers noticed the similarity between Ant Colony Optimization (ACO) and Reinforcement Learning (RL) [Stützle and Dorigo, 2002; Dorigo and Blum, 2005]. Despite having different inspirational sources – ACO is inspired in the foraging behavior of real ants, while RL is based on Optimal Control Theory –, they have several characteristics in common, such as the use of Markov Decision Process as a way to formulate the problem and provide convergence proofs for the algorithms [Stützle and Dorigo, 2002], and the similarity between the action-value function in RL and the pheromone in ACO, among other aspects.

The major differences between ACO and RL are that the first is a distributed approach, with several agents working to find a solution to a given problem and that ACO makes use of a heuristic evaluation of which moves are better. This last difference – the use of heuristics by the ACO – made it difficult to completely model an ACO algorithm as a RL one.

Now these differences can be addressed by using a recently proposed technique, the Heuristically Accelerated Reinforcement Learning (HARL) [Bianchi *et al.*, 2008]. This technique was proposed to speed up RL methods by making use of a conveniently chosen heuristic function, which is used for selecting appropriate actions to perform in order to guide exploration during the learning process. HARL techniques are based on firm theoretical foundations, allowing many of the conclusions obtained for RL to remain valid, such as the guarantee of convergence to an optimal solution in the limit.

This paper presents two contributions: the first one is the proposal of a new class of HARL algorithms, the Distributed HARL (HADRL), and the description and implementation of one algorithm of this class, the Heuristically Accelerated Distributed Q-Learning (HADQL), which extends the Distributed Q-learning (DQL) algorithm proposed by Mariano and Morales [2001]. The second contribution is a demonstration that ACO algorithms can be seen as instances of Distributed HARLs algorithms. In particular, the paper shows that the Ant Colony System (ACS) algorithm [Dorigo and Gambardella, 1997] can be considered a particular case of the HADQL algorithm. This interpretation is very attractive, as many of the conclusions obtained for RL algorithms remain valid for Distributed HARL algorithms, such as the guarantee of convergence to equilibrium.

The domain studied herein is that of the Traveling Salesman Problem, which is used as a benchmark for testing the algorithms with the goal of evaluating HADQL and comparing it with the DQL and the ACS. Nevertheless, the technique proposed in this work is domain independent.

The remainder of this paper is organized as follows: Section 2 briefly reviews the RL problem and the Distributed Q–Learning algorithm, while Section 3 describes the HAQL approach and its solutions. Section 4 shows how to incorporate heuristics in the DQL algorithm. Section 5 presents the ACO and the ACS algorithm, and Section 6 shows how ACS can be seen as a HARL algorithm. Section 7 presents the experiments performed and shows the results obtained. Finally, Section 8 provides our conclusions and outlines future work.

*The authors would like to thanks FAPESP for supporting this project. Reinaldo Bianchi also acknowledge the support of the CNPq (Grant No. 201591/2007-3) and FAPESP (Grant No. 2009/01610-1).

2 Reinforcement Learning and the DQL algorithm

Reinforcement Learning (RL) algorithms have been applied successfully to the on-line learning of optimal control policies in Markov Decision Processes (MDPs). In RL, this policy is learned through trial-and-error interactions of the agent with its environment: on each interaction step the agent senses the current state s of the environment, chooses an action a to perform, executes this action, altering the state s of the environment, and receives a scalar reinforcement signal r (a reward or penalty).

The RL problem can be formulated as a discrete time, finite state, finite action Markov Decision Process (MDP). The learning environment can be modeled by a 4-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where:

- $\mathcal{S}$: is a finite set of states.
- $\mathcal{A}$: is a finite set of actions that the agent can perform.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$: is a state transition function, where $\Pi(\mathcal{S})$ is a probability distribution over $\mathcal{S}$. $T(s, a, s')$ represents the probability of moving from state s to s' by performing action a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$: is a scalar reward function.

The goal of the agent in a RL problem is to learn an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maps the current state s into the most desirable action a to be performed in s. One strategy to learn the optimal policy π^* is to allow the agent to learn the evaluation function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. Each action value $Q(s, a)$ value represents the expected cost incurred by the agent when taking action a at state s and following an optimal policy thereafter.

The Q–learning algorithm [Watkins, 1989] is a well-know RL technique that uses a strategy to learn an optimal policy π^* via learning of the action values. It iteratively approximates Q, provided the system can be modeled as an MDP, the reinforcement function is bounded, and actions are chosen so that every state-action pair is visited an infinite number of times. The Q learning update rule is:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a) \right], \quad (1)$$

where s is the current state; a is the action performed in s; r is the reward received; s' is the new state; γ is the discount factor ($0 \leq \gamma < 1$); and α, is the learning rate.

In the case of the use of the Q–Learning algorithm to solve the TSP, the state s corresponds to the city in which the agent is at a given moment, and the set of actions that an agent can execute corresponds to the set of cities in the problem, excluding the current city s.

To select an action to be executed, the Q–Learning algorithm usually considers an $\epsilon - Greedy$ strategy:

$$\pi(s) = \begin{cases} \arg\max_a \hat{Q}(s, a) & \text{if } q \leq p, \\ a_{random} & \text{otherwise} \end{cases} \quad (2)$$

where q is a random value uniformly distributed over $[0, 1]$ and p ($0 \leq p \leq 1$) is a parameter that defines the exploration/exploitation tradeoff: the larger p, the smaller is the probability of executing a random exploratory action, and a_{random} is an action randomly chosen among those available in state $\mathbf{s}_t$.

Table 1: The DQL algorithm [Mariano and Morales, 2001].

```
Initialize Q(s, a) arbitrarily
Repeat (for n episodes):
    Repeat (for each agent i in the set of m agents):
        Initialize s, copy Q(s, a) to Qc^i(s, a)
        Repeat (for each step of the episode):
            Select an action a, observe r, s'.
            Update the Qc^i(s, a) values.
            s ← s'.
        Until s is terminal.
    Evaluate the m solutions.
    Assign reward to the best solution found.
    Update the global Q(s, a).
Until a termination criterion is met.
```

Several authors have proposed distributed approaches to RL, and various forms of distributed Q–Learning were developed [Pendrith, 2000; Lauer and Riedmiller, 2000; Gu and Maddox, 1996; Mariano and Morales, 2001]. One of these is the Distributed Q-learning algorithm (DQL) proposed by Mariano and Morales [2001], which is a generalization of the Q-learning algorithm where, instead of a single agent, several independent agents are used to learn a single policy.

In Mariano and Morales' DQL, in addition to the global $Q(s, a)$ function, each agent i keeps a temporary copy of Q, the $Qc_i(s, a)$ that is used to decide which action to perform, following an ϵ-greedy policy. Every time an action is performed, $Qc_i(s, a)$ is updated according to the Q-Learning update rule (Equation 1).

The DQL agents explore different options in a common environment and when all agents have completed a solution, their solutions are evaluated, and the global $Q(s, a)$ table is updated: the best solution (the shortest route made by all agents in the TSP) is updated using Equation (1), and receiving a reward r. The DQL algorithm is presented in table 1.

As DQL does not change the update rules of the Q–Learning algorithm, the same proofs of convergence used for the standard Q–Learning remains valid for it [Mariano and Morales, 2001]. Different Distributed Q-Learning algorithms, proposed by other authors, also hold convergence proofs [Lauer and Riedmiller, 2000].

3 Heuristic Accelerated Reinforcement Learning

Formally, a Heuristically Accelerated Reinforcement Learning (HARL) algorithm is a way to solve a MDP problem with explicit use of a heuristic function $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ for influencing the choice of actions by the learning agent. $H(s, a)$ defines the heuristic that indicates the importance of performing the action a when visiting state s. The heuristic function is strongly associated with the policy: every heuristic indicates that an action must be taken regardless of others.

The heuristic function is an action policy modifier which does not interfere with the standard bootstrap-like update

mechanism of RL algorithms. A possible strategy for action choice is an $\epsilon - greedy$ mechanism where a heuristic mechanism formalized as a function $H(s,a)$ is considered, thus:

$$\pi(s) = \begin{cases} \arg\max_a \left[\mathsf{F}(s,a) \bowtie \xi H(s,a)^\beta\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise} \end{cases} \quad (3)$$

where:

- $\mathcal{F} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is an estimate of a value function that defines the expected cumulative reward. If $\mathsf{F}(s,a) \equiv \hat{Q}(s,a)$ we have an algorithm similar to standard *Q–Learning*.
- $\mathcal{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ is the heuristic function that plays a role in the action choice. $H(s,a)$ defines the importance of executing action s in state s.
- $\bowtie$ is a function that operates on real numbers and produces a value from an ordered set which supports a maximization operation.
- ξ and β are design parameters that control the influence of the heuristic function.
- q is a parameter that defines the exploration/exploitation tradeoff.
- a_{random} is an action randomly chosen among those available in state s.

In general, the value of $H(\mathbf{s},a)$ must be larger than the variation among the values of $\mathsf{F}(\mathbf{s}_a)$ for a given $\mathbf{s} \in S$, so that it can influence the action choice. On the other hand, it must be as small as possible in order to minimize the error. If $\bowtie$ is a sum and $\xi = \beta = 1$, a heuristic can be defined as:

$$H(\mathbf{s},a) = \begin{cases} \max_a \left[\mathsf{F}(\mathbf{s},a)\right] - \mathsf{F}(\mathbf{s},a) + \eta & \text{if } a = \pi^H(\mathbf{s}), \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

where η is a small value and $\pi^H(\mathbf{s})$ is a heuristic obtained using an appropriate method.

For instance, let $[1.0\ 1.1\ 1.2\ 0.9]$ be the values of $\mathsf{F}(\mathbf{s},a)$ for four possible actions $[a_1\ a_2\ a_3\ a_4]$ for a given state $\mathbf{s}_t$. If the desired action is the first one (a_1), we can use $\eta = 0.01$, resulting in $H(\mathbf{s},a_1) = 0.21$ and zero for the other actions (see Figure 1). The heuristic can be defined similarly for other definitions of $\bowtie$ and values of ξ and β.

Convergence of the first HARL algorithm — Heuristically Accelerated Q–Learning (HAQL) — is presented in Bianchi *et al.* [2008], together with the definition of an upper bound for the error in the estimation of Q. The same authors investigated the use of HARL in multiagent domain, proposing a multiagent HARL algorithm – the Heuristically Accelerated Minimax-Q [Bianchi *et al.*, 2007] – and testing it in a simplified simulator for the robot soccer domain.

4 The HADQL algorithm

The Heuristically Accelerated Distributed Q–Learning algorithm is a HARL algorithm that extends the DQL algorithm by making use of an heuristic function in the action choice

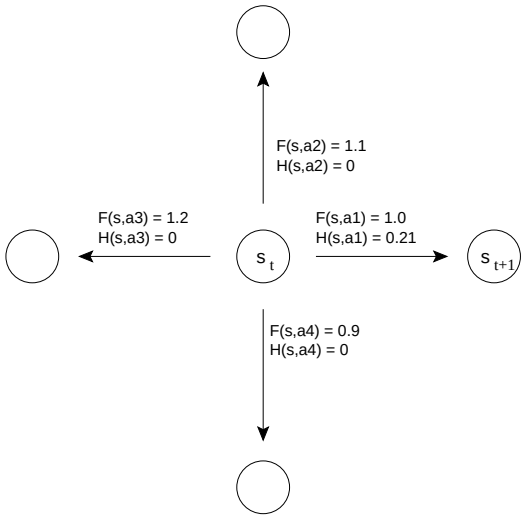


Figure 1: Suppose a state s_t and a desired state s_{t+1}. The value of $H(s_t,a_1)$ for the action that leads to s_{t+1} is 0.21, and zero for the other actions.

rule defined in Equation (3), where $\mathcal{F} = Q$, the $\bowtie$ operator is the sum and $\beta = 1$:

$$\pi(s) = \begin{cases} \arg\max_a \left[\hat{Q}(s,a) + \xi H(s,a)\right] & \text{if } q \leq p, \\ a_{random} & \text{otherwise,} \end{cases} \quad (5)$$

where all variables are defined as in Equation (3). The value of the heuristic can be defined by instantiating Equation 4:

$$H(s,a) = \begin{cases} \max_i \hat{Q}(s,i) - \hat{Q}(s,a) + \eta \text{ if } a = \pi^H(s), \\ 0 \text{ otherwise.} \end{cases} \quad (6)$$

where η is a small real value (usually 1) and $\pi^H(s)$ is the action suggested by the heuristic policy.

As the heuristic is used only in the choice of the action to be taken, the DQL operation is not modified (i.e., updates of the function Q are as in Q–learning), and it thus allows that many of the conclusions obtained for DQL remain valid for HADQL. The HADQL algorithm is presented in table 2.

Theorem 1. *Consider a HADQL system learning in a deterministic MDP, with finite sets of states and actions, bounded rewards* $(\exists c \in \Re; (\forall s,a), |r(s,a)| < c)$*, discount factor* γ *such that* $0 \leq \gamma < 1$ *and where the values used on the heuristic function are bounded by* $(\forall s,a)\ h_{min} \leq H(s,a) \leq h_{max}$*. For this algorithm, the* $\hat{Q}$ *values will converge to* Q^**, with probability one uniformly over all the states* $s \in \mathcal{S}$*, if each state-action pair is visited infinitely often (obeys the Q–learning infinite visitation condition).*

Proof. In HADQL, the update of the value function approximation does not depend explicitly on the value of the heuristic. The necessary conditions for the convergence of DQL that could be affected with the use of the heuristic algorithm

Table 2: The HADQL algorithm

Initialize $Q(s,a)$ and $H(s,a)$ arbitrarily
Repeat (for n episodes):
  Repeat (for each agent i in the set of m agents):
    Initialize s, copy $Q(s,a)$ to $Qc^i(s,a)$
    Repeat (for each step of the episode):
      Compute the value of $H(s,a)$ using Eq. 6.
      Select an action a using Eq. 5.
      Observe r, s'.
      Update the $Qc^i(s,a)$ values.
      $s \leftarrow s'$.
    Until s is terminal.
  Evaluate the m solutions.
  Assign reward to the best solution found.
  Update the global $Q(s,a)$.
Until a termination criterion is met.

HADQL are the ones that depend on the choice of the action. Of the conditions presented in Littman and Szepesvári [1996]; Mitchell [1997], the only one that depends on the action choice is the necessity of infinite visitation to each pair state-action. As equation 5 considers an exploration strategy ϵ– *greedy* regardless of the fact that the value function is influenced by the heuristic, the infinite visitation condition is guaranteed and the algorithm converges. □

The condition of infinite visitation of each state-action pair can be considered valid in practice also by using visitation strategies such as Boltzmann exploration [Kaelbling *et al.*, 1996], intercalating steps where the algorithm makes alternate use of the heuristic and exploration, or using the heuristic during a period of time, smaller than the total learning time.

The domain studied in this work is that of the Traveling Salesman Problem (TSP), which consists in, given a number n of cities $C = c_1, c_2, \ldots c_n$ and the distance $d_{i,j}$ between them, to find the shortest route that visits each city in C at least once and then returns to the starting city.

To compute an heuristic function for the TSP problem, we were inspired by the Nearest Neighbor Heuristic used in several works [Russell and Norvig, 1995]. This heuristic states that the agent starts at some city and from there it visits the nearest city that was not visited so far. Using this rule, a simple heuristic that indicates to which city an agent must move can be defined as the inverse of the distance $d_{i,j}$ between the cities i and j times a constant η:

$$H(s,a) = \frac{\eta}{d_{i,j}}, \tag{7}$$

where s is the current state (i.e., the current city c_i) and a is the action that takes the agent to the city c_j.

The problem with this heuristic is that it does not take into account that the agents must not visit two times the same city. Therefore, the action chosen as the one to be done (the heuristic policy) is the one that moves the agent to the nearest city that was not visited yet. To put this idea in the framework of equation 6, and taking into account the cities that were already visited, the heuristic function becomes:

$$H(s,a) = \begin{cases} \max\limits_{x} \hat{Q}(s,x) - \hat{Q}(s,a) + \eta & \text{if } \delta_{i,j} = \min\limits_{y} \delta_{i,y} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

where η is a small real value (usually 1), a is the action that takes the agent from city i to j and $\delta_{i,j}$ the distances between the city c_i and the cities that have not been visited so far, defined by:

$$\delta_{i,j} = \begin{cases} d_{i,j} & \text{if } j \text{ has not been visited} \\ \infty & \text{otherwise.} \end{cases} \tag{9}$$

Finally, the agents receive only negative reinforcements, defined as minus the distance between the cities, $r = -d_{i,j}$.

5 The Ant Colony System Algorithm

Based on the social insect metaphor for solving problems, the use of Ant Colony Optimization (ACO) for solving several kinds of problems has attracted an increasing attention of the AI community [Bonabeau *et al.*, 1999, 2000; Dorigo and Blum, 2005]. The Ant Colony System (ACS) is an ACO algorithm proposed by Dorigo and Gambardella [1997] for combinatorial optimization based on the observation of ant colonies behavior.

ACS has been applied to various combinatorial optimization problems like the symmetric and Asymmetric Traveling Salesman Problems (TSP and ATSP respectively), and the quadratic assignment problem.

ACS represents the usefulness of moving to the city s when in city r in $\tau(r,s)$, called *pheromone*, which is a positive real value associated to the edge (r,s) in a graph. There is also a *heuristic* $\eta(r,s)$ associated to the edge (r,s). It represents an heuristic evaluation of which moves are better. In the TSP, $\eta(r,s)$ can be the inverse of the distance δ from r to s, $\delta(r,s)$.

An agent k positioned in city r moves to city s using the following rule, called state transition rule [Dorigo and Gambardella, 1997]:

$$s = \begin{cases} \arg\max\limits_{u \in J_k(r)} \tau(r,u) \cdot \eta(r,u)^{\beta} & if\ q \leq q_0 \\ S & otherwise \end{cases} \tag{10}$$

where:

- β is a parameter which weights the relative importance of the learned pheromone and the heuristic distance values ($\beta > 0$).
- $J_k(r)$ is the list of cities still to be visited by the ant k, where r is the current city. This list is used to constrain agents to visit cities only once.
- q is a parameter that defines the exploitation/exploration rate.
- S is a random city from the list of cities $J_k(r)$.

Ants in ACS update the values of $\tau(r,s)$ in two situations: in the local update step (applied when ants visit edges) and in the global update step (applied when ants complete the tour).

Table 3: The ACS algorithm (in the TSP Problem).

Initialize the pheromone table, the ants and the list of cities.
Repeat (for n episodes):
  Repeat (for each ant i in the set of m ants):
    Put each ant at a starting city.
    Repeat (for each step of the episode):
      Chose next city using Equation (10).
      Update list J_k of yet to be visited cities for ant k.
      Apply local update using Equation (11).
    Until (ants have a complete tour).
  Apply global update using Equation (12).

The ACS local update rule is:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\Delta\tau(r,s) \qquad (11)$$

where $0 < \rho < 1$ is a the learning rate, and $\Delta\tau(r,s) = \gamma \cdot \max_{z\in J_k(s)} \tau(s,z)$.

The ACS global update rule is:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau(r,s) \qquad (12)$$

where α is the pheromone decay parameter, and $\Delta\tau(r,s)$ is the inverse of the length of the best tour, given only to the tour done by the best agent – only the edges belonging to the best tour will receive more pheromones.

Thus, the pheromone updating formulas aims at placing a greater amount of pheromone on the shortest tours, achieving this by simulating the addition of new pheromone deposited by ants and evaporation. The ACS algorithm is presented in table 3.

6 ACS as HARL

Now that the HADQL algorithm is defined, we proceed in explaining how ACS can be seen as an HARL algorithm:

- The Pheromone is the ACS counterpart of DQL Q-values, where the city r in which the ant is at a defined moment corresponds to the state s, and the city s the ant should move to, corresponds to the action a to be taken: $\tau(r,s) = \hat{Q}(s,a)$;
- The heuristic $\eta(r,u)$ corresponds to the heuristic function $H(s,a)$;
- The state transition rule (Eq. 10) is the same as the one used by HARL algorithms (Eq. 3), with the $\bowtie$ operator being the multiplication, $\xi = 1$ and β having the same function;
- The list $J_k(r)$ of cities still to be visited by the ant, which causes problems for the convergence proof of ACS (because it cannot be defined as a MDP - see Koenig and Simmons [1996]), can be encoded as an heuristic, as in Equation 8;
- The local update step (Eq. 11) of $\tau(r,s)$ is made in the same way as the updating of Qc in DQL (without giving a reinforcement),
- The global update step (Eq. 12) in ACS corresponds to attributing the reinforcement to the best solution, which is made in the updating of the global Q of DQL, with the pheromone decay parameter α being equal to the learning factor and $\Delta\tau(r,s)$ corresponding to a delayed reward.

ACS and HADQL as proposed in this work differ by the fact that HADQL partial updates are performed over copies of the Q-table, and that HADQL updates the best solution at the same time it provides the reinforcement (that is, the last step of HADQL corresponds to executing both a local and a global ACS update at the same time). According to Mariano and Morales [2001], this avoids multiple updates of the same Q-table, resulting in updating only relevant solutions and allowing faster convergence.

Finally, by modeling ACS as a HARL, it is possible to show that ACS also converges to equilibrium in the limit.

Theorem 2. *Consider ACS in a deterministic MDP, with finite sets of states and actions (a finite number of cities to visit), bounded rewards* $(\exists c \in \Re; (\forall r,s), |\Delta\tau(r,s)| < c)$*, learning rate* ρ *and pheromone decay parameter* α *with values between* 0 *and* 1 *and where the values used on the heuristic are bounded by* $(\forall r,s)\ \eta_{min} \leq \eta(r,s) \leq \eta_{max}$*. For this algorithm, the* $\hat{\tau}$ *values will converge to* τ^**, with probability one uniformly over all the states* $s \in \mathcal{S}$*, if each pair* (r,s) *is visited infinitely often.*

Proof. The only difference between ACS and HADQL is that ACS executes more local updates. But the effect of having more local updates is that the pair (r,s) is visited more often, assigning rewards more frequently. As HADQL converges, without executing the local updates, ACS also converges. □

Other ways by which it could be shown that ACS converges is by modeling it as a HADQL that includes cooperation among the agents, such as one based on Lauer and Riedmiller [2000], or by using a DQL that only posses one Q-table, such as in Pendrith [2000].

7 Experiments in the TSP domain

In order to evaluate the performance of the HADQL algorithm and its relation with ACS, this section compares the performances of these algorithms while solving a set of TSPs. The Distributed Q-learning (DQL) algorithm is also included in this comparison, for the sake of comparing the new algorithm with a traditional RL one.

These tests were performed using a standardized dataset, the TSPLIB [Reinelt, 1995]. This library of problems, which was used as benchmark by both Dorigo and Gambardella [1997] and Mariano and Morales [2001], offers standardized optimization problems such as the TSP, truck loading and unloading and crystallography problems. The results, which are the average of 30 training sessions with 1000 episodes, are presented in Tables 4, 6, and 5. Ten different problems were considered: the first 7 ones are TSPs, and the last 3 ones are ATSPs (Asymmetric TSPs). The number of cities in each problem ranged from 48 to 170, and is shown in the name of the problem (for example, 52 cities in the 'berlin52' problem, and so on).

Table 4 presents the best result found by DQL, ACS and HADQL after 1000 iterations (in 30 trials). It can be seen

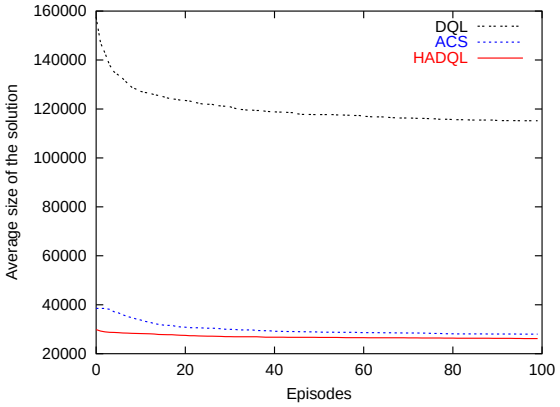


Figure 2: Comparison between the algorithms DQL, ACS and HADQL applied to the kroA100 TSP.

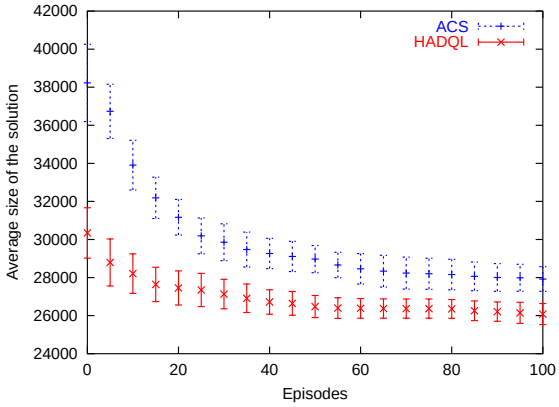


Figure 3: Comparison between the algorithms ACS and HADQL applied to the kroA100 TSP (with errorbars).

Problem	Known Solution	DQL	ACS	HADQL
berlin52	7542	15424	8689	7824
kroA100	21282	105188	25686	24492
kroB100	22141	99792	27119	23749
kroC100	20749	101293	24958	22735
kroD100	21294	101296	26299	23839
kroE100	22068	102774	26951	24267
kroA150	26524	170425	33748	32115
ry48	14422	26757	16413	15398
kro124	36230	122468	45330	42825
ftv170	2755	18226	4228	3730

Table 4: Shortest routes found by the algorithms DQL, ACS and HADQL after 1000 episodes (best of 30 trials).

Problem	DQL	ACS	HADQL
berlin52	16427 ± 540	8589 ± 139	7929 ± 61
kroA100	108687 ± 2474	26225 ± 542	25114 ± 353
kroB100	105895 ± 2949	27337 ± 582	24896 ± 463
kroC100	105756 ± 2710	25985 ± 737	23585 ± 361
kroD100	104909 ± 2293	26188 ± 533	24441 ± 274
kroE100	108098 ± 2652	26723 ± 557	25196 ± 359
kroA150	179618 ± 3397	35816 ± 998	33532 ± 603
ry48	29562 ± 1131	16285 ± 195	15538 ± 58
kro124	127911 ± 2485	46394 ± 599	43482 ± 322
ftv170	19278 ± 373	4532 ± 104	3982 ± 98

Table 5: Average size of the results found by the algorithms DQL, ACS and HADQL after 1000 episodes (average of 30 trials).

that HADQL solutions are better than the solutions from the other two algorithms for all the problems. The same occurs for the average of the best results found after 1000 episodes (Table 5). Figure 2 shows the evolution of the average of the results found by DQL, ACS and HADQL when solving the kroA100 problem, and Figure 3 shows the same results for the ACS and HADQL, with errorbars. It is possible to see in both figures that HADQL converges faster to the solution.

The average time to find these solutions, shown in Table 6, indicates that the DQL is the fastest algorithm. This occurs because DQL converges first, but to a solution of poorer quality. A comparison between mean times to find the best solution between the ACS and HADQL algorithms shows that there is no significant difference between them. It is worth noticing that small improvements may occur at any time, because both algorithms are following an $\epsilon-greedy$ exploration policy. For this reason, the variation in results is very large, which is reflected in the error measure.

Finally, Student's t–test [Spiegel, 1998] was used to verify the hypothesis that the HADQL algorithm produces better results that the ACS. This test showed that all the results of the HADQL are better than the ones obtained with ACO, with a level of confidence greater than 99,99%, a fact that shows that the small differences that exists between both algorithms is enough to make HADQL perform slightly but consistently better than ACS. The same test applied to the average time to reach the best result showed that there is no significant difference in the performance of the two algorithms.

The parameters used in the experiments were the same for the three algorithms: the learning rate is $\alpha = 0,1$, the exploration/exploitation rate is $p = 0.9$ and the discount factor $\gamma = 0.3$. The value of β in the ACS algorithm was set to 2 and the value of $\eta = 10$ in the HADQL (these parameters are identical to those used by Dorigo and Gambardella [1997] and Mariano and Morales [2001]). Values in the Q table were randomly initiated, with $0 \leq Q(s,a) \leq 1$. The experiments were programmed in C++ and executed in a AMD Athlon 2.2MHz, with 512MB of RAM in a Linux platform.

8 Conclusion

In this paper we proposed a new algorithm – HADQL, showed that ACS can be seen as a HARL algorithm and compared the two algorithms in the TSP domain. The results

Problem	DQL	ACS	HADQL
berlin52	7 ± 3	12 ± 7	11 ± 6
kroA100	37 ± 13	89 ± 50	73 ± 43
kroB100	44 ± 17	85 ± 44	88 ± 47
kroC100	51 ± 27	82 ± 48	89 ± 38
kroD100	47 ± 21	98 ± 39	74 ± 39
kroE100	48 ± 22	80 ± 43	80 ± 45
kroA150	91 ± 42	267 ± 136	294 ± 154
ry48	6 ± 3	9 ± 6	3 ± 4
kro124	62 ± 25	89 ± 42	95 ± 43
ftv170	113 ± 73	317 ± 122	333 ± 221

Table 6: Average time (in seconds) to find the best solution using the algorithms DQL, ACO and HADQL, limited to 1000 episodes.

show that HADQL and ACS have similar performances, as it would be expected from the hypothesis that they are, in fact, instances of the same class of algorithms.

Despite the similarity between the HADQL and the ACS algorithms, the results showed a small advantage for the first one. We believe that this advantage is caused by the fact that the HADQL performs partial updates over copies of the Q-table, avoiding updates without rewards, and by the fact that the reward given in both algorithms are different.

Future work include testing other forms of combining Q-values and heuristics in the action selection, and the comparison of other ACO and HARL algorithms, such as comparing the Max-Min Ant System (MMAS) [Dorigo and Blum, 2005] with the HAMMQ [Bianchi *et al.*, 2007].

References

Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, and Anna H. R. Costa. Heuristic selection of actions in multiagent reinforcement learning. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'2007), Hyderabad, India, January 6-12, 2007*, pages 690–695, 2007.

Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, and Anna H. R. Costa. Accelerating autonomous learning by using heuristic selection of actions. *Journal of Heuristics*, 14(2):135–168, 2008.

Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, 1999.

Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. Inspiration for optimization from social insect behaviour. *Nature 406 [6791]*, 2000.

Marco Dorigo and Christian Blum. Ant colony optimization theory: a survey. *Theor. Comput. Sci.*, 344(2-3):243–278, 2005.

Marco Dorigo and Luca M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

Pan Gu and Anthony B. Maddox. A framework for distributed reinforcement learning. In Gerhard Weiß and Sandip Sen, editors, *Adaption and Learning in Multi-Agent Systems, IJCAI'95 Workshop Proceedings, Monréal, Canada, August 21, 1995*, volume 1042 of *Lecture Notes in Computer Science*, pages 97–112, Berlin Heidelberg, Springer, 1996.

Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

Sven Koenig and R. G. Simmons. The effect of representation and knowledge on goal–directed exploration with reinforcement–learning algorithms. *Machine Learning*, 22:227–250, 1996.

Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Standord, CA, USA, June 29 - July 2, 2000*, pages 535–542. Morgan Kaufmann, 2000.

Michael L. Littman and Csaba Szepesvári. A generalized reinforcement learning model: convergence and applications. In Lorenza Saitta, editor, *Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 310–318. Morgan Kaufmann, 1996.

Carlos E. Mariano and Eduardo F. Morales. DQL: A new updating strategy for reinforcement learning based on Q–learning. In Luc De Raedt and Peter A. Flach, editors, *EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5–7, 2001*, volume 2167 of *Lecture Notes in Computer Science*, pages 324–335, Berlin Heidelberg, Springer, 2001.

Tom Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

Mark D. Pendrith. Distributed reinforcement learning for a traffic engineering application. In Carles Sierra, Maria Gini, Jeffrey S. Rosenschein, editors, *Proceedings of the fourth international Conference on Autonomous agents, June 3-7, 2000, Barcelona, Catalonia, Spain*, pages 404–411. ACM, 2000.

Gerhard Reinelt. Tsplib95. Technical report, Universitat Heidelberg, 1995. Technical Report. Universitat Heidelberg.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.

Murray R. Spiegel. *Statistics*. McGraw-Hill, 1998.

Thomas Stützle and Marco Dorigo. A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans. Evolutionary Computation*, 6(4):358–365, 2002.

Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Combining Learning, Deliberation and Reactive Control in Simulated Soccer: The DAInamite Framework

Martin Berger, Holger Endert and **Simon Joecks**
DAI-Labor, TU Berlin
Faculty of Electrical Engineering
and Computer Science
{martin.berger, holger.endert, simon.joecks}@dai-labor.de

Abstract

Developing the behaviour of agents that act in complex domains is often done by following a *divide & conquer* strategy. There, the resulting pieces of work usually have to be finished by applying different solution methods like learning, planning, rule-based systems or a combination of them. In this paper we present a framework for simulated soccer agents that enables the integration and combination of these methods in a straightforward manner. We furthermore demonstrate the fitness of our approach in a case study in the Robotic Soccer domain.

1 Introduction

Since the very beginning of the studies on multi-agent systems, many architectures and frameworks for creating agents for different purposes have been developed and published so far. Usually, the properties of the domain in which the agent's act determine the type of the framework that has to be used. For instance, having hard real-time constraints leads to reactive control, whereas for intelligent applications, BDI frameworks may be suited best. However, in many cases there is no best or unique way of building agents, such that frameworks ideally have to support the development of hybrid approaches.

Implementing the behaviour of agent's for real-world scenarios is one instance of a very challenging task. An example of such a domain is Robotic Soccer, in which teams of robots compete against each other in a soccer challenge. With respect to the software, the current research focus lies mainly in machine learning, whereas especially reinforcement learning was applied extensively (e.g. in [Riedmiller and Withopf, 2005]). In [Stone, 2000], an approach which uses several machine learning algorithms to achieve a joint strategy of a robotic soccer agent has been developed, following an approach called *layered learning*. Further the use of neural networks, decision trees and other AI techniques is rather common.

So far, only few work has been investigated on frameworks that ease the development of agents in such domains. One exception is given in [Berger, 2006], which uses a task-tree that allows for BDI-type reasoning in combination with utility functions in the so called *Doppelpass-Architektur*. Though it addresses many of the requirements, especially the integration of learned behaviours is left open. In [de Boer and Kok, 2002], the agents were even implemented without a specific framework or methodology and this work has been used for a long period of time as a reference. A framework that supports teamwork [Tambe, 1997] was applied in the early stages of RoboCup, but due to the restricted communication bandwidth with only limited success. Finally, a short description of a hybrid framework that incorporates reactive and deliberative control was published by [Sun and Wu, 2006]. The integration of learned behaviours is also still an open matter.

As a result, there is room left for research on architectures and frameworks in the robotic soccer domain. Therefore we approach the issue of designing a framework that offers support in the development of intelligent agents without limiting to either one of the aforementioned solution techniques. Furthermore, well-proven concepts like task-trees or utility functions are incorporated. The framework is fully implemented, and we provide a case study which serves as a proof of concept and gives rise to the underlying methodology.

The remainder of this paper is organized as follows: In Section 2, we provide an overview about RoboCup and the Simulation League 2D. In Section 3, we explain the main components and features of our framework, especially the integration of the different approaches for behaviour development. Thereafter we demonstrate the adequacy of the framework by presenting a case study in Section 4. Finally we close with a discussion of our results in Section 5, and we conclude and give rise to future work in Section 6.

2 RoboCup and Simulated Soccer 2D

The RoboCup initiative[1] [Kitano *et al.*, 1998] aims at promoting research that combines AI, robotics and further related fields. The ultimate goal is to be able to build intelligent and humanoid robots that can perform difficult or even dangerous tasks in the future. To this end, *robotic soccer* was introduced as challenging test bed, and since then researchers from all over the world present their advancements and let their (possibly simulated) robots compete against each other in annual competitions. In order to have comparable results in the different areas of research, sub-leagues with a distinct

[1] See http://www.robocup.org

focus have been established. While most of these leagues have robotics as central theme, the simulation leagues have a strong relation to multi-agent systems and AI.

RoboCup Simulated Soccer 2D [Noda *et al.*, 1998] has been a popular test bed for developing software agents in the past. The simulation runs in a client-server style, whereas the server (simulator) controls every aspect of the environment. The agents connect to the simulator and take control over one player each.

The simulator abstracts from real hardware and uses a rather simple two dimensional model of the environment. Nevertheless it contains almost all relevant properties and restrictions of real-world applications. A discrete time model is used, in which each simulation step (called *cycle*) lasts 100ms. Agents are modeled as circular shapes with an orientation (see Figure 1). They can perform actions like moving or kicking the ball, and they have to build and maintain a world-model from sensory input received by the simulator. The communication bandwidth between agents is very limited and noise is added to both, sensory data and actions, by the simulator.

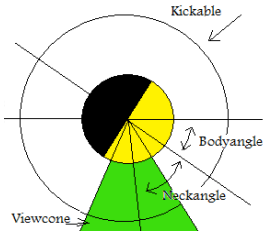


Figure 1: The model of a player in the RoboCup 2D Simulation League

In summary, teams of agents that act in the Simulated Soccer domain have to deal with real-time constraints, high uncertainty in sensing and acting and an adversarial that tries to prevent the fulfillment of the respective goals.

3 The DAInamite Framework

The DAInamite Framework has it's origin in the development of a team for the RoboCup 2D Simulation League called DAInamite[2], which participates since 2006 in the annual championships. Since then, the agent improved in structure and functionality, and later, the framework was extracted from the team's code. Though it was only applied in simulated soccer so far, we believe that it will be usable in similar domains without many adaptations.

In the remainder of this section, the structure and internal control of the agent is presented in depth, focusing on the representation of the behaviours and the action selection algorithm.

3.1 The Agent

The agent consists of three main components which are the *synchronization*, the *world-model* and the *tactic*. The *synchronization* controls the internal workflow of the agent, i.e. when to integrate new sensory data and when to compute and execute the next action. It generally realizes the *sense-think-act* control paradigm [Russell and Norvig, 2003] which is common for robots and agents that act in real-world settings[3]. The *world-model* contains the beliefs of the agent about the environment, and is kept up-to-date by processing sensory input. The *tactic* encapsulates the available behaviours of the agent plus the action selection algorithm. Both, the *tactic* and the *synchronization*, run in concurrent processes.

The World-Model

The world-model consists of two parts. The first is a data storage, which is subsequently called belief-base or simply *beliefs*. It may be accessed and queried from outside the world-model. The second is a list of update routines that are responsible for adding given and derived knowledge from sensor data into the data storage. The update routines are called *facts*. These are only visible from within the world-model. The basic framework contains *facts* that are able to calculate the positions, velocities and other related or important knowledge about the own player, about the teammates, the opponents and about the ball. Further *facts* implement derived knowledge, such as the next expected position, where the ball is intercepted by the fastest player, called the *interception point* [4].

The beleif-base has an interface which is accessible from outside the world model, especially from the tactic.

The Tactic

The structure of the tactic is basically a tree, in which the leafs represent specific behaviours, e.g. dribbling or scoring, and the nodes represent application conditions. An example tree is given in Figure 2. There, the node *Striker* is a condition that evaluates the role of a player, whereas the nodes called *PenaltyTaken* or *PenaltyReady* evaluate the state of the game (i.e. the play-mode). The leafs represent the behaviours for dribbling, scoring, outplay-goalie and intercepting the ball. It is easy to see that all conditions from a behaviour node to the root of the tree must be fulfilled in order to let the corresponding behaviour become active.

We now define the tree structure formally as follows:

Definition 1 *(Tactic-Tree) - A Tactic-Tree $T = (V, E)$ is a directed acyclic graph and consists of a set V of vertices, subsequently called nodes, and a set of edges E. V is furthermore partitioned into the two disjoint subsets V_c, called conditional nodes, and V_b, called behaviour nodes, whereas $V = V_c \cup V_b$.*

Since the behaviour nodes V_b have to be leafs, one further property of the tree must be satisfied:

- $\forall (v_i, v_j) \in E : v_i \notin V_b$

Finally, the following operations are defined for processing the tactic-tree:

- $root : T \rightarrow V$ - retrieves the root-node of the tree.

[2] See http://www.dainamite.de

[3] In the RoboCup 2D Simulation League, the sensor-data arrives at the beginning of a cycle, such that this paradigm works well here

[4] This quantity is very important in Robotic Soccer, since it determines which team is in possession of the ball.

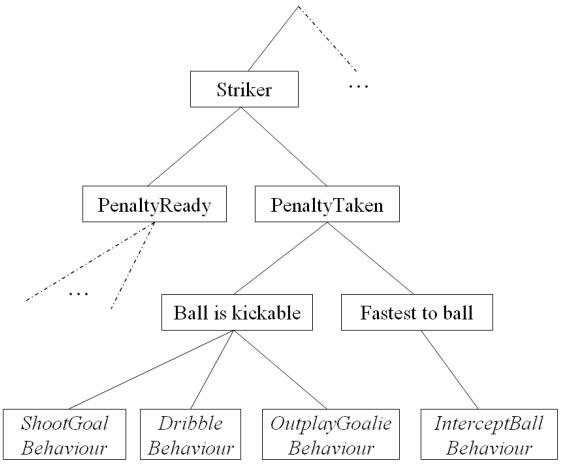


Figure 2: An excerpt from the tactic-tree.

- $children : V_c \rightarrow \{V\}$ - retrieves the all ancestors of a node via the edges of a tree.
- $condition : V \times beliefs \rightarrow \{true, false\}$ - evaluates a boolean statement of a given node that depends on the beliefs of an agent.
- $filter : V \times beliefs \rightarrow \{V_b\}$ - traverses the tree to find applicable behaviour nodes.
- $thinkStep : V_b \times beliefs \rightarrow [0,1] \times Action$ - retrieves action-utility pairs from behaviour nodes, which are considered for execution by the tactic.
- $done : V_b \rightarrow \{true, false\}$ - checks, of the given behaviour node can compute further actions.

These functions are the basic operations for finding actions in a given situation. Note that the utility value returned in $thinkStep$ is defined by the product of two related values. The first is the probability that the effect of the action takes place (success-probabilty) and the second is the utility for it's successful execution (success-benefit). The complete action selection algorithm together with the specification of some of these functions is given in the next subsection.

3.2 The Action Selection Mechanism

The action selection mechanism must fulfill two mayor requirements. First, it has to compute a valid action, and second, it has to return this action in reasonable time. Furthermore, the algorithm must allow to integrate reactive, deliberative and learned behaviours. We first present the general algorithm, and then discuss how these different approaches fit into it.

The first task of the action selection algorithm is to find all applicable behaviours. Therefore let T be a tactic-tree, v be it's root, and B the current beliefs of the agent. Finding all applicable behaviours is done with the $filter$-function, which works as presented in Listing 1:

Listing 1: Algorithm of the *filter*-function:

```
function filter(v, B)
{
  if (v ∈ V_c ∧ condition(v, B) = true)
  {
    return ⋃_{v_i ∈ children(v)} filter(v_i, B);
  }
  else if (condition(v) = true)
  {
    return {v};
  }
}
```

As can be seen, $filter$ finds all behaviours from the root-node on, whereas each node on the path to the behaviour-node must evaluate the condition to the boolean value $true$. Otherwise, the nodes and the following behaviours are ignored. Thereafter, the tactic processes the filtered behaviours iteratively, until either the current simulation cycle draws to a close, all behaviours finished all their $thinkSteps$ or an action with with sufficient utility is found. This algorithm is given in Listing 2:

Listing 2: Algorithm that computes the action to execute:

```
function computeBestAction(TacticTree T,
                           Beliefs B)
{
  // initialise
  r := root(T);
  S := filter(r, B);
  u* := 0;
  a* := null;

  // iterate over behaviours
  while (think)
  {
    select s from S;
    (a,u) := thinkStep(s, B);

    if (done(s))
    {
      remove s from S;
    }

    if (u > u*)
    {
      u* = u;
      a* = a;
    }
    if (timeup() or u* > threshold)
    {
      think = false;
    }
  }
```

While the function $computeBestAction$ is running, the *synchronization* guards concurrently, how much time is left for processing until the next action has to be sent. When time is up, it requests the currently found $a*$ from the *tactic*, and

causes the $timeup$-function to return $true$. This procedure is repeated in every cycle.

Due to the approach of iteratively requesting actions from the applicable behaviour nodes, it can be viewed as a horizontal layered architecture. And since the *synchronization* is able to stop computing, the time-constraints can be satisfied. However, the main requirement is that the computations can be divided into small units that are triggered by thinkSteps. We will explain next, how this work for reactive, learned and deliberative behaviours.

3.3 Reactive Control

Reactive behaviours are the most basic ones. We consider reactivity to be a rule-based response of the agent to a situation, whereas we allow the rules to be defined on the beliefs of the agent and not just on the sensory input. A rule can be represented by a behaviour node in a straightforward manner. The definition of the rule-condition is given by the conjunction of all the conditions of the nodes that lie on the path from the behaviour to the root node. This condition is computed in the $filter$ function. The rule-body is given by the implementation of the $thinkStep$ of the behaviour node. Such a behaviour should always return $true$ on calling the $done$-function.

A complete tactic relying only on reactive behaviours, even without considering the utilities, can be designed. The result would be a very huge tree, which only returns one behaviour for any situation the agent is in. Such a tree is difficult to maintain, and it is not easy to see, if it produces meaningful actions for every situation.

There are a few examples for reactive control already implemented. A very simple example is the goalie catching the ball if it is close to him, possibly resulting from a suboptimal attempt of an opposing player to score a goal.

3.4 Integrating Machine Learning

Machine learning can be applied to many sub-problems of simulated soccer, from predicting world-state values to learning how to behave in specific situations. We limit our scope to action selection and related problems, and especially elaborate on reinforcement learning.

A rather straightforward approach can be chosen, when the success probability of a given action should be determined. Common machine learning algorithms, such as neural networks may be trained in advance, and integrated into a behaviour-node for computing the requested value directly. If the requested value may be important for other behaviour-nodes as well, the learning algorithms should be integrated into a fact which provides the value by writing it into the belief-base.

As reinforcement learning (RL) is a very central issue in RoboCup, we spend some extra consideration on it. With RL an agent learns a policy π by trial and error. After trying an action a $\in$ A in a specific situation s $\in$ S, the agent receives a reward r and updates a value function, usually the state-action value function called Q [5]. We consider the value function to be knowledge which is stored in the belief-base. For this we implemented a fact which manages the access to the abstract reward function and provides methods to load and store the reward to preserve the learned information for the next training phase or match.

Finally, the behaviour node that implements RL is the representation of the policy π. The differences in the two cases of learning or using a trained policy are explained next.

Using a Trained Policy

If the value function provides an acceptable approximation for each state-action pair that may occur, the corresponding behaviour node is used like any other ordinary behaviour-node. If the tactic was able to find a path from the tactic's root through the conditional nodes, the $thinkStep$ is computed. Therefore the behaviour-node simply queries the world-model for the value function and uses it to determine the action that yields the best reward. The reward is returned and the action is stored for execution in case the behaviour-node has the highest utility. If finding the action with the highest reward includes a search over multiple alternatives, the search may be divided into futher $thinkSteps$.

Learning a Policy

Before using a policy effectively, the agent has to learn the policy and the value function. Since the value-function and the policy are distributed in the agent framework, the interaction is not that simple. Learning a policy with RL was integrated as follows:

1. If an RL behaviour-node is selected, an action a_i is calculated depending on the RL configuration (e.g. by following an ϵ-greedy algorithm).
2. The corresponding world-state s_i, and the action a_i are stored in the world-model.
3. The fact that manages the value-function observes the following world-state s_{i+1}, and calculates the rewards r.
4. After that, the fact queries the previous state s_i and the action a_i, and either makes an update to the value function using the common update rules [Sutton and Barto, 1998], or buffers the tuple (s_i, a_i, r) [6] for later batch learning phases.

This way it is possible to have several such learning behaviour-nodes actively learning a behaviour simultaneously (although this may lengthen the learning process if there exist dependencies between these behaviours).

3.5 Integrating Deliberation

Because of the highly dynamic and fluctuating nature of the RoboCup domain, planning more than a few cycles ahead becomes difficult. Planning is nevertheless important, even though the results have to be validated or recomputed very often. Other domains may furthermore have nicer characteristics in terms of dynamics than RoboCup.

We consider planning to be a search in the space of actions towards the fulfillment of a given goal. The requirement we impose on the planning algorithm is the ability to interrupt

[5] Depending on the choice of the RL algorithm.

[6] Also depending on the algorithm

and resume planning. This allows to divide the process into a sequence of $thinkSteps$. If after one step the process has no result, it returns a utility of 0, and will resume planning the next time the behaviour is triggered.

An examples that make use of this approach is the passing behaviour. By simulating different kicks in all directions and with reasonable speed, a good pass may be found. We divided the search by analysing and assessing each kick separately within a $thinkStep$. Behaviours that plan sequences of actions could be implemented as well, but these have to check the applicability of a computed plan during it's execution in every subsequent cycle. Finally we note, that the interval of interrupting the planning process may in many cases be configurable as well, leading to flexibilty in assigning processing time to certain behaviours.

4 Case Study: Penalty Shoot-out

For the sake of determining the fitness of the framework for integrating learned behaviours we decided to take on the penalty shoot-out, a sub-scenario of RoboCup, in which coordination and team play are negligible, hence reducing complexity.

The penalty shoot-out is a direct confrontation between a field player of the attacking party, referred to as striker, and the opposing goalie defending its goal. The striker starts with possession of the ball near the center of the play field. The scenario is not limited to a single kick like a spot kick penalty seen in conventional soccer, so the striker is allowed to move freely with the ball, kicking and sprinting as he desires. The striker tries to score a goal while the goalie of the defending side tries to prevent that. A penalty attempt is aborted, resulting in a penalty *miss* for the striker, if either the specified time (usually 200 simulation cycles) runs out, the ball leaves the play field or is caught by the goalie.

In this scenario we concentrated on the striker. We implemented and aggregated some skills we felt to be helpful to result in a competent tactic for the striker. These skills include the following:

- Scoring: Determining the kick that has the highest probability of success.
- Dribbling: Executing dribblings that don't forfeit possession of the ball, while swiftly approaching the goal.
- Bypass goalie: Executing kicks to bypass a well positioned goal keeper and reach potential scoring situations.
- Intercept ball: Finding and reaching good interception points to quickly regain control over the ball.

Wherein scoring has been learned off-line from a beforehand collected training set, using neural networks to predict the outcome of potential scoring attempts. Dribbling and bypassing the goalie use reinforcement learning to adapt during play. The intercept ball skill uses domain dependent planning to determine the series of actions needed to quickly close in on the ball.

The single approaches are now explained in detail.

In our scenario we defined dribbling as closing in on the goal (and hence goalie) swiftly while not risking to loose ball possession. To archive this behaviour the agent received a reward relative to the change in distance to the goal with the addition of a huge penalty if ball possession was lost as a direct consequence of selecting an action returned for dribbling.

The second learning skill, bypassing the goalie, uses reinforcement learning, too. The target was to find actions that may lead to a situation in which the probability of scoring, as determined by the scoring skill, is higher than the one in the current situation. For this sake the striker was placed into a training scenario, in which he was randomly placed on the play field's half with the target goal, while being in control of the ball (able manipulate it). The opposing goalie was also placed randomly within the vicinity of the striker, with a higher probability for positions somewhere between the goal and the ball.

Our approach to scoring was inspired by [Kok *et al.*, 2002]. We applied the idea of separating the two probabilities for the ball to arrive in the goal, and to be intercepted by the goalie. With the difference that we viewed both cases as a classification problem using neural networks to learn from a recorded data-set and to interpret the confidence in the classification results as probabilities. Making the assumption these two events are independent, the two obtained probabilities could just be multiplied to receive the joint probability. So in the case of scoring we have two separate neural networks, one calculating the probability of the ball entering the goal if the goalie would not intervene, the other calculating the probability the goalie is able to catch the ball or otherwise hinder it from crossing the goal line. These two networks are both loaded and used in a single *fact* to find the kick action with the best joint probability of resulting in a goal. This *fact* continuously calculates this probability when new input arrives. The calculation is done by generating a series of possible kick actions and comparing their probabilities returned by the two neural networks. The action with the highest probability is then stored together with its corresponding rating, the calculated probability, in the agent's world model. Both, the action and the rating are retrieved and exposed by the tactic's behaviour-node for scoring.

The next step in aggregating the single skills to an overall tactic was to create a policy that coordinates them. We started off combining scoring, ball interception and dribbling, later adding the bypassing of the goalie.

Since the intercept ball skill is only active if we are not in possession of the ball, it does not compete for control with the other skills. If the agent is in possession of the ball, however, there are three alternatives to choose from. We implemented our approach based on the fact that scoring should always return the probability of a goal for the current situation as its utility. So the utility of dribbling works as a threshold for executing a scoring attempt.

Later the bypassing goalie behaviour-node was added, synchronizing its own utility with scoring, leading to a higher benefit for a sidekick if it is expected the agent can further improve the scoring rating with it. If this is not the case a scoring attempt from the current situation is generally rated higher than the bypass.

Since the penalty scenario has a time limit and the implemented behaviours could possibly lead to a deadlock, we added some logic for scoring to take over control in certain situations. We execute a scoring attempt if the remaining time is running out and the ball would cross the goal line just a few cycles before the penalty will be aborted due to time-out, when considering a direct kick in direction of the goal.

5 Results

Through testing we found a threshold of 0.4 for the scoring rating to be a good trade off between taking risky shots and coming too close to the goalie to be convenient. This simple tactic alone was enough to reliably score against some of the weaker goalies in the penalty scenario. Here the percentage of number of goals in the number of taken penalty-attempts was greater than 80%. Against more sophisticated goalies however the percentage of successful attempts dropped down to between only 17% and 19%.

In the learning setup the learning curve of the bypass skill using reinforcement learning method displayed a fast learning increase in the beginning. In the following epochs, however, a clear, additional, stable increase could not be observed. Figure 3 shows this increase in the performance of the overall tactic. Note that these values are not from the actual penalty scenario, but were recorded from the very same scenario the skill was trained in, described above.

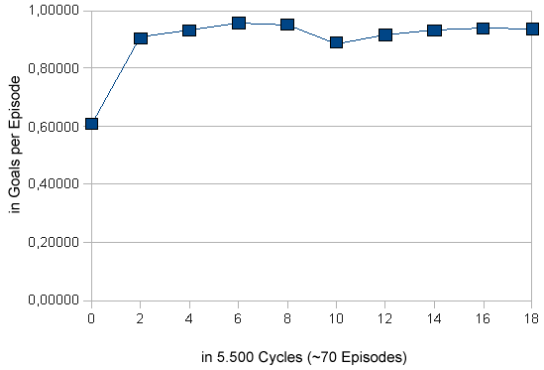


Figure 3: Average goals per episode of the complete tactic in the training scenario of bypass goalie.

With the addition of the bypass skill, we were able to improve the overall performance against stronger goalies, for this tactic, by nearly 70%, after an extended period of learning, in comparison to the same tactic missing this behaviour-node.

6 Conclusion and Future Work

We were able to incorporate on-line and off-line learning methods without much effort into the agent's tactic, without having to deviate from the existing concepts and paradigms used in the DAInamite framework. Even if the learned results are far from being optimal, we were able to demonstrate that there is no difficulty in adding even more complex on-line learning or planning behaviours to the agent.

The next step is to show the possibility to integrate higher level decisions by means of learning and/or planning. This way coordination of a selected set of behaviour-nodes may be learned instead of being coded, and tuned by hand. Even though this was not done up to now it surely does not seem to be an overly complex task if the number of behaviors to select from is small, and the number of decisions is limited, like for example in the penalty-scenario. So we may be able to demonstrate this step, too, in the near future.

References

[Berger, 2006] Ralf Berger. Die doppelpass-architektur verhaltenssteuerung autonomer agenten in dynamischen umgebungen. Diploma thesis, Humboldt-Universitt zu Berlin, Institut fr Informatik, 2006.

[de Boer and Kok, 2002] Remco de Boer and Jelle Kok. The incremental development of a synthetic multi-agent system: The uva trilearn 2001 robotic soccer simulation team. Master's thesis, University of Amsterdam, The Netherlands, 2002.

[Kitano *et al.*, 1998] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawai, and Hitoshi Matsubara. Robocup: A challenge problem for ai and robotics. pages 1–19. 1998.

[Kok *et al.*, 2002] Jelle Kok, Remco Boer, and Nikos Vlassis. Towards an optimal scoring policy for simulated soccer agents. In *RoboCup 2002: Robot Soccer World Cup VI*, pages 292–299. SpringerVerlag, 2002.

[Noda *et al.*, 1998] Itsuki Noda, C Fl Itsuki Noda, Hitoshi Matsubara, Hitoshi Matsubara, Kazuo Hiraki, Kazuo Hiraki, Ian Frank, and Ian Frank. Soccer Server: A Tool for Research on Multiagent Systems. *Applied Artificial Intelligence*, 12:233–250, 1998.

[Riedmiller and Withopf, 2005] M. Riedmiller and D. Withopf. Effective methods for reinforcement learning in large multi-agent domains. *Information Technology Journal.*, 241-249:47(5), 2005.

[Russell and Norvig, 2003] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[Stone, 2000] Peter Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.

[Sun and Wu, 2006] Yong Sun and Bo Wu. Agent hybrid architecture and its decision processes. *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics*, 2006.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[Tambe, 1997] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Hierarchical Activation Spreading: A design pattern for action selection

Michael Müller
Lehrstuhl Intelligente Systeme
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen
michael_mueller@uni-due.de

Abstract

Action selection for a robot means to activate behaviors in a timely manner so that both, short-term and long-term goals, are met, e.g. avoiding an obstacle and completing a complex task. Maes has described an algorithm that spreads activation energy in a network to pursue a set of goals, uses emergence for creating a plan, and selects an action suitable for the current situation. Many extensions to this algorithm have been proposed to achieve a higher degree of flexibility and applicability. In this paper an algorithm for action selection is presented that is based on Maes' algorithm and its extensions. Then a design pattern is described as a guideline for developers to build an action selection system based on the proposed algorithm.

Keywords
Decision Making, Behavior-based Robotics and Emergent Control, Architectures and Architectural Patterns

1 Introduction

For many years robots have been successfully used in industrial assembling. Today applications are developed to make use of robots in other domains like automotive, transport, healthcare, and entertainment. Here both, the complexity of the environment and the necessary autonomy of the robots are considerably higher. Preprogrammed sequences of movements can no longer be used. So a mechanism has to be established that allows a robot to react to its environment, to create a plan and to act situation-dependent.

In [Pirjanian, 1998] different approaches are compared. Some of them mostly react to the current situation without pursuing a long-term plan. Others need a huge amount of information about the environment to build a perfect plan for the actions of the robot. Between these two extremes there are many hybrid approaches.

Maes has formulated a set of properties for action selection in an autonomous robot [Maes, 1990]: In a complex environment it is nearly impossible to collect all information in advance. This makes it impossible to create a complete set of rules for all possible situations. So action selection should be able to do its work even in situations that were not known during the development. It should also be possible to extend an existing action selection system with additional knowledge without the necessity of restarting the whole analysis process from the beginning. Action selection should make use of goals. Goals describe the context; without them each situation has to carry full context information. Goals may change over time. The agent has to be able to cope with conflicts between several goals. Finally, goals enable the agent to learn by giving the feedback which sequence of actions pursues a goal and which does not.

The action selection algorithm proposed by Maes has the formerly described properties and combines reaction and planning [Maes, 1990]. A self-organizing mechanism is used to create a sequence of actions by emergence. But there are also some drawbacks. In the next section the algorithm is described in more detail and solutions to avoid its drawbacks are presented.

2 Emergent action selection in activation spreading networks

First Maes' algorithm is described, followed by a description of a problem and two extensions concerning the activation spreading mechanism. Then two more extensions concerning actions and the network structure are described.

2.1 Maes' action selection algorithm

Maes' action selection algorithm [Maes, 1990] is based on three concepts: monitors, actions, and goals[1]. A *monitor* inspects the conditions found in the environment and calculates the truth value of a given proposition. Although in the original algorithm a strict division into true and false propositions is used, the algorithm can be adapted to use fuzzy results. An *action* becomes executable, if a set of monitors — its precondition — calculates a true value. If an action is executed, then it will change the values of some monitors. Some of them will become true and are part of the action's add list. Others will become false and are part of the action's delete list. Fi-

[1]These three concept names are used throughout this text, because they are more general resp. shorter than the names used in the referenced text. Maes denotes a monitor as *proposition* and an action as *competence module*.

nally, a *goal* is achieved, if a defined set of monitors becomes true.

By designing the algorithm the following features were considered:

- A selected action should directly achieve a goal or at least make the preconditions of other actions true that are closer to a goal.
- Action selection should be situation-dependent and take opportunities.
- If a decision was taken and for the execution of actions some effort was made, the algorithm should prefer actions that continue on working on this decision.
- Forecast helps to avoid selecting actions that lead to a dead end.
- If parts of the system can not be used (shortage of resources, system damage), the remaining system should keep on working robustly.

The algorithm builds a network based on the actions. Each action stores its activation level. Two actions are connected, if the add list (or delete list) of the first one and the precondition of the second one has some monitors in common. Then activation energy is spread through this network, fed in by two sources and distributed over three types of connections:

- If an action has monitors of a goal in its add list, it receives activation energy from this goal.
- If an action has a true monitor in its precondition, it also receives activation energy.
- If an action is not executable, but has received some activation energy, then it distributes it to other actions that make its precondition become true so that it becomes executable.
- If an action is executed, it distributes activation energy to other actions which are connected via its add list. Hereby starting work on a path in the network will make continuing work on this path more likely.
- An action receives negative activation energy through connections via its delete list. Goals and other actions try to inhibit negative effects on their monitors.

Because more and more activation energy is fed in the network, the activation level of all actions needs to be normalized to avoid divergence. To select an action, an initial threshold is set and lowered over time. As long as no action has an activation level higher than the threshold, the network continues distributing activation energy. If the threshold gets lower than the activation level of an action, this action is selected and executed. The activation level of the selected action is reset to zero, the threshold is reset to its initial value, and the process starts over again.

Actions between goals and true monitors receive high feed in of activation energy. Competing paths in the network are formed by emergence and actions along these paths are selected.

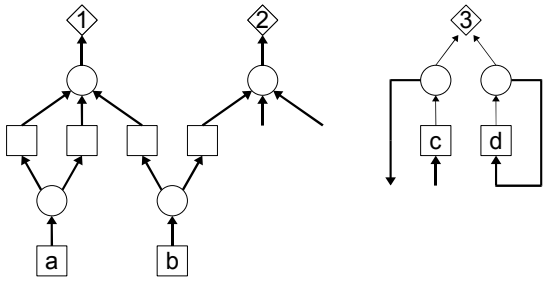


Figure 1: Left: action a indirectly contributes to goal 1 using two paths; action b indirectly contributes to goal 1 and 2. Right: action c and d both contribute to goal 3, but action d is part of a cycle.

2.2 Indirect contribution to goals

Tyrrell has made several experiments to test Maes' algorithm. He found out that in some cases the algorithm distributes the energy in the network in an undesirable manner [Tyrrell, 1994].

On the left side of figure 1 one problem is illustrated. Activation energy is fed in by the two goals 1 and 2, depicted as rhombs. Both are connected to three actions (squares) via a monitor (circle). Although each goal is connected to three actions, only four of them are shown. Activation energy flows over additional monitors to actions a and b

Branching of activation energy flow underneath goal 1 and goal 2 does not differ in any way. In both cases energy comes from one goal and is distributed to three actions. Also merging of activation energy flow above actions a and b is exactly the same. In both cases energy comes from two actions and is distributed to one action. So the locally operating algorithm can not make a difference in these cases — but there is one. Action a contributes to one goal, namely goal 1. In contrast, action b contributes to two goals, goal 1 and goal 2. Therefore action b should receive a higher amount of activation energy than action a, but using Maes' basic algorithm it will not.

Another problem is illustrated on the right side of figure 1. Only goal 3 feeds in activation energy in the network. Activation energy flows over a branch of two monitors to actions c and d. Because there is a cycle in the network graph, activation energy from action d is distributed to action d again.

Branching of activation energy flow underneath goal 3 is the same for both paths. Both paths lead to a monitor that is connected to an action and also receives activation energy from somewhere else. Also at actions c and d the distribution of activation energy is equal. Both actions receive activation energy from one source and distribute it to their predecessor. Because of the cycle in the network structure and the normalization process the activation level in action d will be higher than that in action c — but both actions are working towards the same goal and should have the same activation level. So in this case the locally operating algorithm makes a difference where should be none.

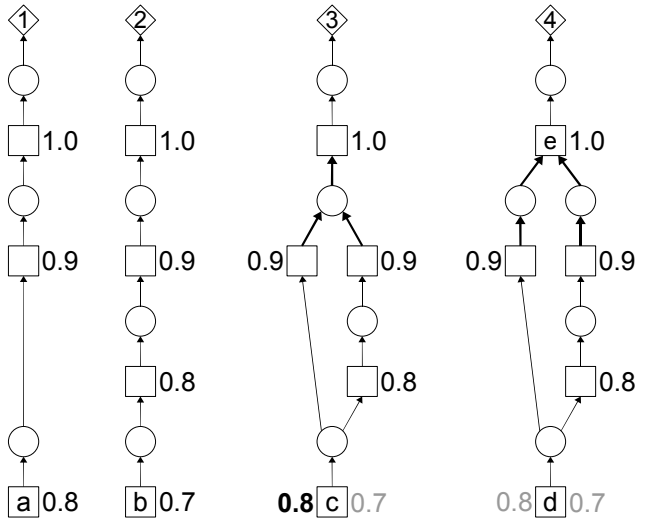


Figure 2: Activation spreading from goals 1 to 4 to actions a to d. Whereas action c only contributes to goal 3, d contributes to two sub-goals generated at the junction at action e. It is not clear which activation value should be assigned to action d.

2.3 Splitting activation energy goal-wise

Dorer has stated that the described problems can be avoided by changing parts of Maes' algorithm [Dorer, 1999]. The problems arise because the locally working algorithm does not have the information to which goal the current action contributes. This is the reason why equivalent paths to the same goal are counted multiple times and why an action can increase its own activation level, if it is part of a cycle.

Dorer suggests to store the activation energy obtained from one goal separately from the others. If an action obtains activation energy from one goal through more than one path, then the maximum is calculated. So only the shortest paths to each goal is considered. To calculate the activation level for an action for all goals, the energies are summarized. But this value is only used for action selection. For distribution activation energy is kept separately.

The proposed change of the algorithm increases the necessary memory for each action node to store the activation energies goal-wise. Still the algorithm can work locally.

2.4 Indirect contribution to sub-goals

Working with the extended algorithm proposed by Dorer a new problem becomes apparent. Figure 2 shows four networks with their goals 1 to 4 (depicted as rhombs) and five labeled actions a to e (depicted as squares). The activation energy fed in by a goal is 1.0 and is decreased by 0.1 from action to action. The path between goal 1 and action a contains only two intermediate actions; therefore action a receives a high value of 0.8. The path between goal 2 and action b contains three intermediate actions; action b receives a lower value of 0.7. To reach goal 3, two parallel paths with different length exist. At action c Dorer's algorithm chooses the higher value of 0.8 standing for the shorter path. The new problem occurs in the fourth graph. To reach goal 4, action e has to be executed. Action e has two monitors as its precondition. So actions on two paths have to be executed before to make the precondition true. The algorithm proposed by Dorer will assign the higher activation level 0.8 to action d. But this is not the correct value. The correct value should be smaller than that of action b, because the path is longer, even though it is split into two paths.

The basic idea to solve this problem is to introduce sub-goals. Because action e has two monitors as its precondition, the network at this node no longer has one goal 4, but instead two new sub-goals: to make the first precondition monitor of action e true **and** to make the second precondition monitor of action e true. It is essential to distinguish between this situation and the situation in the third graph with goal 3. The two paths leading to goal 3 are alternatives (one better than the other), the two paths leading to goal 4 both have to be passed through. In action d the activation energy of both sub-goals is received and can be used to calculate an appropriate activation level. More details about this are given in the section about the revised action selection algorithm.

2.5 Continuous actions

So far actions are considered to solve a complete subtask in a given time span once they are executed. In simulation experiments often a grid world is used, where each action takes one time step and either moves the simulated robot to another cell or carries out a whole subtask. But in real world, actions of a robot can not be discretized so easily. The time span for completion of a subtask will not be the same, even in (almost) the same situation. In addition it is not sure that an action is able to complete a subtask. Both has to be taken into account by an action selection algorithm.

Heumüller et al. describe a small change to Maes' algorithm that allows to use it with actions based on visual servoing [Heumüller *et al.*, 2009]. Visual servoing means that the robot tries to detect an object with a camera and then moves in a way that the projection of the object position in the image moves to a defined final position. This process may end fast, if the projection of the object is already at the defined position. It may take some time, if there is a long distance between the object's projection and the final position. Finally it is also possible that the camera does not see the object and therefore movement can not be calculated.

So the activation level of a selected visual servoing action may not be reset to zero, because it should also be selected in the next iteration if no other action has received a higher activation level till then.

2.6 Hierarchical organization

The method described by Maes does not provide for structure in the activation network. It is argued, that each kind of structure works against the emergent action selection realized with this algorithm. By contrast, many research groups use structured action selection systems, so-called cognitive architectures. Often these cognitive architectures imitate structures found in the human brain by biologists and psychologists. Both, the emergent properties of Maes' algorithm and the possibility to realize cognitive architectures, are desirable.

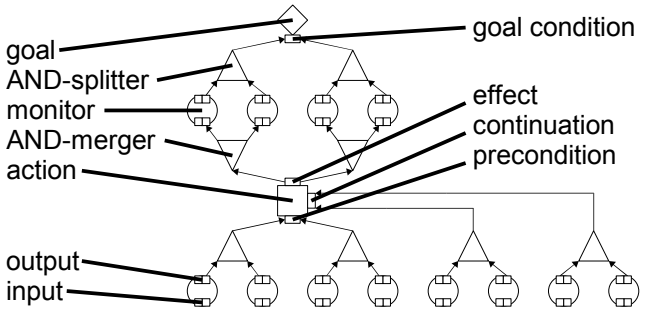


Figure 3: Small network annotated with node types and concepts used in the revised algorithm.

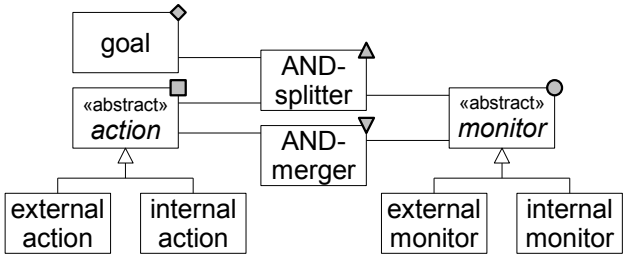


Figure 4: UML class diagram showing activation network node types. The gray icons depict the shape of nodes used in figure 3.

Decugis and Ferber have proposed to use a whole activation network as single action in a higher level activation network [Decugis and Ferber, 1998]. As long as the higher level activation network selects this action, the lower level activation network is active and selects one of its actions. Hereby it is possible to partition large, complex networks into smaller parts.

A more general approach that is able to emulate the approach of Decugis and Ferber is to combine activation networks with finite state machines. To accomplish this, a new kind of monitor has to be introduced that checks, which goals are achieved. Combined with a new kind of action that is able to switch goals on and off respectively change their importance it is possible to model state transitions. More details about this are given in the next section.

3 Revised action selection algorithm

First, an overview of the node types in the network and the concepts used in the algorithm is given. Then the steps of the algorithm are described in detail. Finally, the algorithm is demonstrated on examples based on the problematic networks shown in figures 1 and 2.

3.1 Overview

Action selection is realized using networks with five node types and eight types of connections. Hereby the algorithm is able to precisely distinguish different concepts and to spread activation energy accordingly.

Figure 3 shows a small network with one goal, one action, and a number of monitors with their connections. Each goal has a goal condition. The more a goal condition is true, the more the goal is achieved. Each goal condition is connected to a number of AND-splitters. The value of the goal condition is calculated as a fuzzy disjunction of the values propagated by the AND-splitters. Each AND-splitter is connected to a number of monitors. Its value is calculated as a fuzzy conjunction of the values propagated by the monitors. Each monitor can be connected to AND-splitters through one of two outputs. The first output is called the normal output and propagates a fuzzy value about the property the monitor is keeping track of being in a defined range. The second output is called the inverted output and propagates the negated fuzzy value.

An action becomes the more executable, the more its precondition is calculated to be true. An action will continue its execution, the more its so called continuation is calculated to be true. The more successful the operation of an action was accomplished, the more its so called effect will be calculated as true. Whereas precondition and continuation are calculated the same way as a goal condition is, the connection of an action's effect to monitors follows a mirrored scheme: The effect of an action is connected to a number of AND-mergers, the AND-mergers are connected to the normal and inverted inputs of a number of monitors.

The UML class diagram in figure 4 shows the node types used in activation networks for the revised algorithm. Besides the three main node types (goals, actions, monitors), specialized actions and monitors as well as two nodes for branching are introduced. An external monitor keeps track of some property of the environment, checks, whether this property is in a given range, and reports the result as a value in the range of $[0, 1]$. An internal monitor keeps track of the achievement of a set of goals and also calculates a corresponding value in the range of $[0, 1]$. There exist two kinds of actions. An external action measures a property of the environment and calculates the necessary control commands for the robot hardware so that the property changes towards a defined range. An internal action changes the importance of goals to influence the action selection process.

3.2 Details

The algorithm is organized into four steps.

1. Activation energy is fed in by goals and propagates backwards through the network.
2. Actions, that **were executed**, propagate their accumulated activation energy forward through the network.
3. The activation level for each action is calculated.
4. In each executing action activation energy is accumulated.

After execution of all four steps the activation level of all actions is up-to-date. All actions whose activation level is higher than a threshold θ_a may be executed concurrently. The activation level is intended to support the fusion of conflicting commands.

Propagation of activation energy from goals: An importance is assigned to each goal. This importance is the initial value for activation energy propagated from this goal. Activation energy is propagated independently for each goal and stored separately in each node. In the beginning of this step stored activation energy for all goals in all nodes is removed. If during propagation a node already contains a higher value for a given goal, propagation for this goal stops at this node to avoid self-excitation of actions in cycles. If a node contains a smaller value, then propagation continues, overwriting the smaller value with the higher one, so that activation energy always describes the shortest way to a given goal.

At the disjunctive branch at the goal condition to each AND-splitter activation energy is simply propagated without changes, because each path will have the same effect on the goal condition.

At the conjunctive path of an AND-splitter the propagation for this goal stops. For each path a new sub-goal is created with a corresponding fraction of the activation energy of the original goal, because all paths are necessary to achieve the goal. For each sub-goal the algorithm stores the initiating goal. This information is used in subsequent AND-splitters to avoid recursive creation of sub-goals in cycles.

At a monitor the activation energy is simply passed through from normal output to normal inputs and from inverted outputs to inverted inputs. If activation energy is positive, it is also negated and passed crosswise from normal outputs to inverted inputs and from inverted outputs to normal inputs. By means of this goals and actions can inhibit predecessors that work against their goal- or preconditions. If one action counteracts another action that in turn counteracts a goal, the first action must not necessarily support the goal. Therefore negative activation energy is **not** passed crosswise.

At AND-mergers activation energy is simply passed through.

For actions an assumption is made: if the precondition is met, then the continuation is also met. Later the precondition may become false while the continuation keeps being true. Therefore only paths via the precondition has to be pursued in an action. The activation energy is not simply passed though, but instead changed according to the former activation level of this action. If the former activation level is low, meaning that an action was not executed or that the commands created by this action were not considered in command fusion, preceding actions should receive activation energy to get a chance to make missing preconditions true. If the former activation level is high, this action was already executed and the work of preceding actions is no longer necessary. To achieve this, the passed activation energy is multiplied with the inverted former activation level of an action and a decay factor f_b to also consider the increasing distance from the goal.

Propagation of activation energy from formerly executing actions: Actions that were executed for a while accumulate activation energy (see below) separately from the energy received from goals. When execution of an action stops, the accumulated activation energy is spread to its successors. This makes sure, that actions on paths in the network are preferred, that originates in actions that already have spent time and effort in achieving a goal.

The propagation process is similar to that for activation energy from goals. But now it starts at actions and moves forwards to goals. In the beginning of this step stored activation energy for all actions in all nodes is removed. Again, activation energy is split according to the originating action and stored separately in each subsequent node. In AND-mergers activation energy is not split, in opposite to activation energy originated by goals in AND-splitters. This is because goals represent a state to be achieved in the future and activation energy from formerly executed actions represents a state in the past. Whereas in calculation and distribution of activation energy from goals optional and parallel paths has to be distinguished, for activation energy distributed from actions this is not necessary, because always actions that are closest to goals will be preferred to actions on paths leading to this closest action.

If during propagation a node already contains a higher value for a given action, propagation for this action stops at this node to avoid self-excitation of actions in cycles. If a node contains a smaller value, then propagation continues, overwriting the smaller value with the higher one, so that activation energy always describes the shortest way to a given action. In monitors values are passed through only parallel; no negative value is created. So actions that continue work on a path to a goal gets some advantage over other action. But actions that are not part of this path do not get a disadvantage.

Updating of activation level: To calculate the activation level of an action, the value of its precondition and of its continuation has to be considered, as well as the energy received from goals and predecessors.

The current values of all monitors belonging to the AND-splitters of a precondition have to be calculated. For each AND-splitter a result is calculated by multiplication of the monitor values. So an AND-splitter will only have a high result, if all monitor values are also high. The results of the AND-splitters have to be combined to form the value of the precondition. Using the function $\oplus(a,b) = 1 - [(1-a) \cdot (1-b)]$ with $max(a,b) \leq \oplus(a,b) \leq 1$ the number of AND-splitters with a high value can be considered. In the same way the continuation value can be calculated.

It is possible that an action receives more than one fraction of activation energy from the same goal, possibly via subgoals. Some of these values are positive, denoting a path that leads to this goal. Some of them can be negative, denoting a path of actions that works against this goal. Using the $\oplus$-function a form of sum for all positive values can be calculated as well as a sum for all negative values (a negative value n has to be inverted first: $\bar{n} = 1 + n$). The $\oplus$-function has the special property that $a, b \leq \oplus(a,b) \leq 1$, so adding two values will yield a higher value, but the sum will never exceed 1. Then for one goal a result can be determined by subtracting the sum of negative values from the sum of positive values. To get a result for all goals that distribute energy to this action, the results for each single goal are summed up and divided by the total number of goals; goals that do not dis-

tribute energy to this action are also counted. So in the whole network the denominator is the same and calculated results are comparable independently from the number of goals that are connected to a given node.

Also from predecessors more than one fraction of activation energy could have been received. Using the $\oplus$-function a form of sum can be calculated for these.

The activation level of an action is finally determined considering three conditions:

- The precondition, the continuation, or both has to be true.
- The closer the effect of an action is to one or more goals, the more important this action becomes.
- The more time and effort was spent by predecessors, the more important this action becomes.

So the maximum of precondition and continuation is calculated and multiplied with the maximum of the sums of activation energy received by goals and predecessors. To remove case differentiation from formulas, the maximum function may be replaced by $\sqcup(a,b) = \frac{a^n+b^n}{0.1^{2n}+a^{n-1}+b^{n-1}} \approx max(a,b)$ with $n \geq 2$.

Accumulating activation energy: Two values are calculated to handle accumulated activation energy: an accumulator storing already accumulated activation energy during action execution and a forwarder storing activation energy that is to be distributed through the network after the execution of an action has ended. The activation level of an action is used to weight between accumulating and forwarding.

Let a be the current activation level of an action and m its accumulator. Then its forwarder value is calculated as $f_c \cdot (1-a) \cdot m$. The more the action becomes inactive, the more accumulated activation energy is supplied for distribution. Distribution is slowed down by a decay factor f_c.

The accumulator m is the sum of two parts. First, it contains the remaining part that was not distributed by the forwarder: $1 - (f_c \cdot (1-a) \cdot m)$. Second, it contains the difference between its current value and the current activation level, if the latter one is greater: $f_d \cdot (\sqcup(a,m) - m)$. Again, adjustment to the current activation level is slowed down by a factor f_d. So m will rise if an action starts execution and it will decrease, if an action ends execution and activation energy is distributed via the forwarder value. In both cases changes of m will be delayed.

3.3 Tests

In figure 5 the results of three tests are shown. Activation energy is distributed through three networks based on the problematic cases from figure 1 and 2. The tests yield that the activation energy received by actions now complies with the length of paths to goals.

In the upper part a network with two goals 1 and 2 is shown. Action a receives activation energy via two paths only from goal 1. Because there is one intermediary action between action a and goal 1 and there are two goals in the network, its received activation energy is $\frac{1.0-1\cdot0.1}{2} = 0.45$. Action b receives activation energy via two paths from both goals

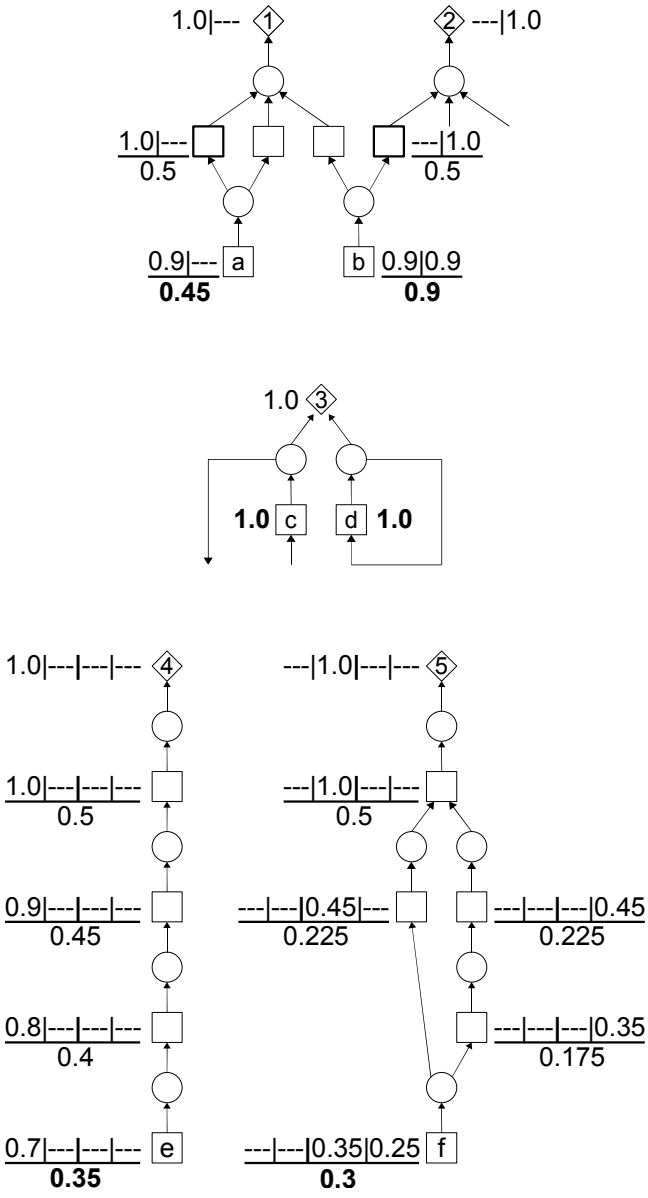


Figure 5: Distribution of activation energy originated by goals. At the top: actions contributing to one and to two goals; in the middle: actions inside and outside a cycle; below: introduction of sub-goals. Activation energy for each goal is shown separated by a vertical bar ('|'). At actions the calculated sum of activation energy is also shown below a horizontal bar.

1 and 2. Because there are also intermediary actions on both paths, the received activation energy is $\frac{1.0-1\cdot0.1}{2}+\frac{1.0-1\cdot0.1}{2} = 0.9$. So action b is preferred to action a because it works towards two goals, whereas action a only works towards one goal.

In the middle a network with a cycle is shown. Although action d is connected to goal 3 directly and indirectly via the cycle, the same amount of activation energy is distributed to it as to action c. So self-excitation is prevented.

The lower part shows a network with two goals 4 and 5, two unnamed sub-goals, and two action e and f. Distribution of activation energy from goal 4 to action e is straightforward: because of three intermediary actions activation energy is $\frac{1.0-3\cdot0.1}{2} = 0.35$. Distribution of activation energy from goal 5 directly stops at the first action and is split into two sub-goals, each with the halved and decayed value. Then distribution of activation energy continues for each path separately. In action f both paths are merged and activation energy is calculated as 0.3. So even if each single path from action f to goal 5 is not longer than the path from action e to goal 4, action f receives a smaller amount of activation energy than action e, because the paths are not optional.

4 Pattern description: Hierarchical Activation Spreading

In this section a guideline is given to use the proposed algorithm to solve an action selection problem. For this purpose an established technique from software engineering is applied:

When constructing complex software systems developers are often confronted with problems that can be solved with approved solutions. These solutions are called patterns [Gamma *et al.*, 1995]. Related patterns can be organized in a network, a so called pattern language.

Patterns have been described for several domains like concurrency, distributed systems, real-time systems, and robotics [Brugali, 2007]. In the following a pattern for action selection is proposed. It is intended to contribute to a pattern language for robotic systems. For description a shortened form of the schema proposed by Buschmann et al. is used [Buschmann *et al.*, 1996]:

Context: An action selection system for a robot should be set up that allows to combine pre-planned action sequences with actions selected by emergence based on the current situation.

Problem description: Four issues (*forces*) influencing the solution have to be considered:

In a robotic system in the majority of cases pre-planned action sequences can be provided by developers. Even if there exist several methods to learn those sequences, providing them to the system will dramatically speed up development.

If an action sequence ends, the action selection mechanism has to take a decision on how to continue. Both, achievement of goals and formerly spent effort have to be considered.

Also the end of one action sequence and the beginning of the next may overlap. The action selection mechanism should smoothly crossfade between both sequences.

Often there are background tasks that have to be activated fast in tricky situations. For example, obstacle avoidance has to be taken care for only if there is an obstacle. But often the finding of obstacles can not be considered in advance. Therefore obstacle avoidance has to be part of all tasks.

Solution: Use the proposed algorithm. Pre-planned action sequences can be described by paths towards a goal and by considering different states by changing the importance of goals. To decide between alternative paths, distance to goals as well as effort spent by formerly executing actions is taken into account. The algorithm allows to execute more than one action at the same time and provides an activation level per action, so that output of concurrently executing actions can be superimposed. Also several system states may be active at the same time to run a main task concurrently to several background tasks.

Implementation: To construct an action selection system based on the described algorithm carry out the following steps:

Step 1: Decompose the robot's task into situations. Here a situation is described by the fact that the state of the robot in this situation differs significantly from the state of the robot in another situation. Two possible situations could be the robot being inside a room or outside. Another example is the robot being far away from an important object or near to it.

Step 2: Decompose each situation into different aspects. An aspect describes a subtask the robot has to pursue concurrently to other subtasks. For example, one aspect of a situation could be to make the robot head for an object. Another aspect in the same situation would be to make the robot avoid obstacles on its way to the object.

Step 3: For each aspect define a goal that supplies the network with activation energy. The goal needs to be connected to a set of monitors that describe the goal condition. Instead of creating one monitor that checks the full goal condition try to create smaller monitors that checks parts of the goal condition and combine them with AND-splitters. Doing so will improve re-use.

Step 4: For each aspect create at least one sequence of actions that forms a path towards the goal. Each action needs monitors to check its precondition and continuation. As in step 3, try to create simple monitors and combine them using AND-splitters.

An action may compute its commands with the help of its continuation monitors. For this purpose the monitors not only have to provide a truth value, but also a measure for the distance between the current robot state and the desired one.

Based on these measures servoing commands can be calculated.

Step 5: For each situation decide about the importance of its constituting aspects. Does all aspects have the same importance? Or is one more important than others? The achievement of which goal signals that the situation has changed?

Step 6: Use the results of step 5 to create internal monitors and internal actions that take charge of setting importance of goals adequate for the current situation. Through this, aspects needed for a situation respectively their goals will be activated, whereas other aspects that are not needed at the moment will be deactivated.

Effects: The proposed pattern leads to the following advances:

- Developers are free to decide, which parts of the action selection system should be provided in advance and which parts should constitute by emergence during operation.
- Action selection is modular, because the mechanism is based on nodes organized in a network. Several basis networks may be combined into a complex network.

The proposed pattern entails the following disadvantages:

- The more emergent properties of the algorithm are used, the more difficult becomes the prediction of action selection.
- Also concurrent execution of actions and fusion of conflicting commands complicate the prediction of action selection.

5 Conclusion

An algorithm has been described that allows to combine emergent action selection with hierarchical finite state machines. Tests have shown that the proposed changes of the original activation spreading mechanism lead to the desired results. The utilization of the algorithm was outlined in a description of a pattern.

Although parts of the method use emergence to cope with new and unknown situations, still a lot of information has to be provided by the developer. Weng et al. have described the necessity for a complex robotic system to autonomously develop its mental structure [Weng *et al.*, 2001]. This is also referenced as *epigenetic robotics*, because changes of the robotic software system take place during operation, analog to nature, where learning takes place parallel to ontogeny based on genes. The proposed algorithm supports this development of mental structures by means of changes to the structure of the activation network. Batory et al. describe a framework that allows to extend existing systems step-wise [Batory *et al.*, 2004] using mechanisms known as *feature-oriented programming*.

Future work aims at utilization of feature-oriented programming for the proposed action selection algorithm to achieve autonomous development of mental structures epigenetically.

References

[Batory *et al.*, 2004] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, 2004.

[Brugali, 2007] Davide Brugali. Stable analysis patterns for robot mobility. In Davide Brugali, editor, *Software Engineering for Experimental Robotics*, 30, pages 9–30. Springer, Berlin, Germany, 2007.

[Buschmann *et al.*, 1996] Frank Buschmann, Regine Meunier, Hans Rohnert, and Peter Sommerlad. *Pattern-Oriented Software Architecture: A System of Patterns*, volume 1. Wiley & Sons, Hoboken, NJ, USA, 1996.

[Decugis and Ferber, 1998] Vincent Decugis and Jacques Ferber. An extension of Maes' action selection mechanism for animats. In *Proceedings of the fifth international conference on simulation of adaptive behavior on From animals to animats*, pages 153–158, Zürich, Switzerland, 17.-21. August 1998 1998.

[Dorer, 1999] Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In *Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1233–1238, Stockholm, Sweden, 31. Juli bis 6. August 1999 1999.

[Gamma *et al.*, 1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reuseable Object Oriented Software*. Addison Wesley, München, Germany, 1995.

[Heumüller *et al.*, 2009] Marvin Heumüller, Jens Hoefinghoff, Michael Korn, and Daniel Pinske. Praxisprojekt 2008 - Gruppe Aktivierungsnetzwerk. Internal report, Lehrstuhl Intelligente Systeme, Universität Duisburg-Essen, Duisburg, Germany, 2009.

[Maes, 1990] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.

[Pirjanian, 1998] Paolo Pirjanian. *Multiple Objective Action Selection and Behavior Fusion using Voting*. PhD thesis, Department of Medical Informatics and Image Analysis, Aalborg University, Aalborg, Denmark, 1998.

[Tyrrell, 1994] Toby Tyrrell. An evaluation of Mae's bottom-up mechanism for behavior selection. *Adaptive Behavior*, 2(4):307–348, 1994.

[Weng *et al.*, 2001] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Artificial intelligence. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2001.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Coverage Under Dead Reckoning Errors: A Hybrid Approach

Victor Shafran, Gal A. Kaminka, Sarit Kraus
Bar Ilan University
{shafrav,galk,sarit}@cs.biu.ac.il
Alcherio Martinoli
Swiss Federal Institute of Technology Lausanne
alcherio.martinoli@epfl.ch

Abstract

Coverage is a task, where a robot is to move about a given a target area until every point in it is visited. Many efficient coverage algorithms cannot be used in practice, because they assume accurate movements by the robot; unfortunately, real robots have navigational errors. A standard costly solution is to utilize a robot that continuously localizes, so as to make course corrections. In this work we present TRIM SAIL, a novel hybrid coverage algorithm that takes as input an exact-movement coverage algorithm, and a maximal dead-reckoning error bound. It optimizes use of the exact-movement algorithm, so as to execute its coverage plan while minimizing movement and localization costs. TRIM SAIL guarantees complete coverage, even under dead-reckoning errors. We present several variants of TRIM SAIL and demonstrate their efficacy in experiments using data collected from real robots.

1 Introduction

Coverage [4] is a canonical robotics task, where robots are given a target work area, and move about the area until every point in the area is covered by a coverage tool associated with each robot. This tool is assumed to be the robots' sensors or specific actuator. There exist a number of elegant and efficient algorithms for single- and multi-robot coverage, that all assume accurate and exact movements by the robot. Among these we include essentially all grid-based and cell-decomposition methods, that divide the target area into smaller cells. [14; 10; 5; 9; 8]. These algorithms output a coverage plan, which—if followed without movement errors—results in complete coverage of the work area.

Unfortunately, real robots have navigation errors—called *dead reckoning errors* [2], which prohibit the direct use of exact-movement algorithms. The problem is that accumulating position errors cause the robot to drift away from its planned trajectory. There are several task-independent approaches to tackling dead-reckoning errors: Calibration or mechanical means [2]; compensation by using relative locations of multiple robots [11]; or using a hybrid system which executes the exact-movement algorithm's coverage plan while continuously executing localization procedures (e.g.,[12; 7; 3; 13]) to correct the motion errors. Coverage presents a unique challenge and opportunity related to dead-reckoning, which is not addressed by task-independent methods. On one hand, coverage requires more accurate movements; unlike other navigation tasks, when a robot is to *cover* some area between A and B, each point in its trajectory must be covered. On the other hand, if the coverage tool is sufficiently large, then some motion errors can be ignored, as long as the points on the trajectories are within the area of the coverage tool.

We present a novel hybrid coverage algorithm, called TRIM SAIL. TRIM SAIL takes as input an exact-movement algorithm, the coverage tool size, and a maximal dead-reckoning error bound. It optimizes use of the exact-movement algorithm, so as to execute its coverage plan while minimizing localization checks and corrections, i.e., minimizing movement and localization costs (e.g., in terms of time and battery). Given the error bound, TRIM SAIL guarantees complete coverage, even under dead-reckoning errors. We present several variants of TRIM SAIL, including a worst-case variant, and average-case heuristics to reduce costs.

To evaluate TRIM SAIL, we experiment using data collected from real robots. We show that the analytical predictions for execution costs match the actual performance of the robot. We additionally show that all versions of TRIM SAIL outperform a task-independent hybrid approach, in which localizations are continuously performed to correct dead-reckoning errors. Finally, we show that TRIM SAIL's performance is not sensitive to cost estimates—thus even if it uses incorrect estimates as to the movement and localization costs, it will still perform well in practice.

2 Related Work

Early investigations of dead reckoning explored mechanical methods that reduce errors, a-priori by mounting additional specialized hardware and calibration of the robot to reduce systematic odometry errors [2]. However, dead-reckoning errors cannot be completely eliminated. There are non-systematic errors that are caused by environmental uncertainties, e.g., wheel slippage.

Increasingly, probabilistic methods [12; 7] are used to carry out the process of fusing information from sensors, over time, to reduce the localization errors (which otherwise accumulate with movement). These technique successfully reduce odometry error by comparing the data obtained from the sensors in a different point of time, taking into account the movements of the robot and the noise in the readings. They also utilize

absolute location information (e.g., from GPS), if available.

In general, such methods require significant resources, and may also interfere with the robot's operation. For instance, in the RoboCup AIBO soccer league, the robots have to physically stop tracking the ball and the opponents, in order to free the camera to identify landmarks for localization. Our work thus focuses on optimizing the use of localization procedures. In particular, our work attempts to schedule localization requests during coverage tasks, so as to reduce costs.

An important motivation for our work is the prevalence of exact-motion coverage algorithms that are highly efficient, yet assume no dead reckoning errors. Choset [4] provides a survey of coverage algorithms. The Boustrophedon coverage algorithm is an efficient method, which relies on perfect localization [5; 9]. Spanning Tree Coverage (STC) [8] is another good example. STC-based algorithms divide the working area into cells of size equal to the robot tool, and build a Hamiltonian cycle that goes through all cells. While STC-based algorithms are efficient and easy to implement, they assume zero dead-reckoning errors, and fail in robots that have restricted capabilities [6].

Simultaneous Localization and Mapping [13] is a related task in which robots are required to map an unknown area, while also overcoming localization errors. The process requires making fusing sensory readings over time, and this puts additional constraints on the movements of the robots, which are not present in coverage. The techniques presented here do not target mapping.

3 Dead-Reckoning in Coverage

We restrict ourselves to *offline complete coverage*, where a map of the work area W, of size $M \times M$, is given, and the algorithms seek to guarantee that a robot visits every point in W. We focus on grid tessellation of the work-area, though in principle the techniques can be extended to other regular tessellation as well.

The robot's tool size is $D \times D$. Thus, when placed at a point p in the work-area, the robot covers a square of size $D \times D$, whose center is at p. The robot is assumed to be omnidirectional, or alternatively, be capable of moving forward and turning in place. We are given the angle α, which is the maximal deviation due to motion error (either left or right of the direction of the movement) as the robot moves in a straight line of a unit distance. The robot has a cost associated with a distance it travels, denoted by C_{drive} for each unit distance. This cost abstracts real-world cost components, such as execution time, battery usage, etc. Table 1 summarizes the notation used in this work.

Now, suppose we have an exact-motion coverage algorithm, denoted Alg_{exact}. This algorithm takes W and D as an input and computes *a coverage plan*—an ordered sequence of movements and heading changes (turns), which take the robot through cells, to completely cover W. Denote by $dist_1$ the distance the robot travels in order to perform this task. Then, the total cost of this coverage task would be equal to $C_{Alg_{exact}} = C_{drive} \cdot dist_1$. If D grows, the robot cover more area in each one of the steps. As a result, the robot needs to travel less to cover the environment, under the assumptions that its movements are accurate.

Notation	Definition
$M \times M$	The size of the work area W
$D \times D$	The size of the tool coverage
α	The dead reckoning error bound
Alg_{exact}	The exact-motion coverage algorithm
C_{drive}	The cost of drive
C_{loc}	The cost of one active localization
C_{total}	The total cost of the algorithm
q	The maximal localization precision error

Table 1: Notations used in this work.

However, dead-reckoning errors interfere in executing the coverage-plan. A robot blindly following the sequence of moves may not go through the intended cells, because dead-reckoning errors will cause its actual course to deviate.

Thus to execute the coverage plan, the robot must use localization procedures to assert its position on the intended trajectory, and to make corrections if necessary. We refer to this process as *localization*. We abstract away from the actual method of localization, and consider only the cost of this operation—in terms of time and battery power—which is denoted C_{loc}. In addition, localization has only a limited precision, bounded by $q \ll D$. If robot is localized at some position p, all we know is that robot stays in a square of size $q \times q$ that is centered at p.

The number of localizations made during coverage is denoted by N. When the robot deviates, it accumulates the additional travel distance. This accumulated distance (which includes course corrections) is denoted by $dist_2$. Then, the total cost of the algorithm is given by:

$$C_{total} = C_{drive} \cdot dist_2 + C_{loc} \cdot N \quad (1)$$

To minimize this total cost (Eq. 1), the robot must carefully balance its use of localization. When such localization checks are relatively expensive (e.g., in the RoboCup AIBO league, where robots must stop tracking the ball in order to localize), increasing the number of localization checks (N) significantly increases overall costs. On the other hand, reducing N too much requires larger corrections after each localization, and thus increases $dist_2$, the travel distance including deviations and their correction. We do this by considering the error bound α, and its relation to N.

Assuming an omnidirectional robot, we address movement in straight lines in arbitrary headings[1]. Without loss of generality, suppose that the path of the robot is in the direction of the x-axis. The ideal robot, without dead reckoning errors, will simply move in a straight line along the x-axis. A realistic robot will diverge from the straight line, with the accumulating dead-reckoning errors accelerating its departure from the x-axis.

Note, however, that localizations—and subsequent corrections—are not *constantly* required, i.e., are only required at some key locations. Suppose the size of each cell in the grid is d, $0 \leq d \leq D - q$. Then the straight line that Alg_{exact} generates goes through a number of $d \times d$-sized

[1]This is equivalent to assuming error-less turns in a robot that can move forward and turn in place. The relaxation of this assumption is straightforward, e.g., by requiring the robot to localize (and correct its position) with every turn.

cells. But because its coverage area $D \times D$ is actually greater than $d \times d$, it can in fact allow some deviation from the intended course. For instance, suppose the robot is to cover cells of size $d \times d$ ($d = \frac{D}{2}$). The robot can deviate by $\frac{D}{4}$ along the y-axis and still cover the cells (Figure 1).

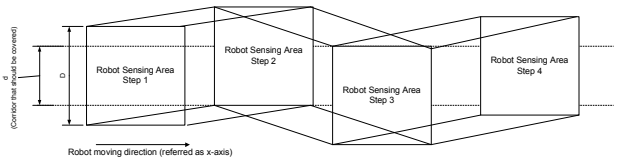


Figure 1: Example of robot motion which covers all cells, while still deviating.

This example presents an opportunity. We can control the value of d (the size of the grid used by an exact-motion coverage algorithm Alg_{exact}), such that it optimizes the use of localizations to minimize total cost. A hybrid algorithm would schedule localization actions (and their corrections) for Alg_{exact}'s coverage plan, augmenting it by periodic localization actions (and subsequent corrections, as necessary), and resulting in a complete coverage, at a minimal cost.

4 A Hybrid Coverage Algorithm

In this section we present an algorithm that utilizes a given grid-cell size parameter d, to provide complete coverage under dead-reckoning, using localizations only when necessary.

The TRIM SAILalgorithm (Algorithm 1) takes as input the exact-motion coverage algorithm Alg_{exact}; the grid-size parameter d; the robot coverage tool size D; the work area W; and α, the maximal dead-reckoning error bound (this assumes the left and right error bounds are equal; this assumption is relaxed in the experiments). It executes Alg_{exact} to create a coverage plan, and then executes the coverage plan while interleaving localization and course-corrections actions, as necessary. This results in movements as in Figure 1.

Algorithm 1 TRIM SAIL ($W, d, D, l, \alpha, Alg_{exact}$)

1: $CP \leftarrow Alg_{exact}(W, d)$ {Exact-motion coverage plan}
2: **for all** Plan step $stp \in CP$ (in order) **do**
3: **if** stp is a turn or heading change **then**
4: execute stp (and localize until pose is correct).
5: **else** {stp is a corridor}
6: **while** corridor Sq is not covered **do**
7: $(x, y, \phi) \leftarrow Localize()$
8: **if** $|Sq \bigcap Sq_{robot}| = d \times d$ **then**
9: $(r, \delta) \leftarrow$ CALCULATE$(d, D, \alpha, x, y, \phi)$
10: Change heading by angle δ
11: Set robot to travel distance of r.
12: **else** back-track until $|Sq \bigcap Sq_{robot}| = d \times d$

The algorithm first calls on Alg_{exact} to receive a coverage-plan, which assumes no dead-reckoning errors (line 1). This coverage plan is an ordered sequence of *turn* (*heading change* for omnidirectional robots) and *corridor* steps, defined as forward movement of some length. For each plan step, TRIM SAIL executes necessary localizations. Turns are executed in lines 3–4). For corridor steps, it interleaves calls to the localization action LOCALIZE() (line 7) with short movements (lines 10–11), whose angle and distance are computed in CALCULATE(), discussed below. TRIM SAIL continues this interleaved execution until the corridor is completely covered.

The robot pose (in the 2D area) is defined by three parameters (x, y, ϕ), which can be read by calling LOCALIZE(). x, y define the robot position, while ϕ defines the robot yaw (heading). We assume LOCALIZE() returns localization information with a precision defined by q.

The interleaving condition (line 8) checks whether the robot is still covering the corridor, or has possibly moved outside of it. The area that the robot currently covers is denoted by Sq_{robot}, and the corridor (of width d) is denoted by Sq, $|Sq|$ denoting the size of the area. If $|Sq \bigcap Sq_{robot}| = d \times d$ then the robot continues to cover the defined corridor. If $|Sq \bigcap Sq_{robot}| < d \times d$ then the robot deviation is too big and there is some portion of the corridor which is not currently covered. In this case, the robot needs to back-track to its previous location to re-cover the corridor (line 12).

CALCULATE (Algorithm 2) calculates the maximum distance r and heading-change δ the robot can travel until the next localization is required, under the assumption of the maximal error bound α. Using CALCULATE ensures that $|Sq \bigcap Sq_{robot}| = d \times d$ is always true, and line 12 in Algorithm 1 is never reached. However, line 12 will be used when α is heuristically estimated (Section 5). Theorem 4.1 asserts the correctness and completeness of TRIM SAIL.

Algorithm 2 CALCULATE ($d, D, l, \alpha, x, y, \phi$)

1: $m \leftarrow \cos 2\alpha(|y| + 0.5(D - l - d)) + 0.5(D - d) - |y|$
2: $n \leftarrow \sin 2\alpha(|y| + 0.5(D - l - d))$
3: $\theta \leftarrow \tan^{-1}(\frac{m}{n})$
4: $\delta \leftarrow \frac{\pi}{2} + \phi - \theta - \alpha$, but $\delta \leftarrow -\delta$ if $y < 0$.
5: $r \leftarrow \frac{|y| + 0.5(D-l-d)}{\cos\theta}$
6: return r, δ

Theorem 4.1 *If $|Sq \bigcap Sq_{robot}| = d \times d$ holds at the initial position of the robot, then Algorithm 1 achieves complete coverage of the environment.*

Proof Not shown for lack of space.

The following corollary is used in Section 5. It is used in alternative methods for determining d, which affects the cost of the coverage.

Corollary 4.2 *For a distance x planned by Alg_{exact}, a robot using Algorithm 1 travels the distance $r \leq \frac{x}{\cos 2\alpha}$.*

5 Reducing Localization Cost

Some of the parameters to TRIM SAIL can be arbitrarily set (d, provided to Alg_{exact}, and the error bound α). Larger values of d will result in smaller sequences of moves, but require more frequent localizations (N increases). Smaller d values allow for less frequent localizations (smaller N) but increase the correction distance. We first analytically determine the optimal value d_{min} for d, based on the maximal dead-reckoning error α, defined earlier. We then discuss estimating an average-case d, which would work well in practice.

Choosing d: Worst Case Analysis. Since the size of the map is $M \times M$, the number of cells of size $d \times d$ is $\frac{M^2}{d^2}$. Under assumption of no errors, a robot travels distance d for

each cell; the total distance the robot travels is therefore $\frac{M^2}{d}$. Based on Corollary 4.2, using TRIM SAIL to overcome errors, we can conclude that the total distance including corrections is bounded by $\frac{M^2}{d \cdot \cos 2\alpha}$. Also, the total number of localizations is bounded by $\frac{M^2 \cdot \sin 2\alpha}{d \cdot (D-d-l) \cdot \cos 2\alpha}$.

Denote $D' = D - q$. We extend Equation 1 and write down the expression for the total cost of the robot's work:

$$C_{total} = C_{drive} \cdot \frac{M^2}{d \cdot \cos 2\alpha} + C_{loc} \cdot \frac{M^2 \cdot \sin 2\alpha}{d \cdot (D' - d) \cdot \cos 2\alpha} \quad (2)$$

Equation 2 is a function of d, which provides an upper bound on the cost of the coverage under dead reckoning errors. To determine an optimal d, we find d values (in the interval $[0.D - q]$) that minimize this function. Note we used a worst case α to find d_{min} value. Because it relies on a worst-case analysis, this variant of TRIM SAIL never makes corrections, but may be more expensive than a riskier variant.

Using a Heuristic α Estimate. Observing the dead-reckoning errors of real robots, we find that most of the errors are much smaller than the worst case robot error α. Thus, we can use smaller values of the α in the TRIM SAIL algorithm (and Equation 2), to reduce the number of localizations. However, this risks greater travel costs, as corrections might be required. When the actual error is larger then the α value used, the robot will need to back-track to the point where its deviation was less or equal to the one allowed by the current d_{min} and α values (Line 12 in Algorithm 1). Thus the selection of a smaller α value must be carefully balanced against the cost incurred for corrections.

We estimate α using error data measured on a real robot. We propose (and empirically compare in Section 6) three heuristics, all based on analysis of the robot errors. Given an estimated α, we utilize the analysis for d_{min} value (Eq. 2):
—*Simple Symmetric Heuristic.* Use the mean of the distribution, ignoring the error sign (errors left of heading have a positive sign, others negative). This mean value is used as α.
—*Absolute-Value Symmetric Heuristic.* Estimate the mean from all errors, while ignoring the sign of the error.
—*Non-Symmetric Heuristic.* Collect the errors of the left and right sides separately; estimate their means separately.

6 Experiments

In this section we complement the analysis from previous sections with experiments with data from real robots. The experiment settings are described in Section 6.1. The first experiment (Section 6.2) compares the data obtained from real robot with the analytic estimates. Then, we compare the performance of the TRIM SAIL coverage algorithm—and the different heuristic estimates for α—with a naïve hybrid, which uses localization continuously (Section 6.3). Finally, we conduct sensitivity analysis to examine the robustness of the techniques to inaccuracies in cost estimates.

6.1 Experiment Settings

In order to evaluate the techniques described above, we obtained error data from a Friendly Robotics RV-400 robot, and used it to simulate the robot's movements across the hundreds of robot runs used in the experiments below. To limit reliance on the choice of the exact-motion coverage algorithm Alg_{exact}, we chose to use a corridor environment, in which all algorithms behave similarly. The robot and coverage algorithm settings are described below.

Robot settings. The RV-400 is a commercial vacuum-cleaning robot, which we fitted with our own control software (Figure 2). The RV-400 runs its own coverage software, but this software was disabled in these experiments. Instead, we run our own coverage algorithms.

Figure 2: An RV-400 robot, used in experiments.

To generate a data set of dead-reckoning errors, the RV-400 robot was commanded to move in a straight line, for a distance of 40cm. This was repeated 50 times, resulting a data set of 50 measurements. For each movement, we measured the error in the robot position at the end of the movement, and calculated the resulting error in heading (angle). This data set forms the basis for the motion error models that we use in this section.

Evaluating the techniques presented above requires measuring a large number of configurations, multiple times. For instance, to evaluate the upper bound computed based on Equation 2, we vary d in the range $[0, D - q]$, and repeat each setting 50 times. We additionally vary the heuristic technique used with Alg_{exact}. This would have made for an impractical number of runs with the physical robots. We therefore chose to conduct controlled experiments by simulating the movements of the robot, using the motion errors described above. With each simulated forward movement (each step) required by the controlling algorithm (TRIM SAIL, Alg_{exact}, etc.), we randomly picked one of the error values and moved the robot under the influence of this error. The simulated robot's movements accurately simulate its movements in our lab.

We use 40cm as the basic distance unit in all experiments, and in reporting all results. The real sensor range D was set to 2 meters (5 40cm units). Using the collected errors, we found that the maximal robot deviation α_{max} is bounded by $15.6°$. All experiment results are averages over 50 trials.

Coverage algorithm settings. For simulation purposes we set the environment area to be equal to $400m^2$ (2500 tiles, 40cm each side). Since the robot's sensors have a range of 2 meters, this corresponds to a corridor of 200m by 2m (500 by 5 of the 40cm steps). The use of a corridor was motivated by two factors: First, all coverage algorithms behave similarly (if not identically) in this environment, and thus the results would not depend on our choice of Alg_{exact}. Second, as TRIM SAIL's localization in turns is the same as any other exact-motion algorithm, this environment highlights TRIM SAIL's differences with existing work. Unless otherwise noted, the different costs were set with a 1:5 ratio (i.e., $C_{drive} = 100$ and $C_{loc} = 500$).

6.2 Calculating d: The Basic Technique

We first evaluate the upper bound in Eq. 2 with real-world data. We compare the cost of using TRIM SAIL (Algo-

rithm 1), with the values obtained from Eq. 2. We vary the virtual sensor size d. This will ensure that the minimum d_{min} computed based on Eq. 2 corresponds to the minimum in the real runs. We set d to 1, 2, 2.5, 3, 3.16 (the d_{min} value, computed based on $\alpha = 15.6°$ in the data), 3.5, 4, and 4.5 40cm steps. For each one of these 'virtual' grid sizes, we run a coverage algorithm for 50 times using the error data we obtained from the real robot.

Figure 3 presents the data obtained in these experiments. This figure compares the cost function of Algorithm 1 run in our simulation with the cost obtained from Eq. 2. It shows that indeed the real cost is bounded by the results from Eq. 2, by 14% in all the measured points. The qualitative behavior of both functions is identical. For both, $d = 3.16$ is a the minimum.

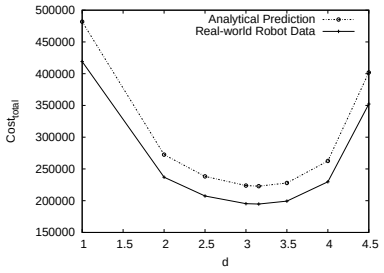


Figure 3: Comparison of running Algorithm 1 with real-world data (averaged over 50 trials), with the predicted cost obtained from Eq. 2.

6.3 Comparing Complete Coverage Algorithms

To establish a baseline for the experiments, we first run Alg_{exact}, as is, to measure its cost and coverage success. Because there are no localizations, Alg_{exact} never turns or travels to correct its location. However, its coverage percentage is poor; in the different trials, Alg_{exact} coverage percentage ran from 13.5% to 73% of the area, with a mean of 43.25%.

These results demonstrate the impact of violating the perfect dead-reckoning assumptions of many exact-motion coverage algorithms. Here, a provably-complete algorithm fails—by a significant margin—to provide complete coverage because its motion is erroneous. Many elegant exact-motion solutions to the coverage problems would suffer from similar problems. Direct comparison of TRIM SAIL to Alg_{exact} therefore does not make sense: Alg_{exact} would fail to provide complete coverage, which TRIM SAIL provides.

TRIM SAIL hybridizes exact-motion coverage algorithms, modifying their use in real-world settings, to maintain their proven properties of efficiency, robustness, etc. *while guaranteeing 100% (complete) coverage*. However, a more direct approach is possible in principle, where an exact-motion algorithm would simply be used together with continuous (repeating) localization. For instance, if landmarks are always sensed by the robot, then the robot can—in principle at least—run localization procedures without pause, resulting in continuous error corrections, and complete coverage.

We therefore turn to empirically evaluate TRIM SAIL and its heuristic variants (Section 5), against a naive use of an exact-motion algorithm with persistent localization. We compare the following techniques: Alg_{loc}, which is Alg_{exact} used with persistent localization (to create the best possible Alg_{loc}, we assume perfect localization); TS_{max} is the worst-case TRIM SAIL using the maximal heading error bound α_{max}; and TS_{simple}, TS_{abs}, TS_{ns} are TRIM SAIL variants using the simple-symmetric, absolute-value symmetric, and non-symmetric heuristics. We remind the reader that these heuristic variants attempt to reduce the number of localizations, at the risk of added travel distance for corrections.

The three heuristic methods TS_{simple}, TS_{abs}, and TS_{ns} all rely on estimating the distribution(s) underlying the error measurements. To do this, we used standard distribution-fitting procedures. We found that the results are best fitted by Pearson's Type 5 distributions, also known as *Pearson5* [1]. The distribution fit was done separately for each heuristic. The fitted mean (in the case of symmetric heuristics) or means (non-symmetric heuristic) were taken as the α value(s) used in the algorithms. For instance, for the simple symmetric heuristic, the fitted distribution had a mean of $\alpha_{simple} = 1.4703°$.

The results of the comparison appear in Table 2. Each row corresponds to a single algorithm, and the values in it are averaged over 50 trials. We use horizontal lines to distinguish the analytically-motivated algorithms Alg_{loc} and TS_{max} from the heuristic-based algorithms TS_{simple}, TS_{abs}, and TS_{ns}. The columns (left to right) provide the total distance traveled (in units of 40cm steps), the number of localization actions, and the distance/localization ratio. The final column indicates the total cost resulting from using the algorithm in question. Table 2 leads to several conclusions, explored below.

Name	Distance	Number of Localizations	Dist-Loc Ratio	Total Cost
Alg_{loc}	790.35	251	3.14	204544.98
TS_{max}	792.15	231.00	3.43	194715.00
TS_{simple}	1418.09	21.04	67.4	152329.00
TS_{abs}	973.28	33.12	29.39	**113888.00**
TS_{ns}	977.25	34.57	28.27	**115010.41**

Table 2: A Comparison of coverage results by different algorithms. All algorithms resulted in 100% coverage. Two best costs are in bold. Results averaged over 50 trials.

First, we see that under the cost ratio defined (100:500), even the worst-performing variant of TRIM SAIL—TS_{max} is better than using the exact-motion algorithm Alg_{exact} with continuous localization calls (Alg_{loc}). The distance traveled by Alg_{loc} is almost the same as TS_{max}, with a much greater number of localizations. This is because Alg_{loc} makes unnecessary corrections. Because it does not consider the geometry/size of the coverage tool, it repositions even if the area is already covered. Thus TRIM SAIL indeed offers a much more effective hybridization of the original algorithm.

Second, the results reveal a qualitative significant difference between the analytical method which seeks to guarantee performance using only the maximal error bound (TS_{max}), and the heuristic methods (TS_{simple}, TS_{abs}, and TS_{ns}) which seek to minimize cost by relying on additional knowledge (here, about the distribution of heading errors). The heuristic methods significantly outperform their worst-case counterpart, demonstrating their effective utilization of the additional knowledge they have. In particular, given that all three methods relying on our fitting the error distribution to

the Pearson5 distribution, we believe that this indicates that indeed this distribution type is appropriate for modeling dead-reckoning errors. To check this, we also experimented with other distribution types, and showed that Pearson5 is indeed superior. We do not provide the details here for lack of space.

Third, while the Absolute Symmetric (TS_{abs}) and Non-Symmetric (TS_{ns}) algorithms are significantly better than all others, their results are in fact non-distinguishable (two-tailed t-test results in $p = 0.32$).

6.4 Sensitivity to Cost Estimations

The distance-localization ratio of the best algorithms (Table 2) is lower than that of TS_{simple}, though higher than that of TS_{max}. The conclusion is that the results in Table 2 might be dependent on the actual cost estimates (travel cost and localization cost), which are used in TRIM SAIL. Here, we explore the sensitivity of the results to errors in the cost estimates provided to the algorithms.

Ratio⟶ Name↓	1:10 (0.1)	**1:5 (0.2) (original)**	1:1 (1)
Alg_{loc}	330035.00	204535.00	104135.00
TS_{max}	310215.00	194715.00	102315.00
TS_{abs}	**130448.00**	**113888.00**	**100640.00**
TS_{ns}	132296.12	115010.41	101181.84
TS_{simple}	162849.00	152329.00	143913.00

Table 3: A Comparison of total costs for each algorithms, under different travel-to-localization cost ratios. Best costs are in bold.

Table 3 shows the total costs for the different algorithms, when the travel-to-localization cost ratio is systematically changed from the original settings (marked, fourth column from left). First, we note that the TS_{abs}, which we found earlier to be the best, remains so under extreme changes to the cost ratio: The result holds from a cost ratio of 1:25 until a cost ratio of 1:1. Thus one conclusion is that the top performing heuristic technique is in fact extremely robust to cost estimate errors.

A second important conclusion is reached contrasting the the top two rows (Alg_{loc} and TS_{max}). We see that TRIM SAIL provides superior performance to that of the other hybrid approach, *in all cost ratios*. This again demonstrates the efficacy of the methods we presented in this paper.

7 Conclusions

In this paper we presented TRIM SAIL, a hybrid coverage algorithm (and associated heuristics, geometric optimizations) for real-world settings. TRIM SAIL takes an exact-motion coverage algorithm, which assumes no dead-reckoning errors, and uses it to guide angled movements that guarantee complete coverage of the target work area, while minimizing the use of localization to that strictly necessary. We presented an analytical worst-case version of TRIM SAIL, and three heuristics which further reduce total coverage costs. We then reported on extensive experiments with TRIM SAIL, using data collected from the RV-400 robot. The experiments demonstrated that (1) the analytical methods accurately predict an upper bound for total costs, and minimum cost, given robot error bounds and coverage range; (2) the heuristic methods outperform the analytical methods in the cost ratio chosen; (3) TRIM SAIL variants are not sensitive to errors in cost estimates; and (4) that the TRIM SAIL algorithm *always* outperforms naive coverage hybridization, where the exact-motion algorithm is simply coupled with continuous localization. In the future, we hope to explore new heuristic directions which take more risks in terms of completeness of coverage, but provide reduced costs.

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, 1964.

[2] J. Borenstein., H. Everett, and L. Feng. *Navigating Mobile Robots: Sensors and Techniques*. A. K. Peters, Ltd., Wellesley, MA, 1996.

[3] A. Burguera, G. Oliver, and J. Tardos. Robust scan matching localization using ultrasonic range finders. In *IROS-05*, pages 1367–1372, 2005.

[4] H. Choset. Coverage for robotics - A survey of recent results. 31(1–4):113–126, 2001.

[5] H. Choset and P. Pignon. Coverage path planning: The Boustrophedon decomposition. In *International Conference on Field and Service Robotics*, 1997.

[6] N. Correll and A. Martinoli. Distributed coverage: From deterministic to probabilistic models. pages 379–384, 2007.

[7] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *ICRA*, pages 1322–1328, 1999.

[8] N. Hazon and G. Kaminka. On redundancy, efficiency, and robustness in coverage for multiple robots. *Robotics and Autonomous Systems*, 2008.

[9] W. H. Huang. Optimal line-sweep-based decompositions for coverage algorithms. volume 1, pages 27–32, 2001.

[10] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[11] I. M. Rekleitis, G. Dudek, and E. E. Milios. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *IJCAI97*, pages 1340–1345, 1997.

[12] S. Thrun. Finding landmarks for mobile robot navigation. In *ICRA*, pages 958–963, 1998.

[13] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[14] A. Zelinsky, R. A. Jarvis, J. C. Byrne, and S. Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *In Proceedings of International Conference on Advanced Robotics*, pages 533–538, 1993.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Bridging the Gap Between Semantic Planning and Continuous Control for Mobile Manipulation Using a Graph-Based World Representation

Roland Philippsen[*†], **Negin Nejati**[‡], **Luis Sentis**[†]
[†]Stanford Robotics and AI Lab, [‡]Stanford Computational Learning Lab
roland.philippsen@gmx.net, negin@stanford.edu, lsentis@stanford.edu

Abstract

We present our ongoing efforts to create a mobile manipulation database tool, a flexible multi-modal representation supporting persistent life-long adaptation for autonomous service robots in every-day environments. Its application to a prototypical domain illustrates how it provides symbol grounding to a reasoning system capable of learning new concepts, couples semantic planning with whole-body prioritized control, and supports exploration of uncertain and dynamic environments.

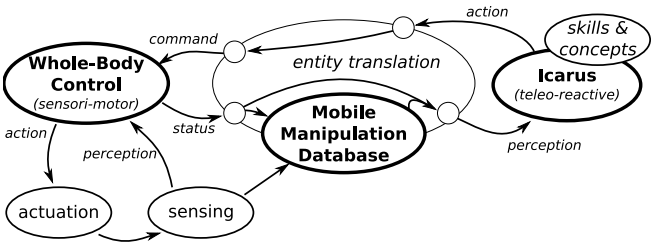


Figure 1: Overview of the system components implemented for the proof of concept presented in this paper, with arrows denoting information flow.

1 Introduction

In this position paper, we describe our approach for bridging the gap between a sensori-motor control framework, developed at the Stanford Robotics Lab, and a flexible symbolic teleo-reactive reasoning system, developed at the Stanford Computational Learning Lab. The context of our research is to integrate several areas of research in autonomous systems, employing learning, planning, perception, and control, to achieve task-oriented whole-body motions for a robot's physical interaction in environments shared with humans.

Our position is that the scalability of skills required for autonomous mobile manipulation in everyday environments can be achieved by combining a reasoning and learning system built on goal-indexed hierarchical task networks with a continuous control framework capable of handling physical interaction behaviors, and that this combination requires a representation that is rich enough to encompass information relevant to both of these components yet lightweight enough to act as a "live" model of the world and the robot.

We rely on a smart database that serves as a white-board between components, in order to *(i)* ground symbols in the robot's low-level sensing and action capabilities, *(ii)* maintain and share information relevant to more than one component, and in the long run *(iii)* support life-long adaptability of the robot in changing and uncertain environments.

Figure 1 illustrates the system architecture. The *mobile manipulation database* (MMDB) is the central component that allows *entity translation* to be a very lightweight process. *Whole-body control* (WBC) implements local continuous-domain task-oriented behaviors, while *Icarus* provides symbolic commands based on global goals and the current state of the world. This architecture is a draft that is strongly inspired by the classical three-tiered approach. It serves the purpose of illustrating a proof of concept, but the boundaries between layers are not strictly defined: Icarus is a teleo-reactive system that encompasses reasoning and execution, and the MMDB handles information that pertains to all components of the system.

Robotics and AI researchers have investigated everyday manipulation tasks for a long time, and addressing this application requires integrating approaches from several subfields. Thus, the amount of related work is quite vast, and we limit ourselves to an overview of the contributions that have most strongly influenced our collaboration so far. Related work is given at the beginnings of the sections presenting Icarus, WBC, and MMDB.

Part of the integration challenge stems from differing technical terms and unstated assumptions. Thus, one of the starting points is to define common terminology, of which we give a very short summary here.

Tasks can mean either *(i)* high-level specifications of desired actions expressed in formal logic clauses (in Icarus), or *(ii)* a task is a specific facet of continuous control (in WBC). A **goal** is a set of states which correspond to a desired outcome, where the achievement of the goal must be measurable. **Planning** is the process of finding a sequence of actions to take the system from a known initial state to a spec-

[*]This work has been supported by the Swiss National Science Foundation under the Fellowship for Advanced Researchers, grant number 115346

Table 1: Examples of percepts, concepts and skills in Icarus.

Percepts

```
(object 26 tag window state clean)
(close-to 29)
```

A concept definition example

```
((all-windows-clean)
  :percept ((object ?window tag window))
  :relations ((not (dirty ?window))))
```

A skill definition example

```
((clean ?window)
  :subgoals ((close-to ?window)
    (action-clean ?window)))
```

ified goal state. **Control** refers to real-time computation of actuator commands such that operational constraints are satisfied. **Manipulation** refers to actions that *(i)* influence the arrangement of objects in the robot's environment or *(ii)* interact physically with other agents (e.g. helping a person cross the street). **Perception** acquires and interprets information about the environment, yielding information grounded in sensor readings and ranging from low-level geometrical features to high-level abstract states.

The whole-body controller is not built on formal logic, and thus it is necessary to define a taxonomy of behaviors it can execute in order to integrate it with Icarus. This is an ongoing process, which gives each whole-body behavior a *name*, lists what *task types* it includes (e.g. table 2), specifies the types and ranges of *parameters* it accepts (e.g. controller gains, velocity and acceleration bounds), what it's *inputs* are (e.g. desired posture, visual feedback), and what kind of *output* or feedback it provides to higher levels.

2 Icarus: Reactive Symbolic Planning

An autonomous agent needs the ability to robustly choose actions that lead to its goal while continuously considering its changing perceptions of the dynamic environment. Teleo-reactive programming is a formalism for computing and organizing actions for an agent that provides this capability [Nilsson, 1994]. Icarus [Langley and Choi, 2006] is a cognitive architecture for physical agents with a commitment to teleo-reactive logic programs as the representation which supports execution and acquisition of complex procedures (figure 2). At each cycle, Icarus perceives the environment, recognizes situations, and chooses an action based on the situation and the goal, aided by two knowledge bases: *(i)* **conceptual** knowledge allows recognizing relevant situations and describes them in a higher level of abstraction, and *(ii)* **skill** knowledge encodes how the agent can affect its environment. Concepts are encoded as hierarchical monotonic inference rules with a syntax similar to Horn clauses. Skills are represented with **goal-indexed** Hierarchical Task Networks (HTNs) [Nau *et al.*, 2003]. Each skill is a recipe for decomposing a high level task into lower level ones, providing a partial ordering between them, and specifying a precondition that needs to be satisfied in the environment before it can get selected. We chose goal-indexed HTNs as skill representation because *(i)* they provide transferable solutions to similar

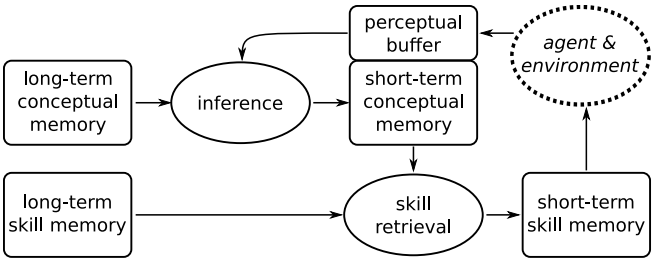


Figure 2: The Icarus cognitive architecture employs long- and short-term memories for concepts and skills in order to produce goal-directed actions that take into account a changing and uncertain environment.

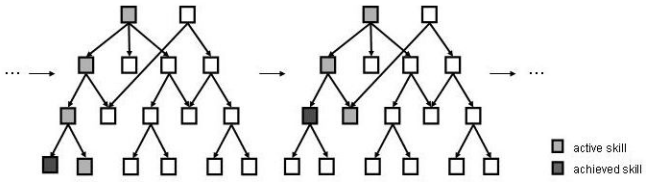


Figure 3: Icarus uses HTNs for teleo-reactive skill-selection. Skills are goal-indexed: at the highest level are the global goals, and primitive skills are leave nodes corresponding to executable actions.

problems and *(ii)* they can be automatically acquired. Table 1 provides examples for percepts, concepts and skills.

Once a goal is chosen, Icarus works out the relationship among different actions and propose a suitable one at each step (see figure 3). These hierarchies are built by Icarus dynamically at each runtime cycle and the skill path is selected in a top-down manner starting at the skill indexed by the intended goal and rooted in a primitive skill which is applicable in the current state of the world. This is more scalable than traditional controller programming: once new behaviors and goals are added into the system, Icarus can automatically use them together with the previous knowledge. Icarus keeps the skill selection path as similar as possible across cycles, but if a previously achieved goal becomes untrue again, it can interrupt its current activity to re-achieve the older goal.

Goal-indexed HTNs can be time consuming to craft, making it worthwhile to investigate automatic ways of acquiring them [Choi and Langley, 2005], [Nejati *et al.*, 2006]. The latter introduces LIGHT, an approach for HTN learning by observing sequences of operators taken from expert solutions to a problem. By analyzing the solution in the context of a background knowledge, LIGHT learns skills for achieving complex tasks, their preconditions, and partial ordering among their subgoals. It is important to note that the hierarchical nature of the learned skills is crucial for scalability as shown in [Nejati *et al.*, 2006] and that learning flat macros e.g. [Mooney, 1990] is more equivalent to the finite state machine approach.

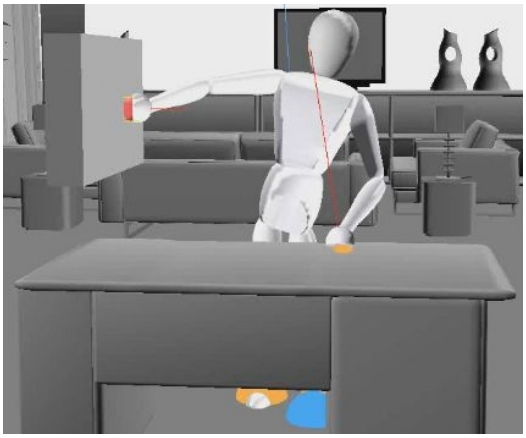


Figure 4: Example WBC behavior for cleaning a vertical surface while maintaining several operational constraints

3 Prioritized Multi-Objective Control: Executing Complex Behaviors

Choosing a symbolic action does not make a robot move its motors. The discrete symbolic structure has to be translated into continuous sensori-motor feedback, and for this we rely on model-based control applicable to mobile manipulators and humanoid robots. We consider the problem of coordinating the physical behavior of the robot operating in the complex environment. The robot is required to accomplish arbitrarily complex tasks, which involve manipulation and locomotion behaviors as well as the handling of environmental constraints.

In particular, for this work we exploit our work on interactive control of a humanoid robot [Sentis and Khatib, 2005]. This framework leverages potential field control techniques to address the simultaneous optimization of multiple low-level criteria characterizing the skills of the robot. It combines the potential fields from all desired criteria using a prioritized control hierarchy, producing motion behaviors such that all criteria can be optimized while satisfying the assigned priorities. Figure 4 illustrates an interactive behavior with multiple potential fields to clean windows. The criteria designed to execute this behavior are shown in table 2.

The difficulty of operating in every-day human surroundings arises from their inherent complexity and variability. Using whole-body skills as described above, we address all aspects of the motion including both goal-based tasks and constrained behaviors. To deal efficiently with constraints we build models that address the complex contact and topological interactions with the environment.

For example, we have recently developed a model called the virtual linkage model to characterize the contact state of the robot. Using optimization techniques it enables the design of internal force behavior and locomotion policies that comply with frictional and rotational contact constraints.

To implement potential field control strategies, we create a generalized dynamic model of the robot that relates actuator and body accelerations to generalized control torques as well as to contact forces with the environment. This model provides an effective interface to project artificial potential fields into actuator space. Working at the torque level and aided by the dynamic and contact models mentioned earlier, we create force compliant behaviors that are capable of dealing with unplanned contact events and contact variability in the environment.

Table 2: Decomposition for the task shown in figure 4.

Task Primitive	*Coordinates*	*Control Policy*
Contact support	internal forces	optimal contact
Joint Limits	joint positions	locking attractor
Self Collisions	distances	repulsion field
Balance	CoM(x, y)	position
Right hand	Cartesian	force and position
Gaze	head orientation	position
Upright posture	marker coordinates	captured sequences

Another important characteristic of the execution layer is its hierarchical architecture, which is designed to analyze and handle action conflicts by imposing priorities between the control objectives. Priorities are used as a mechanism to temporarily override certain non-critical criteria in order to fulfill critical constraints. To reinforce the planning process, the execution framework estimates at runtime the feasibility of the commanded actions and returns detailed information on the causes of the conflicts. Aided by Icarus, feasibility information is aimed at triggering the replanning process of the robot's behavior, which will result in finding alternative paths that optimize the desired chores.

4 Mobile Manipulation Database

As depicted in figure 1, the role of the mobile manipulation database (MMDB) is to collect information relevant for the interaction between the various components of the system, and to help mediating the data flowing between them. It allows components to retrieve collections of entities matching some search criteria. For example a path planner could request all location nodes and locomotion links along with their associated path costs. However, the MMDB goes beyond "simple" database operations by using a graph-structure that naturally encodes multiple semantic aspects of the world, and providing an event infrastructure that allows components to be notified when certain types of entities are added, removed, or changed.

Related work that influenced the formulation of the MMDB comes from three main sources. The Semantic Spatial Hierarchy of [Kuipers, 2000] is one of the fundamental contributions allowing to ground an abstract topological representation of space in the noisy and uncertain sensori-motor system of autonomous robots. Later work of the same lab (e.g. [Beeson, 2008]) pursues this line of research, with aspects of sensori-motor learning increasing in importance. [Vasudevan, 2008] solves representation of space using a hierarchy of objects and their relationships, in order to support spatial cognition, and gives a good overview of further related work. Concerning planning and execution, the works of [Haigh and Veloso, 1998] and [Veloso *et al.*, 1995] provide system architectures and planner systems that integrate learning through knowledge structures that are interpretable across

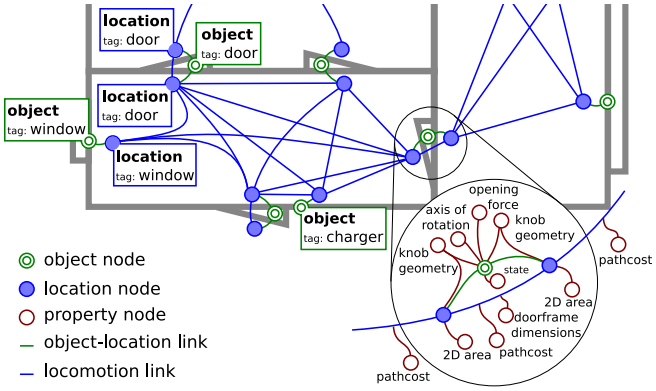


Figure 5: Example of a room with doors, windows, and a charging station, illustrating the graph underlying the MMDB.

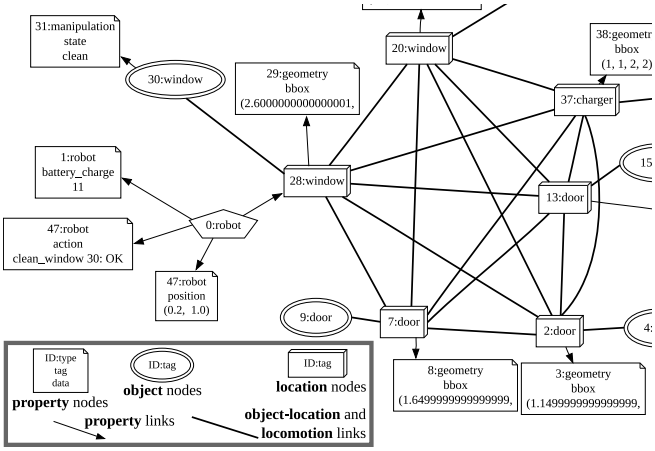


Figure 6: Snapshot of the MMDB (partial view) after the robot has cleaned the window represented by node 30. At this stage, the battery has been depleted to 11% which causes Icarus to interrupt the window-cleaning task in order to go and recharge the robot.

several components. Yet all these contributions are related to mobile robot navigation or HRI and do not directly allow us to integrate manipulation using the whole-body control approach [Sentis and Khatib, 2005].

Intuitively, the MMDB has to *(i)* provide spatial and functional information about the world, *(ii)* represent object identities and connections between regions and objects, and *(iii)* allow components to easily store and retrieve information relevant for their functioning (e.g. "retrieve all locations with links to an object labelled as a cup"). Figure 5 shows an illustrative example.

The MMDB contains *entities* that are either *nodes* or *links*. Each entity has a unique *ID*, an associated *type* and *tag*, and optional mutable *data* which are initialized when the entity is created. **Nodes** represent pieces of information that are relevant for the robot, e.g. an object node representing a cup that it is supposed to wash. **Links** can be directed or bidirectional and represent relationships between nodes, e.g. a collection of SIFT features that the robot can use to detect and localize a cup prior to grasp planning. The **ID** is essential for symbol grounding, it ensures that the various components "talk" about the same "thing", e.g. when Icarus requests locating object 17, the vision system ends up retrieving the correct set of SIFT features in order to find that specific cup. The **type** and **tag** are essential for searching the database and filtering events, they encode an ontology for an application domain. **Data** is usually set only for property nodes, it encodes an actual piece of information, such as a point cloud coming from stereo vision.

We are aware that there is a large body of work on knowledge bases and ontologies that can be exploited for reasoning in the domain of autonomous mobile manipulation. At this early stage of our research, in order to demonstrate that the overall approach is feasible, we chose to use a simple ad-hoc ontology, outlined below. However, given that the *type* of entities is stored as a string, the MMDB remains ontology-agnostic, at least for the time being.

The currently implemented node and link types are: location, object, object-location, locomotion, agent, geometry, and manipulation. **Location** (e.g. door, window, room) permits reasoning about space. **Object** (e.g. door, window, charger) is fundamental for modeling, recognition, and manipulation of objects. **Object-location** links a region of space with a physical entity. **Locomotion** links two regions of space that are known to be reachable via locomotion from each other. **Agent** nodes collect a robot's internal state information as far as it pertains to more than one component. **Geometry** properties (e.g. bounding box, length, point-cloud, mesh) collect metric information about entities, e.g. for grasp planning or fusion over longer exploration periods. **Manipulation** properties (e.g. state, rotation axis, maximum forces) store data necessary for planning or controlling manipulation tasks.

5 Evaluation in an Example Domain

As a prototypical evaluation scenario, we have chosen to implement a world with rooms, doors, windows, and a charger. The task of the robot is to clean all the windows, exploring all rooms and recharging itself as required (all operations use up some amount of energy). Some of the doors are locked, in order to verify that failure detection at the WBC level can be propagated to Icarus. However, integration with WBC is not yet done, so at this stage we verify the symbol grounding of geometrical entities by assigning bounding boxes to objects and simulating controller failures and successes depending on the location of the robot with respect to these bounds (e.g. it needs to be close enough to a window to clean it). When a door to a previously unknown room is opened, the windows and doors contained in the new room are injected into the MMDB. Thus, the controller is not aware of any symbols, Icarus is not aware of any bounding boxes, and the MMDB provides the translation between the two. The result of cleaning a window gets reflected in the *manipulation state* property of window objects, which starts out as *dirty* and transitions to *clean* only if the controller succeeded. Similarly, doors can be in an *open*, *closed*, *locked*, or *unknown* state.

Icarus is a Lisp program and the MMDB has been prototyped in Python. The two are connected using XMLRPC, and at each step the entire graph is logged, yielding output similar to figure 6 (after some manual layout adjustments). This example demonstrates how MMDB translates between symbols and geometric entities, how Icarus makes skill selection flexible and scalable (adding door opening skills and exploration goals are simple additive adjustments that do not require rewiring any program) and how the window-cleaning task can be interrupted by battery-charging and then resumed.

6 Conclusions & Outlook

In this position paper we have presented early stages of our work on a mobile manipulation database and have motivated this research in the context of autonomously performing tasks that can be useful in every-day human environments. Symbolic representations and reasoning methods on the one hand allow to make globally informed decisions and ensure that execution is goal-oriented, by "abstracting away" much of the low-level details in order to make the problem tractable. On the other hand, continuously controlling the motion of an autonomous robot such that it safely and effectively operates in physical contact with objects and even humans focusses on more local details such as real-time control and rigid body dynamics.

The evaluation we presented demonstrates that two of the three objectives mentioned in the introduction are presently fulfilled by the MMDB: *(i)* ground symbols in the robot's low-level sensing and action capabilities, *(ii)* maintain and share information relevant to more than one component. Concerning our objective to *(iii)* support life-long adaptability of the robot in changing and uncertain environments, we give a motivating example as outlook: Suppose that the perceptual apparatus can detect doors but does not provide information needed for operating it, such as whether it is a sliding or rotating door, or how exactly to grip the handle. WBC provides sensori-motor exploration to figure out the missing details by trying out various alternatives. Once the robot has discovered how a particular door can be opened, the relevant data is stored in the MMDB, attaching this WBC-specific information to an entity that is shared between all components. Later, the robot perceives another door of similar appearance. We can then use the parameter set of the first door as an initial guess at how this new door can be opened.

The MMDB provides representational support for system integration, aimed at fulfilling a specific set of requirements for a more general problem, namely to expose those parts of a component's internal data structures that are needed for useful interaction between approaches coming from various sub-fields of AI and robotics. In this position paper, we only present a prototypical implementation of the MMDB, which lacks the event mechanisms that shifts the burden of detecting environment dynamics and the action of other components into a representational system that has all the required information at its disposal. For the proof-of concept, the lack of events is not a limiting factor: the number of nodes and links is very small, and events are emulated by iterating over all entities at each step. However, it is already apparent that these capabilities will promote loose but effective coupling between components. In particular, we are interested in adding learning at the HTN level, exploration and SLAM at the geometric and spatial-topological levels, and providing a rich set of sensory inputs.

One promising direction of future research concerns feeding high-level contextual information from Icarus into the MMDB. We expect this to help with disambiguating object perception, and to provide more fine-grained specification of WBC behaviors. For example, it would be possible to inject or relax manipulation constraints such as maintaining an upright orientation of a container depending on whether it contains liquid or is empty. Exploration of unknown environments is another application that we feel will benefit from the MMDB.

References

[Beeson, 2008] Patrick Beeson. *Creating and Utilizing Symbolic Representations of Spatial Knowledge using Mobile Robots*. PhD thesis, The University of Texas at Austin, August 2008.

[Choi and Langley, 2005] Dongkyu Choi and Pat Langley. Learning teleoreactive logic programs from problem solving. In *Proceedings of the Fifteenth International Conference on Inductive Logic Programming*, pages 51–68, 2005.

[Haigh and Veloso, 1998] Karen Zita Haigh and Manuela M. Veloso. Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots*, 5(1):79–95, 1998.

[Kuipers, 2000] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000.

[Langley and Choi, 2006] Pat Langley and Dongkyu Choi. A unifed cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 51–68, 2006.

[Mooney, 1990] R. J. Mooney. *A general explanation-based learning mechanism and its application to narrative understanding*. Morgan Kaufmann, San Mateo, CA, 1990.

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

[Nejati *et al.*, 2006] Negin Nejati, Pat Langley, and T. Konik. Learning hierarchical task networks by observation. pages 665–672, 2006.

[Nilsson, 1994] Nils Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[Sentis and Khatib, 2005] Luis Sentis and Oussama Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(4):505–518, 2005.

[Vasudevan, 2008] Shrihari Vasudevan. *Spatial Cognition for Mobile Robots: A Hierarchical Probabilistic Concept-Oriented Representation of Space*. PhD thesis, ETH Zürich, Switzerland, 2008. Diss. ETH no. 17612.

[Veloso *et al.*, 1995] Manuela M. Veloso, Jaime Carbonell, Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: the prodigy architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Approaches to Learning for Hybrid Dynamical Cognitive Agents

Eric Aaron and **Henny Admoni**
Department of Mathematics and Computer Science
Wesleyan University
Middletown, CT 06459

Abstract

As a foundation for goal-directed behavior, a hybrid agent's reactive and deliberative systems can share a single, unifying representation of intention. In this paper, we summarize a framework for *hybrid dynamical cognitive agents* (*HDCAs*) that incorporates a single representation of *dynamical intention* into both reactive and deliberative structures of a *hybrid dynamical system* agent model, and we introduce the first proposed approaches to learning for such intention-guided agents. The HDCA framework is based on ideas from *spreading activation* models and *belief-desire-intention* (*BDI*) models: Intentions and other cognitive elements are represented as interconnected, continuously varying quantities, employed for goal-directed intelligence by both reactive and deliberative processes. Methods for learning that modify interconnections among an HDCA's cognitive elements —such as *Hebbian* associations based on co-active elements, and *belief-intention* learning of task-specific relationships— can therefore improve goal-directed performance without additional reliance on deliberation. We also present simple demonstrations of agents that learned geographic and domain-specific task relationships in a virtual grid world, and we discuss limitations and potential extensions of our approaches to HDCA learning.

1 Introduction

In hybrid reactive / deliberative agents, if the reactive and deliberative levels share a single, unifying representation of intention, goal-directed intelligence can be effectively distributed over both levels. Reactive-level learning could therefore improve intention-guided behavior without entailing additional reliance on deliberation for intelligent performance.

As an example, consider three hybrid agents completing tasks in a simulated grid city, running several errands (e.g., deposit a check at the bank, borrow a book at the library, buy a book at the bookstore) starting from a shared initial position and navigating to various target locations. Along with various desires and beliefs, each agent starts out with *dynamical intentions* in its cognitive system, one for each task it might perform; each dynamical intention has a cognitive *activation* value, representing the intensity of commitment to the corresponding task, the task's relative *priority*. All cognitive activation values vary continuously during the agents' run, reflecting continuous changes both in agents' relative task priorities and, more generally, in their overall cognitive systems. Therefore, cognitive states change due to both deliberative and sub-deliberative processes: Deliberative methods can re-plan agents' task sequences; sub-deliberative, continuous cognitive evolution also causes tasks' relative priorities to rise and fall, which can also re-order task sequences.

One of these three agents, A_R (for *Rules*), is a hybrid agent in which some geographic and task-specific intelligence arises from explicit deliberative rules. In particular, when deliberating to re-plan its task sequence, A_R employs a sorting-based *distance bias*, giving higher priority to tasks that can be completed closer to its current position, which is intended to result in faster errand runs. Agent A_R also follows the *one-book rule*: As A_R runs errands, propositional rules encode that it either buys or borrows a book, but not both; importantly, the one-book rule has no effect on which of the two book-related tasks is performed first—indeed, the rule does not affect agent cognition or behavior until after a book-related task is completed. A second hybrid agent, A_{NR} (for *non-Rules*), is identical to A_R except that the deliberative system of A_{NR} does not encode either the distance bias or the one-book rule. Instead, the task selection intelligence of A_{NR} relies on reactive priorities, simply selecting a maximal priority task at every opportunity. Unsurprisingly, as they run errands, agents A_R and A_{NR} take different paths through their world. They begin at the same location, and their paths are initially identical, but after each buys a book, they diverge: Agent A_{NR}, not following the one-book rule or the distance bias, eventually borrows a book and finishes its errands somewhat late; in contrast, agent A_R follows its rules and finishes sooner.

The third simulated agent, A_L (for *Learning*), is identical to A_{NR} in its deliberation, with no explicit distance bias or one-book rule, but it has learned certain useful reactive-level associations in its cognitive system. Due to training runs in which it perceived the target locations of the errands, A_L has learned to co-associate intentions corresponding to tasks with geographically proximate targets, so when a task T has high priority, priorities are raised on all tasks T' with target lo-

cations near that of T. In addition, A_L has learned that the completion of one of the *BuyBook* and *BorrowBook* tasks — represented by explicit beliefs about task completion— is to be negatively associated with the intention to do the other book-related task. As a result, when A_L runs errands, even though it has no deliberative encodings of the distance bias or the one-book rule, it behaves similarly to A_R: Agent A_L buys a book but does not borrow one, and it sequences its errands to finish faster than A_{NR} does (although not as fast as A_R). Because dynamical intention-guided intelligence is distributed over both the reactive and deliberative levels of agent A_L, it learned rule-like behavior from cognitive associations without explicit propositional rules, improving its goal-directed performance without additional dependence on deliberation.

In this paper, we summarize a framework for *hybrid dynamical cognitive agents* (*HDCAs*, for short) that supports such dynamical intention-guided intelligence. The design of HDCAs' cognitive systems is influenced in a somewhat unconventional way by the *belief-desire-intention* (or *BDI*) theory of intention [Bratman, 1987]; the theory and its related implementations (e.g., [Georgeff and Lansky, 1987; Rao and Georgeff, 1991] and successors) suggest that BDI elements (beliefs, desires, and intentions) are an effective foundation for goal-directed intelligence. Unlike typical BDI agent implementations, HDCAs' cognitive models interconnect BDI elements in a continuously evolving system inspired by (though different from) *spreading activation* frameworks of [Collins and Loftus, 1975; Maes, 1989]. Each BDI element in an HDCA is represented by an activation value, indicating its salience and intensity "in mind" (e.g., how intensely committed an intention), and cognitive evolution is governed by differential equations, so elements' activation values affect rates of change of other elements' activations. HDCAs employ these dynamical cognitive representations on both reactive and deliberative levels, enabling smooth hybrid integration and enhancing agent performance by distributing goal-directed intelligence over both levels. For example, as described in [Aaron and Admoni, 2009a], HDCAs' cognitive systems can productively, dynamically re-order planned task sequences and resolve inconsistencies among cognitive elements —such as an intention to mail a letter co-occurring with a belief that there is no letter to be mailed— without invoking deliberation.

HDCAs' reactive cognitive models are also influenced by some *distinguishing properties* that differentiate intention from desire (noted in [Bratman, 1987]). For examples, in HDCAs, an intensely committed intention I diminishes impacts of other intentions on the intensity of I; the strongest intentions (i.e., intentions with the most intense commitment) need not correspond to the strongest desires; and intentions, not desires, govern HDCAs' task priorities. In these ways, HDCAs' dynamical intentions function as conventional BDI-based intentions in goal-oriented behavior.

In addition, we introduce the first proposed approaches to learning for HDCAs —the kinds of learning employed by agent A_L, above— two methods based on modifying interconnections among cognitive elements: *Hebbian* learning, which strengthens associations based on co-active elements; and *belief-intention* learning that can flexibly encode a range of focused, task-specific relationships. In section 5 of this paper, we describe simple demonstrations of agents learning in a virtual grid world, establishing the effectiveness and general functionality of these methods. We also discuss potential extensions of these approaches to learning, emphasizing how the *hybrid dynamical system* framework underlying HDCAs constrains and illuminates what such extensions might require.

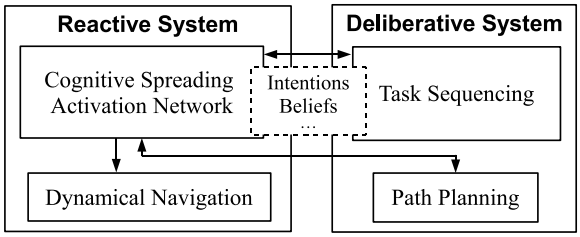


Figure 1: System-level architecture of an HDCA, showing deliberative and reactive levels. Cognitive representations are shared by the sub-deliberative spreading activation network and the deliberative task sequencing process.

2 Model Structure

The underlying HDCA model draws upon several different conceptual frameworks, including *hybrid automata*, *BDI* modeling, and *spreading activation* models.

- At its foundation, an HDCA model is a finite state machine. Each state corresponds to a continuous action or mode of behavior; in each state, differential equations govern agent behavior, and transitions between states are instantaneous. HDCAs are thus modeled as *hybrid automata*, as described in section 2.1.
- Cognitive elements of HDCAs are primarily *BDI* elements —beliefs, desires, and intentions— represented by continuously evolving *activation* values. In each state of an HDCA model, evolution of activation values is governed by the differential equations in that state, as described in section 2.2.
- Cognitive elements are interconnected in a (somewhat unconventional) *spreading activation* framework: Elements serve as variables in differential equations, so the activations of cognitive elements affect the (rates of change of) activations of other cognitive elements. These influences and interconnections among cognitive elements are described in section 2.2.

In this section, we further explain and illustrate these ideas.

2.1 Hybrid and Deliberative Structure

The reactive / deliberative structure of HDCAs is illustrated in Figure 1, showing sub-deliberative cognitive and dynamical navigation processes; deliberative task sequencing and path planning processes; and cognitive representations shared across levels. Each level employs cognitive representations in its own manner, but the representations fully support both levels, for straightforward hybrid integration. The particular

deliberative task sequencing and path planning processes of HDCAs in this paper are simple, although other, more complicated methods could be readily employed. Planners essentially derive "utility" values for each option, each task or path segment, based on geographic information, task-specific knowledge, and cognitive activations. Plans, then, are essentially sequences (e.g., of tasks or path segments) in decreasing order of utility; higher commitment to an intention (i.e., higher intention activation) translates to higher utility for the associated task but does not solely determine task sequence.

Deliberation is designed to be invoked only in situations that are not well handled by fully reactive processes. For agents in this paper, deliberation occurs only in two circumstances: if the current task is unexpectedly interrupted (e.g., by a blockaded street in the grid world); or if the agent is called upon to change its current task —due to completing the previous task, evolutions of intention activations, or any other cause— and must select from multiple candidates with essentially equivalent intention activations. Unlike *reactive task resequencing* —the changing of relative task ordering due only to continuous evolutions of intention activations (i.e., task priorities)— HDCAs' deliberative task sequencing incorporates constructs such as the distance bias and the one-book rule. Deliberation also re-evaluates an agent's entire task sequence, adjusting activations of cognitive elements so that, e.g., tasks earlier in the sequence have higher activations on corresponding intentions, and precluded tasks have intentions with highly negative activations. After deliberation, an agent simply continues with its new cognitive activation values in the reactive behavior of its new highest priority task.

In addition to being a hybrid reactive / deliberative system, an HDCA is a *hybrid dynamical system* (*HDS*, for short), a combination of continuous and discrete dynamics, modeled by a *hybrid automaton* [Alur *et al.*, 2000]. A hybrid automaton is a finite state machine in which each discrete state (or *mode*) is a continuous behavior, containing differential equations that govern system evolution in that mode. Transitions between modes (including those from a mode to itself) are instantaneous, occurring when *guard* conditions are met, and may have discontinuous *side effects*, encoding discrete system dynamics. Hybrid dynamical systems can be apt models for navigating robots or animated agents (e.g., [Aaron *et al.*, 2002b; Axelsson *et al.*, 2005]), and HDCAs' reactive and deliberative structures naturally correspond to HDS elements: Each task of an HDCA is a reactive behavior, implemented as an HDS mode; deliberation in HDCAs only occurs during transitions between tasks.

2.2 Reactive Structure

For this paper, an HDCA's physical state (position and heading angle) continuously varies as it navigates, with steering based on [Goldenstein *et al.*, 2001] and simple intersection-to-intersection navigation in a grid world similar to the method in [Aaron *et al.*, 2002a], although other dynamical approaches could be equally effective. This continuous physical state cleanly integrates with an HDCA's cognitive system, which is based on continuously evolving activations of BDI elements (beliefs, desires, intentions); differential equations govern continuous evolutions of all elements, physical and cognitive. (Element values can also be changed discretely, as effects of mode transitions; after completing a task, for example, HDCAs' mode transitions set the activation of the corresponding intention to the minimum possible value and the activation of the belief that the task has been completed to the maximum possible value.) Figure 2 shows BDI elements (and abbreviations for their names) and the mode transition model for HDCAs in this paper, which is simplified to a one-to-one correspondence between intentions and tasks.

Activation values of BDI elements are restricted to the range $[-10, 10]$, where near-zero values indicate low salience and greater magnitudes indicate greater salience and intensity of associated concepts; thus, for example, more active intentions represent more commitment to and urgency of the related tasks. Negative values indicate salience of the opposing concept, such as, for intentions, intention not to perform the related task. For this paper, we restrict beliefs to only two values, -10 (*false*) and 10 (*true*), although the system could in principle express intermediate degrees of belief.

Cognitive activations are interconnected in differential equations; equation 1 is a partial cognitive system (with many elements omitted), where beliefs, desires, and intentions are represented by variables beginning with B, D, and I, and time-derivative variables are on the left in each equation:

$$\begin{aligned} \dot{D_{DC}} &= a_1 B_{HC} + a_3 I_{DC} - a_5 I_{GC} + \cdots \\ \dot{I_{DC}} &= b_1 B_{HC} + b_3 D_{DC} - b_6 D_{HH} + \\ &\quad b_8 I_{DC} - b_{10} I_{GC} + \cdots . \end{aligned} \tag{1}$$

This illustrates interconnectedness: Elements exert *excitatory* or *inhibitory* influence by increasing or decreasing derivatives, due to positive or negative connections —i.e., addition or subtraction of the relevant terms— in the cognitive system. Variables stand for activations of cognitive elements (e.g., desire to deposit a check, D_{DC}). Coefficients encode impacts of connections; many are constants, but intention coefficients also contain components that encode *distinguishing properties of intention* (see section 3) or are employed in implementing learning (see section 4). For this paper, we make the simplifying assumption that any two intentions mutually conflict, and from that, the excitatory or inhibitory nature of cognitive connections is generally intuitive—each intention is negatively connected to all other intentions; each belief that a task is completed is negatively connected to the corresponding intention (and each belief that a task is not completed is positively connected to an intention); desires are positively connected to corresponding intentions, including the desire to get a book being positively linked to the intentions for both *BuyBook* and *BorrowBook*; etc.

In demonstrations for this paper, learning in HDCAs consists of altering relationships among cognitive elements, i.e., changing coefficients in cognitive differential equations such as equation 1. The processes by which this learning occurs are discussed in section 4; other ways by which an HDCA might potentially learn are discussed in section 6.

There is also a mechanism for *perception* in HDCAs, by which proximity to items (e.g., a building, a street blockade) affects activations in agents' cognitive systems. Current HDCAs have only limited perceptual structure; potential for substantial extensions exists but is not discussed in this paper.

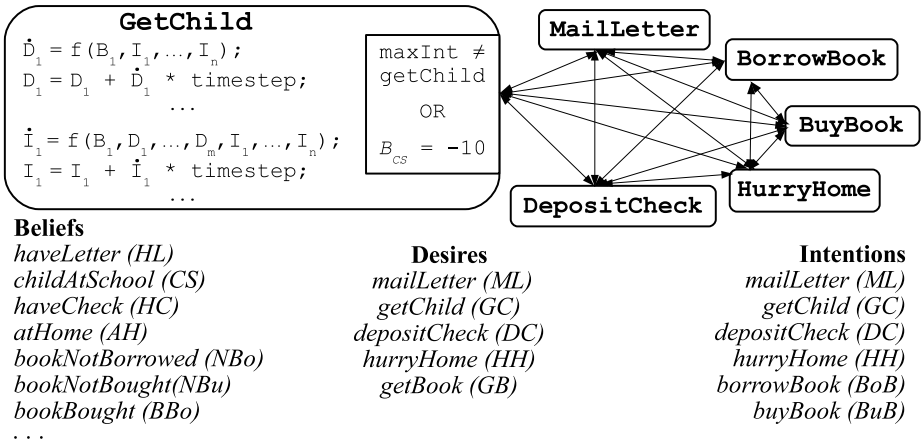


Figure 2: Hybrid dynamical system modes and BDI elements (including abbreviations for names) for HDCAs in this paper.

3 Properties of Intention

By employing representations of BDI elements in HDCAs' cognitive systems, we implicitly invite comparison of our somewhat unconventional BDI application with the philosophical foundations of BDI agents in [Bratman, 1987]. In particular, it might initially seem possible that the entities called intentions in HDCA cognition are not *genuinely* BDI intentions: HDCA intentions might be inconsistent with properties noted in [Bratman, 1987] that distinguish intentions from other cognitive elements (most notably from desires). We carefully implement HDCA intention, however, so as to be consistent with theoretical *distinguishing properties* that apply to our dynamical account of intention: Intentions are *conduct controlling* elements that, when salient, *resist reconsideration* and *resist conflict* with other intentions.[1]

For *reconsideration resistance*, we encode two criteria: any *high-active* intention I_a (i.e., having high activation magnitude) tends to minimize impacts on I_a from other intentions; and the magnitude of this effect grows as the activation (magnitude) of I_a grows. To enable this, for intentions I_a and I_b, for every $a \neq b$, the differential equation for $\dot{I}_a$ includes the following structure:

$$\dot{I}_a = \ldots \; - k_n \cdot PF(I_a) \cdot I_b \; \ldots. \tag{2}$$

For example, in equation 1, the coefficient of I_{GC} in the equation for $\dot{I}_{DC}$ has the form $b_{10} = k_{10} \cdot PF(I_{DC})$, with *persistence factor PF* defined as

$$PF(I_a) = 1 - \frac{|I_a|}{\sum_i |I_i| + \epsilon}, \tag{3}$$

where noise term $\epsilon > 0$ prevents division by 0, and i ranges over all intentions. For $b \neq a$, $PF(I_a)$ multiplies every intention I_b in the equation for $\dot{I}_a$, so as $PF(I_a)$ nears 0 (i.e., as I_a grows in magnitude relative to other intentions), contributions of every such I_b are diminished, and when $PF(I_a) = 1$ (i.e., $I_a = 0$), such contributions are unaffected. The denominator encodes that I_a is less reconsideration-resistant when other intentions are highly active.

Demonstrations of *PF* and similarly activation-oriented mechanisms for *conduct control* and *conflict resistance* establish that dynamical intentions are consistent with the distinguishing properties of intention noted above; see [Aaron and Admoni, 2009b] for additional discussion about HDCA intention that is beyond the scope of this paper. Supported by our results, we treat dynamical intentions in HDCAs as conventional BDI intentions, rather than as some other cognitive elements inconsistent with [Bratman, 1987]; this is a necessary, if subtle, part of the foundation of HDCAs.

4 Learning

Because HDCA intelligence is based on agents' cognitive activation networks, we propose two methods of learning for HDCAs that modify interconnections among cognitive elements: *Hebbian* learning, which strengthens associations based on concurrently salient cognitive elements; and *belief-intention* (*BI*) learning, which illustrates the flexible adaptivity of HDCA's cognitive networks to encode domain-specific task relationships. In particular, we describe how HDCAs can learn to approximate two aspects of goal-directed intelligence discussed in section 1: We apply Hebbian learning to train agents to approximate the *distance bias*, incorporating geographic proximity information into task sequencing; and we apply BI learning to train agents to approximate the *one-book rule*, relating beliefs about the completion of the *BuyBook* and *BorrowBook* tasks to the intentions to perform the tasks.

4.1 Hebbian Learning

We implement Hebbian learning (inspired by ideas of synaptic plasticity and neuronal interconnections in [Hebb, 1949]) to enhance connections based on concurrently high-active elements in agents' cognitive systems. In particular, HDCAs in this paper learn to intensify connections among intentions corresponding to geographically proximate target locations.

[1] These are not the only properties of intention that are emphasized in [Bratman, 1987]; they are, however, properties that can apply to reactive-level intention, not requiring, e.g., future-directedness incompatible with reactive implementations.

For our Hebbian learning, training consists of an HDCA making a single circuit through its grid world, taking a prespecified route that passes in proximity to all possible task-targets (i.e., locations at which tasks are completed, such as a post office or library). During this training session, an agent represents these targets as *ground concepts*, cognitive elements corresponding to entities perceived in its environment; there is a one-to-one correspondence between task-targets and tasks, and thus between task-target ground concepts and tasks (hence intentions, as well). Ground concepts have baseline activation values of 0, but when an agent is within its *radius of perception* of a task-target, activation on the associated ground concept instantaneously rises; except in extraordinary cases, ground concept activations do not increase beyond initial levels. When the agent moves outside of its radius of perception from the target, the target loses salience, and the corresponding activation gradually decreases to zero. (For implementation details, see the supplementary website [Aaron and Admoni, 2009b].) In this way, targets are co-active — i.e., with activation values both greater than 0— only when they are geographically proximate, with greater co-activation when the targets are perceived closer to each other on an agent's training run.

Based on these concept activations, agents learn to associate intentions corresponding to co-active (i.e., proximate) task-targets. At every timestep during a training session, for any task-target ground concepts a and b ($a \neq b$) with activations greater than 0, the following adjustment is made:

$$IC(I_a, I_b) = IC(I_a, I_b) - \frac{\beta}{c_1}. \quad (4)$$

In this equation, intention I_a is the intention corresponding to the task with target represented by ground concept a (similarly for I_b), $IC(I_a, I_b)$ is the coefficient on the I_b term in the differential equation for $\dot{I_a}$, β is the activation of ground concept b, and c_1 is a scaling constant.

Because all intentions are negatively interconnected in HDCAs in this paper, this Hebbian learning weakens the inhibitory links between intentions corresponding to co-active targets; the extent to which the inhibitory effect of I_b on $\dot{I_a}$ is diminished, moreover, is proportional to the activation of b, so more highly active concepts have stronger effects. Therefore, in agents trained this way, positive activation on an intention I is less inhibitory to intentions corresponding to task-targets near that of I, with the extent of this effect intuitively corresponding to perceived proximity of the related task-targets.

4.2 Belief-Intention Learning

To enable HDCAs to flexibly relate intentions to perform certain tasks with conditions on whether other tasks are or are not completed —relationships that might naturally be encoded in propositional rules (e.g., the one-book rule)— we implemented *belief-intention* (*BI*) learning, which alters cognitive connections among intentions and task-completion beliefs. In particular, we apply BI learning to train HDCAs to behave consistently with the one-book rule: to perform exactly one of the two complementary book-related tasks, *BuyBook* and *BorrowBook*, in such a way that the relative priorities of the two tasks are not altered by the rule until one of the tasks is complete. Training for BI learning consists of an agent simply running errands in its grid world; when the agent reaches home at the end of its run, if it did not perform exactly one of *BuyBook* and *BorrowBook*, coefficients are adjusted in its cognitive system, and it undertakes another training run from the same initial position. Training stops when the agent performs exactly one of the two book-related tasks.

To enable BI learning of the one-book rule, some coefficients in HDCAs' cognitive systems were implemented as:

$$\begin{aligned} IC(I_a, B_b) &= k_b(\frac{B_{\bar{b}} - 10}{-20}) \quad (5) \\ IC(I_a, B_{\bar{b}}) &= k_{\bar{b}}(\frac{B_{\bar{b}} - 10}{-20}). \end{aligned}$$

In these equations, a and b range over the two book-related tasks, restricted to $a \neq b$, so I_a is the intention for task a, B_b is the belief that task b has been completed, $B_{\bar{b}}$ is the belief that task b has not been completed, and $IC(I_a, B_b)$ ($IC(I_a, B_{\bar{b}})$, respectively) is the coefficient for term B_b ($B_{\bar{b}}$) in the differential equation for intention I_a. Values $k_b, k_{\bar{b}}$ are constants, and the term $\frac{B_{\bar{b}}-10}{-20}$ in each coefficient ensures that neither B_b nor $B_{\bar{b}}$ affect I_a when $B_{\bar{b}}$ is true (value 10), i.e., when task b is not completed. To learn the one-book rule, inhibitory connections are strengthened between beliefs that books have been obtained and intentions to obtain books. Specifically, after each training run that did not adhere to the one-book rule, coefficients are altered as follows:

$$IC(I_a, B_b) = IC(I_a, B_b) * c_2 \quad (6)$$

(and similarly for $IC(I_a, B_{\bar{b}})$), where $c_2 > 1$ controls the extent of the modification of coefficients. Thus, this BI learning strengthens the inhibitory links between beliefs that one book-related task has been completed and the intention to perform the complementary task, leading to one-book rule-like behavior: Before either task is completed, the beliefs have no enhanced effect on intentions, but after one is completed, activation on the complementary intention rapidly decreases. This is an example of the relationships that can be learned with BI learning; other relationships, linking different beliefs and intentions under different circumstances, could be learned by the same general method.

4.3 Integrating Hebbian and BI Learning

For the particular applications of Hebbian and BI learning in this paper, because the two methods alter disjoint sets of cognitive connections, HDCAs can straightforwardly employ both methods without structural complications. In this paper, a training run for this integrated Hebbian-BI learning consists of an agent running errands in the grid world along an autonomously determined path. An agent concludes training when its most recent training run meets conditions similar to the distance bias and one-book rule—i.e., the agent performs exactly one of the two book-related tasks, suggesting adequate one-book rule learning; and the entire errand run takes no less time than the previous run did, suggesting adequate distance bias learning. (All HDCAs in our simulations move at identical, constant speed, so time and distance are equivalent measures.) Deeper accounts of HDCA learning could

explore different training conditions and applications of Hebbian and BI learning to overlapping sets of cognitive connections, but our simplified conditions suffice for the illustrative demonstrations in section 5.

5 Demonstrations and Experiments

We simulated HDCAs in the grid world of Figure 3 to demonstrate effects of learning on HDCAs' intention-guided intelligence. In each simulation, one or more HDCAs navigated to the targets in Figure 3, completing a series of tasks (listed on the display as *MailLetter*, ..., *BuyBook*). Some agents were designed with deliberative systems encoding the distance bias and one-book rule, while others were trained to approach such performance by altering their reactive cognitive systems; we tested agents by comparing performance on autonomous errand runs, evaluating agents' task sequences and times at which tasks were completed. We compactly summarize our results here; the supplementary website for this paper [Aaron and Admoni, 2009b] has animations and additional information about these simulations, including parameter values employed in equations of the learning methods.

5.1 Hebbian Learning

To establish that Hebbian learning can improve HDCA performance, a purely reactive agent —an HDCA with its deliberative processes disabled— learned to approximate the distance bias, with training as described in section 4.1: Agent A_H (for *Hebbian*) made a single training run with radius of perception $r_p = b + i$, where b and i are the lengths of a block and an intersection in the grid world; its pre-specified path passed within r_p of each target location. At each timestep, cognitive coefficients were updated as in equation 4.

Agent A_H was then tested by comparing its errand-running performance to that of agent A_{NH} (*non-Hebbian*), which was identical to the original, pre-training A_H. Both agents made single errand runs from the same position with the same initial cognitive activations; the initial location and cognitive activations were those for the training of A_H. Each agent's cognitive system and task priorities evolved individually during the test run, but because the initial activations were highest on intentions corresponding to remote target locations — e.g., $I_{MailLetter}$, because the post office is not near any other target— the effects of learning were not immediately apparent: With no high-priority targets proximate to other targets, both agents began their runs on similar paths, completing their first two tasks simultaneously. After that, however, the agents' behavior diverged, suggesting that the reactive-level learning of A_H affected navigation in accord with the distance bias: After both agents completed *MailLetter* and *DepositCheck*, agent A_{NH} next went to the school to complete *GetChild*, whereas A_H went to the bookstore and then the library, the next task-targets in order of geographic proximity.

5.2 Belief-Intention Learning

To establish that belief-intention (BI) learning can improve HDCA performance, a purely reactive agent A_{BI} learned to approximate the one-book rule, with training as described in section 4.2: Agent A_{BI} autonomously ran errands, adjusting cognitive coefficients as in equation 6 and undertaking further training runs when a run did not include exactly one of the book-related tasks; each of its 7 training runs began with the same cognitive element activations and from the same position near the library on the left of the grid world.

After its training had concluded, A_{BI} was tested by comparison to agent A_{NBI}, which was identical to the original, pre-training A_{BI}. Tests were run from 16 starting locations, the intersections in the four-by-four grid world, which did not include the training location; for each test run, the agents autonomously ran errands with identical initial cognitive element activations (the same as those for the training of A_{BI}) from their shared starting point. Agent A_{BI} completed exactly one book-related task on 15 test runs, whereas A_{NBI} completed one book-related task on 8 test runs, suggesting that BI learning enables reactive-level changes in HDCA cognition to encode one-book rule-like behavior.

5.3 Integrated Hebbian and BI Learning

A demonstration similar to the three-agent errand-running example in section 1 of this paper illustrated the integration of Hebbian and BI learning, showing that HDCAs can learn behaviors consistent with the distance bias and one-book rule without explicit deliberative encoding of either. Training of learning agent A_L was in accord with the description in section 4.3; each of its 18 training runs began with the same cognitive element activations and from the same position near the library on the left of the grid world. (See supplementary website [Aaron and Admoni, 2009b] for relevant parameter values and other implementation details.)

After training, A_L was tested with two other agents: agent A_{NR}, which was identical to the pre-training A_L; and agent A_R, which was identical to A_{NR} except that it had explicit, deliberative encodings of the distance bias and one-book rule. Tests were run from 16 starting locations, the intersections in the four-by-four grid world, which did not include the training location; for each test run, agents autonomously ran errands with identical initial cognitive element activations (the same as those for the training of A_L) from their shared starting point. On every run, A_R bought a book but did not borrow one, due to its explicitly encoded one-book rule; by comparison, on 15 of the 16 test runs, A_L bought but did not borrow a book, whereas A_{NR} completed both book-related tasks on every test run. Additionally, A_L always finished the run in less time than A_{NR}, though later than A_R. Indeed, on 11 of the 16 test runs, A_L and A_R performed the same task sequence, and A_L finished less than 0.75 seconds behind A_R, with the difference seemingly due to the time immediately after completing *BuyBook* in which the activation on I_{BoB} in A_L decreased as an effect of BI learning. Together, these results support our results about individual Hebbian and BI learning methods, suggest the effectiveness of integrated Hebbian-BI learning in this task domain, and suggest the potential for learned behavior to successfully generalize beyond a training set.

6 Discussion

The simple demonstrations in section 5 contain agents with limited perceptual mechanism and little world interaction, but

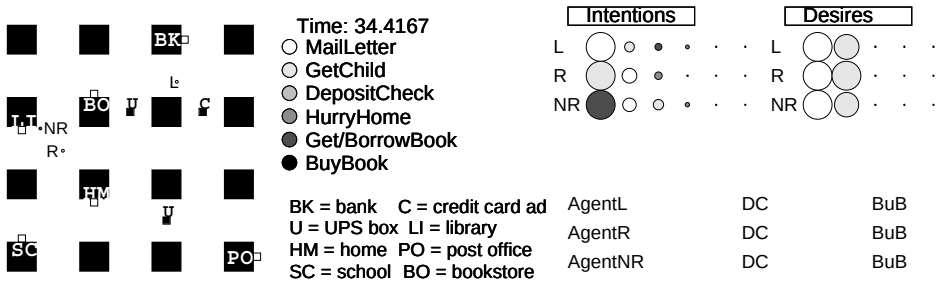


Figure 3: Screen display of a simulation in progress. A map of the four-by-four grid world, left, shows buildings and obstacles (black squares), targets (white squares abutting buildings), and three agents (*L*, *R*, and *NR*). Visual representations of agents' desire and intention activations are on the right, beneath which are lists of tasks completed by each agent.

the underlying ideas are designed to be more generally applicable. HDCAs' deliberative and reactive systems employ the same cognitive representations, and there is nothing necessarily unconventional about HDCAs' deliberative levels, so this dynamical intention-based framework could potentially support more powerful learning, perception, and deliberative inference methods for HDCAs than those in section 5, with similarly clean integration of reactive and deliberative intelligence. Here, we discuss some limitations and possible extensions of HDCAs' cognitive models and of the approaches to learning for HDCAs presented in this paper.

6.1 Hybrid Dynamical Cognitive Modeling

In the demonstrations in this paper, HDCAs' representations of BDI elements are overloaded: We simplify agent cognition by conflating salience and cognitive intensity or commitment "in mind," so an activation value represents both the intensity and the salience of an element. If future applications required agents to separate those features —e.g., to be very aware (high salience) of a mild desire (low intensity)— the HDCA framework could readily adapt, supplying multiple activations for each cognitive element. A cost of such expansion, however, would be a larger, potentially less computationally efficient cognitive system. We have not yet addressed efficiency concerns —our code was in MATLAB and not fully optimized for speed— or whether sophisticated HDCAs are practical for large-scale applications.

To further simplify the implementations and explanations in this paper, HDCA deliberation is treated as if it were instantaneous: We modeled deliberation as part of instantaneous mode transitions in hybrid automata rather than by explicit HDS modes that model time spent during deliberation, although such deliberation-modes could in principle be designed and incorporated without altering HDCAs' cognitive representations. Modeling HDCAs as hybrid automata also supports the clean interconnections among deliberative and reactive processes, and it enables analysis methodologies for HDSs (e.g., [Asarin *et al.*, 2007], which accommodates nonlinear systems) to potentially algorithmically verify some aspects of agent behavior, although practical and theoretical restrictions on HDS analysis [Alur *et al.*, 2000], make HDCA behavior verification an extremely challenging problem.

6.2 Learning for HDCAs

Because cognitive representations are shared by HDCAs' reactive and deliberative structures, the reactive-level learning methods in this paper can improve intention-guided behavior without additional reliance on deliberation, and more powerful HDCA learning methods might achieve even stronger results. Because of the sophisticated goal-directed intelligence that could be learned, a supervised approach closer to the BDI agent learning in [Subagdja and Tan, 2008] may well be more successful than an evolutionary algorithm-based approach [Bugajska *et al.*, 2002] or a parameter fitting-based approach to hybrid dynamical system learning (even, e.g., the multi-phase approach in [Kawashima and Matsuyama, 2005]). Further investigation is needed to determine which approaches might be most productive for HDCA learning.

The learning demonstrations in this paper are incomplete in several ways. We have not fully investigated many critical aspects of the methods in section 4, such as choices of cognitive interconnections to alter during learning, parameter settings for the alterations performed during learning, and conditions for the learning processes to commence and conclude. Further testing is also needed to determine how our results might generalize to a wide range of agents' initial cognitive activations or to environments other than our grid world. (In particular, the Hebbian learning of the distance bias may not easily generalize to other environments.) We hand-coded a mechanism specifically to train agents' cognitive networks to approximate the one-book rule, but in a more generally applicable and developmentally insightful system, this learning might instead arise organically from primitive factors —e.g., desires to have a book, finish errands quickly, and not carry too much— that could also serve as foundations for other rules and constraints on agent behavior.

No matter how general a foundation for HDCA learning might be, however, adjusting interconnections among cognitive elements would not result in learning an *explicit propositional rule*; instead, the HDCAs would learn a *tendency* that approximates a rule, as in the examples in section 5. Different learning methods, however, might potentially enable explicit propositional rule learning in HDCAs. Ultimately, such rules are encoded by guards on HDS modes and transi-

tions between modes (see section 2.1); in principle, it seems possible to construct an HDS model in which the guards and mode transitions are parameterized, and the defining parameters could be altered at run time, thus permitting rule learning. This is related to the problem of learning new *behaviors* —i.e., incorporating new modes into an agent's HDS model— which would require adding new modes, mode transitions, and guards during an execution, essentially reshaping the mode-transition system to include the new behavior. The formal structure of HDSs thus illuminates how unconventional HDS models might potentially support such guard-level or mode-level learning.

7 Conclusion

Hybrid dynamical cognitive agents are based on continuously varying cognitive representations that are shared by deliberative and reactive processes, distributing intention-guided intelligence over both levels. The Hebbian and belief-intention learning methods in section 4 are the first approaches to learning proposed for HDCAs, and demonstrations of these methods show that HDCAs' cognitive networks can encode geographic or task-specific factors that might otherwise be elements of deliberative processes, augmenting goal-directed intelligence without additional dependency on deliberation. Variations of these approaches to learning might potentially train an HDCA's deliberative system by directly altering the mode-transition structure of its underlying hybrid dynamical system model; the formal HDS structure illuminates what such learning would entail, although it remains unclear how HDS analysis methods could be adapted for such unconventional, learning-oriented HDS models.

Although our simple demonstrations contained small numbers of HDCAs, with limited perceptual mechanism and world interaction, the underlying ideas are more generally applicable, and more sophisticated approaches to learning might have far greater effects on both reactive and deliberative performance. Ultimately, dynamical intention-based approaches to learning could lead to intention-guided agents that emphasize reactive intelligence and employ deliberation only when needed, making them more robust, efficient performers in multi-agent scenarios and other dynamic applications.

Acknowledgments

The authors thank Jim Marshall, Michael Littman, and HYCAS paper reviewers for their insightful and helpful comments on previous versions of this paper.

References

[Aaron and Admoni, 2009a] E. Aaron and H. Admoni. A framework for dynamical intention in hybrid navigating agents. In *Hybrid Artificial Intelligence Systems*, 2009. To appear.

[Aaron and Admoni, 2009b] E. Aaron and H. Admoni. Supplementary HYCAS 2009 material, 2009. Available at http://eaaron.web.wesleyan.edu/hycas09_supp.html.

[Aaron *et al.*, 2002a] E. Aaron, F. Ivančić, and D. Metaxas. Hybrid system models of navigation strategies for games and animations. In *Hybrid Systems: Computation and Control*, pages 7–20. 2002.

[Aaron *et al.*, 2002b] E. Aaron, H. Sun, F. Ivančić, and D. Metaxas. A hybrid dynamical systems approach to intelligent low-level navigation. In *Proceedings of Computer Animation*, pages 154–163. 2002.

[Alur *et al.*, 2000] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 88(7):971–984, 2000.

[Asarin *et al.*, 2007] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of non-linear systems. *Acta Informatica*, 43(7):451–476, 2007.

[Axelsson *et al.*, 2005] H. Axelsson, M. Egerstedt, and Y. Wardi. Reactive robot navigation using optimal timing control. In *American Control Conference*, 2005.

[Bratman, 1987] M. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[Bugajska *et al.*, 2002] M. D. Bugajska, A. C. Schultz, J. G. Trafton, M. Taylor, and F. E. Mintz. A hybrid cognitive-reactive multi-agent controller. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2807–2812, 2002.

[Collins and Loftus, 1975] A. M. Collins and E. F. Loftus. A spreading activation theory of semantic priming. *Psychological Review*, 82:407–428, 1975.

[Georgeff and Lansky, 1987] M.P. Georgeff and A.L. Lansky. Reactive reasoning and planning. In *AAAI-87*, pages 677–682, 1987.

[Goldenstein *et al.*, 2001] S. Goldenstein, M. Karavelas, D. Metaxas, L. Guibas, E. Aaron, and A. Goswami. Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers And Graphics*, 25(6):983–998, 2001.

[Hebb, 1949] D. O. Hebb. *The Organization of Behavior*. John Wiley & Sons, Inc., New York, NY, USA, 1949.

[Kawashima and Matsuyama, 2005] H. Kawashima and T. Matsuyama. Multiphase learning for an interval-based hybrid dynamical system. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E88-A(11):3022–3035, 2005.

[Maes, 1989] P. Maes. The dynamics of action selection. In *IJCAI-89*, pages 991–997, 1989.

[Rao and Georgeff, 1991] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proc. of Principles of Knowledge Representation and Reasoning*, pages 473–484, 1991.

[Subagdja and Tan, 2008] B. Subagdja and A. H. Tan. Planning with iFALCON: Towards a neural-network-based BDI agent architecture. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 231–237, 2008.

1st International Workshop on Hybrid Control of Autonomous Systems — Integrating Learning, Deliberation and Reactive Control (HYCAS 2009)
Pasadena, California, USA, July 13 2009

Extended Markov Tracking with Ensemble Actions

Zinovi Rabinovich and **Nicholas R. Jennings**
Electronics and Computer Science,
University of Southampton,
Southampton SO17 1BJ
{zr,nrj}@ecs.soton.ac.uk

Abstract

In this paper we extend the control methodology based on Extended Markov Tracking (EMT) by providing the control algorithm with capabilities to calibrate and even partially reconstruct the environment's model. This enables us to resolve the problem of performance deterioration due to model incoherence, a negative problem in all model based control methods. The new algorithm, Ensemble Actions EMT (EA-EMT), utilises the initial environment model as a library of state transition functions and applies a variation of prediction with experts to assemble and calibrate a revised model. By so doing, this is the first control algorithm that enables on-line adaptation within the Dynamics Based Control (DBC) framework. In our experiments, we performed a range of tests with increasing model incoherence induced by three types of exogenous environment perturbations: catastrophic, periodic and deviating. The results show that EA-EMT resolved model incoherence and significantly outperformed the best currently available DBC solution by up to 95%.

1 Introduction

Model based control methodologies have found their expression in a wide range of AI techniques. From basic planning methods like STRIPs to complex PID controllers, the main principle remains the same: the decision on what action to take is based on a mathematical model of the environment response to an action application. However, in spite of being mathematically sound with provable properties, model based control methods suffer from one common pitfall. If the model is incoherent, that is a discrepancy exists between the actual reaction of the environment to an action application and the reaction described by the model, the decision made by a model based controller will be suboptimal. Now, a common approach to resolve this problem is model calibration: either through off-line or on-line interaction with the environment, the model is adjusted (or even entirely reconstructed) to reduce incoherence and facilitate better decision making.

Although model calibration has received increasing attention in recent years, the existing approaches make significant behavioural assumptions on the environment, ranging from frame assumptions [Pasula *et al.*, 2004] to structured motion of physical robots [Eliazar and Parr, 2004; Stronger and Stone, 2005]. Against this background, in this paper we relax these environment behavioural limitations by concentrating on a control framework with discrete abstract state models. More specifically, we concentrate on solving on-line model calibration for the Dynamics Based Control (DBC) framework [Rabinovich *et al.*, 2007], which allows application of the model based control methodology to an even wider range of domains (e.g. environments with abstract discrete state spaces and generic behaviour) than hitherto was possible.

In more detail, the DBC framework is almost unique in its ability to capture dynamic control tasks from the subjective point of view, i.e. in terms of agent's beliefs and observations of the changes that occur in the environment, and is based on two key principles. First, a part of the perceptual control paradigm [Powers, 1973], it states that changes in the environment are a means to altering and controlling perceptions. For instance, if we feel cold, we adjust temperature in a room to feel warmer, thus changing the environment to produce the required perception. Second, is that the dynamics of the system, rather than a momentary system state, are a means of describing the control task and modulation of environment dynamics are a means of solving the task. Notice, for example, that in our cold room example it was necessary to produce a change – increase the temperature – rather than bring it to a certain value. Combined together, these principles were implemented in a model based control algorithm termed *Extended Markov Tracking (EMT) control*. The algorithm has been shown to be an effective polynomial time solution to the DBC framework in discrete Markovian environments, where the next system state depends only on the current system state and the control action taken [Rabinovich *et al.*, 2007].

However, as with any model based control algorithm, EMT control is subject to deteriorating effects of model incoherence. In particular, our experiments further reveal that the standard EMT controller can not recover from persistent or catastrophic incoherence, where the environment behaves in a way not captured by the model. Nevertheless, EMT Control remains the sole solution within the DBC framework. Therefore, it is the only algorithm capable of operating in environments with a control task description that is both subjec-

tive and dynamic. The algorithm's polynomial running time underlines its importance even further, making its extension imperative. We thus modify the EMT control algorithm to include model calibration, which resolves the performance deterioration induced by a model incoherence.

The adaptive algorithm we have developed, the Ensemble Action EMT (EA-EMT), enables model calibration through the use of expert ensembles [Cesa-Bianchi and Lugosi, 2006]. For our purposes, each expert in the ensemble represents an alternative way to capture and model effects that an action has on the environment. EA-EMT dynamically merges the expert alternatives together, thus building a new environment model. Over time EA-EMT changes the properties of that merger, reflecting the performance of each expert in capturing environment behaviour, thus calibrating the environment model it uses.

The rest of the paper is organised as follows. In Section 2 we detail the operation of the standard EMT Control algorithm. Section 3 follows with the description of our new EA-EMT algorithm, detailing how it reconstructs and calibrates the environment model through the use of expert ensembles. Experimental support for the effectiveness of our approach is given in Section 4. The experiments take special focus on the on-line property of the EA-EMT model calibration, underlining the algorithm's ability to work in environments with changing behavioural trends. Section 5 summarises the results and gives future directions of this research.

2 EMT Control

The standard EMT algorithm continually maintains an estimate of system dynamics. To do so, the algorithm assumes that the system is an autonomous discrete Markov chain. That is, the system state stochastically develops over time without external influence, and the next system state depends on the current state only. This allows EMT to describe the estimate of the system dynamics by a single stochastic matrix. To maintain the estimate, the EMT algorithm performs a conservative update of the system dynamics matrix, minimising the Kullback-Leibler divergence between the new and the old estimate, with the limitation that the new estimate has to match the observed system transition that triggered the update.

To put it formally, assume that two probability distributions, p_t and p_{t+1}, are given that describe two consecutive states of knowledge about the system, and τ_t^{EMT} is the old estimate of the system dynamics. Then the EMT update τ_{t+1}^{EMT} is the solution of the following optimisation problem, where D_{KL} is the Kullback-Leibler divergence:

$$\begin{aligned} \tau_{t+1}^{EMT} &= \arg\min_{\tau} D_{KL}(\tau \times p_t \| \tau_t^{EMT} \times p_t) \\ \text{s.t. } p_{t+1}(x') &= \sum (\tau \times p_t)(x', x) \\ \text{and } p_t(x) &= \sum_{x'}^{x} (\tau \times p_t)(x', x) \end{aligned}$$

The update is abbreviated: $\tau_{t+1}^{EMT} = H\left[p_t \to p_{t+1}, \tau_t^{EMT}\right]$.

Although EMT can work with more general environmental descriptions (see e.g. [Adam *et al.*, 2008]), it has been more commonly used with a discrete Markovian environment with partial observability, described by a tuple $MEnv =< S, s_0, A, T, O, \Omega >$, where:

- S is the set of all possible environment states;
- s_0 is the initial state of the environment (which can also be viewed as a distribution over S);
- A is the set of all actions applicable in the environment;
- T is the environment's probabilistic transition function: a mapping $T : S \times A \to \Delta(S)$. That is, $T(s'|a, s)$ is the probability that the environment will move from state s to state s' under action a;
- O is the set of all possible observations. This is what the sensor input would look like for an outside observer;
- Ω is the observation probability function: a mapping $\Omega : S \times A \times S \to \Delta(O)$. That is, $\Omega(o|s', a, s)$ is the probability that o will be observed given that the environment moved from state s to state s' under action a.

This naturally connects with the EMT algorithm, as knowledge about the system is summarised by a distribution vector over the system states $p_t \in \Delta(S)$, in which case the system dynamics estimator created by EMT has the form of a conditional probability $\tau : S \to \Delta(S)$.

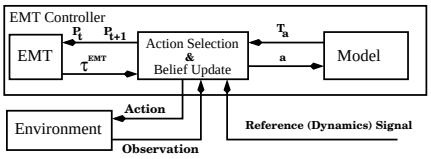


Figure 1: Closed loop of EMT Control

The overall control algorithm, termed *EMT Control*, forms a closed loop control with a reference signal [Stengel, 1994] (Figure 1 depicts the scheme). The reference signal encodes the task to be performed and takes the form of the conditional probability $\tau^* : S \to \Delta(S)$. The standard EMT Control (see Figure 2) can be summarised as a greedy one-step look ahead correction action selection. At every point in time, the algorithm attempts to predict the reaction of an estimation algorithm (EMT in this case) to the changes induced by an action (lines 2-7 of the algorithm), and then chooses the action that shifts the EMT estimator closest (line 8) to the reference dynamics τ^*. Once the action has been applied, the response of the EMT estimator to the changes in the environment is registered (line 11), and the control loops to make its next decision.

Notice that the controller action selection in lines 2-8 is heavily dependent on the environment model, as it uses the mapping T_a to predict action effects. If the model is incoherent the reaction of EMT can not be estimated correctly, which in turn will lead to selection of a suboptimal action. In what follows, we modify the action selection process to vary the environment model it uses.

3 Ensemble Action EMT

As already stated, the performance of the standard EMT Control algorithm deteriorates if the environment model is incoherent. However, by providing the algorithm with an additional method to correct model incoherences, it is possible to rectify the deterioration.

Require:
Set the system state estimator $p_0(s) = s_0 \in \Delta(S)$
Set the system dynamics estimator
$\tau_0^{EMT}(\bar{s}|s) = prior(\bar{s}|s)$
Set time to $t = 0$.
1: **loop**
2: **for all** $a \in \mathcal{A}$ **do**
3: Set $\bar{T}_a = T_a$ {use transition model T directly}
4: Set $\bar{p}^a_{t+1} = \bar{T}_a * p_t$
5: Set $D_a = H\left[p_t \rightarrow \bar{p}^a_{t+1}, \tau_t^{EMT}\right]$
6: Set $V(a) = \langle D_{KL}\left(D_a \| \tau^*\right)\rangle_{p_t}$
7: **end for**
8: Select $a^* = \arg\min_a V(a)$
9: Apply a^*, receive observation $o \in \mathcal{O}$
10: Compute p_{t+1} due to the Bayesian update:
$p_{t+1}(s) \propto \Omega(o|s,a) \sum_{s'} \bar{T}(s|a,s') p_t(s')$
11: Compute $\tau_{t+1}^{EMT} = H\left[p_t \rightarrow p_{t+1}, \tau_t^{EMT}\right]$
12: Set $t := t + 1$
13: **end loop**

Figure 2: EMT control algorithm.

Now, there are many incoherences a Markovian model, $MEnv =< S, s_0, A, T, O, \Omega >$, may have. While the choice of the state, action and observation spaces, as well as the observability function, may be dictated by subjective considerations (e.g. to make it more readable for the human domain designers), the transition function T is always dictated by the environment. Thus, in this work we choose to concentrate on the quality of the transition function T. This function maps actions into stochastic matrices, so that for each action $a \in A$ the matrix $T_a = T(\cdot|\cdot, a)$ models the effects of that action on the system state. The difference between the matrix T_a and the true effects of the action $a \in A$ is the incoherence type we have resolved in the EA-EMT algorithm (Figure 3). Thus, while the standard EMT Control views the transition mapping, $a \mapsto T_a$, to be constant, the EA-EMT algorithm modifies its transition mapping over time, reducing the mapping's incoherence. However, before we go into the details of how it was implemented, we would like to explain the principles of the approach taken by EA-EMT.

EA-EMT assumes that, although the mapping $T : A \rightarrow \Delta(S)^S$ is incoherent, the set of matrices $T_A = \{T_a = T(\cdot|\cdot, a)\}_{a \in A}$ represents feasible effects that the actions may have. The algorithm then attempts to assemble a better mapping, $\bar{T} : A \rightarrow \Delta(S)^S$, based on the set T_A. More specifically, for each action $a \in A$ the transition matrix $\bar{T}_a$ is a weighted linear combination of matrices in the set T_A, that is $\bar{T}_a = \sum_{b \in A} T_b * w_a(b)$. Intuitively, the weight $w_a(b)$ represents the similarity between the matrix $T_b \in T_A$ and the effects that the action $a \in A$ has on on the environment state. As the interaction between the EA-EMT algorithm and the environment progresses, the weights $w_a(\cdot)$ are updated, modifying the mapping $\bar{T} : A \rightarrow \Delta(S)^S$ to reduce its incoherence with the environment.

The update of the weights $w_a(\cdot)$ is based on the approach of predictions with expert ensembles [Cesa-Bianchi and Lugosi, 2006]. The intuition behind this approach is that, when making a prediction or a decision, a readily available set of feasible alternatives (the expert ensemble) can be merged together to form a prediction which is potentially better than any of the alternatives standing alone. In our case the *expert ensemble* is the set T_A, where each expert attempts to predict the effects an action would have on the environment state. From this point of view, the weight $w_a(b)$ expresses how much the expert $T_b \in T_A$ is trusted to capture the effects of the action $a \in A$ correctly. Once EA-EMT has applied an action, a^*, it measures the discrepancy between the effect a^* had and the effect predicted by expert T_b. The lower the discrepancy, the higher will be the weight $w_{a^*}(b)$ when the next control decision is made.

Require:
Set the system state estimator $p_0(s) = s_0 \in \Delta(S)$
Set the system dynamics estimator
$\tau_0^{EMT}(\bar{s}|s) = prior(\bar{s}|s)$
Set action weight vectors $w_a(a') \propto \delta_a(a') + \epsilon$
Set time to $t = 0$.
1: **loop**
2: **for all** $a \in \mathcal{A}$ **do**
3: Set $\bar{T}_a = \sum_{a'} T_{a'} * w_a(a')$
4: Set $\bar{p}^a_{t+1} = \bar{T}_a * p_t$
5: Set $D_a = H\left[p_t \rightarrow \bar{p}^a_{t+1}, \tau_t^{EMT}\right]$
6: Set $V(a) = \langle D_{KL}\left(D_a \| \tau^*\right)\rangle_{p_t}$
7: **end for**
8: Select $a^* = \arg\min_a V(a)$
9: Apply a^*, receive observation $o \in \mathcal{O}$
10: Compute p_{t+1} due to the Bayesian update:
$p_{t+1}(s) \propto \Omega(o|s,a) \sum_{s'} \bar{T}_a(s|s') p_t(s')$
11: Compute $\tau_{t+1}^{EMT} = H\left[p_t \rightarrow p_{t+1}, \tau_t^{EMT}\right]$
12: **for all** $a \in \mathcal{A}$ **do**
13: Set $\bar{p}^a_{t+1} = T_a * p_t$
14: Set $D_a = H\left[p_t \rightarrow \bar{p}^a_{t+1}, \tau_t^{EMT}\right]$
15: Set $V(a) = \left\langle D_{KL}\left(D_a \| \tau^{EMT}_{t+1}\right)\right\rangle_{p_t}$
16: Set $w_{a^*}(a) \propto w_{a^*}(a)\beta^{V(a)}$
17: **end for**
18: Set $t := t + 1$
19: **end loop**

Figure 3: The EA-EMT control algorithm.

Given the above principles, we have modified the standard controller algorithm. Specifically, line 3, previously directly substituted into the calculations the transition function from the provided model. Whereas now it uses a weighted combination of the matrices in T_A. The rest of the computations proceed as before until the EMT estimate, τ_{t+1}^{EMT}, of the action outcome is computed in line 11: the algorithm predicts the effects of each action on the EMT estimate, chooses the action that would bring τ_{t+1}^{EMT} closest to the reference signal τ^*, applies the action and receives an observation. At that point, the algorithm has to measure the performance of each expert, and update the weights. Now, recall that the algorithm operates in terms of subjective beliefs, the relevant effects of the action are thus those expressed in the EMT estimate τ_{t+1}^{EMT}. This means that the performance of each expert

can be expressed by the distance between the estimate τ_{t+1}^{EMT} and the estimate that would have been obtained based on the expert prediction. This distance is computed in lines 13-15, and the weight of the expert is updated accordingly. Specifically, the old weight of the expert is multiplied by β^d, where $\beta \in (0,1)$ is the parameter of the update and d is the distance above. Once all weights are updated, they are normalised to sum to 1, so that $\bar{T}_a$ at the next step will be a stochastic matrix. Notice that all these operations take time polynomial in the model parameters, such as the size of state, action and observation spaces.

4 Experimental Evaluation

To test the effectiveness of the EA-EMT algorithm, we have devised a set of comparative tests with the standard EMT Controller. To support comparability with previous work in this area, all tests were based on modifications of the Drunk Man (D-Man) domain: a controlled random walk over a linear graph (see Figure 4 for the principle structure) with actions weakly modulating the probability (only a small discrete set of probabilities in the range $(\epsilon, 1-\epsilon)$ with $\epsilon \gg 0$ is attainable) of the left and the right steps. A task within the domain is represented by a conditional probability $\tau^*(s'|s)$, the reference signal for the controller, specifying what sort of motion through the state space has to be induced. During an experiment run, the control algorithm was provided with a Markovian environment model, $MEnv =< S, s_0, A, T, O, \Omega >$, incoherent with the true behaviour of domain. The incoherences were created by introducing exogenous perturbations to the behaviour of the D-Man domain. In particular, three perturbations, making the model of the standard D-Man domain increasingly incoherent with the actual environment behaviour, were used:

- **Deviating**. An additional deterministic step (to the right) was done.
- **Periodic**. An additional deterministic step was done, but its direction changed with time.
- **Catastrophic**. A random permutation of actions was selected $\sigma : A \to A$. When the controller applied action $a \in A$, the environment responded instead to $\sigma(a)$.

Three baselines where obtained in various combinations: standard EMT Control algorithm operating in a perturbed environment, standard EMT Control operating within an unperturbed environment, and standard EMT Control operating in a perturbed environment with its model correctly encoding the environment perturbation. At least two baselines are present in each experimental setting to provide comparative performance bounds. Unless specified otherwise, the confidence envelope of 99.5% is depicted in all data graphs.

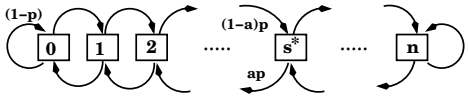


Figure 4: Principle structure of the Drunk Man domain.

In all our experiments the reference dynamics for the controller is given by $\tau^*(s'|s) \propto \delta_{s^*}(s') + \epsilon$, where $\epsilon > 0$ is small. In other words, the target prescribes that the environment should almost surely move to the ideal state s^* from any other state. In our experiments the state space was $S = \{0, ..., 12\}$, and the ideal state $s^* = 6$.

Notice that, due to the probabilistic nature of the domain, any reasonable[1] control scheme set to accomplish the task would result in a bell shaped empirical distribution of the system state. Success of the control scheme can then be readily appreciated visually by the difference of the expected value and the ideal system state, as well as the standard deviation of the empirical state distribution. To present an overall evaluation of a control scheme's performance, rather than a comparison of multiple parameters, we also measured the distance between the empirical distribution and δ_{s^*} using l_1 norm.

4.1 Deviating Perturbation

In this experiment we introduce a deviating perturbation. That is, beyond the usual probabilistic step, the environment has also deterministically shifted in one direction along the linear graph. For example (referring to Figure 4) if the system reached state $k \in \{0, ..., n-1\}$, the additional step will shift it to state $k+1$.

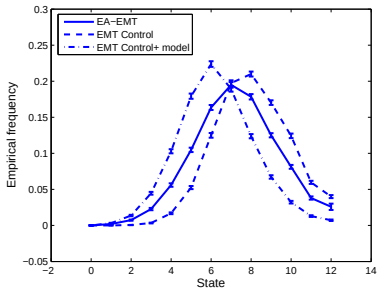
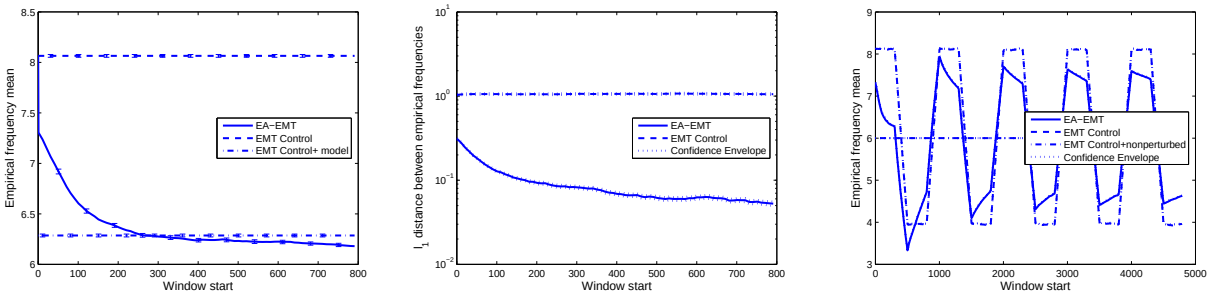


Figure 5: EA-EMT performance under persistent shift

In this context, Figure 5 shows the empirical distribution of system states under three control strategies: the EA-EMT controller and the standard EMT Controller equipped with the standard D-Man model (thus excluding the shift modelling), and the standard EMT Controller equipped with the environment model that explicitly captures the additional shift. The figure shows the complete empirical distribution of the EA-EMT obtained during the first 200 control choices made in this experiment, and marks a definitive improvement in performance. This can be seen from the fact that the standard EMT Control fails to enforce the reference dynamics τ^*, with the system spending the majority of its time away from the ideal state, $s^* = 6$, while EA-EMT manages to force the state distribution to concentrate closer to s^*. In fact, the distance between δ_{s^*} and the EA-EMT distribution induced in

[1]Unreasonable, for instance, would be choosing a constant action to equalise the left and the right step probabilities, as this would result in an almost uniform distribution, utterly defeating the controller purpose.



(a) (b) Notice the log-scale of the Y axis. (c)

Figure 6: EA-EMT adaptation to various perturbations: (a) Persistent Shift, (b) Random Permutation, (c) Switching Shift

the first 200 steps is 40% less than the comparable distance for the EMT controller. This, however, does not fully reflect the adaptability of EA-EMT. To this end, Figure 6(a) shows how the mean of the empirical distributions of the 200 step windows behave. The distributions induced by EMT Control do not change over time, resulting in straight horizontal lines depicting the constancy of the mean. On the other hand, the data shows that EA-EMT quickly adapts, the algorithm induces the empirical state distribution with the mean approaching the ideal state $s^* = 6$. In this respect, EA-EMT even slightly surpasses the performance of the standard EMT algorithm with the correct environment model. This is due to the adaptive portion of EA-EMT contributing to the tie breaking when considering similar actions – this tie breaking is rigid in EMT Control. Similar pictures occur with respect to the variance of the empirical distributions. This means that EA-EMT overcomes the model incoherence and increasingly concentrates the state empirical distribution around the ideal state, which is exactly what the reference dynamics, τ^*, requires.

4.2 Catastrophic Perturbation

The action space of the D-Man domain has a simple intuitive interpretation – the action sets how quickly the system state will shift left or right. The deviating perturbation did not exceed this interpretation, it simply meant that the system will naturally move in one direction faster than the other. In a way it also meant that the perturbation induced a very mild model incoherence – principally the model remained correct. However, EA-EMT can adapt to much more severe model incoherences. In fact, in the next set of experiments the environment model is completely incorrect. For each run in this experiment set a random permutation $\sigma : A \to A$ was selected. Then, when action $a \in A$ was applied, the environment reacted as if the action was $\sigma(a)$.

In more depth, Figure 7 shows the empirical distributions obtained in the first 200 steps of decision making. Permuting the action breaks any connection between what EMT Control expects the action to do and what actually occurs in the environment, essentially the actions are scrambled and the EMT Control chooses a random action. This results in the failure of the algorithm – the empirical state distribution is equivalent to that of applying no control at all[2]. In contrast, EA-EMT easily adapts to scrambling and performs increasingly well, as can be seen in Figure 6(b). Following the development of the empirical distribution within a sliding 200 step window, the figure shows the l_1 norm distance from the distribution formed by the standard EMT algorithm in the non-perturbed environment. This data demonstrates that EA-EMT exponentially quickly discovers the true effects of actions and approaches the performance of the EMT control in a non-perturbed environment. Even though the empirical distribution of the first 200 steps includes the first decisions made based on the scrambled model, it already recovers 70% of the performance lost due to the model incoherence and, through further adaptation, it reaches 95% recovery.

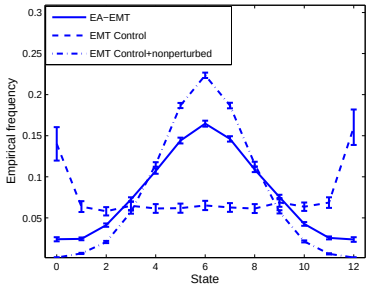


Figure 7: EA-EMT performance under random permutations.

4.3 Periodic Perturbation

Finally, it is important to evaluate whether the algorithm can also perform well in a dynamically changing environment. For example in robotics, even if everything else remains the same, the robot body will behave differently over time due to natural wear-and-tear. To test EA-EMT in such environments, we consider yet another perturbation: an additional deterministic step is made, and the direction of the step switches between left and right with constant period (500

[2]Since left and right steps fail in respective terminal states, the empirical probability there is higher.

control steps in our experiments). The shape of the distributions formed by the controllers are equivalent to those in the persistent shift experiment (see Figure 5), and we omit the respective graph. On the other hand, the development of the empirical distribution over time is quite different. In particular, Figure 6(c) shows the behaviour of the mean value for empirical distributions calculated within a 200 step sliding window. While the standard algorithm literally switches from one value to another, depending on the direction of the shift, the performance of EA-EMT always shows recovery after a direction switch occurs. Notice also, that the magnitude of the mean variation at the switch point becomes significantly (25%) less for EA-EMT than the standard EMT. This suggests that, beyond its ability to recover from irrelevant adaptations, the adaptive controller version learns to reduce the control inertia. In other words the algorithm reduces the impact of the sudden change in the environment behaviour, making the overall performance more stable.

5 Conclusions and Future Work

In this paper we present the Ensemble Action EMT algorithm – a polynomial time solution to the Dynamics Based Control framework with capabilities of on-line adaptation to environment model incoherences. The EA-EMT algorithm, uses the transition matrices contained within the model as an expert library for the feasible action effects in the environment, and treats any possible action as an ensemble predictor based on this experts set. Following the relevance of experts to the exhibited effects an action has on the EMT dynamics estimate, the weights within each ensemble action predictor are updated. We have experimentally verified the efficiency of the EA-EMT algorithm and shown that it quickly adapts to deviating, periodic and catastrophic exogenous perturbations of the environment. Furthermore, the data from the periodic perturbation experiment suggests reduced control inertia.

These adaptive capabilities of the EA-EMT algorithm allow a wider range of problems to be solved within the DBC framework than could hitherto be addressed. For example, previously a precise environment model was required to solve a type of search problem called area sweeping [Rabinovich *et al.*, 2007]. In contrast, the EA-EMT algorithm is capable of operating with environment behaviour described by a set of dynamic primitives that may occur in response to an action. This allows the algorithm to be applied in environments whose behaviour is only partially known, making precise modelling impossible.

Speaking more generally, the use of the expert ensemble method has a close association with plan recognition techniques [Bui, 2003; Riley and Veloso, 2002; Pynadath and Wellman, 2000], where a library (ensemble) of potential plans is commonly used. Plan recognition algorithms, through observation and causal interpretation of events, select a most likely explanation from the library. Similarly, EA-EMT views expert alternatives as explanations to changes in the environment state (which are not unlike the behaviours of plans in a library). Specifically, EA-EMT evaluates the performance of each expert based on the observed effects of an action within the environment. This parallel opens up the possibility of using EA-EMT as an opponent recognition and classification method in multi-agent adversarial scenarios.

However, EA-EMT also fuses and arbitrates between the various expert predictions. Specifically, the action model is essentially a weighted combination of the expert alternatives, which links it to behaviour-based robotics (BBR). In BBR [Arkin, 1998] a collection of simple (usually reactive) control algorithms is fused by an arbitration mechanism that combines their control signal. Given that this arbitration can include mutual inhibition or activation of other control signals, the resulting system can exhibit complex, adaptive behaviour (see e.g. [Mataric, 1998; Buffet *et al.*, 2002] and references therein) through the modulation and adaptation of the arbitration process itself. Ideologically similar adaptation occurs in EA-EMT, where individual experts gain higher weight in assembling the model with respect to their performance in predicting the effects of an action. Thus we plan to exploit this connection to construct new hybrid control schemas that combine the subjective dynamics task specification of EMT control with the structural task decomposition of BBR.

Finally, we also would like to investigate the possibility of altering the weight adaptation to include *forgetting*. That is, over time the weights should have an inherent tendency to equalise. By so doing, the controller could possibly produce higher frequency adaptability, enabling even better response to periodic perturbations.

References

[Adam *et al.*, 2008] A. Adam, Z. Rabinovich, and J. S. Rosenschein. Dynamics based control with psrs. In *7th AAMAS*, pages 387–394, 2008.

[Arkin, 1998] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[Buffet *et al.*, 2002] O. Buffet, A. Dutech, and F. Charpillet. Learning to weigh basic behaviors in scalable agents. In *1st AAMAS*, volume 3, pages 1264–1265, 2002.

[Bui, 2003] H. Bui. A general model for online probabilistic plan recognition. In *18th IJCAI*, pages 1309–1315, 2003.

[Cesa-Bianchi and Lugosi, 2006] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[Eliazar and Parr, 2004] A. I. Eliazar and R. Parr. Learning probabilistic motion models for mobile robots. In *21st ICML*, pages 32–??, 2004.

[Mataric, 1998] M. J. Mataric. Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. *Trends in Cognitive Sciences*, 2(3):82–86, 1998.

[Pasula *et al.*, 2004] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning probabilistic relational planning rules. In *14th ICAPS*, pages 73–82, 2004.

[Powers, 1973] William T. Powers. *Behavior: The control of perception*. Aldine de Gruyter, Chicago, 1973.

[Pynadath and Wellman, 2000] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *16th UAI*, pages 507–514, 2000.

[Rabinovich *et al.*, 2007] Z. Rabinovich, J. S. Rosenschein, and G. A. Kaminka. Dynamics based control with an application to area-sweeping problems. In *6th AAMAS*, pages 785–792, 2007.

[Riley and Veloso, 2002] P. Riley and M. Veloso. Recognizing probabilistic opponent movement models. In *The 5th RoboCup Competitions and Conferences*. 2002.

[Stengel, 1994] R. F. Stengel. *Optimal Control and Estimation*. Dover Publications, 1994.

[Stronger and Stone, 2005] D. Stronger and P. Stone. Simultaneious calibration of action and sensor models in a mobile robot. In *Proceedings of the ICRA*, 2005.