Proceedings of the IJCAI-05 Workshop on Agents in Real-Time and Dynamic Environments

Ubbo Visser
Gerhard Lakemeyer
George Vachtesevanos
Manuela Veloso
(editors)

Held on July 30, 2005 in conjunction with the
International Joint Conference on Artificial Intelligence
Edinburgh, Scotland

.

IJCAI-05 Workshop on Agents in Real-Time and Dynamic Environments

Held on July 30, 2005 in conjunction with the International Joint Conference on Artificial Intelligence Edinburgh, Scotland

Recent developments in multiagent systems (MAS) have been promising by achieving autonomous, collaborative behavior between agents in various environments. However, most of the agents, both software agents and physical agents, still have problems if the environment is dynamic and the agents have to act in real time. Examples are obstacle avoidance with moving obstacles or world models which are composed from egocentric views of numerous agents. Another aspect is the need for quick responses. In an environment where a number of agents build a team and both single agent decisions and team collaborative decisions have to be made methods have to be fast and precise. This workshop addresses various problems that occur with respect to these issues.

The main focus of this workshop deals with methods from various areas such as world modeling, planning, learning, and communicating with agents in real-time and dynamic environments. Within this general theme we aim to bring together researchers to discuss the following topics:

- World modeling (quantitative, qualitative)
- Coaching (one agent gives advice to a group of agents)
- Planning with resources (especially time)
- Learning (both offline and online)
- Cooperation between agents (robot and/or humans)
- Communication between agents (implicit, non-verbal, or verbal one)
- Real-time systems software issues (often ignored but important if serious about real-time issues in robotics)
- Scalability and robotics interfacing issues (demands a great deal of support from the initial design of the system)

In the last decade, a lot of effort has been invested to develop methods that can be used with multi-agent systems. The language development in the area of communication between agents (ACL) might act as the first example. Speech acts serve as the basic principle and various protocols have been invented (e.g. auctions, contract-nets, etc.). Can we transfer these results to environments where quick decisions have to be made? Consider planning as another example: there are promising methods for path planning, but do they still hold if the observed obstacles are moving? Learning is another example: we need on-line learning in a real-time scenario to give agents the option to learn more about their environment. Usually, learning takes a fair amount of time but sometimes this time is not available. Can we find methods which will consider these restrictions?

This workshop addresses researchers from various areas in AI who want to discuss the mentioned issues from their point of view. How can we develop new methods or adapt existing methods to meet these demands?

13 contributions have been selected for oral presentation at the workshop. These proceedings contain all papers, which can be roughly categorized with the help of the following sketch of a general multiagent architecture. Please note that this is only a rough categorization and that there are a number of papers that belong to more than one component.

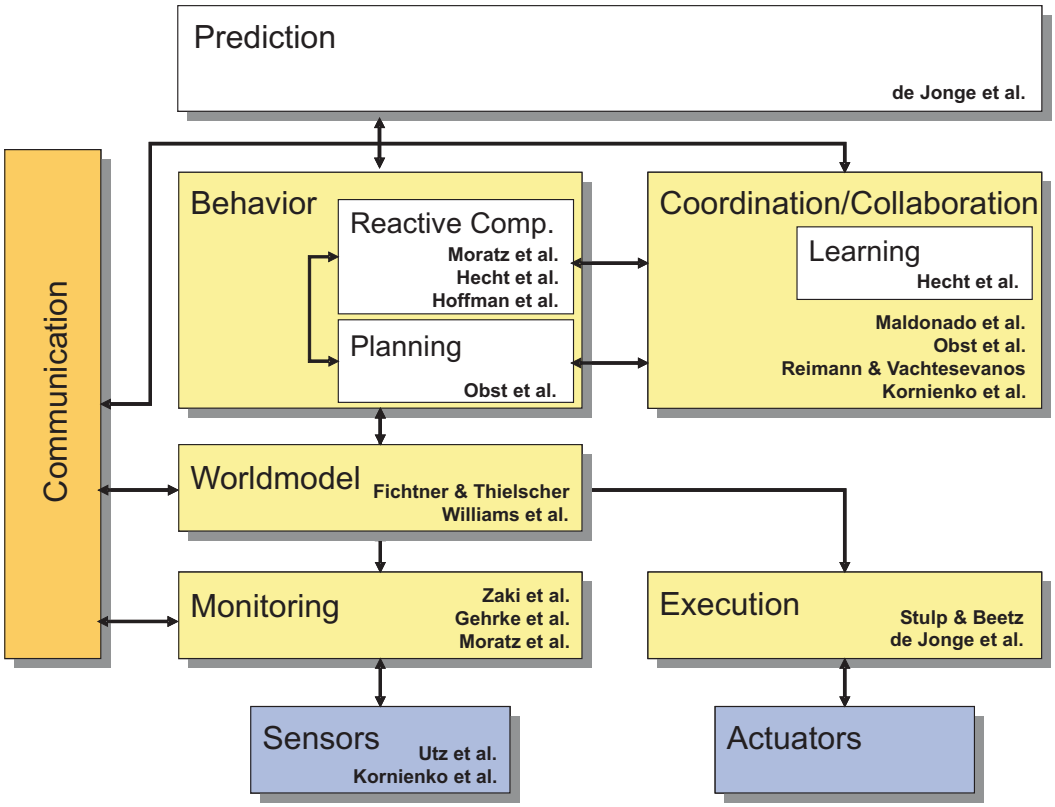


Figure 1: A general multiagent systems architecture. The papers are roughly categorized

Two papers can be categorized into the **world modeling** component. *Fichtner & Thielscher* discuss the importance of establishing and maintaining the correspondence of high-level symbols or concepts and the actual sensor signals. Their approach is based on the Fluent calculus and allows to anchor symbols to percepts using definite and indefinite references. Among other advantages, it also supports reasoning over object properties during planning. Anchoring is also the topic that is the focus of the contribution by *Williams et al.*. They provide a framework for evaluating groundedness of representations in systems. This framework provides means to measure how well a system is grounded, i.e. how well the high-level symbol correspond to the entities they represent.

Two papers are closely related to sensor data and thus the **sensor** component. *Utz et al.* state that a robot vision architecture needs to encapsulate the constraints of the application domain in order to keep a vision application flexible. Their approach introduces a video image processing (VIP) framework for multi threaded control flow modeling in robot vision. The authors discuss the VIP design and implementation as well as an experimental evaluation of its performance in parallel image processing tasks. *Kornienko et al.* present research results in the field of perception for a real micro robotic swarm. The proposed hardware and software solution uses IR-based reflective measurements for individual perception and a Dampster-Shafer evidential reasoning method for hypothesis refinement in collective perception. The authors focus their paper to a reliable identification of encountered geometries and to a reduction of local communication. This paper can also be categorized in the **coordination/collaboration** component.

Three papers can be categorized into the **monitoring** component. *Zaki et al.* demonstrate an approach that diagnoses faults in an analogue electrical circuit. This kind of circuit is an example for a dynamic continuous non-linear and time invariant system. The paper shows how the type of system has an impact on the choice of the modeling techniques. *Gehrke et al.* propose a vocabulary for a qualitative representation for a traffic scenario. The authors present and evaluate a prototype that does the qualitative abstraction for knowledge-based behavior control. *Moratz et al.* propose a relative orientation algebra which features an adjustable granularity. The key idea behind this approach is to find an appropriate granularity for an applied calculus in order to reason about space for orientation. It turned out that their approach is feasible for robots in real-time conditions as a reactive component when deploying constraint-based reasoning methods. The granularity of the calculus allows to select only relevant feature changes in dynamic environments. The paper also fits into the **reactive component**.

Four contributions touch the areas of behavior modeling. Behavior modeling (for a single agent) is divided into **reactive components** and **planning**. *Hoffmann et al.* discuss how negative information such as the absence of expected sensor signals and proprioception (a quality measure for an actual odometry comparing target and current robot joints) can be used to localize robots. They incorporate negative information into a known Monte-Carlo-Localization method and do fine-tuning using the proprioceptive information. The contribution of *Hecht et al.* focuses on a learning task that aims to train robots in a team using static game situations in diagrams that has been drawn by a human coach. The network learns to generalize and give advice for the best option for a player. The authors claim that the method improved their control method for five small size robots within the RoboCup domain significantly and that the method can be adopted to various other domains. The outcome of this approach has been successfully used in robot soccer games at the RoboCup German Open 2005. This paper also fits into the **coordination/collaboration** component. *Obst et al.* propose an approach that coordinates the behavior of a multiagent team using an HTN planning procedure. They formalized domain knowledge (the RoboCup domain in the experiments) and used this within the planning methods in order to subdivide the given tasks. The hierarchical structure helps to speed up the planning task significantly. This paper also fits into the **coordination/collaboration** component. The fourth paper is from *Moratz et al.* that has already been discussed above.

Among the two mentioned papers that also deal with team behavior or **coordination/collaboration**, *Reimann & Vachtesevanos* propose a game-theoretic approach to solve a differential pursuit-evasion game which involves multiple pursuers and a single evader. Their idea consists of a computational algorithm which is developed to determine a suboptimal control strategy that a swarm of pursuers can use to intercept a single evader. The authors use a solution that is based on simulated annealing to reduce the complexity of the task and show in an experiment that their approach is feasible. The contribution of *Maldonado et al.* focuses on using auctions for real-time scheduling. They implemented their methods in the ARTIS (Architecture for Real-Time Intelligent Systems) agent architecture and conclude with promising results.

Two contributions can be categorized into the **execution** component. *Stulp & Beetz* propose a novel computation model for the execution of abstract action chains. A robot first learns situation-specific performance models of abstract actions and then uses these models to automatically specialize the abstract actions for their execution in a given action chain. The authors state that this specialization results in refined chains that are optimized for performance. The central idea behind this approach is that actions can be tailored to the task context by adapting action parameters. *de Jonge et al.* present an approach that enables agents to control their plan execution health and to regain health if necessary. The agents can utilize the model to predict consequences of occurring disruptions and thus detect unhealthy situations. With the help of the models predictions,

agents can correct the execution of tasks within the plan in such a way that conflicts will be avoided and health is regained. This paper can also be categorized in the **prediction** component.

Program Committee and Reviewers

We are grateful to the following members of the international program committee for helping us to make this a IJCAI-05 workshop:

- Hans-Dieter Burkhard, Berlin, Germany
- Uli Furbach, Koblenz, Germany
- Dieter Fox, Seattle, USA
- Joachim Hammer, Gainesville, USA
- Gal Kaminka, Bar Ilan, Israel
- Gerhard Lakemeyer, Aachen, Germany
- James Lawton, Rome, NY, USA
- Paul Levi, Stuttgart, Germany
- John-Jules Meyer, Utrecht, NL
- Elena Messina, Gaithersburg, USA
- Daniele Nardi, Rome, Italy
- Bernhard Nebel, Freiburg, Germany
- Frank Pasemann, St. Augustin, Germany
- Angel del Pobil, Jaume I, Spain
- Patrick Riley, Pittsburgh, USA
- Peter Stone, Austin, USA
- Fiora Pirri, La Sapienza, Rome, Italy
- Alessandro Saffiotti, Örebro, Sweden
- Michael Thielscher, Dresden, Germany
- George Vachtesevanos, Atlanta, USA
- Manuela Veloso, Pittsburgh, USA
- Ubbo Visser, Bremen, Germany
- Brian Williams, Boston, USA

The organizers would also like to thank the additional reviewers Jan Murray, Oliver Obst, John Reimann, Thorsten Scholz, and Ingo Timm.

.

Contents

Anchoring Symbols to Percepts in the Fluent Calculus

Matthias Fichtner **Michael Thielscher**

Department of Computer Science
Technische Universität Dresden
Dresden, Germany

Abstract

For an autonomous agent operating in a real, dynamic environment it is crucial to assure that symbols and signal-level models refer to the same physical object. The problem of establishing and maintaining the correspondence between a high-level symbol and sensor data is called the anchoring problem. Here we present preliminary results on an approach to the anchoring problem based on the Fluent Calculus. While properties of objects are used to distinguish between different objects, reasoning about knowledge supports to identify the particular object in mind. An example shows how the approach can deal with multiple hypotheses for correspondences.

1 Introduction and Related Work

Modern control architectures for autonomous robots often comprise a signal level and a symbolic level. Although providing a rich expressiveness, knowledge representation at the symbolic level causes a severe problem: A symbol that refers to a real-world entity lacks direct, unmediated connection and hence its reference may be wrong. *Anchoring* of symbols to percepts aims at providing this link and is defined as "the process of creating and maintaining the correspondence between symbols and sensor data that refer to the same physical objects [Coradeschi and Saffiotti, 2003]." The anchoring problem denotes how to perform anchoring in an autonomous artificial system. It is a special case of the symbol-grounding problem: "How is symbol meaning to be grounded in something other than just more meaningless symbols [Harnad, 1990]?"

As a matter of fact, any cognitive system facing the real world somehow has to solve the anchoring problem. Most systems implement an ad-hoc approach to the anchoring problem which is hidden in the implementation. Only since recently explicit approaches arise.

We claim that object recognition alone is insufficient to solve the anchoring problem, because it only helps to *distinguish* between different (kinds of) objects. Besides discriminating, cognitive agents need to describe, manipulate and most importantly to *identify* objects of interest in real environments. Moreover, without anchoring symbols to percepts autonomous agents like mobile robots cannot cope with dynamic worlds in which objects may appear, move and disappear. In such environments, the agent has to face uncertainty in action execution and significant noise involved in recognition. The agent might know little about an object's properties, too.

In our approach, both sources of valuable information are used, top-level knowledge and signal-level recognition. Aiming at robust anchoring, correspondences between symbols and percepts are maintained in both directions, top-down and bottom-up: On the one side, knowledge and constraints about object properties are used in an expressive reasoning system; on the other side, an object recognition and tracking system extracts useful perceptive information describing real-world objects.

In [Coradeschi and Saffiotti, 2000] one of the first formal approaches to the anchoring problem was described. There, correspondences between symbols and percepts of objects are based on the object's properties. Given a description of the desired object symbol, a correspondence is considered possible if each predicate of the description matches the perceived values. This basic principle is common to most formal approaches to anchoring including our's.

An important aspect of the anchoring approach in general is to be able to use definite as well as indefinite references, as has been suggested in [Coradeschi and Saffiotti, 2000] besides others. Definite and indefinite references are integral aspects of our approach.

In [Coradeschi and Saffiotti, 2002] it has been recognised that perceptive ambiguity requires to maintain multiple hypotheses of anchoring symbols to percepts. As opposed to [Coradeschi and Saffiotti, 2000] and [Chella *et al.*, 2004] which lack this functionality, our anchoring approach maintains multiple possible hypotheses.

[Saffiotti, 1994] proposed to use context-dependent information for anchoring objects like "the large one". Being based on the Fluent Calculus, our approach to anchoring symbols can make use of the expressivness of reasoning about knowledge. In this way, much more than context-dependent information alone can be exploited for anchoring.

The document is organised as follows. In Section 2 we will introduce the basic notions of the Fluent Calculus, which is the underlying theory of this work. Next, Section 3 describes our approach on anchoring in detail. The example in Sec-

tion 4 is meant to illustrate it. A comparison between our and other's approaches as well as future work can be found in Section 5. We conclude in Section 6.

2 Preliminaries: The Fluent Calculus

The Fluent Calculus is an axiomatic theory of actions that provides the formal underpinnings for agents to reason about their actions [Thielscher, 1999]. Formally, it is a many-sorted predicate logic language which includes the two standard sorts FLUENT (i.e., an atomic state property) and STATE. States are built up from fluents (as atomic states) and their conjunction, using the binary function $\circ : \text{STATE} \times \text{STATE} \mapsto \text{STATE}$ along with the constant $\emptyset : \text{STATE}$ denoting the empty state.[1]

A fundamental notion is that of a fluent f to *hold* in a state z. For notational convenience, the macro $\mathit{Holds}(f, z)$ serves as an abbreviation for an equational formula which says that z can be decomposed into f and some state z':

$$\mathit{Holds}(f, z) \stackrel{\text{def}}{=} (\exists z')\, z = f \circ z'$$

This definition is accompanied by the following foundational axioms of the Fluent Calculus, which ensure that a state can be identified with the fluents that hold in it.

Definition 1 Assume a signature which includes the sorts FLUENT and STATE such that FLUENT is a sub-sort of STATE, along with the functions $\circ, \emptyset$ of sorts as above. The foundational axioms Σ_{state} of the Fluent Calculus are[2]

1. Associativity and commutativity,

$$\begin{aligned} (z_1 \circ z_2) \circ z_3 &= z_1 \circ (z_2 \circ z_3) \\ z_1 \circ z_2 &= z_2 \circ z_1 \end{aligned} \tag{1}$$

2. Empty state axiom,

$$\neg \mathit{Holds}(f, \emptyset) \tag{2}$$

3. Irreducibility and decomposition,

$$\mathit{Holds}(f_1, f) \supset f_1 = f \tag{3}$$

$$\mathit{Holds}(f, z_1 \circ z_2) \supset \mathit{Holds}(f, z_1) \vee \mathit{Holds}(f, z_2) \tag{4}$$

Axioms (1)–(4) essentially characterize "$\circ$" as the union operation with $\emptyset$ as the empty set of fluents. Associativity allows us to omit parentheses in nested applications of "$\circ$".

The Fluent Calculus employs the standard sorts ACTION and SITUATION (i.e., sequence of actions) as in the Situation Calculus [McCarthy, 1963]. The initial situation is denoted by the constant S_0, and $\mathit{Do}(a, s)$ denotes the situation reached by performing action a in situation s. As a denotation for the state in situation s, the pre-defined function $\mathit{State}(s)$ allows to define what it means for a fluent to hold in a situation thus:

$$\mathit{Holds}(f, s) \stackrel{\text{def}}{=} \mathit{Holds}(f, \mathit{State}(s))$$

[1] Throughout this paper, the function "$\circ$" is written in infix notation. Free variables in formulas are assumed universally quantified. Variables of sorts FLUENT and STATE are denoted by the letters f and z, respectively.

[2] The full axiomatic foundation of the Fluent Calculus contains two further axioms [Thielscher, 1999].

Representing State Knowledge

The knowledge that an agent has of its environment can be represented in the Fluent Calculus via the notion of *possible states* [Thielscher, 2000]. The predicate $\mathit{KState}(s, z)$ has the intended meaning that, according to the knowledge of the agent, z is a possible state in situation s.

3 Anchoring in the Fluent Calculus

Our approach to the anchoring problem is based on perceptive as well as symbolic descriptions of properties of objects in the world. Such a perceptive description can be obtained from an object recognition and tracking system by extracting a number of distinctive features from each percept.

High-level knowledge of an object's properties as well as certain conditions to be met by some of its properties form a symbolic description. Regarding object properties, a number of perceptive features (attributes) is useful. To this end, the set $\mathbf{a}$ specifies essential attributes for a particular object category o by means of predicate $\mathit{Attributes}(o, \mathbf{a})$. For instance,

$$\mathit{Attributes}(\mathit{Cup}, \{\mathit{Colour}, \mathit{Location}, \mathit{Width}, \mathit{Height}\}) \tag{5}$$

denotes perceptive attributes of objects of the category *Cup*.

3.1 Representation

Numerical estimates maintain the current knowledge about the object's properties. The more precise the estimate becomes, the better it allows to distinguish the object at hand from other objects. For each attribute $a \in \mathbf{a}$, the estimate $\gamma = (a, (b_1, b_2))$ represents the possible range of values in the bounds $[b_1, b_2)$, while its extent shows uncertainty in this attribute. By this, uncertainty in the sensor model is *explicitly* represented, while its representation is still very concise.

Following previous formal approaches on symbol anchoring, a structure called *anchor* is used to represent information about a symbol-percept correspondence [Coradeschi and Saffiotti, 2000]. To this end, the Fluent Calculus signature is extended by the fluent $\mathit{Anchor}(x, \Gamma, o, \pi)$. It comprises symbol x, estimate Γ, object category o, and percept ID π. We use "$\perp$" to denote the symbol argument of an anchor if the anchor is not associated with an object symbol, and to denote the percept argument of an anchor if the anchor is not associated with a valid percept.

At any time, the meaning of fluent $\mathit{Anchor}(x, \Gamma, o, \pi)$ wrt. the correspondence between symbol x and percept π takes one of the following forms:

1. If both x and π are defined, i.e. $x \neq \perp \wedge \pi \neq \perp$, then a complete correspondence between symbol x and percept π has been established.
2. If x is defined but π is not, i.e. $x \neq \perp \wedge \pi = \perp$, then no appropriate percept has been identified and assigned to symbol x yet.[3]
3. If $x = \perp$ and $\pi \neq \perp$, then the corresponding symbol for percept π has not yet been found.

[3] We use $\perp$ to say that something is not defined.

We define and maintain an anchor only for those symbols that denote objects of interest.

The following constraints rule out states that are impossible regarding the fluent *Anchor*.

$$(\forall s, z)(\mathit{KState}(s, z) \supset \phi_1(z) \wedge \phi_2(z)) \tag{6}$$

$$\begin{aligned}\phi_1(z) \stackrel{\text{def}}{=}\ & \big(\mathit{Holds}(\mathit{Anchor}(x, \Gamma_1, o_1, \pi_1), z) \wedge \\ & \mathit{Holds}(\mathit{Anchor}(x, \Gamma_2, o_2, \pi_2), z) \wedge x \neq \perp \\ & \supset \Gamma_1 = \Gamma_2 \wedge o_1 = o_2 \wedge \pi_1 = \pi_2\big)\end{aligned} \tag{7}$$

$$\begin{aligned}\phi_2(z) \stackrel{\text{def}}{=}\ & \big(\mathit{Holds}(\mathit{Anchor}(x_1, \Gamma_1, o_1, \pi), z) \wedge \\ & \mathit{Holds}(\mathit{Anchor}(x_2, \Gamma_2, o_2, \pi), z) \wedge \pi \neq \perp \\ & \supset x_1 = x_2 \wedge \Gamma_1 = \Gamma_2 \wedge o_1 = o_2\big)\end{aligned} \tag{8}$$

Condition (7) for state z says that if an anchor for a valid symbol x exists, then it is the unique anchor of this symbol in this state. (Universal quantification of variables is assumed in this article if not specified.) Note that a state z represents a single hypothesis regarding possible correspondences for anchoring, while multiple hypotheses are considered in Section 3.4.

The object recognition system is required to assign unique percept IDs π to each individual percept. We simply assign a running natural number for each new percept which is recognised. Accordingly, formula (8) conditions a state z to contain anchors with unique percepts π if the anchor for π exists and π is defined.

Together, both conditions form the basic representational requirement for a state to be consistent:

$$\mathit{Consistent}(z) \stackrel{\text{def}}{=} \phi_1(z) \wedge \phi_2(z) \tag{9}$$

The test of the correspondence relation is based on comparing the properties of the object pointed at and will be explained next.

3.2 Matching as Definite Reference

The term definite reference is used if a particular entity is in question, e.g., "my cup". Anchoring a definite reference tries to associate a specific object symbol with appropriate percepts by means of the object's properties and knowledge about it. In general, this process may yield a number of hypotheses for possible correspondences according to the information available at that time. Then, intelligent techniques should allow to determine the correct one sooner or later, as pointed out in Section 3.4.

Anchoring a definite reference employs the predicate *MatchDef*. Given a symbol x and a percept with properties Γ' in state z it checks for all required attributes whether the estimate of x matches that of Γ':

$$\begin{aligned}&\mathit{MatchDef}(x, \Gamma', z) \stackrel{\text{def}}{=} (\exists \Gamma, o, \pi, \mathbf{a}) \\ &\quad \big(\mathit{Holds}(\mathit{Anchor}(x, \Gamma, o, \pi), z) \wedge \mathit{Attributes}(o, \mathbf{a}) \wedge \\ &\quad\ (\forall a \in \mathbf{a})\,(\exists \gamma \in \Gamma, \gamma' \in \Gamma', i_1, i_2) \\ &\quad\quad \big[\gamma = (a, i_1) \wedge \gamma' = (a, i_2) \wedge \mathit{Intersect}(i_1, i_2)\big]\big)\end{aligned}$$

where *Intersect* is true iff the intersection of the intervals $[\,c_1, c_2)$ and $[\,b_1, b_2)$ is not empty:

$$\begin{aligned}&\mathit{Intersect}((c_1, c_2), (b_1, b_2)) \stackrel{\text{def}}{=} \\ &\quad |c_2 - c_1| > 0 \wedge |b_2 - b_1| > 0 \wedge \\ &\quad ((b_1 \leq c_1 < b_2) \vee (b_1 < c_2 \leq b_2) \vee (c_1 \leq b_1 < c_2))\end{aligned}$$

Notice, that the direction of anchoring at this level of detail can be considered bottom-up — given an object specification in terms of estimates of perceptive features, appropriate percepts are retrieved. Because this results in multiple possible instances in general, the strategy of anchoring will use knowledge and constraints in order to determine valid hypotheses in top-down fashion.

3.3 Matching as Indefinite Reference

In contrast to the definite reference, an indefinite reference considers objects that meet certain conditions in terms of perceptive properties. For instance, one might simply look for "a white cup". Anchoring an indefinite reference tries to establish a complete correspondence between symbol and percept, both representing an object that meets given conditions.

A symbolic reasoning system appeals for intuitive symbols as a convenient specification of object properties. We define *Grounded* which relates predicates to ranges of attribute values. For instance,

$$\mathit{Grounded}(\mathit{Cup}, \mathit{Height}, \mathit{Regular}, (8, 11)) \tag{10}$$

specifies a regular height of cups between 8 cm to 11 cm. Together with

$$\begin{aligned}&\mathit{GroundingVal}(o, a, p, v) \stackrel{\text{def}}{=} \\ &\quad (\exists b_1, b_2)(\mathit{Grounded}(o, a, p, (b_1, b_2)) \wedge (b_1 \leq v < b_2)) \\ &\mathit{GroundingInt}(o, a, p, (c_1, c_2)) \stackrel{\text{def}}{=} (\exists b_1, b_2) \\ &\quad \big(\mathit{Grounded}(o, a, p, (b_1, b_2)) \wedge \mathit{Intersect}((c_1, c_2), (b_1, b_2))\big)\end{aligned}$$

these predicates allow to compare a symbolic description with an estimate comprising values or intervals of values.

Predicate *MatchIndef* provides the tool to check whether the percept at hand meets a given object description. It determines whether the symbolic description Σ and the anchor for symbol x' match, based on whether the estimate of attribute a of this anchor coincides with the perceptive range denoted by each predicate p of Σ:

$$\begin{aligned}&\mathit{MatchIndef}(\Sigma, x', o, z) \stackrel{\text{def}}{=} (\exists \Gamma', \pi', \mathbf{a})\big(\mathit{Attributes}(o, \mathbf{a}) \wedge \\ &\quad \mathit{Holds}(\mathit{Anchor}(x', \Gamma', o, \pi'), z) \wedge \\ &\quad (\forall \sigma \in \Sigma)(\exists \gamma' \in \Gamma', a \in \mathbf{a}, p, i') \\ &\quad\quad \big[\sigma = (a, p) \supset \gamma' = (a, i') \wedge \mathit{GroundingInt}(o, a, p, i')\big]\big)\end{aligned}$$

Notice the bottom-up direction of anchoring in *MatchIndef* — given a particular percept (in terms of attribute values), appropriate symbols denoting objects in question are retrieved.

As in the case of a definite reference, the anchoring strategy will have further means for determining which ones of several possible instances are "appropriate" given high-level knowledge and constraints. The formal account of the anchoring strategy is part of our future work.

3.4 Space of Hypotheses

As has been pointed out above, several correspondences between a symbol and a percept may be possible in general. All of them have to be taken into consideration in order for the anchoring approach to be sound, i.e., not to neglect potential solutions and thus yielding wrong correspondences eventually. In our approach, multiple hypotheses are represented

by a knowledge state in the Fluent Calculus (see Section 2). While a single state z specifies one hypothesis regarding correspondences for all considered objects, *KState*(s, z) associates several possible states with a certain situation s. For example, starting in the initial situation S_0 where the agent knows of the cup C, the knowledge state of situation S_2 after recognising two cups and anchoring C could be this:

$$\begin{aligned}&\textit{KState}(S_2, z) \equiv (\exists z_1, o, \Gamma_1, \Gamma_2, \pi_1, \pi_2)\\ &[z = \textit{Anchor}(C, \Gamma_1, o, \pi_1) \circ \textit{Anchor}(\perp, \Gamma_2, o, \pi_2) \circ z_1 \vee\\ &\ z = \textit{Anchor}(\perp, \Gamma_1, o, \pi_1) \circ \textit{Anchor}(C, \Gamma_2, o, \pi_2) \circ z_1] \wedge\\ &o = \textit{Cup} \wedge \textit{Consistent}(z)\end{aligned}$$

This formula describes that either the percept π_1 is related to symbol C and percept π_2 has no correspondence to a symbol, or vice versa. Moreover, for both possible hypotheses the object category *Cup* applies.

4 Example

Now we will illustrate the computational framework for anchoring presented above in an example. Figure 1 depicts the evolution of knowledge states of a mobile delivery robot during the course of action.

In the initial situation S_0, Mike has requested the mobile robot to bring his cup C_M from the kitchen to his place. The robot has learned further that Mike's cup is white and should be located on the table in the kitchen T_k, as usual on the left hand side just next to a green cup. Both cups are of regular size. Consider the corresponding knowledge state (11). The robot starts at the location near Mike's table, *At*(T_M). In this knowledge state, constraints on the given properties of cup C_M are derived from the symbolic description Σ of the object's properties.[4] Notice the incomplete state specification by means of z_1, which allows other fluents to hold in state z, except another fluent *Anchor* for symbol C_M due to the domain constraint (9).

After moving to the table in the kitchen T_k in situation S_0, model-based object recognition regarding cups is performed. The knowledge state of situation S_2 is described in (13). Three additional anchors have been added to the previous knowledge state representing three cups which were recognised in the current sensory data, each bearing a unique percept ID. While the object recognition system extracted individual values of attributes for each of them, no correspondences between symbols and the new percepts have been established yet, as indicated by $\perp$ for each symbol argument in the anchor representation. For the sake of readability, the estimates White, Green and $on\ T_k$ abbreviate perceived (ranges of) attribute values like those for *Height*.

Notice the formula in brackets in (13). Since the knowledge state z is incomplete, as indicated by state variable z_1, and since the list of current percepts of cups with $\pi \neq \perp$ listed in z is complete, the restriction in brackets conditions state z_1 to contain no more anchors for percepts of this kind.

Which one of the three recognised cups is the one that symbol C_M refers to? According to knowledge state (13), two of them match Mike's description wrt. immediate object properties — the two white cups of regular size on table T_k. Both percepts render potential correspondences with symbol C_M and have to be considered as possible hypotheses.

Next, action *AnchoringDef* tries to establish the correspondence with the definite reference to symbol C_M and yields the knowledge state (15). Anchoring has determined that either the anchor with $\pi = 1$ corresponds to symbol C_M, or the anchor with $\pi = 3$ does so. In both cases, the other anchor must not correspond to C_M, too, which can be inferred from (9). Formula (15) demonstrates that the anchoring strategy fuses two anchors referring to the same object, where the symbol is undefined in the one anchor and the percept is undefined in the other one, if both match. Recall from Section 3.1 that a complete correspondence between a symbol and a percept is represented by a unique *Anchor* fluent in the state at hand. The resulting perceptive estimate Γ of such a fusion of two anchors is obtained from the intersection of both attribute value ranges for each attribute, such that a more precise estimate Γ is gained.

Part of the knowledge of this example task was that the desired cup C_M can be found next to a green cup on its left hand side. The last two lines of (15) specify this high-level knowledge. Given this constraint, the ambiguity regarding correspondences of symbol C_M with the correct percept can be solved and knowledge state (16) can be derived.

Besides sensory information concerning the colour and height of cup C_M, its location is actually known as well. Hence, the preconditions of the action to grab the desired object should be fulfilled and the robot can continue solving the given task.

5 Discussion and Future Work

In the anchoring approach of [Coradeschi and Saffiotti, 2000], predicates like "small" are related to ranges of values. The work of [Chella *et al.*, 2004] extended and adapted this framework to so-called conceptual spaces. Apparently the conceptual space introduces an intermediate representational level for referring to the underlying domain of values for attributes of objects. While yielding modularity on the one side of this intermediate level, it seems that domain-dependent definitions again have to refer to specific properties of attributes on the other side. Our approach also requires domain-dependent definitions; e.g. in (5) or (10).

In [Coradeschi and Saffiotti, 2002] a context-dependent meaning of predicates describing object properties wrt. perceptive attributes was proposed. Our approach is designed for this independency by always referring to a given object category, e.g. in *MatchDef* and *MatchIndef*. Hence, predicates like "small" become comparable wrt. this category. The object category is used to restrict the object recognition system to this object model at the same time, gaining computational efficiency.

Multiple hypotheses for possible correspondences of anchoring symbols to percepts must be maintained in case of perceptive ambiguity, as was pointed out in [Coradeschi and Saffiotti, 2002] for instance. The approach described in [Lang *et al.*, 2003] allows to represent multiple hypotheses by as-

[4] For the sake of readability, the location T_k is specified in this way, too. Considering a location in the world to be a vector, this mechanism could be easily extended to ranges in space.

$$\mathit{KState}(S_0, z) \equiv (\exists z_1, \Gamma_M, \Sigma_M)\big(z = \mathit{Anchor}(C_M, \Gamma_M, \mathit{Cup}, \bot) \circ \mathit{At}(T_M) \circ z_1 \wedge \mathit{Consistent}(z) \wedge \Sigma_M = \{(\mathit{Height}, \mathit{Regular}), (\mathit{Colour}, \mathit{White}), (\mathit{Location}, on\ T_k)\} \wedge \mathit{MatchIndef}(\Sigma_M, C_M, \mathit{Cup}, z)\big) \tag{11}$$

$$S_2 = \mathit{Do}(\mathit{Sensing}(\mathit{Cup}), \mathit{Do}(\mathit{Goto}(T_k), S_0)) \tag{12}$$

$$\begin{aligned}\mathit{KState}(S_2, z) \equiv (\exists z_1, \Gamma_M, \Sigma_M)\big(&z = \mathit{Anchor}(C_M, \Gamma_M, \mathit{Cup}, \bot) \circ \mathit{At}(T_k) \circ \\ &\mathit{Anchor}(\bot, \{(\mathit{Colour}, \mathrm{White}), (\mathit{Height}, (8.5, 9.5)), (\mathit{Location}, on\ T_k)\}, \mathit{Cup}, 1) \circ \\ &\mathit{Anchor}(\bot, \{(\mathit{Colour}, \mathrm{Green}), (\mathit{Height}, (9.0, 10.0)), (\mathit{Location}, on\ T_k)\}, \mathit{Cup}, 2) \circ \\ &\mathit{Anchor}(\bot, \{(\mathit{Colour}, \mathrm{White}), (\mathit{Height}, (7.5, 8.5)), (\mathit{Location}, on\ T_k)\}, \mathit{Cup}, 3) \circ z_1 \wedge \\ &\mathit{Consistent}(z) \wedge \Sigma_M = \{(\mathit{Height}, \mathit{Regular}), (\mathit{Colour}, \mathit{White}), (\mathit{Location}, on\ T_k)\} \wedge \\ &\mathit{MatchIndef}(\Sigma_M, C_M, \mathit{Cup}, z) \wedge \neg(\exists x', \Gamma', \pi', \Sigma')\big[\mathit{Holds}(\mathit{Anchor}(x', \Gamma', o, \pi'), z_1) \wedge \\ &\quad \pi' \neq \bot \wedge \Sigma' = \{(\mathit{Height}, \mathit{Regular})\} \wedge o = \mathit{Cup} \wedge \mathit{MatchIndef}(\Sigma', x', o, z_1)\big]\big)\end{aligned} \tag{13}$$

$$S_3 = \mathit{Do}(\mathit{AnchoringDef}(C_M), S_2) \tag{14}$$

$$\begin{aligned}\mathit{KState}(S_3, z) \equiv (\exists z_1, x_1, x_2, x_3, l_M, o)\big(&z = \mathit{At}(T_k) \circ \\ &\mathit{Anchor}(x_1, \{(\mathit{Colour}, \mathrm{White}), (\mathit{Height}, (8.5, 9.5)), (\mathit{Location}, L_1)\}, o, 1) \circ \\ &\mathit{Anchor}(x_2, \{(\mathit{Colour}, \mathrm{Green}), (\mathit{Height}, (9.0, 10.0)), (\mathit{Location}, L_2)\}, o, 2) \circ \\ &\mathit{Anchor}(x_3, \{(\mathit{Colour}, \mathrm{White}), (\mathit{Height}, (7.5, 8.5)), (\mathit{Location}, L_3)\}, o, 3) \circ z_1 \wedge \\ &o = \mathit{Cup} \wedge x_2 = \bot \wedge \big[x_1 = C_M \wedge x_3 = \bot \wedge l_M = L_1 \ \vee \ x_1 = \bot \wedge x_3 = C_M \wedge l_M = L_3\big] \wedge \\ &(\exists x', \Sigma', \Gamma', \gamma' \in \Gamma', l', \pi')\big[\mathit{Holds}(\mathit{Anchor}(x', \Gamma', o, \pi'), z) \wedge \Sigma' = \{(\mathit{Colour}, \mathrm{Green}), (\mathit{Location}, on\ T_k)\} \wedge \\ &\quad x' \neq C_M \wedge \mathit{MatchIndef}(\Sigma', x', o, z) \wedge \gamma' = (\mathit{Location}, l') \wedge \mathit{LeftNextTo}(l_M, l')\big] \wedge \mathit{Consistent}(z)\big)\end{aligned} \tag{15}$$

$$\begin{aligned}\Rightarrow \quad \mathit{KState}(S_3, z) \supset (\exists z_1)\big(&z = \mathit{Anchor}(C_M, \{(\mathit{Colour}, \mathrm{White}), (\mathit{Height}, (8.5, 9.5)), (\mathit{Location}, L_1)\}, \mathit{Cup}, 1) \circ \\ &\mathit{At}(T_k) \circ z_1 \wedge \mathit{Consistent}(z)\big)\end{aligned} \tag{16}$$

Figure 1: Example: knowledge states of a delivery robot during the course of action.

signing scores to individual anchor components regarding different sensory modalities. It seems that multiple hypotheses are only maintained temporarily until sensory information is collected. Moreover, the scores are based on a single attribute only. In our approach, the more perceptive attributes are used for anchoring, the better two objects can be told apart based on their perceptive properties.

The tagged-behaviour approach described in [Horswill, 2001] achieves anchoring of symbols to percepts by means of a number of object trackers running at high frequency and specific programs associated with input rules. Since there exists virtually no symbolic representational level, its expressiveness is strictly limited in comparison to our approach.

In future work we will axiomatise the strategy of anchoring in form of knowledge update axioms in the Fluent Calculus. Specifying domain-dependent attribute values used for grounding predicates may not be the desired way. Learning of these values as well as deriving properties from an ontology of object categories seem to be interesting directions. We also want to investigate reasoning over time with object properties. By that, one could specify "the cup that I held recently", for instance.

6 Conclusion

Anchoring symbols to percepts is crucial for autonomous agents in real and dynamic environments. We have presented an approach to the anchoring problem based on the Fluent Calculus. Its powerful expressiveness lies in representing multiple hypotheses for possible correspondences as well as reasoning about knowledge and object properties. Our approach allows to anchor symbols to percepts using definite and indefinite references. Besides other advantages, the proposed anchoring technique supports reasoning over object properties during planning on the one hand, and establishing and maintaining correspondences during run-time on the other hand. Thus the robustness of an agent control system in terms of possible failures when performing in realistic environments can be increased significantly.

Acknowledgments

This research has been funded by the *Deutsche Forschungsgemeinschaft* (TH 541/8-3). We are thankful to our anonymous reviewers for helpful comments.

References

[Chella *et al.*, 2004] A. Chella, S. Coradeschi, M. Frixione, and A. Saffiotti. Perceptual anchoring via conceptual spaces. In *Proc. AAAI-04 Workshop on Anchoring Symbols to Sensor Data*, Menlo Park, USA, 2004. AAAI.

[Coradeschi and Saffiotti, 2000] S. Coradeschi and A. Saffiotti. Anchoring symbols to sensor data: preliminary report. In *Proc. 17th AAAI Conf.*, pages 129–135, Menlo Park, USA, 2000. AAAI.

[Coradeschi and Saffiotti, 2002] S. Coradeschi and A. Saffiotti. Perceptual anchoring: A key concept for plan execution in embedded systems. In M. Beetz, J. Hertzberg, M. Ghallab, and M. Pollack, editors, *Advances in Plan-Based Control of Robotic Agents*, number 2466 in LNAI, pages 89–105. Springer, Berlin, 2002.

[Coradeschi and Saffiotti, 2003] S. Coradeschi and A. Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43:85–96, 2003.

[Harnad, 1990] S. Harnad. The symbol-grounding problem. *Physica / D*, 42:335–346, 1990.

[Horswill, 2001] Ian Horswill. Tagged behavior-based architectures: Integrating cognition with embodied activity. *Intelligent Systems*, 16(5), 2001.

[Lang *et al.*, 2003] S. Lang, M. Kleinehagenbrock, S. Hohenner, J. Fritsch, G. A. Fink, and G. Sagerer. Providing the basis for human-robot-interaction: A multi-modal attention system for a mobile robot. In *Proc. Int. Conf. on Multimodal Interfaces (ICMI-03)*, pages 28–35, Vancouver, Canada, 2003. ACM.

[McCarthy, 1963] J. McCarthy. *Situations and Actions and Causal Laws*. Stanford Artificial Intelligence Project, Memo 2, Stanford University, USA, 1963.

[Saffiotti, 1994] A. Saffiotti. Pick-up what? In C. Bäckström and E. Sandewall, editors, *Current Trends in AI Planning*, pages 266–277. IOS, Amsterdam, NL, 1994.

[Thielscher, 1999] M. Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299, 1999.

[Thielscher, 2000] M. Thielscher. Representing the knowledge of a robot. In *Proc. Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-00)*, pages 109–120, Beckenridge, USA, April 2000. Morgan Kaufmann.

A Framework for Evaluating Groundedness of Representations in Systems: From Brains in Vats to Mobile Robots

Mary-Anne Williams
University of Technology, Sydney
Australia
mary-anne.williams@uts.edu.au

Peter Gärdenfors
Lund University Cognitive Science
Sweden
peter.gardenfors@lucs.lu.se

Alankar Karol
University of Technology, Sydney
Australia
alankar@it.uts.edu.au

John McCarthy
Stanford University
USA
jmc@cs.stanford.edu

Christopher Stanton
University of Technology, Sydney
Australia
cstanton@it.uts.edu.au

Abstract

In order for a system to achieve its objectives it must *ground* its representations: a grounded representation is one where the entities in the representation correspond *meaningfully* to the entities they represent.

In this paper we develop the first framework for analysing grounding capabilities in systems. The framework can be used at a theoretical level to analyse grounding capabilities in systems, and it also offers a practical guide to assist the design and construction of systems with more effective grounding capabilities.

1 Introduction

This paper addresses the issue of grounding representations which is an important, fundamental and challenging problem in Artificial Intelligence. Grounding involves building and maintaining coherent representations that correspond *meaningfully* to the entities they represent. We develop an innovative framework for evaluating *how well* system representations are grounded, and then demonstrate the framework's usefulness and impact.

We build on a wide variety of pioneering and important work on grounding [2; 3; 5; 12; 13; 18; 19; 20; 21; 22] by developing a new understanding of grounding and the first framework for analysing and evaluating grounding capabilities.

Systems from airline reservations to autonomous mobile robots rely on *grounded* representations. Grounded systems possess grounded representations. An airline reservation system must manage information about flights and passengers in a way that corresponds to real flights and real passengers over time. An autonomous mobile robot that navigates a physical space will be more effective in acheiving its objectives if its internal representations of physical barriers correspond to real physical barriers in its environment. A sound grounding capability provides basic infrastructure for cognition and intelligence. Consequently, *how*, and *how well*, internal states and representations are grounded is of significant interest and crucial importance in AI.

Grounding capabilities are system specific, domain specific, and context specific. Our framework strongly supports the idea that when it comes to assessing grounding capabilities there are few absolute measures. Typically groundedness[1] of a system is measured relative to the groundedness of other systems, e.g. it is common to evaluate the grounding capabilities of systems relative to human grounding capabilities often coupled with additional sensors and instruments. The framework we develop can be used to understand grounding capabilities in existing systems and to support the design and implementation of intelligent systems whose representations need to be grounded in order for them to achieve their respective design goals.

In section 2 we describe our broad notion of representation. In section 3 we describe grounding capabilities of systems and provide a set of principles that guide the groundedness framework which is developed in section 4. The framework is designed to measure the quality of grounding capabilities. In section 5 we illustrate the power of the framework by demonstrating its use in (i) analysing a specific system, (ii) comparing the grounding capabilities of several systems, and (iii) developing a quality ranking for the system development

[1] Groundedness is a noun and it refers to the property possessed by grounded systems.

lifecycle. Finally in the last section we highlight the major benefits and applications of the framework.

2 Representations

The value of representations has been hotly debated in the literature, for example according to Brooks [3] "the world is its own best representation[2]". Regardless of the debate surrounding the need for representations the unassailable fact is that the better system representations are grounded, the more effectively the system will acheive its goals.

Representations for our purposes come in all shapes and sizes. They range from low level sensorimotor representations all the way up to high level logic and linguistic expressions. A grounded representation does not require that every entity in the representation be *linked* to a corresponding physical manifestation, but a meaningful relationship should exist between the entities in the representation and the entities being represented. Maintaining a correspondence between representations of physical objects and the objects themselves is important but so too are representations of object functionalities and relationships between objects, as well as descriptions of ways to interact with specific objects, etc.

For the purpose of understanding grounding it is insightful to classify representations using the hierarchy of Gärdenfors [10], illustrated in Figure 1, which describes the crucial relationships between three key representational entities: sensations, perceptions, and imaginations. Representations in the hierachy can take two forms: cued and detached. *Cued representations* are based on the perception of things that are present, and detached representations focus on entities that are not currently perceived. Cued and detached representations may or may not be grounded.

Sensations are immediate sensorimotor impressions, *perceptions* are interpreted/processed sensorimotor impressions, and *imaginations* are detached representations. Sensations provide systems with an awareness of the external world and their internal world. They exist in the present, are localised in the body/system, and are modality specific, e.g. visual, auditory, not both. Perceptions encapsulate more information than raw sensorimotor information. They can represent accumulated sensorimotor information and sensorimotor information reinforced with simulations [2]. Sensations involve signals from sensors or from inside the system itself, but perceptions require additional information derived from previous experiences and/or outcomes of learning. In contrast to sensation, perception is cross-modal, and perceptions can generate permanence, e.g. object permanence.

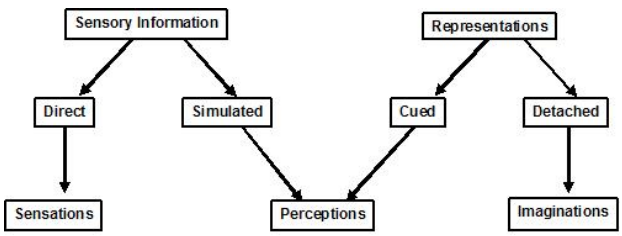


Figure 1: *Cued and Detached Representation Hierarchy.*

[2]Future world states in general, however, are not a feature of the current world state consequently systems that need to plan effectively and anticipate future world states require representations.

Representations in our framework include low level sensorimotor information such as YUV or RGB values of pixels in a digital image through to information about entities that can no longer be experienced like dinosaurs and melted ice cubes and imaginary entities like hobbits[3]. Detached representations of objects exist as well as detached representations of relationships, actions, events, and processes.

Representations can be derived from information that has been gathered from a wide range of sources e.g. internal and external sensors, internal and external effectors, external instruments, external systems, etc. In addition they can result from fusing sensorimotor information with high level representations such as perceptions, concepts and linguistic expressions. Consider a doctor who not only grounds his own sensorimotor information, but information from colleagues, books, lab tests, instruments such as thermometers, and equipment used to visualise heart beat, and to measure blood pressure and oxygen content of the blood.

We illustrate several kinds of representations in Figure 2 based on a Robot Soccer System [1] which are constructed from raw robot camera data. Figure 2(a) is a 2D visualisation of the *raw* camera data, and Figure 2(b) is a processed version of Figure 2(a) where specific colours (YUV values) of pixels are used to determine if they *belong* to specific objects of interest - a ball, a beacon and a goal are clearly identified.

The information represented in Figure 2(b) can be used to find the distance, heading and elevation, from the robot's camera, of the various objects of interest which in turn can be used to calculate the pose of the robot in a global reference frame. Information represented in Figure 2(b) can be combined with a relational representation of robot location, i.e. $robot(id, x, y, \phi)$[4], to create a relational representation of the location of objects, i.e. $object(o, r, \phi, \theta)$[5]. The set of $object$ relations can be visualised for ease of interpretation as soccer objects such as goals, robots, ball in specific locations on a simple 2D representation of soccer field.

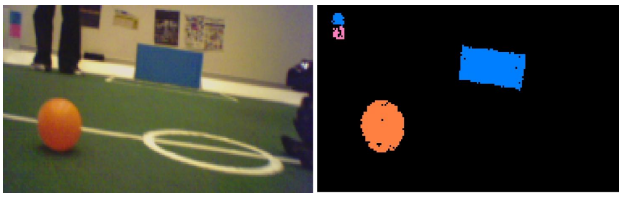


Figure 2: *(a) a digitial image derived from a robots camera, (b) a perceptual representation of the ball, a beacon and a goal.*

Detached representations are extremely powerful. They can be manipulated independently of the external world, i.e. can be conceived and do not need to be perceived. Some examples of detached representations are absent objects, past and potential future world states, etc.

[3]Tolkien, J.R.R. The Hobbit, Ballantine Books, 1937.

[4]id is a robot's identifier, x and y are coordinates of the robot and ϕ is the heading of the robot in a predefined world coordinate system

[5]o can be one of [*ball, beacon, goal, team-mate, opposition-robot, obstacle*], r is the distance from the camera of the robot to the object, ϕ is the heading to the object and θ is the elevation to the object relative to the robot's camera system

3 Grounding Capabilities

Grounding plays an important role and provides critical infrastructure for cognition and intelligence. Groundedness is intimately related to, but not the same as, cognition and intelligence and as a result the framework we develop in the next section does not make judgements about the quality of specific representations, e.g. whether one representation is better than another, nor does it directly measure how well a system exploits its representations.

Instead, the framework focuses on the grounding capability, namely, the ability to maintain relationships between entities in representations and the entities themselves where the entities can be physical, abstract, sensed, perceived, or fanciful.

A grounding capability can capture information and manage information exchanges to and from the external world, it can also create, interpret, manage and maintain internal and external world representations.

Grounding capabilities support system goals and objectives, and therefore measuring the quality of a grounding capability should be conducted with respect to the system goals and objectives. The purpose of a grounding capability is to construct and maintain coherent internal representations that correspond meaningfully to the things being representated so that the system can achieve its aims and objectives. Clearly the *quality* of a system's grounding capability will have an crucial impact on the success of the system, and on what it can achieve.

3.1 Grounding in Traditional System Development

The correspondence between representations and the entities represented in a wide range of artificial systems is often established and maintained over the system's life time by the human designers. As a result it is fair to say that the systems produced are (partially) grounded externally via the human mind. In other words, the human mind plays a role in creating, subsequently interpreting, and maintaining the correspondence between entities in representations and the actual entities themselves throughout a systems lifetime.

Systems are, typically, reviewed by (a team of) human reviewers during the various phases of the system development lifecycle and through those reviews the groundedness of the system is evaluated.

Depending on the level of sophistication of the system under review once humans have established the correspondence between entities in the "external world"and the internal systems representations such as a database conceptual schema, systems such as Database Management Systems can manage the correspondence relationship over time but only in restricted ways. For example, a DBMS can add, remove, modify and validate relational tuples via application programs without human intervention, but should database conceptual schema require modification due to changes in requirements then a human will likely determine how best to accommodate the changes and ensure that the database remains grounded, i.e. in appropriate correspondence with its *external world*.

Changes to grounding requirements due to changes in the environment and/or systems goals, for example, result in changes to the system and those changes are typically determined by human designers, not the system itself for simple applications like database management systems.

3.2 Principles for Grounding and Groundedness

In this section we present the principles that will be used to guide the development of our groundedness framework in section 4.

- *Grounding is a capability* which involves the creation, management and maintenance of the associations between entities in representations and the entities themselves. It can involve a single system or extend beyond the boundaries of the system, e.g. rely on components beyond the system such as external sensors, resources, tools, instruments, other systems, etc.
- *Groundedness is a property of a grounding capability*, and it should be measured relative to system goals.
- *Systems can ground their representations in a variety of different ways*: top-down via the process of design; bottom-up via sensors, effectors, and interfaces, and through information obtained via external objects (e.g. physical tools), external sensors (e.g. radar), and external systems (e.g. medical monitoring system); or a complex combination.
- *Groundedness is graded and multidimensional.* A system is not simply grounded or ungrounded. There are degrees of groundedness. Furthermore, since groundedness should be measured relative to system goals the salient dimensions will vary depending on what is determined to be important for the purposes of the grounding analysis at hand.
- *Measures of groundedness can be qualitative or quantitative, continuous or discrete*. A groundedness framework should not impose restrictions on the form and measure of assessment, only its capacity to support the systems goals.
- *A groundedness framework should not place restrictions on what could or should be grounded.* Anything can be grounded: physical things, abstract things, nonexistent things, and things that have never been experienced. Things can include: objects, relationships, states, actions, processes, events, etc.
- *A groundedness framework should cater for a wide range of systems from artificial to biological.*

4 Our Groundedness Framework

Our framework is motivated by the need to understand and build sophisticated systems that do (some of) the grounding themselves rather than systems that are completely grounded with the assistance of human grounding capabilities. It comprises five essential elements which can be as detailed as required for the purpose of the analysis:

1. System Objectives
2. Architecture of Grounding Capability
3. Scope of the Analysis
4. Nature of the Grounding Capability
5. Groundedness Qualities.

All five components are related, e.g. the objectives and the scope will often determine how the qualities of groundedness are chosen, interpreted, and assessed.

4.1 System Objectives

The first part of the framework involves developing a description of the system(s) objective, goals, tasks and activities. The level of detail will be determined by the nature of the system grounding analysis being undertaken.

4.2 Architecture of Grounding Capability

The second component of the framework is a description of the underlying system architecture that supports or implements the grounding capability.

An architecture defines the structure and organisation of the main system components and their channels of communication. There are a wide range of potential architectures e.g. layered, embedded, cognitive, etc. Furthermore, a grounding capability can be described in terms of a number of different architectures. The architecture should be described so as to maximally expose the grounding capability. The description of the underlying system architecture should focus on the components and processes involved in systems' grounding activities.

If systems are being compared then it is desirable to describe the architectures using similar concepts and components. Often representations are translated from one form to another. Details about the relationships among the representations can also be given including details of elements of representations that are perserved and those that are changed during such transformations.

4.3 Scope of the Analysis

The third component of the framework is a detailed description of the scope of the analysis. For example, the scope of the analysis might be restricted to a specific component of the system, a specific set of interfaces or system activities, or specific grounding activities such as the creation of associations between representations and external entities. An example is given in section 5.1.

4.4 Nature of the Grounding Capability

The fourth component describes the nature of the grounding capability under analysis. A useful approach to describing the nature of the grounding capability is with respect to the underlying architecture. Important charactistics of a grounding capability are described in the example given in section 5.1.

4.5 Groundedness Qualities

The fifth component of the framework includes a description of the pertinent groundedness qualities, and an assessment of them relative to each architectural component of the grounding capability where appropriate. The instruments for measuring the qualities should also be identified. It is important to be able to compare and contrast grounding capabilities in different systems, consequently lower level features of groundedness need to be determined in order to evaluate grounding capabilities in systems.

Grounding is multi-dimensional and graded. The components of a grounding capability, and the dimensions/qualities of groundedness need to be identified in order to better understand and ultimately evaluate groundedness. In order to evaluate groundedness we identified a set of important features that can be used as key performance indicators for assessing the quality of a grounding capability.

In what follows we describe some groundedness qualities which are appropriate for assessing an intelligent agent. For any particular groundedness analysis we envisage that a set of appropriate qualites will be identified based on the objectives of the system(s) under analysis, as well as the scope and nature of grounding analysis. Some of the qualities in the example, below, are so fundamental to the grounding endeavour that they could be used as candidate qualities for almost all grounding analyses.

Expressiveness: is the breadth of objects, relationships, processes, actions, events, states, etc that are representable internally and require grounding. Expressiveness measures the richness of representations.

Relevance: determines the degree of relevance of the entities that are represented by a system. Relevance is related to, but different from, expressiveness. It focuses on issues related to those aspects of the world that are important for a system to achieve its goals. For example, a robot soccer player may not perceive the audience, or field lines painted on the field because they are not relevant to its tasks or it can achieve its goals without specifically representing them. Changes in task, goals and environment are considered elsewhere and so in the assessment of relevance we only consider current goals; not potential or future goals. Representations are selective in terms of what they can represent - a representation cannot capture every feature or aspect of the world. Choices have to be made with regard to the entities that are important, relevant, and necessary for the system to complete its tasks and achieve its goals.

Faithfulness: is the relationship between entities in internal representations and the entities themselves, e.g. the relationship between an internal world model and the world itself. Faithfulness is a matter of degree and the pertinent question is how *closely* does a system's representations correspond to the entities being represented. Determining the degree of faithfulness is sometimes achieved by measuring the ability of the system to model the world states and world state transitions in terms of prediction and explanation.

Correctness: is the ability of a system to represent information in accordance with its specification. For example, a robot soccer player's ability to determine the location of the ball on the field would be an example of a task which has a well defined specification and whose correctness could be measured. The correctness of the task could be context dependent. For example, a robot's ball location ability may be better when it is stationary than when it is in motion.

Accuracy/Precision: is related to faithfulness and correctness and involves the degree of fidelity of information being represented. For example, the robot soccer players perception of its position and the ball's position on the field might be required to be measured to different degrees of accuracy, e.g. to the nearest millimetre or meter.

Robustness: is the ability of a system's representations to behave appropriately to unexpected or abnormal conditions. For example, the ability of a robot soccer player to handle changes in the environment such as changes in lighting, changes in background noise, changes in playing surface texture. More dramatic environmental changes would include a change of ball, e.g. different size, different colour, different degree of hardness, and/or different density.

Adaptability: is the ability of representations to adapt to task and goal changes. For example task changes might involve a robot's ability to change soccer positions e.g. from Defender to Striker. More dramatic changes involve changes to the rules of robot soccer or a change in the number of robots on a team. Adaptabilty can be measured by determining the nature/difficulty of the changes that the system can tolerate [17]. To what extent the system can change itself, and when does it require human assistance if we introduce new objects, new relationships between objects, new actions, new events, etc.

Timeliness: is the ability of representations to respond (appropriately) to the environment in a timely fashion. For example, a robot soccer player's ability to dive for a ball as the

ball approaches rather than after it has passed it by.
Efficiency: is the ability of a representation to place as (few) demands as possible on hardware resources such as processor time, communication bandwidth, internal and external storage, sensors, effectors, and actuators.
Self-Awareness: Since systems with self-awareness are typically embedded or embodied the degree of self-awareness is of interest. For example, the question of whether a robot is aware of the state of its body parts such as its forearm is cocked at a 45^o angle, will be important when assessing a grounding capability. Self-awareness is a representation that is graded from physical awareness up to intention awareness. It also raises issues concerning the role of trust in grounding, e.g. being aware of one's own sensor limitation can impact grounding capabilities.
Awareness of Others: Awareness of others is graded: the spectrum of awareness of others spans from the existence of others to the awareness of the intention of others. The degree of awareness about the grounding capability of others and the intentions of others is important for communication and collaboration because such an understanding facilitates the sharing of information in meaningful ways. The issue of trust is also important, e.g. awareness of other's limitations and biases can impact grounding capabilities.
Functionality: involves identifying the system abilities that require grounding. For example, some basic functionality of a robot soccer player includes the ability to recognise the ball, move to the ball, grab the ball, and kick the ball. Different robot players may have different abilities, for example a goalie may be able to dive for the ball whilst a forward may not.
Transparency: is the ability of a system to represent its internal information and knowledge in a way that is assessible to a human or other system. For example, is the representation of information explicitly represented or implicit, clearly derivable or buried in a black box processor. Transparency is a crucially important quality for some systems. A strong transparency quality allows a system to be compared with other systems across a wide range of dimensions with confidence.
Testability: is the ease of testing system grounding capabilities and associated activities such as behaviour and decision making. Building more effective systems in the future will be advanced by learning from grounding capabilities in existing systems, and clearly more will be learned from transparent, easily understood and testable systems.
Uncertainty Management: It can be important to identify, qualify and quantify uncertainty in the grounding capability. This will involve determining the strategies used by the system to address the uncertainty. The focus is on how the system reduces the uncertainty of information gathering and internal information mangement rather than what techniques are used to manage uncertainty in representations.

Important interrelationships exist among the qualities described above such as faithfulness, correctness and accurracy. Transparency and testability are also clearly related. Other qualities may be dervied from those listed above such as reliability which could be the probability of an agent to malfunction or the probability that a system will behave similarly in similar situations, flexibility which is related to robustness, and adaptability, and performance which captures the responsiveness of a grounding capability which could be measured by the time required to respond to stimulus or the number of events processed in some interval of time. Performance is related to a number of qualities including efficiency, expressivity, timeliness, faithfulness and relevance.

4.6 Evaluating Groundedness Qualities

In order to assess the quality of grounding it is helpful if the entities to be grounded are identified. Such entities might include objects, relationships, actions, states, events, plans, and other processes. Objects can be physical e.g. a ball, or abstract e.g. a penalty, and internal e.g. a forearm angle, or external e.g. teammate. They can be permanent, temporary or ephemeral. Relationships typically exist between objects such as the ball is on the field; the ball is located in the yellow half; the goalie has possession of the ball; the ball is out of bounds; the ball is in the goal area; the ball is dead.

Our approach to evaluating groundedness is to assess and/or measure constituent quality dimensions relative to system goals and architecture. A wide range of instruments can be used in concert to assess and measure specific qualities including:

- Direct observation and analysis of working system behaviour.
- Design of test cases and scenarios that push the limits of system grounding capabilities.
- Analysis of artifacts produced by and for the system should they exist, e.g. design documents, software code.
- Development of formal measures e.g. the *closeness* of a soccer field configuration to the actual field configuration can be measured using techniques developed in [14].

Some evaluation methods for certain systems are external such as direct observation, others involve internal analysis. Some qualities can be evaluated via external methods, others need to be measured internally, whilst others measured using a combination of a both modes.

4.7 Benefits of using Logic Driven Systems

Logic-driven systems from database applications to more sophisticated knowledge systems form an important, privileged, and well studied class of systems. Major benefits flow from the possession of clear semantics in particular building, managing, testing, and measuring grounding capabilities can become straightforward when the representations have a fully specified semantics. For example, the faithfulness quality often collapses to an evaluation of truth/falsity, and as a result properties of the methods and algorithms used to determine truth/falsity are at focus. A clear semantics can also enhance the qualities of expressiveness, relevance, correctness, accuracy, timeliness, understandability, transparency and testability. Typically accounts of robustness, adaptability, self-awareness, awareness of others can also be given.

Many types of logical representations and systems have been developed to enhance standard logics' ability to represent more complex, imprecise, incomplete, uncertain, and dynamic information such as nonmonotonic reasoning [16], possibility logic [6], belief revision [8; 26], and languages for action and change [18].

5 Power of the Groundedness Framework

In this section we highlight the power of the framework by demonstrating how it can be used to (i) measure the groundedness of the UTS Unleashed! 2003 Robot Soccer System [1], (ii) compare the groundedness of the UTS Unleashed! 2003 Robot Soccer System with the UTS Unleashed! 2004

Robot Soccer System [4], and (iii) develop a grounding quality ranking for use in systems design.

5.1 Measuring a Systems Groundedness

In this section we illustrate the use of the framework by outlining an analysis of a sophisticated robot soccer system's grounding capability, i.e. robots can perceive the ball, search for it when it is not in view, chase it, kick it, etc. The robots build and maintain a representation of the state of the soccer field from their sensors and internal body data, and then use that representation to make decisions about the best action to perform. The system is based on the classical *sense-think-act* processing cycle [7].

1. System Objective: To play soccer in the RoboCup 4-Legged League[6] at an internationally competitive level.

2. Architecture of Grounding Capability:

The system is the UTS Unleashed 2003 robot team [1]. It is composed of four SONY AIBOs which are 4-legged mobile autonomous robots[7]. Each robot has a camera, and only uses visual cues to communicate. The architecture of the system is illustrated in Figure 3 where the grounding capability is viewed as involving four major subsystems: interaction, perception, conception, and problem solving. *Interaction* involves the exchange of information across interfaces, sensors, and actuators. *Perception* involves the creation, acquisition, management and maintenance of sensorimotor and other cued representations. *Conception* involves the creation, acquisition, management and maintenance of concepts. *Problem Solving* involves the creation, acquisition, management and maintenance of high-level representations such as declarative, procedural, and tacit knowledge used for problem solving, reasoning, and decision making activities.

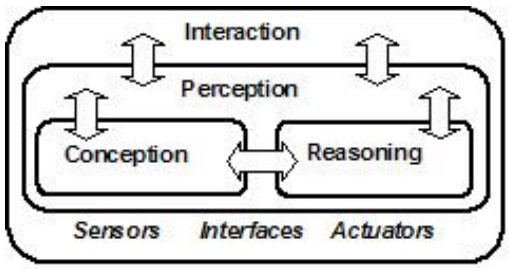


Figure 3: *2003 UTS Unleashed! Robot Soccer Architecture.*

All interaction between the outside world and the internal representations takes place via the *interaction subsystem*. The *conception subsystem* and the *problem solving subsystem* are embedded in the *perception subsystem*. The conception, problem solving, and perception subsystems can communicate with each other directly. Overall robot behaviour is driven by the problem solving subsystem which communicates to the actuators in the interaction subsystem via the perception subsystem.

3. Scope of the Analysis: The analysis will focus on the representation of visual and actuator information. All the architectural subsystems will be involved in the analysis. Only activities related to grounding within the robot are to be considered, i.e. the human designers role in grounding is outside the scope of the analysis.

4. Nature of the Grounding Capability: The world model (field configuration) representation of the problem solving subsystem is grounded through visual information aquired via the robots' camera and a high level model of the robot body. The world model representation can be visualised via a 2D picture of the field with objects identified place in their perceived location, and subsequently evaluated.

5. Groundedness Qualities: Due the lack of space we briefly describe a few of the more pertinent groundedness qualities from section 4.5 below.

Expressiveness: The robots interact with the environment through sensors and actuators. The sensor under analysis is the camera which uses YUV values for each pixel. Parameters for motion are sent and received from motors in the robot's body. Perception for the purpose of this analysis involves vision and control of actuators to achieve bodily movements such as walking and kicking. Conception creates and maintains the following concepts: *physical objects*[ball, beacons, goal, team mates, opposition robots], *abstract objects*[player positions, attack, defend, strategy], *physical relationships*[behind, inside penalty area], *actions*[search, kick, walk], *events*[game start, game restart, game end, kick-off]. The problem solving subsystem constructs a representation of the location of objects such as the ball, team mates, opposition robots, and based on it the robot determines its next action.

Relevance: Only relevant soccer related entities are represented.

Faithfulness: Each robot builds and maintains a representation of the field configuration. The extent to which the field configuration representation is faithful to the real configuration can be measured using the similarity measure developed in [14] which measures the *distance* from one field configuration to another, and it provides a means to explicitly measure the distance/similarity between the *real* configuration of the field and its representation built by the robot as it moves its body and analyses its raw camera data.

Timeliness: The robots are fairly responsive to changes in field configurations and in particular to changes of ball locations. Robot response times can be easy measured and quantified using a wide range of methods at many levels of granularity.

Transparency: is low due because almost all representation management is buried in C++ code.

Robustness: The grounding capability is robust to field surfaces, but not robust to minor changes in lighting. Specific measurements can be made regarding the lighting levels and the roughness of playing surfaces to determine the range of tolerance.

Adaptability: The grounding capability is not adaptable. It cannot make any changes to itself.

Self-Awareness: the grounding capability is aware of some of its internal settings such as neck angles, appendages, touch button states, motor parameters. It can recognise its own body parts if it perceives them, i.e. it can recognise its own feet.

Awareness of Others: is achieved through visual cues only.

5.2 Comparing Systems Grounding Capability

In this section we briefly compare the UTS Unleashed! 2003 System described above with the UTS Unleashed! 2004 System [4]. The 2004 System is built on the 2003 System and it possesses the same overall objectives and underlying grounding infrastructure with a few important extensions. The scope

[6] See http://www.tzi.de/4legged for details

[7] See http://www.sony.net/Products/aibo/ for details.

of the analysis is the same as for section 5.1 and the nature of the grounding capability of the 2004 System has been extended to include sharing vision information about field configurations among the robots via a wireless network.

Groundedness Qualities:

Expressiveness: The 2004 system can represent all the entities representable in the 2003 verison as well as the following:

Interaction:- Robots share information from their world models via a wireless network, in other words full field configurations are represented in the interaction subsystem. In addition improvements were made to the walking engine so that machine learning techniques such as reinforcement learning with self detection and correction could be applied to improve walking speeds.

Perception:- Major improvements in the 2004 Systems include (i) the relationship between YUV values of pixels and symbolic colours can be one to many, rather than one-to-one as in the 2003 system which allows for overlapping colours and more flexibility in identifying objects[23], (ii) the velocity of the ball is perceived which supports new high level skills such as passing, catching, and diving, and (iii) field line recognition by perception subsystem.

Conception:- new object recognition for field lines, new skills conceived [dodge, dive, catch, pass] and new strategies that exploit the new skills and perception grounding capabilites.

Problem Solving:- Robots in the 2004 System can *share information* derived from their world model representation such as the location of the ball and the location other robots[15]. Robots on the team who cannot percieve obects directly can be alerted to their location from team mates. In addition, using the shared information they can localise using the ball's location and their internal body sensors.

For the purposes of illustration we make brief comments about some of the other qualities. All *relevant* entities are represented in both the 2003 and 2004 systems. *Faithfulness* is measured using the visualisation of the world model representation build by each robot. The similarity measure developed in [14] which measures the *distance* from one field configuration to another, allows us to explicitly measure the distance between the *real* configuration of the field and the configuration represented by the robot. Based on our experimental testing the 2004 system was more faithful than 2003. The 2004 System also turned out to be significantly more *accurate*, *responsive*, *transparent*, and *robust* to changes in lighting conditions (due to the one-to-many relationship between pixels and symbolic colours) than the 2003 System. The 2004 System was *aware* of its internal power levels and the 2003 system was not, and furthermore it had a heightened *awareness of others* because high level representations regarding the configuration of the field were communicated between robots directly via the wireless network, and as a result it was more *adaptable* because if a robot was unable to "see"the ball then his teammates could broadcast the ball's location via the wireless network. In addition in the 2004 System the robot's movements were more adaptable due to the incorporation of machine learning techniques in the walking engine.

5.3 Measuring Groundedness in System Design

Quality rankings can be generated from the framework by attaching levels of priority to the groundedness qualities. The resultant priority rankings can then be used to evaluate grounding capabilities during system designs. Tailored rankings can be developed for each system and used to develop system requirements.

Design decisions should respect the priority ranking. Clearly design and implementation decisions will impact on various qualities in different ways and the key idea is to ensure that high priority qualities are maintained in preference to lower ranked qualities whenever faced with a choice. Given the interrelationships that can exist between the groundedness qualities, sometimes trade-offs will be necessary. Increasing efficiency is well-known to negatively impact most other qualities regardless of how we choose to rank them. Identifying a priority ordering of qualities is standard practice in software quality assessments. Some groundedness qualities could be identified as so crucial that they must be part of the design and should not be sacrificed for the sake of improving other qualities.

Dimensions of grounding can be graded according to their importance. A ranking that reflects the importance of the qualities determined in requirements allows system developers to understand and evaluate grounding capabilities. A typical Grounding Quality Ranking is illustrated below:

Rank 1: Essential - Failure to meet the stated degree of qualities will result in complete failure of the system.

Rank 2: Important - Failure to meet the stated degree of qualities will result in a system with certain kinds of problems.

Rank 3: Desirable - Failure to meet the stated degree of qualities will result in less flexibility than desired.

Different rankings for different systems will reflect the design goals. Different design goals will lead to different priorities. For example we would expect that a robot soccer system designed for winning would have a different ranking of grounding qualities that a system designed for innovative play!

6 Discussion

Grounding of representations is an important capability for intelligent systems. Despite its importance there has not been a practical way up to now to measure the groundedness of systems. In this paper we develop a novel framework for measuring how well a system is grounded. The framework supports the identification and articulation of important similarities and differences in grounding capabilities across systems, and can be used to demonstrate how and why one system is grounded *better* than another. For the purpose of designing more effective intelligent systems it is important to be able to articulate that one system has a better grounding capability than another, or to say things like if system A's grounding capability had certain properties then it would have an equivalent or better grounding capability than system B.

The framework has lead to a deeper and richer understanding of grounding capabilites. Furthermore, it provides guidance on how to evalute grounding capabilites, to compare grounding capabilites across several systems, and to build more effective grounding capabilities. Moreover, by developing a better understanding of grounding the framework has allowed us to isolate new research problems, challenges, and directions. For example the framework raises the following research questions: (i) Tarski [25]developed a powerful *Theory of Truth* but what should a *Theory of Reference* look like? (ii) Is there a relationship between the hierarchy of representations in section 2, the qualities of groundedness in section 4, and consciousness?, and (iii) How can we build systems capable of reasoning about their own grounding capability and that of other systems?

Robot soccer offers a standard and rich domain that would benefit from a comprehensive analysis of grounding capabilities. In future work the framework will be used to analyse, compare and contrast the grounding capabilities of the top 10 teams in the RoboCup Legged League. The analysis will include a survey for developers, as well as direct observation of running systems.

References

[1] Agnew, N., Brownlow, P., Dissanayake, G., Hartanto, Y., Heinitz, S., Karol, A., Stanton, C., Trieu, M., Williams, M-A, and Zeman, A., Robot Soccer World Cup 2003: The Power of UTS Unleashed!, http://www.unleashed.it.uts.edu.au/TeamReports

[2] Barsalou, L., Perceptual Symbol Systems, Behavioural and Brain Sciences (1999) 22, 577 - 660.

[3] Brooks, R. A. (1993) The Engineering of Physical Grounding. Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society, pp. 153154. Hillsdale, NJ: Lawrence Erlbaum.

[4] Chang, M., Dissanayake, G., Gurram, D., Hadzic, F., Healy, G., Karol, A., Stanton, C., Trieu, M., Williams, M-A, and Zeman, A., Robot World Cup Soccer 2004: The Magic of UTS Unleashed!, http://www.unleashed.it.uts.edu.au/TeamReports

[5] Coradeschi, S. and Saffiotti, A. (eds), Special Issue of Robotics and Autonomous Systems on Perceptual Anchoring Anchoring symbols to sensor data in single and multiple robot systems, Robotics and Autonomous Systems, Volume 43, No. 2-3, 2003

[6] Dubois, D., Lang, J., Prade, H., Possibilistic Logic, in Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning, Oxford University Press, Oxford, 1994.

[7] Dudek, G. and Jenkin, M., Computational Principles of Mobile Robotics, Cambridge University Press, 2000.

[8] Gärdenfors, P., Knowledge in Flux, A Bradford Book, MIT Press, Cambridge, MA, 1988.

[9] Gärdenfors, P., Conceptual Spaces, A Bradford Book, MIT Press, Cambridge, MA, 2000.

[10] Gärdenfors, P., How Homo became Sapiens, Oxford University Press, 2003.

[11] Gärdenfors, P. and Williams, M-A, Reasoning about Categories in Conceptual Spaces, Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 2001.

[12] Gärdenfors, P. and Williams, M-A, Building Rich and Grounded Robot World Models from Sensors and Knowledge Resources: A Conceptual Spaces Approach, Proc of International Symposium on Autonomous Minirobots for Research & Edutainment, 2003.

[13] Harnard, S., 1990, "The symbol grounding problem", Physica D, Vol.42, pp.335-346.

[14] Karol, A., Nebel, B., Stanton, C., and Williams, M-A, Case-Based Game Play in the RoboCup Four-Legged League: Part I The Theoretical Model, Robot Soccer World Cup VII Series: Lecture Notes in Computer Science, Vol. 3020, 2004.

[15] Karol, A. and Williams, M-A., Distributed Sensor Fusion for Object Tracking, in the Proceedings of the International RoboCup Symposium, Springer Verlag , 2005 (in press)

[16] McCarthy, J., "Circumscription - A form of Non-monotonic Reasoning", Artificial Intelligence, Vol. 13 pg. 27-39, 1980.

[17] McCarthy, J., Elaboration Tolerance http://www-formal.stanford.edu/jmc

[18] McCarthy, J. and Hayes, P. J., 1969, Some Philosophical Problems from the Standpoint of Artificial Intelligence, in D. Michie (ed), Machine Intelligence 4, New York: American Elsevier.

[19] Newell, A. (1980) Physical Symbol Systems. Cognitive Science vol. 4, 135183.

[20] Schank, R. C. and Abelson, R. P. (1977) Scripts, Plans, Goals, and Understanding. Lawrence Erlbaum.

[21] Searle, J. (1980) Minds, brains and programs. Behavioral and Brain Sciences vol. 3, 417457.

[22] Sharkey, N. E. and Jackson, S. A. (1996) Grounding Computational Engines. Artificial Intelligence Review vol. 10, 6582.

[23] Stanton, C. and Williams, M-A., A Novel and Practical Approach towards Color Constancy for Mobile Robots, in the Proceedings of the International RoboCup Symposium, Springer Verlag , 2005 (in press)

[24] Williams, M-A., Applications of Belief Revision, in Transactions and Change in Logic Databases, edited by B. Freitag, LNCS 1472, Springer Verlag, 15 - 47, 1998.

[25] Tarski, A., 1956, "The Concept of Truth in Formalized Languages", Logic, Semantics, Metamathematics, Oxford: Oxford University Press.

[26] Williams, M-A, Transmutations of Knowledge Systems, in J. Doyle, E. Sandewall, and P. Torasso (eds), in the Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann Publishers, 619 - 629, 1994

[27] Ziemke, T., Rethinking Grounding, Perspectives from Cognitive Science, Neuroscience, Epistemology, and Artificial Life, pp. 8794, Austrian Society for Cognitive Science, ASoCS Technical Report 9701, Vienna, Austria, May 1997

Advanced Video Image Processing on Autonomous Mobile Robots

Hans Utz and **Ulrich Kaufmann** and **Gerd Mayer**
Neuroinformatics, University of Ulm
D-89069 Ulm, Germany
{hutz|kaufmann|gmayer}@neuro.informatik.uni-ulm.de

Abstract

Vision is one of the most valuable sensors of an autonomous mobile robot, but advanced robot vision systems are still rare. A major obstacle in applying advanced computer vision to robotics are the additional constraints, that need to be fulfilled in the domain of robot vision.

We propose that in order to make computer vision applicable to robotics, it needs thorough support from the robot's software architecture. A robot vision architecture needs to encapsulate the constraints of the application domain to keep a vision application flexible and maintainable.

This paper introduces the video image processing (VIP) framework, a software framework for multi threaded control flow modelling in robot vision. It discusses its design and implementation as well as an experimental evaluation of its performance in parallel, priorised image processing.

1 Introduction

Vision is one of the most valuable sensors for autonomous mobile robots. Cameras are relatively low cost and offer a huge and diverse set of information that can be used for very different sensing tasks. Unfortunately, there is a severe lack of advanced vision processing methodologies applied in todays robotic applications. Especially in highly dynamic environments and predominantly reactive scenarios, research is still focused on model based colour blob detection. In consequence, vision processing in robotics lacks flexibility and scalability, which makes it impossible to use such a vision system for different tasks and multiple scenarios. This hinders advances in the scientific view on the problem domain.

Applying advanced vision processing methodologies to autonomous mobile robotics is difficult, as the requirements of this application domain add a whole set of additional complexity to the original task of image understanding. For instance, image processing binds a lot of computational resources and most higher level image processing operations are difficult to apply within the timeliness constraints of a real-time reactive autonomous system. Addressing such issues for a robotics vision system requires extensive architectural support, which is not available in currently available image processing systems.

This paper introduces the video image processing (VIP) framework. A software architecture for real time oriented video image processing for autonomous mobile robots. We show how the framework enables and facilitates the use of computer vision methodologies within the heavily time-constraint environment of an autonomous mobile robot within a highly dynamic environment, like the RoboCup mid-size league.

The remainder of this paper is organised as follows. In the next section the challenge of doing computer vision on autonomous mobile robots and related work in this area is discussed. Our solution approach is then introduced in section 3 and illustrated by a short example in section 4. An assessment of the real-time oriented features of the VIP framework is presented in section 5 before the paper ends with application examples, conclusions and the indication for future work.

2 Image Processing on Autonomous Mobile Robots

Vision systems for mobile robots bring together the two very challenging problem domains of image processing and autonomous mobile systems. E.g. most of the state of the art computer vision algorithms are computationally rather expensive, even when efficiently implemented. So a very careful assessment of their individual applicability is necessary. This on the other hand often discourages experts in computer vision to work on robot vision, as most of the advanced algorithms seem to be ruled out per se by timing constraints. In consequence solutions in robot vision are often: (1) hard coded quick hacks, that try to enable micro optimisations by doing multiple operations at once, (2) heavily model based or heuristic, exploiting special circumstances with little validity despite the one scenario they are targeted for, (3) in consequence hardly maintainable and little flexible.

So to mediate between the partially contradictory requirements of advanced vision processing in a real-time constraint environment, proper conceptual support from the vision processing architecture is necessary, to encapsulates the vision application within this application domain. In order to better understand the different requirements that need to be supported, we first take a brief look at the two problem domains.

2.1 Computer Vision and Image Understanding

The basic concept of computer vision is the application of operators to image data, such as the conversion of a colour image into gray-scale, or filtering the image for edges. Often operations transform more then one input image into a new output image as e.g. a Canny edge detector [Canny, 1986] usually needs two images which are convolved using a horizontal respectively a vertical Sobel operator. Other operators may use the same image result from different time stamps as for example a operator using two timely consecutive images to detect the optical flow [Horn and Schunck, 1980].

More sophisticated operations do not only cover filter-like processing steps, but all possible input-output mappings in general. So the result of a computer vision operation don't have to be again an image but can be every possible data as e.g. a colour histogram, a similarity value between two images or any other image statistic measure.

Sequences of such image operators reveal features within the image that can be used to identify regions of interest (ROIs). So filter don't need to work on the whole image but only on parts of the image. This is done either to speed up the processing loop or to be sure not to tamper the result with unwanted image structures from outside the region. Further operators derive image features from these ROIs that enable a reliable object recognition. Various feedback loops such as integration over time [Kalman, 1960] can speed up processing and improve classification results.

2.2 Robot Vision

Performing the above sketched operations on an autonomous mobile robot on the video image stream of the robots camera(s) within a medium sized robotics application adds a whole bunch of additional challenges to the problem set.

Efficient organisation of control and data flow. Video image processing on a mobile robot is usually sensor triggered and is started as soon as a new image is available to the robot as an image taken one second before does not necessarily resemble anymore the actual situation in a dynamic environment. At the same time, the performed processing needs to be demand driven, to not misspend the available computational resources.

Parallel and asynchronous evaluation. More and more robots are equipped with multiple cameras for stereo vision, or to extend their field of view. Multiple image sources, but also dual CPU boards as well as the upcoming hyper-threading and multi-core processor technologies call for asynchronous, parallel processing capabilities. Multiple image sources allow for interleaving processing, and the true parallelism of the advanced hardware features stay unused by single-threaded applications. The actual challenge however, lies in the proper synchronisation between different image processing tasks for the fusion of their results.

Timeliness and resource management. Due to the computational cost of most image operations, and the fact that the CPU is also used by other concurrent tasks of the system, the available processing power will usually not be enough, to perform all possible evaluations on every single image. In order to still meet the timeliness constraints of the reactive systems, different perceptual tasks (e.g. obstacle avoidance and face recognition) need to be properly priorised. E.g. the data for obstacle avoidance needs to be evaluated as often as possible, while the face recognition for greeting known pedestrians can be evaluated whenever some CPU cycles are left. Additionally, not all image processing tasks have to be performed over the whole time. The robots' situatedness enforces the use of special vision routines for different purposes.

Communication of results. Last but not least, images as well as extracted symbolic information of objects need to be accessible to the other modules of the robot software. Interfacing is an issue in the context of image processing on autonomous robots, as the information requested by client modules usually determines which information needs to be extracted from the image in a given situation. Robot applications, e.g. multi robot scenarios are most often distributed and therefore support for communication in a distributed environment has to be available, too.

2.3 Related Work

Common vision related architectures and publications can be roughly divided into three types: subroutine libraries, command languages and visual programming languages.

Subroutine libraries are the most commonly used ones. They mostly concentrate on the efficient implementation of image operators. Therefore they consist of normal functions, each responsible for a different image processing operation. Classical examples are e.g. the well known SPIDER system [Tamura *et al.*, 1983] or NAG's IPAL package [Carter *et al.*, 1989] written in C or Fortran. More recent approaches are e.g. LTI-Lib [lti,] or VXL [vxl,] which both are open-source, written in C++ and consist of a wide range of operations, ranging from image processing methods, visualisation tools and I/O functions. The commercial Intel Performance Primitives

[ipp,] are an example for highly (MMX and SSE) optimised processing routines with a normal C-API. What they all have in common is there lack of support for some kind of flow control support. Yet another collection of mutex or semaphore helper classes and some kind of thread abstraction is the maximum of assistance for this.

More advanced command languages for image processing are mostly implemented as scriptable command line tools, that a developer can use to direct the vision package. In case of the imlib3d package [iml,], the image processing operators can be called from the Unix command line, the CVIPtools [Umbaugh, 1998] are delivered with an extended tcl command language. So both packages have the ability to include conditional and looping facilities. But again the programmer not only has a flexible way of complete control over the system, but also the full liability over the processing cycle. Additionally the scripting approach makes it hard to meet the required performance constraints of this application domain.

The most sophisticated solutions are the visual programming languages. They allow the user to connect a flow-chart of the intended processing pipeline using the mouse. They combine the expressiveness and the flexibility of both above groups. Often they contain not only a real mass of image processing functions and statistical tools, but also a complete integrated development environment. Most of these systems are commercial products. One of the most advanced one is VisiQuest (formerly known as Khoros/Cantata). According to there web site, it supports distributed computing capabilities for deploying applications across a heterogeneous network, data transport abstractions (file, mmap, stream, shared memory) for efficient data movement and some basic utilities for memory allocation and data structure I/O.

To the best of our knowledge, there is no image processing framework, that combines all of our above described features like processing on demand of complete parts of the filter tree in a flexible yet powerful way, making the system suitable for a wider range of image processing tasks, like e.g. active vision problems on autonomous mobile robots.

3 Solution Approach

The principal idea of the VIP framework is to manage the control flow and organise the data flow of the vision application, for a clean separation of the two problem domains. That is, the vision application programmer only needs to implement the individual image operations (if not already available in form of a library) and direct the data flow for the target application. The VIP framework then executes the implemented code as the execution logic implies.

The basic processing unit is called a filter. This denotes not only a (non-)linear image transformation function like a Sobel operator, but every input-output mapping such as a neural classificator on image features. While the control flow is evaluated in a tree in depth first order, the data flow is much more flexibly organised as a directed acyclic filter graph (DAG). The framework ensures the correct evaluation order. Freely definable so-called meta-information, such as a list of regions of interest, histogram values etc. can also be passed through the DAG to successor filters. This actually extends each filter instance to a general image processing node.

Support for the intrinsic problems of robotic vision is supplied on the basis of the configurability and adaptivity of the framework and its execution logic, by special purpose filters and also by additional development tools. VIP is currently implemented as a C++ white box framework for Linux platforms.

3.1 Robotics Support

To prevent excessive polling or context switching between waiting threads the framework performs sensor triggered evaluation of filters. In order to maximise performance in this highly time constraint environment, VIP keeps track of which filters are actually queried by client modules. Based on this connection management, a dynamic graph pruning is performed to process only the minimal required filter tree. If a client module connect to a new filter, the filter is guaranteed to be part of the processing tree, as soon as the next image becomes available. The integration of the VIP framework into the middleware Miro [Utz *et al.*, 2002] provides support for network transparent as well as co-location optimised access to images or higher level sensory results from the filter DAG to client applications. For co-located image queries a shared memory based approach is used with zero-copying.

3.2 Source Nodes of a Filter DAG

Video devices are also modelled as filters within the framework and form the root node of a processing tree, that is source nodes in data flow graphs. The framework supports various camera connections such as BTTV, IEEE 1394 and USB-cameras and also multiple cameras in parallel. Each processing tree its executed within its own thread and is processed in parallel with other source nodes, while the data flow can stay connected. The framework then ensures appropriate synchronisation between the image streams. Note that, as the framework takes care of synchronisation, developers do not need to worry about locking issues and the right usage of synchronisation primitives.

Additional processing trees can be added to decouple time-consuming image operations (a slow path), that can not be performed at the full frame rate of the input source, from fast image evaluations, needed at full framerate for reactive tasks in the robotics application.

3.3 Real-time Constraint Image Processing

As one of the dominant features of robot vision is the timeliness constraint, VIP integrates multiple concepts for real-time processing. Each processing tree can be executed with its own thread priority and scheduler choice,

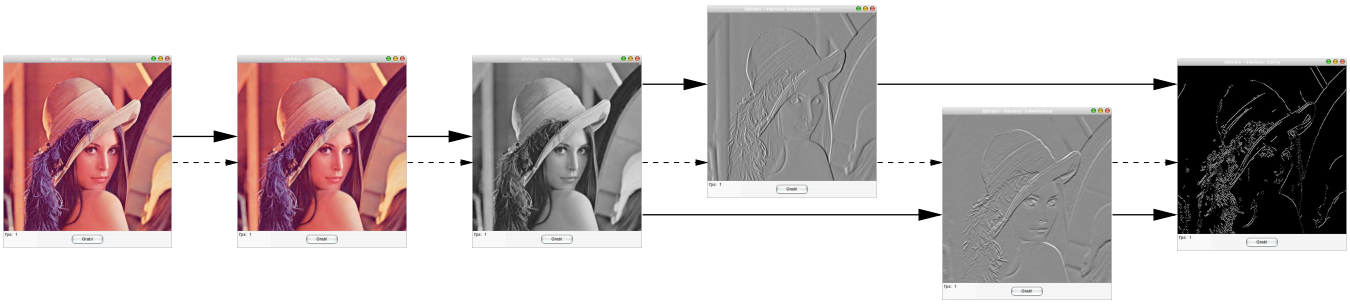


Figure 1: Original image, intermediate processing steps (blurred, grayed and convolved images) and resulting edge detection. The thick, solid lines denote the data flow, the thinner, dashed lines the control flow.

which is directly mapped on the OS-native process scheduler by the framework. This is necessary to minimise jitter and ensure correct priorisation, especially under high load situations. Additionally detailed timing statistics are provided for each filter. Different models for synchronisation of filters between different processing trees can be used to either optimise synchronisation of image sources (stereo vision) or minimise locking overhead and context switching between threads (slow/fast path processing).

3.4 Development Support

Applications in robot vision require extensive testing and tuning of filter configurations. Therefore VIP provides various concepts to ease the development process. The extensive use of the middleware provided configuration management support allows to specify meta-information about newly developed filters for various means, especially the flexible specification of filter graph configurations by the help of an XML-based description language. Such configurations can be built conveniently under a graphical user interface, as illustrated in section 4. Also, every filter, and therefor every intermediate result, can be queried (e.g. for visualisation) by simple assigning it a name for the according interface. The middleware integration also enables to change filter parameters on the fly from client applications in reaction to changes in the environment. By exchanging the physical video device for an image file set based virtual device that replays a stored image stream, the whole processing tree can equally used on- and offline.

4 Example Configuration

The above described feature set of the VIP framework is best understood by a small illustrative example. Figure 1 illustrates the derivation of an edge image from the classical test image of computer vision. The original image is Gaussian blurred and transformed into a grey image. Then a horizontal and vertical Sobel operator is applied and in the last step the Canny operator is applied. The screenshots are taken from the generic inspection tool. Meta-information is not provided by these simple filters. The data flow and control flow is are illustrated in figure 1. The thick, solid lines denote the data, while the dashed lines illustrate the control flow.

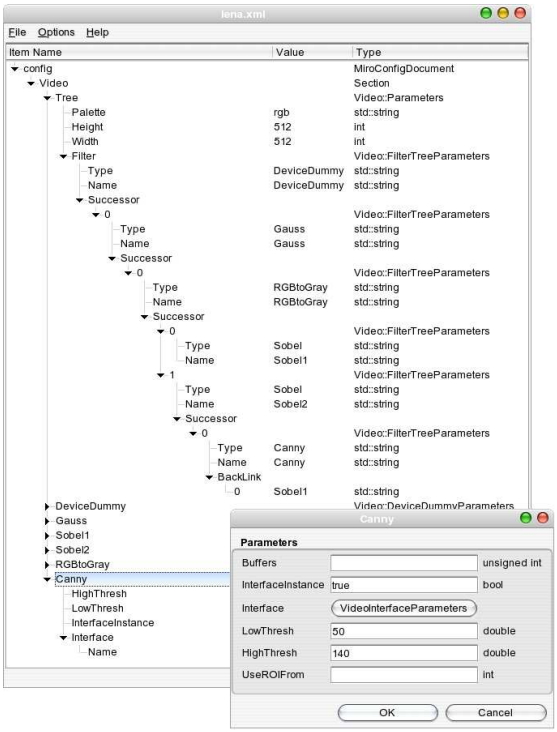


Figure 2: Graphical user interface illustrating the processing steps for a simple edge detection (lower window). The upper dialog shows the parameter configuration window for the Canny filter.

In Figure 2, this configuration is shown in the graphical user interface for the filter graph configuration. In addition of the control- and data-flow also the parameters (such as the threshold for the canny operator) can be edited in typesafe dialog fields. The parameter management framework provided by the underlying robotics middleware MIRO allows to add user defined filters with their specifications for filter parameters and filter meta-information for use by the configuration editor and the runtime-inspection tools.

5 Performance Assessment

A critical part of robot vision is the timely processing of image data. The VIP framework does not try to provide faster implementations for standard image operations, as sufficient libraries for this purpose exist. These can be easily utilised for the use by VIP, as done for IPP in our applications. Instead this framework concentrates on improving the responsiveness of a vision application, by allowing for proper priorisation and synchronisation of image processing tasks with parallel and asynchronous control flow.

A typical use case for the processing of multiple filter trees, is the combination of a fast path with an asynchronous slow path of vision processing, which then needs correct priorisation. We therefor assess in this section the capabilities of the framework to correctly preserve processing priorities under high-load situations.

The typical scenario would be one camera-synchronous processing tree that runs at full frame rate and extracts sensory information for the reactive control module and one or more asynchronous processing trees, that are connected to the data flow of the first tree and perform time-consuming computations not possible at full frame rate, extracting information for higher level cognitive processes with relaxed timing constraints.

The configuration of the VIP module for this experiment consist therefore of one high priority tree with the camera as source node, running with a round robin real-time scheduler (the fast path) and one, resp. two low priority asynchronous processing trees that are connected to the camera tree, running with default priority (the slow paths). The configuration is illustrated in figure 3. The low priority load is increased incrementally in the experiment. In the first run, the synchronous processing tree is run alone. In the second run one low priority processing tree is added to the configuration, but still all processing threads can be completed at frame rate (30Hz). In run three a second low priority tree is added and the system load reaches saturation. The results are compared against the equivalent setup without priorisation.

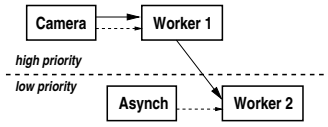


Figure 3: The filter configuration of the experiment.

Table 1 shows statistics on the overall time, the different processing trees need for completion. In the unpriorised configuration, the completion time of the camera-synchronous tree drops significantly in the third configuration, as the thread is preempted before completion to perform work on the other processing trees. This would cause significant delay for the consumers of this sensor information (e.g. the reactive control unit).

Figure 4 illustrates this effect by plotting the individual timings for 100 runs of the fast path. While the priorised processing three still runs with predictable completion time, the timings of the unpriorised configuration worsen significantly under high load.

Processing tree	Priorised		Unpriorised	
	Mean	Std. Dev.	Mean	Std. Dev.
Fast path only				
fast path	7,17	0,035	7,18	0,052
Medium load				
fast path	7,22	0,017	7,26	0,479
slow path 1	30,46	25,197	8,32	10,000
High load				
fast path	7,22	0,025	8,55	3,192
slow path 1	53,31	69,921	60,77	94,601
slow path 2	57,66	5,065	56,84	5,240

Table 1: Different timing statistics for the individual processing tree in both, the priorised and unpriorised case. The values are stated in milli-seconds.

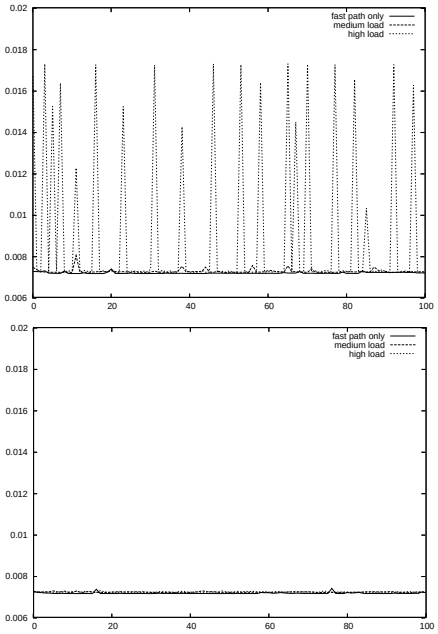


Figure 4: Timings of the fast path with none, one and two slow paths running concurrently. The left plot shows the unpriorised case, whereas the right plot shows the fast path running with enabled real-time scheduling.

Another visualisation of these preemptions is shown in figure 5. From the third setup a small section of the interleaving processing of the three processing trees is plotted. Each tree is assigned a different colour. Yellow was chosen for the fast path, the slow paths are coloured read and blue. To fit into the column, a new line is added each time the processing of both slow paths is finished. The completion of a processing tree is marked with a black box at the end of the coloured bar. While the real-time scheduled fast path always runs to completion before its processing stops, it is occasionally interrupted without priorisation. Additional load on the system will worsen this effect. A medium complex robotics application performs many other tasks in parallel to image

processing, which will contribute to the latencies in high load situations.

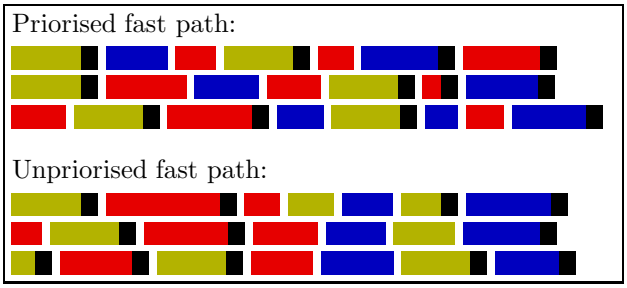


Figure 5: Illustration of alternation between the different running processes with a priorised resp. unpriorised fast path.

6 Application

The VIP framework is successfully in use in different robotic scenarios, such as biologically motivated neural learning and object classification [Fay *et al.*, 2004] and reliable high speed image processing in the RoboCup mid-size league [Kaufmann *et al.*, 2004][Mayer *et al.*, 2004] . It also provides the basis of a large filter library shared between the different scenarios.

It's application in RoboCup consists of a dual camera setup, combining a directed camera for object classification with an omni-directional camera for obstacle avoidance and near range ball tracking. The application combines 66 filters with 108 connections. One of the fastest path, a simple colour based football goal detection takes around 4 msec to complete, while one of the slowest paths (a complete neural robot classification) needs around 20 msec on average when seeing one robot per image (measured on a 1.4GHz Pentium M processor).

Currently, priorisation of and synchronisation between processing trees is not yet used by the RoboCup application. The use of an omni-directional camera as well as the real-time features of the framework were both added fairly recently. But the promising results of section 5 will definitely encourage their prompt application.

7 Conclusions and Future Work

This paper discusses the difficulties of meeting the requirements of th application domain, when applying advanced computer vision to autonomous mobile robots in dynamic environments. The VIP framework is introduced, which was designed to facilitate the application of computer vision in robotics, by managing the additional challenges of robot vision in this domain. The middleware based framework approach especially enables to support roboticists with the non-functional aspects like configuration, priorisation and performance assessment. Extensive development support is provided in the form of parameter management, GUI-based configuration as well as generic inspection of images and meta-information. The performance assessment confirmed the suitability of the design for real-time constraint processing, as mandatory in highly-dynamic environments.

Future work will be directed in two different directions. The first is to assess carefully the optimisation potential for used system resources, especially memory consumption. Improving cache hit rates for instance can tremendously increase the performance of image algorithms and control flow and memory management have a significant impact on it. One possibility is to switch to in-place processing of filters, if the filter and the filter graph configuration allow it. The other direction is to connect the priorisation of the image processing tasks with the real-time capabilities of the underlying distributed systems middleware (RT-CORBA), to ensure end to end quality of service between sensory and actuatory processes.

Acknowledgements:

The work described in this paper was supported by DFG SPP 1125 within the project *Adaptivity and Learning in Teams of Cooperating Mobile Robots*. MIRO, including its video image processing facilities and documentation, is available at `http://smart.informatik.uni-ulm.de/MIRO/`.

References

[Canny, 1986] John F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 679–698, 1986.

[Carter *et al.*, 1989] M. K. Carter, K. M. Crennell, E. Golton, R. Maybury, A. Bartlett, S. Hammarling, and R. Oldfield. The design and implementation of a portable image processing algorithms library in fortran and c. In *Proceedings of the 3rd IEE International Conference on Image Processing and its Applications*, pages 516–520, 1989.

[Fay *et al.*, 2004] Rebecca Fay, Ulrich Kaufmann, Friedhelm Schwenker, and Günther Palm. Learning Object Recognition in a NeuroBotic System. In Horst-Michael Groß, Klaus Debes, and Hans-Joachim Böhme, editors, *3rd Workshop on SelfOrganization of AdaptiVE Behavior SOAVE 2004*, pages 198–209. VDI, Düsseldorf, 2004.

[Horn and Schunck, 1980] B.K.P. Horn and B.G. Schunck. Determining optical flow. Technical report, Massachusetts Institue of Technology, 1980.

[iml,] imlib3d. Available via http://imlib3d.sourceforge.net/.

[ipp,] Intel performance primitives (ipp). More information on http://www.intel.com/software/products/perflib/.

[Kalman, 1960] Emil Kalman, Rudolph. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[Kaufmann *et al.*, 2004] Ulrich Kaufmann, Gerd Mayer, Gerhard Kraetzschmar, and Günther Palm. Visual robot detection in robocup using neural networks. In *Proceedings of RoboCup-2004 Symposium*, Lecture Notes in Artificial Intelligence, Berlin, Heidelberg, Germany, 2004. Springer-Verlag.

[lti,] Lti-lib. Available via http://ltilib.sourceforge.net/doc/homepage/index.shtml.

[Mayer *et al.*, 2004] Gerd Mayer, Ulrich Kaufmann, Gerhard Kraetzschmar, and Günther Palm. Neural robot detection in robocup. In *27th German Conference on Artificial Intelligence (KI2004), NeuroBotics Workshop*, University of Ulm, 2004. to appear in the book "Biomimetic neural learning for intelligent robots".

[Tamura *et al.*, 1983] H. Tamura, S. Sakane, F. Tomita, and N. Yokoya. Design and implementation of spider-a transportable image processing package. *Computer Vision, Graphics and Image Processing*, 23(3):273–294, 1983.

[Umbaugh, 1998] Scott E. Umbaugh. *COMPUTER VISION and IMAGE PROCESSING: A Practical Approach Using CVIPtools*. Prentice Hall, June 1998.

[Utz *et al.*, 2002] Hans Utz, Stefan Sablatnög, Stefan Enderle, and Gerhard K. Kraetzschmar. Miro – middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, 18(4):493–497, August 2002.

[vxl,] Vxl. Available via http://vxl.sourceforge.net/.

.

Cognitive micro-Agents: individual and collective perception in microrobotic swarm

S. Kornienko, O. Kornienko, C. Constantinescu, M. Pradier and P. Levi,
Institute of Parallel and Distributed Systems, University of Stuttgart,
Universitätsstr. 38, D-70569 Stuttgart, Germany

Abstract

In this paper we present the research results in the field of perception for real microrobotic swarm. The proposed hardware and software solution uses IR-based reflective measurement for individual perception and the Dampster-Shafer evidential reasoning for hypothesis refinement in collective perception. Especial attention is paid to a reliable identification of encountered geometries and a reduction of local communication. Based on the experimental results we make a conclusion about cognitive capabilities of individual microrobots and the whole swarm.

1 Introduction

Miniaturization represents now a very important trend in many areas of research. Molecular-scale or nanotechnological devices jumped from science-fiction novels to research papers. Even the today's technology allows creating complete autonomous systems, such as robots, in the size of 1 mm^3. As demonstrated by a progress in the I-Swarm project [I-Swarm, 2003 2007], the swarm of thousand such microrobots gets reality as well as come into the reality impressive applications of this technology.

The scaling down of the hardware influences almost all important parameters of microrobots, as e.g. running time, communication distance and channel capacity, computational power, movement and so on. However we ask ourselves about *"intelligence" of such a microrobot; is it also scaled down so that we get finally some "stupid moving thing"* [Kornienko *et al.*, 2004] ? Since many years there exists in the scientific literature the opinion that "artificial intelligence" for very small systems drifts towards "collective artificial intelligence", like those in social insects [Bonabeau *et al.*, 1999]. For collective systems the "individual intelligence" gets some pre-intelligence form. The question is *which minimal degree of individual intelligence does allow growing "collective intelligence"* ?

In this paper we consider such an aspect of cognitive intelligence as perception. In a microrobotic swarm the size of a robot is essentially smaller than the size of most environmental objects. The recognition of these objects is primarily done in collective way. However here we encounter the same question about "individual aspects" of collective perception. Is a microrobot able to provide enough sensory information for the collective perception ? Which sensing and processing steps should be done individually and which collectively ?

For answering these questions we designed and prototyped a sensor system for our own test microrobot. This is is actually larger as envisioned in I-Swarm project however is very cheap and easy to reproduce without specific equipment. Based on this prototype we can investigate questions about "individual/collective intelligence" so that the results, e.g. principles, methods, algorithms can be later implemented in the $1mm^3$ robot. The size of the sensor system is $23\times23\times5$mm. It uses the Megabitty board ($23\times23\times2$mm) with Atmel AVR Mega 8 microcontroller, having 8 kB ROM and 1 kB RAM [Megabitty, 2005]. Besides perception, the board supports 6-directional robot-robot and host-robot communication, with the average communication radius 0-140mm (with special solution for deadlock reduction) and a maximum of 300mm. The sensors are also used for proximity sensing in navigation. The communication subsystem for a large microrobotic swarm is described in [Kornienko *et al.*, 2005]. In this paper we present the development of the perception system for the sensor board and the problems of individual and collective perception in microrobotics.

The rest of paper is organized as follows. In the next two sections the problem of individual perception and the development of IR-perception system are described. Then, we discuss the nonlinearities of this perception and the algorithms of feature extraction and surfaces classification. The last two sections are devoted to the problem of collective classification and preliminary experiments.

2 Problems of individual perception in microrobotic swarms

As mentioned before, the recognition of large objects by small microrobots is primarily performed in a collective way. However the prerequisite for collective perception is the surface identification and classification that is performed by each microrobot. We name further this process as individual perception. From the collective perception point of view the following types of surfaces are required to be identified:

1) *infinite-size surfaces* (from a robot's viewpoint), as huge objects or borders;

2) *finite-size surfaces* (a microrobot has to calculate the visible size of a surface) which are classified, at least, into small, medium and large;
3) *convex and concave corners*;
4) *2-side and 3-side concave surfaces*;
5) *one-surface/many-surfaces geometry*.
Additionally, the microrobots have to be able to perform the following activities:

1) *detection of holes (gangways) in surfaces*;
2) *classification of the perceived surfaces into defined classes and providing a probability of correct classification*;
3) *recognition of robot's own position in relation to a corner* (left/right from a corner) or even its own slope to a surface.

When each robot identifies the surface in its own sensing areal, further collective processing consists in fusing individual observations into many hypotheses and collective identification of most probable hypothesis about the observed object (see also [Ye *et al.*, 2002]).

Returning to the issue of individual perception, we identified the following implementation possibilities:

1) **vision-based** way by e.g. using some small micro(faced)-cameras;
2) **reflection-based** way by using laser or infra-red light, ultra-sound etc.;
3) **wavelength-based** way such as color sensing;
4) by using **specific** chemical, temperature, vibration, magnetic and so on sensors (we do not consider them here).

The vision-based way represents the most information intensive mode. However its application in microrobotics has several difficulties caused by very limited computational capabilities and small memory. Algorithms of image processing are difficult to be implemented in this hardware. Moreover due to very small size we prefer to use the same sensors for navigation (proximity sensing and obstacle detection) and communication (robot-robot and host-robot) purposes as well. Finally, the geometrical features from deep images are essentially more useful for collective perception than edges and regions from camera's grey-value images. Thus, the vision-based as well as wavelength-based ways, although they have found a large application in mini- and usual robotics, unfortunately are less useful here. The reflection-based perception uses the principle of sending and receiving a signal, that can be also used for navigation and communication.

Considering different alternatives for reflection-based perception we focus primarily on laser, electro-magnetic/inductive and infra-red systems. Ultra-sound systems do not satisfy the size limitation. Though the laser provides the most exact measurement and long range, there are several technical difficulties to use it with the microrobot. So, choosing between electro-magnetic/inductive and infra-red systems, we prefer the last ones due to their simplicity, relative long working range and small energy consumption.

Generally, the IR-systems are recently dominant in so-called small-distance-domain, as e.g. for communication between laptops, hand-held devices, remote control and others. The IR-solution is not new in robotic domain, see e.g. [Kube, 1996], [Suzuki *et al.*, 1995]. There are many approved schemes or even industrial sensors for IR-communication. However the fusion of perception and communication using IR-devices does not find too many applications, perhaps because of a high nonlinearity of IR-based perception and availability of more appropriate solutions in the domain of usual robotics. Therefore the microrobotic domain of integrated IR-solution (perception, communication, navigation) is more or less unexploited.

The IR-based perception consists on sending an IR radiation beam and receiving the reflected light. The intensity of this light contains information about the geometry of reflecting surface (primarily a distance between IR-recever/emitter and surface). As mentioned, the IR-based perception is highly nonlinear. The most large influence on accuracy of perception exerts the resolution of the distance sensor. In the center of radiation ray, the intensity of IR radiation is highest. Closely to the bounds of this ray, this intensity becomes gradually degraded (Figure 1). The main component of a reflecting light consists of the energy of the central radiation stream. However low-intensity "secondary streams" spread the reflecting light so that object's edges and gaps between objects get non-recognizable. With a poor resolution of distance sensor, small geometrical elements cannot be perceived and so cannot be used as features for recognition. Therefore

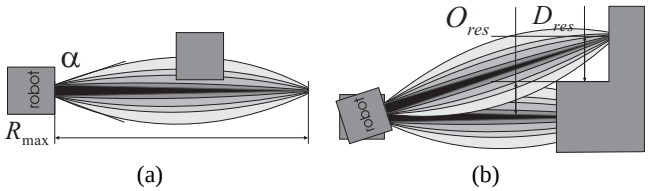


Figure 1: *Perception by using the IR beam, R_{max} recognition distance, D_{res}, O_{res} distance/object resolution.* **(a)** *Thickness of radiation beam and influence on the size measurement;* **(b)** *Nonlinearity in the identification of many-surfaces geometry.*

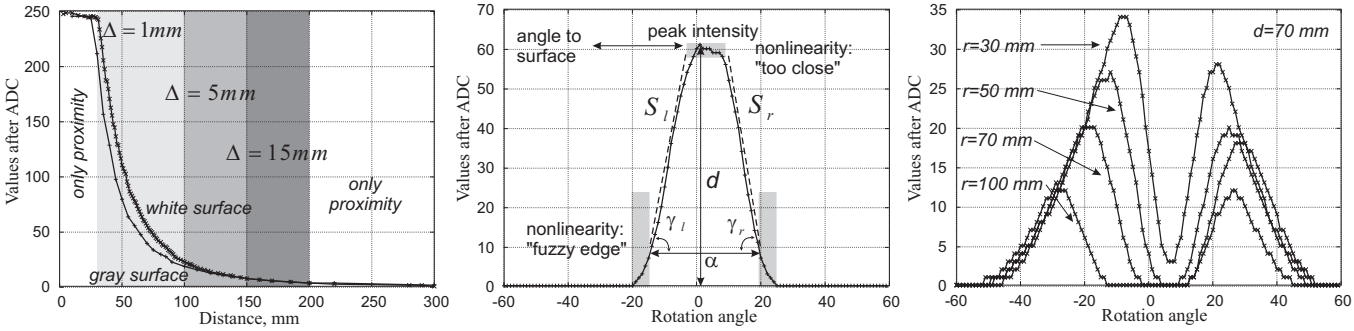
for perception are suitable only such IR-emitters that have an as small as possible opening angle of the beam.

Secondly, the accuracy of measurement depends on the distance to a reflecting surface[1]. In Figure 2(a) we demonstrate this effect for the developed sensor system. Nonlinear accuracy essentially influences the further recognition of features.

The reflecting light is also very sensitive to the color of reflection object. In Figure 2(a) we show the distance measuring values for white and gray objects. Further in experiments we use only white color objects. The distance measuring also depends on the object's slope to a radiation ray. In Section 4 we discuss in detail these nonlinearities and suggest some approaches to absorb them.

Since we did not found a suitable integrated IR-solution for the microrobot, we decided to develop our own required hardware and the corresponding processing algorithms. In the next sections we describe them.

[1]The dependence between reflecting light and distance is also nonlinear however this problem can be easily solved by a look-up table or some approximation functions.



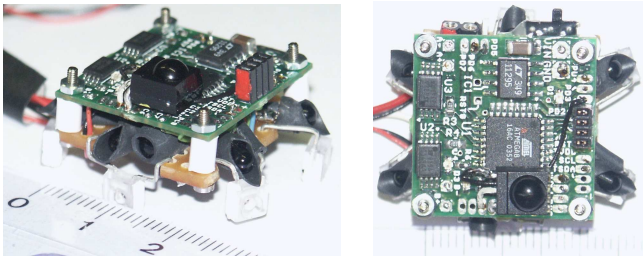
(a) (b) (c)

Figure 2: **(a)** *Dependency between ADC values of emitter voltage on phototransistor and the distance to reflecting object. Shown are values for the white reflecting object (white paper) and the grey reflecting object (grey cardboard);* **(b)** *The used features of IR-diagrams relevant for identifying the surfaces;* **(c)** *The "thickness effect" of radiation beam by scanning a gap with different size r. The distance between a microrobot and the gap is 70 mm.*

3 Development of the IR-based perception system

The main requirement on the IR-perception is given by as small as possible opening angle of the radiation ray. Additionally, IR-emitter has to provide a high energy beam, being able to get good deep images. Finally, IR-emitter and receiver should be able to work in a communication mode.

The perception system of the microrobot is a part of IR-system used for proximity sensing, obstacle detection, distance measurement and communication, as well (Figure 3). For the perception and objects recognition we use only the



(a) (b)

Figure 3: **(a)** *The megabitty board and the sensors board used in the prototype of a microrobot;* **(b)** *The 6-directional sensor system for directional communication and proximity sensing.*

distance measuring sensor, so that only this sensor is further considered. This sensor consists of a receiver with a wide opening angle (used also for communication and proximity sensing) and an emitter with as small as possible beam angle (used for perception and long-range communication). We utilize the Si phototransistor TEFT4300 (60^o, peak sensitivity 950 nm) and the high power GaAs/GaAlAs emitter TSAL6100 (radiant intensity >80 mW/sr, 20^o, the real opening angle is of 18-22^o, 950 nm). This combination is a result of many experiments with different sensors (over 30 pairs), with integrated receiver/emitter like SFH9201, as well as non-integrated ones. The TEFT4300-TSAL6100 pair demonstrated the best spectral coupling, the longest sensing distance and the acceptable nonlinearity of sensing. Although the IR-emitter is relatively large for the microrobot (8xϕ5 mm), the specific construction of the chassis allows to hide it inside the robot.

Since IR-emitter and receiver are non-integrated and are placed side by side in the chassis, they have to be optically isolated. The optical isolation of the emitter allows also reducing the opening angle of the beam up to 10-15^o (it reduces also a perception distance). However the main problem here is to provide similar optical characteristics of isolation for a large number of different microrobots in a swarm (to avoid later the problem of individual calibration of each microrobot).

The principle of object recognition is the following. As soon as a robot detects (by means of proximity sensors) an obstacle in front of itself, it switches on the high power IR-emitter and after 1ms delay (needed to get reliable reflecting light) measures voltage on the emitter of phototransistor. The dependence between emitter voltage (after ADC) and the distance to an object is shown in Figure 2(a). Generally, this sensor perceives distances up to 300 mm. However accuracy of measurement is different. For the pair *distance-accuracy* where Δ is the accuracy, we obtained the following values: 30-100 mm $\rightarrow$ Δ=1 mm, 100-150 mm $\rightarrow$ Δ=3-5 mm, 150-200 mm $\rightarrow$ Δ=10-15 mm and after 200 mm $\rightarrow$ Δ=30-50 mm. Therefore, the reasonable measuring distance for object recognition lies within 30 mm-100 mm (with the accuracy of 1-2 mm).

Not only the resolution of the IR-sensor is important for scanning the objects. During scanning, a microrobot turns on some degrees. The more exact is this turning, the more precise is the spatial resolution of sensor data. Microrobot does not possess any devices allowing to measure positions and orientation of chassis or wheels. Therefore there is only one way to rotate a robot, namely to turn the motors on and after some delay turn them off. This delay has to be so chosen, that a robot rotates on some fixed degree. The motors are controlled through the H-bridge SI9988, that can change a polarity of supplying current. Choosing normal polarity for one motor and inverse polarity for the second motor, the ro-

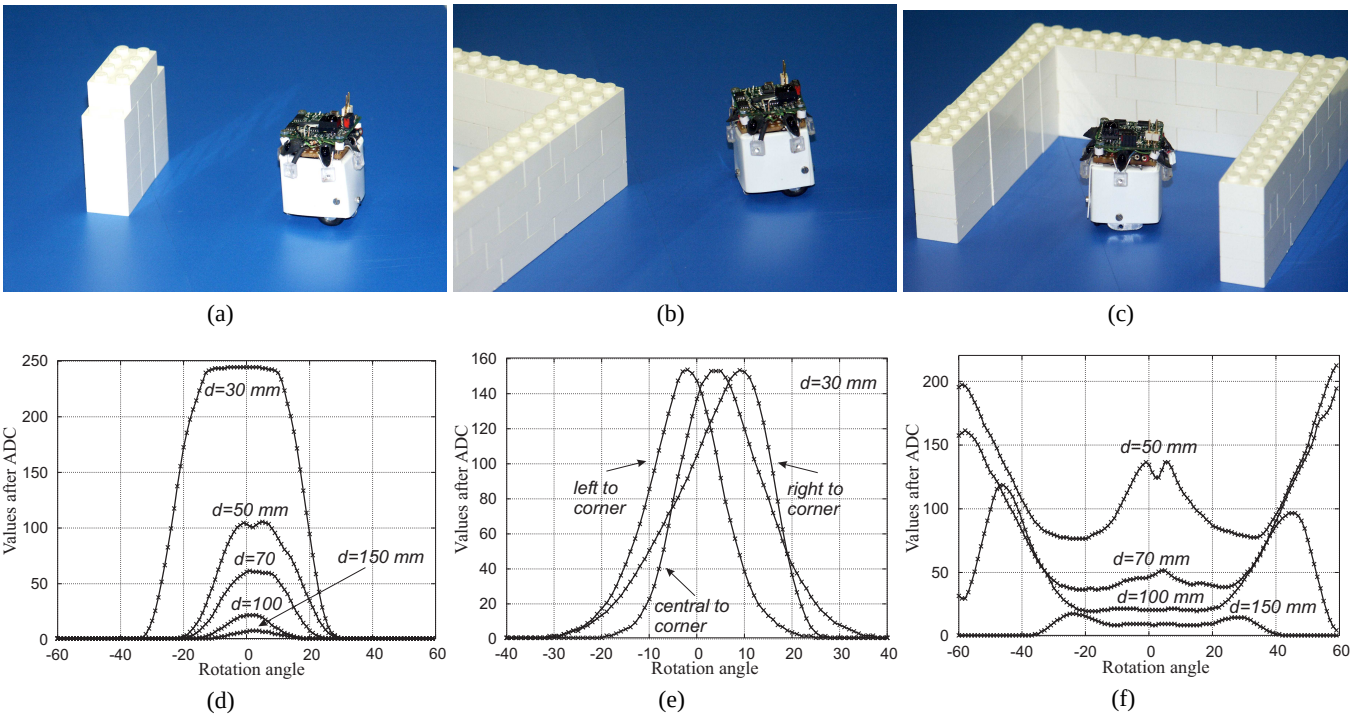


Figure 4: *"Jasmine", the prototype of the microrobot, scans different surfaces, where d is the distance to surfaces.* **(a)** *Scanning of the finite-size surface, object 48 mm;* **(b)** *Scanning of the convex surface;* **(c)** *Scanning of the 3-concave surface;* **(d)** *The IR-diagram for finite-size surface;* **(e)** *The IR-diagram for convex surface;* **(f)** *The IR-diagram for 3-concave sides surface of 95×95×95 mm;*

bot can rotate without changing its own position. In this way we get relatively shift-errorless deep images. After some tests we achieved the resolution and accuracy of rotation 1^o (taking into account different friction between weels/chassis and floor surface).

In our experiments, when a robot detects an obstacle on the distance of 70 mm $\pm$ 10 mm, it stops and then rotates 60^o left. After that it scans the obstacle with the distance sensor by rotating 120 ^{o} right. During this scanning it writes the obtained values of distances each 1 degree into an integer array. In this way we have 120 values describing a visible geometry of the encountered obstacle. In Figure 4 and 5 we demonstrate some geometries of encountered obstacles and the scanned surfaces.

4 Features extraction from IR-deep images

After performing the first experiments, we faced the following challenge: which features of the obtained IR-diagrams are relevant for identifying the geometry of the surfaces ? By analyzing the IR-diagrams in Figure 4 and 5, we find the following features as representative and useful in the IR-based individual perception (Figure 2(b)):

1. *The angle* α, which represents the scanning angle between the first visible edge and the last visible edge of the surface;

2. *The peak intensity of the diagram,* I_{max}. This corresponds to the maximal intensity of reflecting light and, in turn, to the minimal distance d between the surface and the microrobot. For the most types of surfaces (beside convex corners) this minimal distance is measured as a perpendicular to a surface. This feature allows calculating the visible size of a surface by using trigonometric relations;

3. *The left and right slopes*, denoted as γ_l and γ_r are useful for identifying the size-type of the surface (unlimited, big, medium, small). They are calculated as slopes of the approximation lines S_l, S_r. The slope denotes also the "degree of a distance decreasing" and enable us to identify the so-called "convex surfaces" that cannot be recognized in the trigonometrical way;

4. *The position of the "center" of the IR-diagram,* P_{imax} in relation to the scanning angle ("0", origin point on the X axis). Displacement of the center points to a slope between the front of robot and surface. In this way we can identify a directional orientation of the microrobot.

Now we formalize the nonlinearities mentioned in Section 2 and present their impact on the corresponding features:

1. *Nonlinear thickness* of the IR radiation ray and so different distribution between high-energy beam and low-energy beam. The first effect of this nonlinearity consists in spread edges (Figure 2(b)). This nonlinear effect can be absorbed by calibration. The second effect is shown in Figure 2(c). At scanning many-surfaces geometry (a gap between objects) a robot cannot reliable differentiate between 2-concave surfaces and surfaces that belong to different objects;

2. *Nonlinear measurement for small distances*. As known from other IR-distance measurement systems (e.g. [Caprari and Siegwart, 2003]), the maximal intensity of measurement

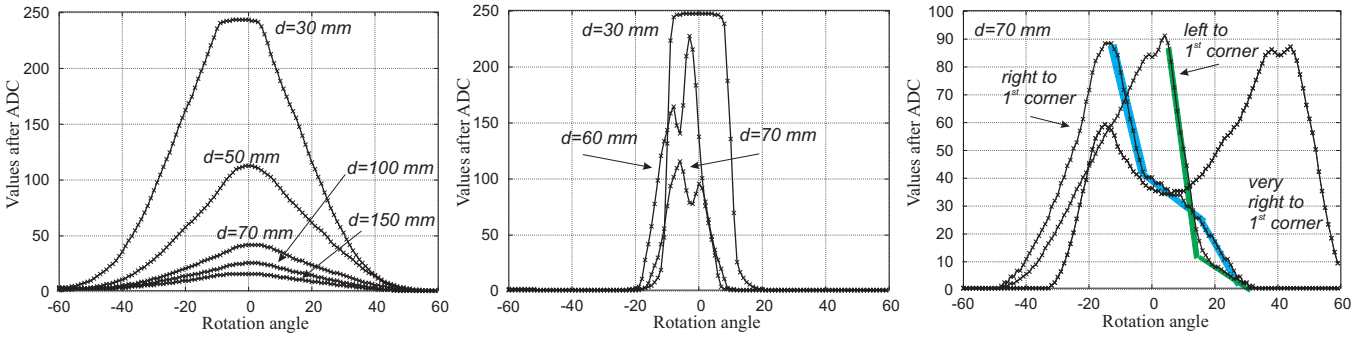


Figure 5: *IR-diagrams for different types of surfaces, d is a distance to surfaces.* **(a)** *"Infinite-size" surfaces with flat geometry;* **(b)** *Convex round (external diameter 125 mm) surface;* **(c)** *Many-surfaces geometry (1^{st} convex corner 122×60 mm and 2^{nd} concave corner 60×95 mm), robot positioned 70 mm before the middle part.*

lies in 10-25% before the front of IR-receiver, after that the intensity goes down (therefore small distances cannot be measured by these systems at all). Due to the specific restriction and the application of high-power GaAs/GaAlAs emitter, we removed this effect. However the surfaces that lie less then 40 mm away from a robot are represented only by values 245-250. In this way, for close measurement (30 mm) we get a flat horizontal diagram. Another undesired effect in small-average distances (40-70 mm) consists in a spontaneous decreasing of peak intensity (this is observable in all IR-diagrams in Figures 4 and 5). We cannot identify the nature of this nonlinearity and assume multiple IR-reflections as a reason for them;

3. *Nonlinear accuracy* of distance measurement. This requires nonlinear correction (it is done as a look-up table) of trigonometric relation in dependence of distance. However this nonlinearity is very "tricky". Even when a robot starts a measurement in the "good" area of 40-120 mm, a part of geometry can lie over 150 or 200 mm away. The effect of this nonlinearity appears in unreliable identification of many-surfaces geometry (Figure 5(c) "left to 1st. corner");

4. *Nonlinear rotation* of the robot. This can lead to different left γ_l and right γ_r slopes even for symmetric surfaces. The most easiest solution here is to calibrate γ_l and γ_r;

5. *Nonlinearity in measuring convex surfaces*. The identification of all types of convex geometries is performed by γ_l and γ_r. The difference between slopes for e.g. round objects (Figure 5(b)), convex corners (Figure 4(b)) and finite-size flat objects (Figure 4(a)) is small, moreover due to a nonlinear intensity diagram, these slopes change with distances. This problem has some basic character and we hardly belief that with all nonlinearities of IR-perception we are able to reliable identify the type of convex surfaces.

The main problem of these nonlinearities represents the necessity to maintain many look-up tables for corrections. This, in turn, is limited by a small memory of Atmel microcontroller. The assumption is that this problem can be solved in collective way. We can reduce the accuracy of individual recognition (so that to satisfy all hardware constraints) till such a degree which still allows a reliable collective recognition. Now, based on the discussed features and nonlinearities, we can briefly analyze the types of surfaces.

1. Surfaces with flat geometry. The flat type of geometry is primarily characterized by only one peak value on the IR-diagram. Finite-size surfaces are also characterized by large left and right slopes and scanning angle $\alpha \ll 120^\circ$, Figure 4(a). The size L_{vis} can be calculated as $2d\tan(\alpha/2)$, taking into account the "fuzzi edge" nonlinearity.

"Infinite-size" surfaces (Figure 5(a)) have small slopes of IR-diagrams and $\alpha \sim 120^\circ$. To absorb the nonlinearity of slopes for small and large distance, we apply the polygonal approximation [Pitas, 1993] and use in calculation the relation $\gamma_{\{r,l\}}/S_{\{r,l\}}$ instead of simple $\gamma_{\{r,l\}}$, where $S_{\{r,l\}}$ is the length of approximating line. In the performed experiments the probability of correct identification is very high and the accuracy of size calculation is of 5 mm (15 mm in the worst case).

2. Surfaces with convex geometry. Surfaces with convex geometry possess also only one peak value, however larger slopes then flat geometries. This type of geometry has to be identified before the calculation of size, which has no sense in this case. There are several types of convex geometry: convex corners and convex round surfaces (Figure 4(b)), convex many-surface geometry (can be recognized only collectively)(Figure 4(f)). We identify this geometry by $\gamma_{\{r,l\}}/S_{\{r,l\}}$ in the IR-diagrams. The difference between them points to a position in relation to a corner (left to a corner, right to a corner). The probability of correct identification of convex round geometry is very high, however convex corners are often classified as flat geometry. One approach to avoid this problem is the so-called "active exploration" (simple move towards the surface and scan again induces the appearance of a large "flat region" in the peak intensity which points to the flat type of geometry).

3. Many-surfaces and concave geometries. Concave geometries manifest primarily as multiple peaks in IR-diagrams. Based on the number of peaks we can differentiate between 2-concave (concave corners) and 3-concave sides geometry (Figure 4(c)). Concave many-surfaces geometries (Figure 4(b)) can be also classified by one robot. They have one peak value, however multiple left or right slopes. Many-

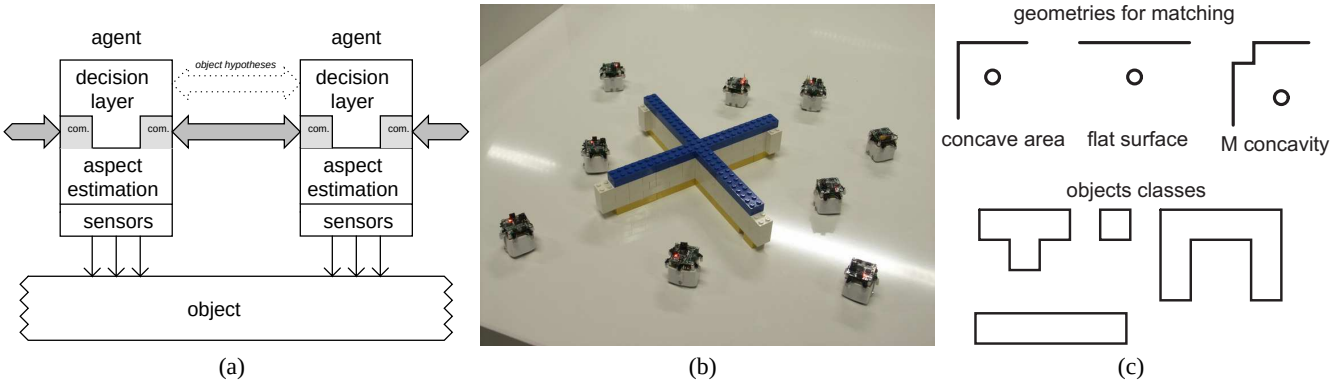


Figure 6: **(a)** *Distributed architecture for collective observation;* **(b)** *Spatial distribution of the robots around the observed object;* **(c)** *Geometries for matching and objects classes.*

surfaces geometry can also be composed from surfaces that belong to different objects. Generally, concave geometries can be identified with high reliability, however some fine differentiation between them is not always possible.

4. Estimation of probability. Since the robot cannot reliable classify the type of surfaces, it calculates a probability of correct classification. The calculation is done in the following way. We measure the possible values of α, d, $\gamma_{\{r,l\}}/S_{\{r,l\}}$, P_{imax} and estimate L_{vis} for all types of surfaces. The robot uses last square metrics to calculate the relation between the measured values and these presaved types. For collective perception a robot sends all possible classifications that have the probability over 30%.

Through the presented features of the IR-deep image we tried to classify several surfaces and to identify the classification probability as well, as base steps or components required for the individual perception.

5 Collective Perception

The described in the previous sections individual perception provides the sensor input for the collective perception. The approach for collective perception presented here proposes that each robot *talks* to its neighbors to exchange information about the surrounded object. In this task we limit ourselves only to the problem of *collective classification* [Pradier, 2005]. The robot possesses the objects models and have only to order the collective sensor input to one of the presaved model.

The distributed architecture for collective perception is shown in Figure 6(a). There is no privileged agent with a special role: all robots perform the same operations. The suggested method is homogeneous, i.e. all robots act the same and there is no need for a leader. Due to the homogeneous architecture the approach is robust, scalable, moreover new robots can join the team dynamically without any need to readjust any task assignment. Figure 6(b) shows how robots are deployed during collective observation. There are two possible implementations for the propagation of hypotheses: a single agent collects the information needed to identify an object by moving around it and performing the sensing operations; a single agent acquires local evidences and propagates hypotheses for the further fusion.

5.1 Object model

Given the limitations on the sensing capabilities of the robots, object classes can only be defined in terms of their geometries, as mentioned in Section 4. Figure 6(c) shows the 2D geometries of the four object classes which will be used subsequently. Once robots are situated around the object, they can estimate the local properties of the object as seen from their current positions called *viewpoints*. The actual measurement obtained from a viewpoint v can be noted as $S(v)$; $S : V \mapsto \text{feature vector}$ and represents the output of the distance sensors. Given an object class, it is possible to establish the expected sensor outputs for a number of views. A number of viewpoints $n_V^{K_i}$ for each object class K_i are chosen, along a trajectory situated in the center of the measurement domain, and noted as $\mathfrak{V}^{\mathfrak{K}_i} = \left(v_n^{K_i}\right)$. The corresponding expected measurements for objects of class K_i are $S\left(\mathfrak{V}^{\mathfrak{K}_i}\right) = \left(S\left(v_1^{K_i}\right), \ldots, S\left(v_{n_V^{K_i}}^{K_i}\right)\right)$. Therefore, the object model for a class K_i incorporates an ordered sequence of views for different *successive* positions around objects of that class. The starting position is arbitrary: only the ordering is relevant. The direction — clockwise or counterclockwise — can be chosen arbitrarily, but must be the same for all object models.

Additionally, object models include information about the reachability of different viewpoints, taking into account both geometrical constraints and the limitations imposed by the communication capabilities of the robots. It is noted as $W^{K_i} = \left\{ \left(v_j^{K_i}, v_k^{K_i}\right) \middle| \; v_k^{K_i} \text{ reachable from } v_j^{K_i} \right\}$. Finally, the corresponding distances between viewpoints in W^{K_i} are added to the object model, as $d_{\mathfrak{V}} : \mathfrak{V}^{\mathfrak{K}_i} \times \mathfrak{V}^{\mathfrak{K}_i} \to \mathbb{R}$.

The set of all canonical measurements — corresponding to sets of observable features, called *aspects* — in the model is noted $A = \left\{S\left(v_i^{K_j}\right)\right\}$ and its cardinality can be reduced by clustering the expected measurements. In that case, a sequence of canonical views could match several $(identity, position)$ pairs.

The goal of collective classification in a swarm of robots is

to estimate that class K_i the object being observed belongs to. When n_r robots are situated in an area surrounding the object (measurement domain) in positions $w_1, \ldots, w_{n_r}$, they are ordered implicitly depending on their position around the object as the perimeter of the latter is explored in a given trigonometric direction. Given these positions, the robots will measure $(S(w_1), \ldots, S(w_{n_r}))$. The proposed collaborative classification method will try to estimate the corresponding canonical viewpoints $\left(v^{K_n}_{n(1)}, \ldots, v^{K_n}_{n(n_r)}\right)$ given the above measurements; the end result, namely the class K_n the object belongs to, is implicit. *This means that not only the class of the object, but also the relative positioning of each robot can be obtained.*

Without any other a-priori information, and based only on the features observed by a robot, the latter can already generate an hypothesis regarding its current viewpoint, and implicitly which object it is observing, if the matched view is only present in that object model. If the observed features match closely the features corresponding to a view that is unique to an object class, the latter can be retained as a likely hypothesis for the whole object.

5.2 Hypotheses fusion

By observing the object from a given position, a robot can only generate local, basic hypotheses. Generally this is not enough to determine the class the object belongs to. The information obtained from different measurements should be fused via exchange of hypotheses between different robots. Amongst the many fusion processes introduced in the literature [Abidi and González, 1992; Hall, 1992; Klein, 1999], the Dempster-Shafer (DS) formalism [Hutchinson and Kak, 1992] was retained because it does not require a-priori class probabilities and is able to capture the notion of uncertainty. The often-cited drawback of the DS method is that its complexity grows exponentially with the cardinality of the primitive hypothesis set. However, due to the way hypotheses are generated from the object models, the complexity can be proven to be polynomial [Hutchinson and Kak, 1992].

Dempster-Shafer (DS) evidential reasoning [Shafer, 1976] is an extension to Bayesian inference that allows each source of information to contribute only to the evidence it has gathered, without overcommitting or trying to make hasty choices based on incomplete information. The Dempster-Shafer approach allows to express the lack of information by separating belief for a proposition from its mere plausibility, assigning probability masses to sets of propositions in such a way that the latter is free to move to any subset.

Probability mass assignment. Information sources can distribute probability masses among subsets of Θ, where Θ is the set of all statements about the possible outcomes of a random experiment. It is represented by the *frame of discernment* (FOD). The FOD is a set of mutually exclusive and exhaustive statements named *singletons*. When a probability mass is assigned to a set of singletons, it is free to move to any subset. Consequently, assignment of probability mass to Θ represents ignorance, since the probability mass can move to any element of Θ. When a source of evidence cannot differentiate between two propositions, it can assign a probability mass to a set including both.

The *probability mass assignment* function associates a probability mass to the sets in the power-set 2^Θ of Θ; it is therefore a function $m : 2^\Theta \longrightarrow \mathbb{R}$ verifying the following properties $m(\emptyset) = 0$, $0 \leq m(X) \leq 1$, $\sum_{x \in 2^\Theta} m(x) = 1$. The subsets $\{x_i\}$ of Θ such that $m(x_i) > 0$ are called *focal elements*; the union of those subsets is termed *core* of the probability assignment m.

Dempster's orthogonal sum. Two different sources of information will yield different mass distributions m_1 and m_2. Dempster's *rule of combination*, or *orthogonal sum*, can combine them if they are relative to the same FOD Θ, according to $m = m_1 \oplus m_2$, $m(X) = K \sum_{X_1 \cap X_2 = X} m_1(X_1) m_2(X_2)$. K is a normalization term defined as $K = \dfrac{1}{1 - \sum_{X_1 \cap X_2 = \emptyset} m_1(X_1) m_2(X_2)}$, which normalizes the new probability masses so that their sum is still unity. It can be seen as a measure of the degree of conflict between the two sources of information. When $\sum_{X_1 \cap X_2 = \emptyset} m_1(X_1) m_2(X_2) = 1$, the information is completely inconsistent and it is impossible to integrate it: the orthogonal sum is then undefined.

Hypothesis refinement. General, non-basic hypotheses are noted $H^{level} = \{(a_k, \ldots) | \, a_k \in A\}$. It is important to note that a_k could correspond to the output from several canonical viewpoints. The set of all possible hypotheses is noted H. Clearly the sequences of canonical measurements can only correspond to valid view sequences in some object model; impossible sequences, such as those having views that cannot belong to the same object, will not be generated.

In general, a robot will propagate its current beliefs about the object to the "next" neighboring robot along the perimeter of the object — initially $m\left(H^0\right)$. When this information is sent, the receiving robot can access the following:
- belief of the previous robot $m_1(H^n)$;
- distance to the robot whose message is being received d_{pre};
- its own beliefs about the observing part of the object $m_2\left(H^0\right)$.

The Dempster-Shafer combination rule for two hypothesis sets in a compatible frame of discernment $m(H_n) = \dfrac{\sum_{H_i \cap H_j = H_n} m(H_i) m(H_j)}{1 - \sum_{H_i \cap H_j = \varnothing} m(H_i) m(H_j)}$ is slightly modified to use the information about the relative positions of the robots as follows. Given an hypothesis set H^n, the refined hypotheses will be $H^{n+1} = \{U(h \oplus a) | \, h \in H^n, a \in A, h \oplus a \in H\}$, where the last condition means that the new view sequence must be possible for at least one object class. The operation $\oplus : H \times A \longmapsto H$ is defined as $h \oplus x = \left(h_1, \ldots, h_n, \ldots, a_{m_1}, \ldots, a_{m_p}, x\right)$, where the views $a_{m_1}, \ldots, a_{m_n}$ are a "filler", and x is the view that is to be added to the sequence. An additional restriction can be imposed to the $\oplus$ operation, namely that the filler has to be no longer than some arbitrary number of viewpoints k with $p < k$ in the above expression. The output of the function $U(h)$ is defined as the shortest hypothesis equivalent to h, that is, an hypothesis that corresponds to the same $(\text{object}, \text{offset})$ matches.

Algorithm 1 Hypothesis refinement (pseudocode).

```
function new_hypotheses(incoming_message)
  in_hypotheses = decode(incoming_message)
  new_hypotheses = {}
  for hypothesis in in_hypotheses do
    for feature in initial_feature_estimates do
      if exists_successor(hypothesis, feature)
        for succ in successors(hypothesis, feature)
          pm = hypothesis.pm * feature.probability *
           distance_factor(incoming_message.distance, succ.dist)
          if succ not in new_hypotheses
            add succ to new_hypotheses with p.m. pm
          else
        add pm to the probability mass of succ in new_hypotheses
          end endfor endif endfor endfor
  trim_hypotheses(new_hypotheses)
  total_pm = sum of all prob. masses in new_hypotheses
  for hypothesis in new_hypotheses
    hypothesis.pm /= total_pm  endfor
  return new_hypotheses
endfunction
```

The new probability mass assignment is calculated with $m'\left(h_i^{n+1}\right) = \sum_{h^n \oplus x = h_i^{n+1}} m_1\left(h^n\right) m_2\left(x\right) \xi\left(d_{pre}, d_{model}\right)$, $m\left(h_i^{n+1}\right) = \frac{m'\left(h_i^{n+1}\right)}{\sum_k m'\left(h_k^{n+1}\right)}$, where an additional normalization is required due to the usage of the distance term $\xi\left(d_{pre}, d_{model}\right)$. The latter reuses the known distances between the last canonical viewpoint of h^n and the viewpoint that is chosen to match x. ξ is taken as the normal distribution $\xi\left(d_{pre}, d_{model}\right) = \frac{1}{\sqrt{2\pi}\gamma d_{model}} e^{-\frac{\left(d_{pre} - d_{model}\right)^2}{2\gamma^2 d_{model}^2}}$ whose standard deviation depends on the expected distance, to cope with the increasing inaccuracy as the latter grows; in practice, values around $\gamma \sim 0.5$ yield good results. The overall process is described in Algorithm 1.

5.3 Hypothesis encoding and compression

Once a number of robots have acquired information about the object they are observing, hypotheses can be refined through exchanges. The associated communication cost is proportional to the volume of data being communicated. It is possible to bound the cost of the communication associated to collective classification as follows. It can be seen that there can only be at most $n_V = \sum_i n_V^{K_i}$ hypotheses being considered at any point in time, representing the number of differentiable object identities and poses. The information about the hypothesis to be transmitted can be encoded either by explicit encoding on a per-hypothesis basis, or by factoring out information common to multiple hypotheses and using implicit information (like ordering) across message fragments.

Per-hypothesis encoding. A unique identifier for each hypothesis can be encoded using only $\left\lceil \log_2 \sum_i n_v^{K_i} \right\rceil$ bits. Due to memory constraints, hypotheses can be encoded alternatively as $h = \langle i, l, o \rangle$, where K_i is an object model, l is the number of aspects of the hypothesis and o is the offset in the canonical views sequence. Thus, each hypothesis can be encoded in at most $\lceil \log_2 |K_i| \rceil + \left\lceil 2 \log_2 \max_i n_V^{K_i} \right\rceil$ bits. The amount of information transmitted for k hypothesis is directly proportional to the latter, resulting in $k\left(\left\lceil \log_2 \sum_i n_v^{K_i} \right\rceil + c_{pm}\right)$ bits normally, where c_{pm} is the amount of bits needed to encode the probability mass itself.

Implicit encoding. Only *hypothesis selector* can be sent, indicating which hypotheses are actually transmitted and a sequence of probability masses. It consists of n_V bits: the n-th bit specifies if the probability mass of the hypothesis whose identifier is n is attached to the message. In order to transmit k hypotheses $\sum_i n_v^{K_i} + k c_{pm}$ bits are needed. Therefore this encoding approach is only practical when $k \gg 1$, that is, when a large number of probability masses are to be transmitted, so the overhead is amortized.

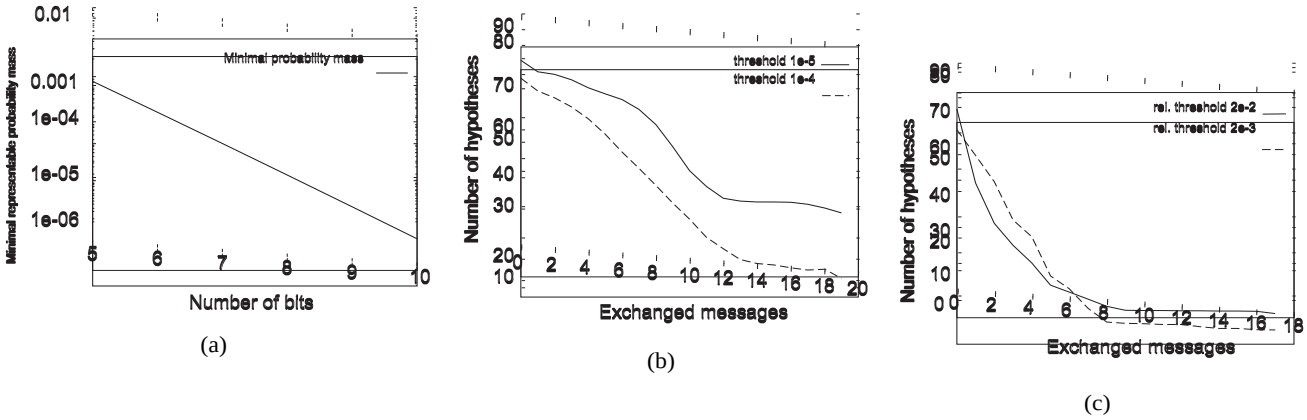
Linear encoding. If a simple linear, fixed-point scheme is employed, and the resolution is chosen to be a fraction of the average probability mass $\frac{\overline{m}}{k}$, as many as $\left\lceil \log_2 \left(\frac{k}{\overline{m}}\right) \right\rceil$ bits would be needed. For a reasonable value of $k = 10$ and $n_V = 60$, a fixed-point encoding would require $\lceil \log_2 (10 \times 60) \rceil = 10$ bits per probability mass.

Dynamic range compression. ITU-T G.711 [ITU, 1988] introduces two *compression* algorithms based on the following key idea: the signal is compressed according to a logarithmic expression. The simplest one, μ-law, applies the transform $y = \operatorname{sign}(x) \frac{\ln(1+\mu|x|)}{\ln|1+\mu|}, -1 < x < 1$ where μ is chosen according to the desired output resolution; for 8 bits, $\mu = 255$. The similarity with probability mass encoding is striking. Indeed, based on the μ-law expression, probability values can be encoded using $E(m) = 2^n \frac{\ln(1+(2^n-1)m)}{n \ln(2)}$ so that the encoded probability mass fits in n bits, and the distortion ratio is minimal. Figure 7(a) shows the minimal representable probability mass for different encoding lengths.

Hypothesis set compression. Regardless of the method used to encode the hypothesis set, the cost, in terms of amount of information to be transmitted, grows with the number of hypotheses propagated. It is thus desirable to minimize the cardinality of the hypothesis set before transmission. This can be performed either by *trimming* (discarding hypotheses whose probability mass is comparatively or in absolute terms small) or by *coalescing* (grouping several hypotheses into one corresponding to the union of the corresponding propositions).

Trimming. The cardinality of an hypothesis set can be reduced by simply ignoring unlikely hypotheses. The simplest way is retaining only hypotheses whose probability mass is higher than some absolute threshold. Figure 7(b) shows the cardinality of the hypothesis set when the latter is trimmed according to different absolute thresholds. The hypothesis can also be made smaller by removing all the hypotheses whose probability mass is below a threshold relative to the most likely hypothesis, i.e. those that satisfy $pm < r \times \max_i pm_i$, where r is the relative threshold and pm_i are the probability masses. The performance of this method is illustrated in Figure 7(c).

Coalescing. It is possible to further minimize the cost of transmitting an hypothesis set by transferring only some hypotheses. Those not specified explicitly can be coalesced into



(a) (b) (c)

Figure 7: **(a)** *Minimal representable probability mass using dynamic range compression;* **(b)** *Number of hypotheses with reduction based on absolute threshold;* **(c)** *Number of hypotheses with reduction based on relative threshold.*

a more general hypothesis carrying the sum of the probability masses using $m(X) = \sum_{Y \subseteq X} m(Y)$. This scheme also makes the overall communication more robust, since it can be interrupted without adverse consequences at any point during the transmission of the probability masses.

Termination. When information fusion is successful, the whole group of robots will converge as a whole towards a common decision regarding the nature of the object. The final decision of each robot can be taken as the hypothesis with the highest associated probability mass. It is therefore necessary to know when a given hypothesis set can be considered as "refined enough". The key idea is that hypothesis refinement can be considered finished when enough evidence has been collected, i.e. the ambiguity of a set of hypotheses is larger than a given threshold. [Hutchinson and Kak, 1992] defines the ambiguity of an hypothesis set, closely related to the concept of entropy in information theory, as follows: $A(\Omega) = -K \sum_{\theta \in \Omega} p(\theta) \log p(\theta)$, $p(\theta) = \frac{\sum_{\theta \in H} m(H)}{|H|}$. This definition takes into account the fact that an hypothesis might correspond to several individual statements or singletons. It can be seen than the ambiguity measure of the probability mass assignment $\{\Omega \Longrightarrow 1\}$, i.e. complete ignorance, corresponds to the entropy of an equiprobable distribution over $|\Omega|$ possible outcomes.

6 Preliminary experiments and discussion

Preliminary experiments have been performed with 10 prototypes of the microrobots Jasmine in the field of individual and collective perception. In experiments we measured the feature extraction and surface's recognition, as described in Section 4 and collective hypothesis refinement, as described in Section 5. The robots are placed in the situations like those depicted in Figures 4, 5. Table 1 contains the probability mass assignments for the three stored patterns "flat surface", "concave area" and "M concavity", represented in Figure 6 (c). The calculated probabilities from experimental scans confirm the results predicted by the simulation. The collective classification process was tested in hybrid approach, where the real scan data are taken from the microrobots, however the hypothesis fusion was performed in the host computer. The reason is a lack of bidirectional communication in the prototypes, that is currently under improvement. Figure 8(a) shows

Feature	Distances			Probability masses		
	Flat	Concv.	M	Flat	Concv.	M
Concavity	1020	543	1096	0.26	0.49	0.24
	765	872	1359	0.41	0.36	0.23
	664	764	1251	0.42	0.36	0.22
	1275	861	995	0.27	0.39	0.34
	702	215	1105	0.20	0.67	0.13
	1020	1020	1418	0.37	0.37	0.26
Flat surface	258	812	1864	0.69	0.22	0.10
	259	954	1846	0.71	0.19	0.10
	510	872	1862	0.54	0.31	0.15
M concavity	1785	1785	1646	0.32	0.32	0.25
	1530	1343	789	0.25	0.28	0.48
	1436	1288	1190	0.30	0.34	0.36
	1444	1331	895	0.27	0.29	0.44
	1530	1376	1053	0.28	0.31	0.41
	1624	1570	1312	0.31	0.32	0.38
	1457	1294	861	0.26	0.29	0.44
	1275	1061	559	0.22	0.27	0.51

Table 1: Probability mass assignments according to Jasmine's scan data.

the belief of a robot after its initial estimation, which is based only on the information obtained via distance sensors, and after reception of messages from other robots. The belief values converge quickly towards the correct value.

Figure 8(b) illustrates the evolution of robots placed around a "T shaped" object. The curves "correct", "wrong class" and "wrong pose" indicate respectively the fraction of robots that took the correct decision, those which made a mistake in the class of the object, and finally those which were able to determine the class of the object correctly but could not estimate their relative positions accurately. The graphs corresponds to an average value for several successful processes.

Figure 8(c) shows the success rate for different convergence rates. It can be interpreted as follows: a pair $\left(\frac{x}{100}, \frac{y}{100}\right)$ in the curve means that in y percent of the runs the rate of correct decisions remained stable at x percent or higher after thirty message exchanges. We can therefore see that in around 66% of the processes all robots took the right decision regarding the object identity and their relative position

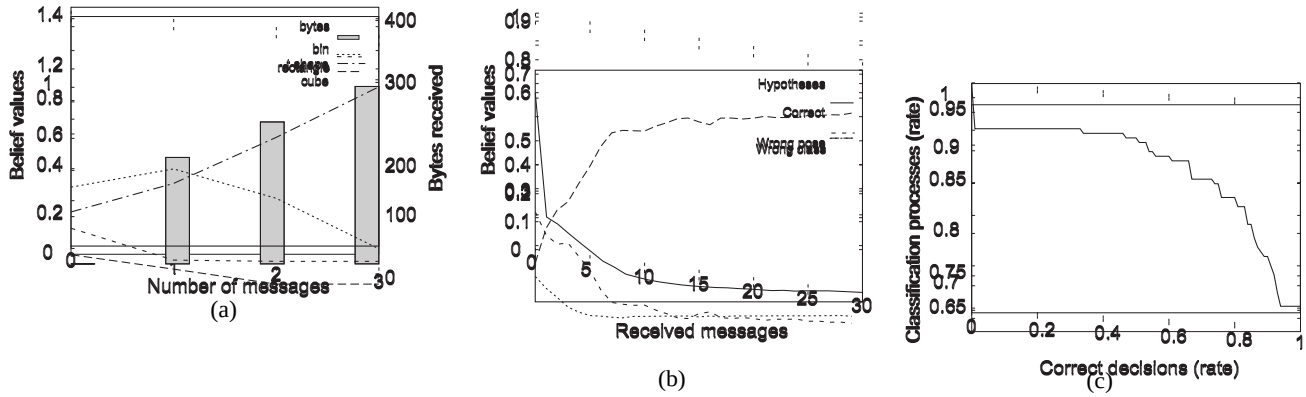


Figure 8: **(a)** *Evolution of the classifying estimations of a robot. The belief value evolves as the robot obtains information from its peers, while observing an object of class "T shape";* **(b)** *Convergence in successful collective classification processes (T);* **(c)** *Success rate for different convergence thresholds.*

(the rate for a convergence equals to or greater than 80% exceeds 82%), more than one half of the robots reached correct decisions regarding both object identity and position in over 90% of the classification operations. The group of microrobots converged towards a wrong decision regarding the identity of the object in around 5% of the classification processes. Around 10% of the classification processes end up with less than one robot out of ten with correct identity but wrong positional decisions. Around 15% of the classification processes failed to converge to either a correct decision within a 20% rate or to an erroneous decision.

Summary. In this paper we addressed the specific problem of perception in a swarm of microrobots. We investigated the process of individual perception by designing and implementing the IR sensory system. We researched also the problems related to IR-based perception and developed/tested the hardware and the corresponding algorithms allowing sensing and classifying the geometry of the surfaces. The collective classification was performed by fusing local hypotheses by using a formalism based on the Dempster-Shafer evidential reasoning. Communication needs were analyzed. Experiments demonstrated that, the size of robot is scaled down (over 20 times in comparison with the middle-size league in RoboCup), however the microrobot still possesses cognitive features. However we also observe that the smaller the size (the more reduced capabilities) of a separate robot is, the more functionalities can be achieved only in collective way.

References

[Abidi and González, 1992] Mongi A. Abidi and Rafael C. González, editors. *Data fusion in robotics and machine intelligence*. Academic Press Professional, Inc., 1992.

[Bonabeau *et al.*, 1999] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, New York, 1999.

[Caprari and Siegwart, 2003] G. Caprari and R. Siegwart. Design and control of the mobile micro robot alice. In *Proc. of the 2nd Int. Symposium on Autonomous Minirobots for Research and Edutainment, AMiRE'2003*, pages 23–32, 2003.

[Hall, 1992] David Lee Hall. *Mathematical Techniques in Multisensor Data Fusion*. Artech House, Inc., 1992.

[Hutchinson and Kak, 1992] S. A. Hutchinson and A. C. Kak. Multisensor strategies using dempster-shafer belief accumulation. In *Data fusion in robotics and machine intelligence*, chapter 4, pages 165–209. Academic Press Professional, Inc., 1992.

[I-Swarm, 2003 2007] I-Swarm. *I-Swarm: Intelligent Small World Autonomous Robots for Micro-manipulation, 6th Framework Programme Project No FP6-2002-IST-1*. European Communities, 2003-2007.

[ITU, 1988] ITU. G.711 pulse code modulation (pcm) of voice frequencies. nov 1988.

[Klein, 1999] Lawrence A. Klein. *Sensor and Data Fusion Concepts and Applications*. Society of Photo-Optical Instrumentation Engineers (SPIE), 1999.

[Kornienko *et al.*, 2004] S. Kornienko, O. Kornienko, and P. Levi. Generation of desired emergent behavior in swarm of microrobots. In *Proc. of the 16th European Conf. on AI (ECAI 2004), Valencia, Spain*, 2004.

[Kornienko *et al.*, 2005] S. Kornienko, O. Kornienko, and P. Levi. Collective ai: context awareness via communication. In *Proc. of the IJCAI 2005, UK*, 2005.

[Kube, 1996] C.R. Kube. A minimal infrared obstacle detection scheme. *The Journal for Robot Builders*, 2(2):15–20, 1996.

[Megabitty, 2005] Megabitty. see http://groups.yahoo.com/group/megabitty/. 2005.

[Pitas, 1993] I. Pitas. *Digital Image Processing Algorithms*. Prentice Hall, 1993.

[Pradier, 2005] M. Pradier. *Collective Classification in a Swarm of Microrobots*. Master Thesis, University of Stuttgart, Germany, 2005.

[Shafer, 1976] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[Suzuki *et al.*, 1995] S. Suzuki, H. Asama, A. Uegaki, S. Kotosaka, T. Fujita, A. Matsumoto, H. Kaetsu, and I. Endo. An infra-red sensory system with local communication for cooperative multiple mobile robots. In *Proc. of International Conference on Intelligent Robots*, pages 220–225, 1995.

[Ye *et al.*, 2002] Y. Ye, S. Boies, J. Liu, and X. Yi. Collective perception in massive, open, and heterogeneous multi-agent environment. In *Proc. of the AAMAS'02*, pages 1175–1182, Bologna, Italy, 2002.

Multi Agents Modelling for a Real-Time Dynamic System: an Analogue Electronic Circuit

Osama Zaki, Keith Brown, and Dave Lane
School of Engineering and Physical Sciences
Electrical, Electronics & Computer Engineering
Heriot-Watt University, Edinburgh, Scotland
o.zaki@hw.ac.uk

Abstract

This paper demonstrates the use of Multi Agent Systems (MAS) to model a real-time dynamic system. The specific goal here is to diagnose faults in an analogue electrical circuit. This modelling represents a layer from the seven layers architecture described on [Zaki *et al*, 2005]. The design is described and the implementation is carried using the Jade – Multi Agent toolkit.

1 Introduction

An analogue electrical circuit is an example of a dynamic continuous non-linear and time invariant system. Certain faults diagnosis approaches can fit better for different types of systems. For instance, while a rule-based system and Modal-Based Diagnosis MBD are suitable for a system involving complicated interactions and whose outcomes are hard to predict, they are not yet effective for a real-time continuous system [Sampth, 1995].

This paper shows that how the type of system has impact on the choice of the modelling techniques and that modelling is a crucial phase when building diagnostic tasks. It also demonstrates the effective use of Bond Graphs to model dynamic systems. Similarities and commonality is drawn between Bond graphs and agents. The seven diagnostic layers, where each layer is represented by one or more dedicated software agents, are revisited. Emphasis, however, is on two of the middle layers: modelling and controlling. Many of the design issues for those two layers are explained, such as threading and behaviours, transferring effort and flow between agents, agents' relationships, controlling the agents, and the automatic creation of the model.

1.1 Models and Modelling

The characteristics of the system will impose on the modelling techniques used, for example Automata and Petri-Nets are suitable for discrete event systems (known as Active Systems), but not for a dynamic continuous systems. In this paper a MAS is used to model a real-time dynamic system, an analogue electrical circuit.

Modelling is a crucial phase when building diagnostic tasks [Console, 2001]. Modelling may be accomplished at different level of abstraction. Models can be classified into two main categories quantitative models (sometimes referred to as analytical models) and qualitative model (sometimes referred to as conceptual models). MAS provide a suitable suite to utilize both types of modelling.

1.2 Faults and Failure

Component life-time can rarely be estimated accurately. Hardware can be viewed at three different levels: component level, board level, and system level (i.e. detecting the replicable unit). Failure in one of the components or broken links (wires) can lead to an open-circuit or a short circuit. The fault diagnostic process is broken into three steps: 1) fault detection, 2) fault isolation and 3) fault diagnosis. In some cases the fault diagnosis includes fault recovery. While detecting faults can be accomplished by a real-time inspection of the system, isolation and diagnosis requires a reasoning mechanism. Faults can be classified into different classes: 1) based on the time taking to occur (abrupt and incipient faults), 2) based on the period of time they occur (intermittent and permanent faults), 3) based on the number of faults (single and multiple faults), and 4) based on failure coverage (complete and partial). MAS, intuitively, can cope with most type of faults.

1.3 Diagnostic Techniques

Diagnostic tasks are mostly offered by the two communities; FDI (Fault Detection and Isolation) and DX (Based on Intelligent techniques). Both methods use an explicit model, which is known as model-based diagnosis (MBD) to predict normal behaviour. The process (algorithms) detects faults from inconsistencies between the observed (from real physical system) and the predicted behaviour by the model. Then they interlink a set of components with the detected inconsistencies to isolate the faults. But techniques and hypothesis are different. FDI uses Analytical Redundancy to expresses a constraint among possible observations that hold when the system is working correctly. The Analytical Redundancy builds quantitative

mathematical models. Quantitative models require knowledge of differential equations, transfer functions, signals and pattern estimation. The DX approach uses consistency-based logic and logical inference to determine the symptoms, then determines the minimal conflicts, and hence determines the minimal diagnoses. The DX approach builds qualitative models for the systems. There were common agreements among researchers from both communities that a bridge can be made between the techniques. Studies showed that major parts of the two theories fits into a common framework. As a result, technology such that OBD II (on-board diagnosis), which make use of both quantitative and qualitative modelling [Struss, 2001].

Because of the nature of the diagnostic tasks – uncertainty and incomplete data probabilistic and fuzzy methods may be used. Incipient faults, in particular may require the use of probabilistic models.

2 Related Work - Diagnosis Using Agents

A multi agent system was developed to monitor industrial turbine start-up sequences. It was also used for data interpretation in electrical plant monitoring [Mangina *et al.*, 2001; Hossack *et al.*, 2003]. Zeus (an agent building toolkit from BT) was used.

Schroeder [Schroeder, 1998] proposed an architecture for autonomous model-based diagnosis agents. He developed a logic programming approach for model-based diagnosis and introduced strategies to deal with more complex diagnosis problems, and then embedded the diagnosis framework into the agent architecture of agents. Two algorithms were developed; a bottom-up algorithm to remove contradiction from extended logic programs and top-down evaluation of extended logic programs. PVM-Prolog was used to implement the algorithms. Both algorithms are evaluated in the circuit domain including some of the ISCAS85 benchmark circuits. Many diagnosis problems were modelled such as: digital circuits, traffic control, and integrity checking of a chemical database, alarm-correlation in cellular phone networks, diagnosis of an automatic mirror furnace, and diagnosis of communication protocols.

Multi-Agents-based Diagnostic Data Acquisition (MAGIC) project was funded by the European Commission [Köppen-Selige *et al*., 2001]. One of the main goals of MAGIC is the on-line detection and diagnosis of incipient or slowly developing faults in complex systems. The MAGIC basic architecture was based on multi-level approach. The idea is that the task of the complex embedded system's diagnosis and operator support is distributed over a number of intelligent agents, which perform their individual tasks nearly autonomously and communicate via the MAGIC architecture. The tasks of the six levels are as follows: process specification, information acquisition, diagnosis, diagnosis monitoring, decision, and operator support. Because agents allow you to model the device, the process and the topologies in one shot, a MAS approach is adopted to perform diagnostic tasks. The strengths and advantages of MAS architecture allow us to integrate different diagnosis tools and techniques. It would also support heterogeneous distributed systems. It was suggested to separate between different tasks of diagnostics using what it was called the Seven Diagnostic layers [Zaki *et al.*, 05].

The work in this paper is advancements in the state of the art in the area of fault diagnosis. The idea is new and the approach is interesting. No similar work was reported before.

3 Agents and Bond Graphs

Bond Graphs are widely used for modelling dynamic systems [Mosterman and Biswas, 1998]. Similarities and commonality can be drawn between Bond Graphs and agents

Bond Graphs is a modelling language and it is domain independent. The Bond is the connection to enable Energy transfer among components. Two components A and B will have link between them with two associated values *e* (effort) and *f* (flow). Bond Graphs force you to make explicit assumptions about the physical system and it is based on small number of primitives: dissipative elements, energy storage elements, source elements and junctions. Physical system can be mechanics, electricity, hydraulic and thermodynamic.

In an electrical system, the *effort* e is the voltage and the *flow* is the current. While in a mechanical system, the *effort* e is force and the flow is velocity. The product of the *effort* and the *flow* is *power* and integration of *power* is the *energy*. The state of the system is determined by the *energy* transfer between components, more accurately, the rate on energy transfer.

For passive 1-port elements (energy storage elements); resistor R, $e = R \cdot f$, $e(t) = R(t) \cdot f(t)$. For capacitor C, $e = 1/C \int f \, dt$. Other 1-ports are Effort Source (Se) and Flow Source (Sf). To connect elements together, for 2-ports, transformers for examples, $e2 = (b/a) \cdot e1$ and $f1 = (b/a) \cdot f2$. This leads to $e1 \cdot f1 = (a/b) \cdot e2 \, (b/a) \cdot f2 = e2 \cdot f2$.

For 3- ports there are 1-junction (Common flow junction), which enforces Kirchhoff's voltage law, and 0-junction (Common effort junction), which enforces Kirchhoff's current law. The Common flow junction is equivalent of series junction where there is no loss of energy at junction and net power in is equal to net power out, all flows is equal to 0. The Common effort junction is equivalent of parallel junction where there is no loss of energy at junction and net power in is equal to net power out and all efforts is equal to 0. Figure 1 shows Bond Graph for a circuit consists of battery, capacitor and resistor.

Causality relations are usually used with Bond Graphs to aid the generation of equations among system variables. As described above a Bond Graph considers variables as interacting variables pairs. The cause effect relation considers the *effort* as push and the *flow* as response. There are various types of causality relation these are: necessary causality, restricted causality, integral causality, derivative causality, and arbitrary causality.

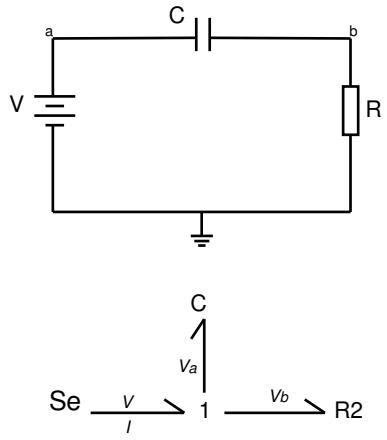


Figure 1: Bond Graph representation for a simple circuit

All of these causal relations are based on algebraic relations. Figure 2 shows Bond graph with causality relationship for electrical circuit with voltage source, inductor and resistor (R1) in series connected with capacitor and resistor (R2) in parallel. In the Bond Graph diagram, the relation between Se, L, and R1 is restricted causality. The relation between 1, C, and R2 is also restricted causality.

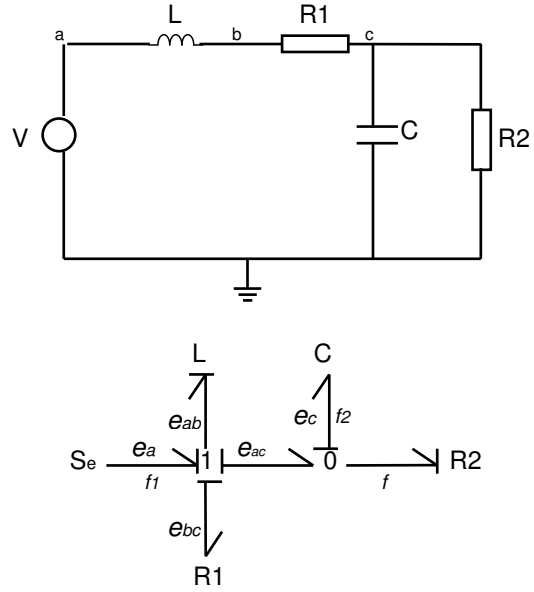


Figure 2: Bond Graph with causality relations for electrical circuit

Agents are a suitable implementation environment for Bond Graphs where many of the features and concepts in a Bond Graph can be adopted easily by agents. For example messages exchange between agents can be viewed as the energy transfer (the arrow) between the elements on the electric circuit. Direct mapping can be drawn between the agents and the nodes on the electric circuit. Causality relations can be also represented on agents as it will be described in the next section.

4 Overall Architecture

The overall architecture is composed of 7 layers, called the Seven Diagnostic layers. Each layer of the Seven Diagnostic layers receives data and depends on the lower layer. Practically each layer can be represented by one or more dedicated software agents. The Seven Diagnostic layers depict both; hardware and software.

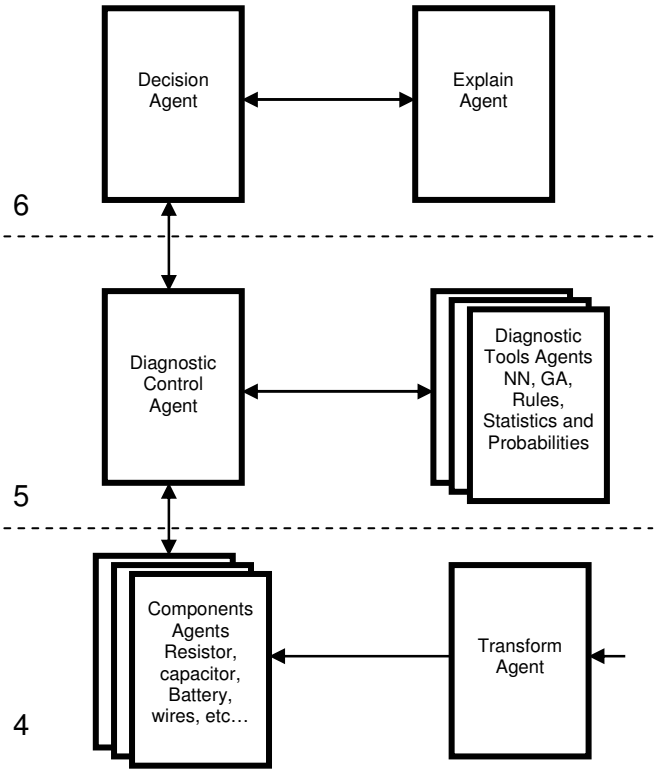


Figure 3: The middle three levels – the core

At the first level is the physical hardware: board, device or systems (e.g. a dynamic electrical circuit contains a battery and two resistors). The second level includes both hardware and software for data acquisition purpose, reading on-line real-time data from the device. This would include analogue to digital conversion. The third level - the communication level – is responsible for preparing, formatting and packetisation of the data to be ready for the diagnostic engine at higher levels. Levels 4, 5 and 6 are the core of this

research and they represent the diagnostic engine. Figure 3 shows these three levels in more details.

At the fourth level, Figure 3, the modelling level, we try to use the dynamistic of the agents to be used as a modelling language for dynamic systems. A qualitative Model-based diagnosis is implemented using the agent's architecture. Bond Graphs are used for modelling dynamic systems. Similarity and commonality was drawn between Bond Graphs and Agents in the previous section. At this level also, a Transform agent will be used to transform quantitative models (developed by third software package, e.g. Matlab) into the qualitative Model-based diagnosis agents. However, mathematical models will be still used inside the agents. At the fifth level, the diagnostic engine level, there is the main agent which monitors and controls the overall mechanism of diagnostic tasks. At this level there are also agents that host different tools, such as an agent for neural networks (NNs) and other for genetic algorithms (GAs) and etc.

At the sixth level, there are two agents. One agent is to make the final decision and select among different hypothesis. The other agent is to produce a suitable explanation to the end-user. At the higher level, there is a user friendly interface to interact with the user

5 Design and Implementation Issues

In the rest of this paper the focus is on some design and implementation issues for levels 4 and 5. Figure 4 shows the two layers out of the complete architecture: the lower layer, in Figure 4, is the modelling layer.

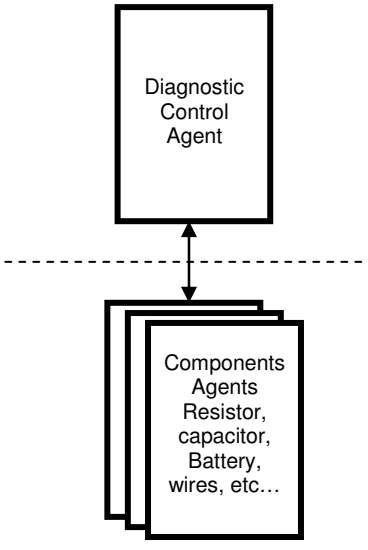


Figure 4: Diagnostic layers architecture

The dynamics of the agents allow to perform the modelling. A qualitative model based diagnosis in implemented using the agent's platform. The upper layer, in Figure 4, is the diagnostic engine level - one agent which monitors and controls the overall mechanism of diagnostic tasks.

For each single component on the electrical circuit there is an agent to represent it and monitor it. In a real-time each agent reads two values; current and voltage (*flow* and *effort*), at different nodes on the electrical circuit. These nodes of measurements are pre-selected near the monitored components. The real-time observed values are compared with the expected values which are stored inside the agent. Expected values are taken from the numerical model that was already built by one of the electronics design tools, e.g. PSpice and it was transformed to the agents by the Transform agent.

Each dual reading (the *energy* transfer which is discussed in the previous sections) determines the status of the component. The status of the component can be either: GOOD, FAULTY, DEGRADING, UNKNOWN, and UPNORMAL. The status of the components are different form the actual status of the agents, Jade (agent toolkit/library) provides the following status for the agents: INITIATED (agent build, not registered, and has neither name nor address), ACTIVE (registered, has a regular name and address), SUSPENDED (agent is currently stopped, no agent behaviour is being executed, internal thread of agent is suspended), WAITING (agent is blocked, waiting for something, its internal thread is sleeping, wake up when message arrive), TRANSIT (mobile agent is migrating to new location), and DELETED (agent is definitely dead)

There is no direct mapping between the status of the components and the status of the agents. This means that when the component is FAULTY it doesn't necessarily mean that the agent is DELETED.

The dual readings allow capturing the open and short circuit failures on the electrical circuit. Here we assume that the data acquisition software is embedded into the agent itself. In the complete architecture, this would take place at layers 1 and 2. Values should be read at milliseconds intervals. Current sensors should be fitted into the circuit. The accuracy of data read depends on the efficiency of the hardware and software used for data acquisition. A good data acquisition tool should have precision timing and high-speed sample buffer. There are two modes for data logging from the electrical circuit: stream and burst. Software timed acquisition (also called command/response) allow the PC to send a command to the device and it responds with data in either modes; stream or burst. In burst mode, for a typical device, up to 1,024 samples per channel will be acquired and then stored in a buffer. In stream mode, 300 samples/second per channel will be acquired and then stored in the buffer. Simultaneously the data is transferred from the device buffer to the PC buffer. The device should also have reasonable numbers of analogue and digital I/O channels. For I/O digital up to 50 Hz per bit is needed.

Inside each agent, there is one or more fault model which describes the expected energy transfer values; current and voltage when the named component fails. For example, a simple circuit consists of a DC battery and two resistors, shown in Figure 5, demonstrates how agents can be used to model and then detect a fault in a real-time manner for as an example for a dynamic system.

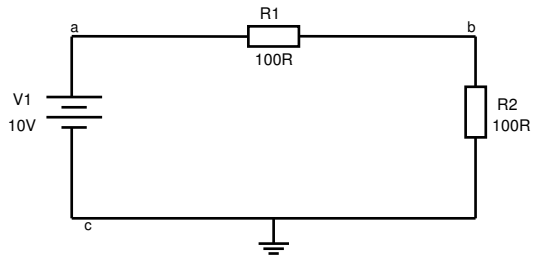


Figure 5: A simple circuit consists of DC battery and two resistors

For this simple circuit we would have three agents to model the circuit: V1 agent, R1 agent, and R2 agent. Each agent monitors one single component. Three positions (a, b, and c are selected to acquire the energy transfer values, flow and effort. This means we six values will be available as follows: IR1 and VR1 at point a, IR2 and VR2 at point b, IV1 and VV1 at point c. For this circuit, I = 50mA and P = 0.5 watt. The agent which is representing and monitoring R1, named A_R_R1, will be reading the IR1 and VR1 values from the circuit, and among other data the following fault model:

R1 is faulty when:

IR1= 0	VR1= 10
IR2= 0	VR2= 0
IV1= 0	VV1= 0

The observed values are compared with the expected values and these are: IR1= 50mA and VR=10. If the values are not matching the agent, as one of the task to do among other tasks, is to investigate the fault models.

The agent which is representing and monitoring R2, named A_R_R2 observes the IR2 and VR2 values and compares them with the expected values and then investigate following fault model:

R2 is faulty when:

IR1= 0	VR1= 10
IR2= 0	VR2= 5
IV1= 0	VV1= 0

The agent which is representing and monitoring V1, named A_R_V1 observes the IV1 and VV1 values and compares them with the expected values and then investigates the following fault model:

V1 is faulty when:

IR1= 0	VR1= 0
IR2= 0	VR2= 0
IV1= 0	VV1= 0

5.1 Creating the Model/Agents Automatically

The transform agent which will be used to transform quantitative models into the qualitative Model-based diagnostic agents can create the agent model automatically. This can be done by parsing the Netlist file (the output of a design package, such as PSpice) and creates agent for each node in the electrical circuit. For example, the following part of the Netlist file for circuit in Figure 5 when modelled with PSpice is as follows:

Name	+Node	+Node	Value
R_R1	$N_0002	$N_0001	100R
R_R2	$N_0001	0	100R
V_V1	$N_0002	0	10V

The transform agent can parse the Netlist file and creates agents (the model) automatically. This eases the implantation of the model, adds extra flexibility to the architecture, and allows integration with different packages.

5.2 Controlling Agents

There are three ways to make agents communicate together. The interest here is on the communication at the abstract level. The low level communication (interaction protocols) is handled by the agent toolkit as it will be described in the next sections. These are:

- Agents communicate together using an external controller (a control agent as in Figure 2).
- Agents communicate together using a build-in controller inside each agent.
- Agents communicate using external and build-in controllers.

When acquiring the values from the dynamic systems and when communicating between agents, time slices is a crucial issue and this should be handled carefully by the implementer and the by actual agent toolkit.

5.3 Using a Rule-Based Engine to Control Agents

It is recommended to use an agent-reasoning engine to control the behaviours (activation and deactivation) of agents. Reactive-deliberative agent architecture can be built. The agent-reasoning engine plays the deliberative role and agent behaviours play the reactive role. The external control agent in our architecture embeds JESS – a rule-based engine. However, the architecture allows the agents to also have reactive relations among themselves without the interference of the external control. This is because we have decided to use both external and internal controllers at the same time.

5.4 Agent Relationships

Each agent will have more than one thread (task) to deal with all sorts of communications coming from or going out to the outside world. One or more thread read the values from the dynamic system. Others threads communicate with neighbour agents. More threads communicate with related (relative) agents. And other threads communicate with related (relative) neighbour agents. This adds more work on the agent to handle all these type of traffics. Neighbour agents are those that are directly connected to the named agent Relative agents are those that have relation with the current agent. Relative neighbour agents are those that have relations and they are directly connected to the named agent.

This social arrangement of the agents into: neighbour, relative, and relative neighbour agents, allow the agent to deal with the incoming and outgoing information (messages) in prioritized manner. In this way the priority of communications should be classified into the following classes:

1. Controller (external)
2. Self-control
3. Relative-Neighbour
4. Relative
5. Neighbour

Therefore, each thread running inside the agent should belong to one of these classes. For, example, the thread which is communicating with the Controller agent is of type Controller thread and this one has the highest priority. And the thread that is communicating with the dynamic system is of type Self-control which has priority 2. This is somehow maps with the casualty relations in Bond Graph.

5.5 Threading and Behaviours

In Jade toolkit threads are implemented in what is called, behaviours. Behaviours are of two types: primitive and composite. Primitive behaviours can be one of three choices: 1) Simple behaviour which models simple atomic behaviour, 2) OneShot behaviour which is executed only one, 3) Cyclic behaviour which is executed forever.

The Composite behaviour is made up by composing a number of other behaviours, called children. This means that the actual operation of the composite agent is defined inside its children. The composite agent its job is to schedule its children. Three types of composite classes are defined: 1) Sequential behaviour which executes its subclasses sequentially and terminates when all sub-behaviours are done, 2) Parallel behaviour which executes its sub-behaviours concurrently and terminates when a particular condition on its sub-behaviours is met, or when any one of its sub-behaviours are terminated. Or when a user defined number N of its sub-behaviours have finished. 3) Finite State Machine behaviour which executes its children according to a user defined Finite State Machine. The activity of its child performed within a state of the FSM. When state Si (one of its child) completes, its termination value is used to select transition to fire and new state Sj is reached. This goes on until the final state is reached. For our problem Parallel is most appropriate because the threads executes concurrently.

5.6 Transferring Effort and Flow between Agents

Effort and *flow* in the electrical circuit are mapped as messages in the agent environment. The form of message exchanged between agents is defined in the FIPA communicative act and the Coder/Decoder classes for the Semantic Language for messages. The FIPA standards ease the communications between different agents written in different languages and running on different platforms.

The message is composed of *content* and auxiliary parts. The actual *content* can be encoded in different ways according to the Agent Communication Language (ACL). *Content* can be encoded as Strings, Java objects, or ontology objects.
The most basic way consists of using strings. It is convenient for atomic data but not for structured data or object. The meaning of the String is application dependent. The second way to code c*ontent* as serialized Java object (not readable). This method is useful when all agents are written in Java. The third method is to define ontology (own vocabulary and protocols). This method can be used if more flexibility is required. At this stage for our application, String is sufficient to transfer the energy elements: effort and flow information between objects.

The auxiliary parts of the message (specific to Jade) are: sender (agent ID), message type (performative), Recipients (agent ID), Protocol type, ReplyWith, InReplyTo, ReplyBy, and ConversationID (which is useful when having parallel negotiation with several other agents). Message types can be for example: INFORM (agent gives another some useful information), QUERY (agent asks question), PRPOSE (agent starts bargaining). Answers to performative include AGREE and REFUSE.

Jade provides Template messages. Template messages are useful to our application since it allows us to filter messages and set up distinct behaviours to handle messages from various agents or various kinds of messages.

5.6 Inside the Agents

Each agent contains two Parallel behaviours; controllers and listeners. Listeners are implemented as sub-behaviours. Listeners listen to incoming messages. Four listeners are implemented; ObserveVaules (listens to data from the hardware), ListenToAllRN (listens to Relative-Neighbour agents), ListenToAllR (listens to all Relative agents), and ListenToAllN (listens to all Neighbour agents). Agents, behaviours and sub-behaviours all work in parallel. Controllers are also implemented as sub-behaviours. Two controllers are used; ControltaskInt (manages the internal communications between the different behaviours) and

ControltaskExt (manages the external communications between the agents). A UML can be used to describe the contents of each agent. More tools and notations need to be added to UML to complement the extra features in agents.

5.7 An Example and Results

The hardware for the DC circuit described on Section 5 was built. The circuit consists of a DC battery and two resistors. The software agents were also implemented - using Java and Jade. A simple scenario was prepared for testing. The scenario is using the build-in controller which resides inside each agent (no external controller was used and therefore no rule-based engine was used to control the agents). Also no fault models were presented inside the agents.

For this scenario three software agents were designed manually (no automatic creation of agents). Faults were injected into the circuit to different components. To speed and ease the testing procedures, faults were injected directly into the software. Assume an incorrect reading (or null reading) was inspected by the ObserveVaules sub-behaviour. The ObserveVaules informs the ControltaskInt via internal communications. The ControltaskInt sub-behaviour send messages to all related agents (Relative-Neighbour, Relative and Neighbour) and terminates the agent who is monitoring the faulty component. When other agents receive the messages from the agent who is monitoring the faulty component, they communicate with their own internal controllers and then these agents terminate too, because they would have been affected by the faulty component/agent.

The system was able to detect all type of faults which were injected to different nodes on the circuit. Although, this simple electrical circuit and the scenario give indication of the success of the approach more complex examples are needed. At the time of writing more complex electrical circuit was built which consists of about 115 components. To create the software agents for all these components manually is complex and can led to errors. For this reason, investigation is carried out to find possible ways to implement the agents automatically and to find out whether the automatic creation of the agents is sufficient to solve the problem of a big electrical circuit.

Conclusion

The use of a Multi Agent Systems (MAS) to model a real-time dynamic system has been described, specifically for use with an analogue electrical circuit. The described design and implementation was carried out using the Jade – Multi Agent toolkit. It was shown that there are similarities and commonality between Bond Graphs and agents. A number of agents at the modelling layer (of the Seven layers architecture) were used to model the circuit (in other words, to implement the Bond Graph). It was shown that the model can be created automatically via a transform agent. Parallel behaviours for the threads (tasks) inside each agent were suggested. Each task inside the agent has priority 1 to 5, tasks are prioritized. It was decided that Template messages are useful to transfer energy (effort and flow) between agents, since it allows us to filter messages down to the threads. It was found that both type of controls: external (in which a rule-based engine can be used) and build-in controlled are needed to manage the activation and deactivation of the agents. The architecture was tried out with a simple electrical circuit using three agents. The preliminary results from these tests were good, with the system being able to detect all type of faults that were injected to different nodes on the circuit. This is to be extended to more complex systems to further investigate the agents' behaviour.

Acknowledgments

The work in this paper is supported by ESPRC and SeeByte Ltd.

References

[Console, 2001] L. Console. *Model-Based Diagnosis: histroy and state of the art*. Monet School, 2000.

[Hossack *et al*., 2003] J. Hossack, J. Menal, S. D. J. McArthur, and J. R. McDonald. *A Multiagent Architecture for Protection Engineering Diagnostic Assistance.* IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 18, NO. 2, 2003.

[Köppen-Selige *et al*., 2001] B. Köppen-Selige, S. X. Ding, and P. M. Frank¸ *European Research Projects on Multi-Agents-based Fault Diagnosis and Intelligent Fault Tolerant Control*. 2001.

[Mangina *et al*., 2001] E. E. Mangina, S. D. J. McArthur, J. R. McDonald, A. Moyes. *A Multi Agent System for Monitoring Industrial Gas Turbine Start-up Sequences.* IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 16, NO. 3, 2001.

[Mosterman and Biswas, 98] P. Mosterman and G. Biswas. *A Modeling and Simulation Methodology for Hybrid Dynamic Physical Systems,* Journal of the Society for Computer Simulation*,* June 1998.

[Sampath, 1995] M. Sampath. *A Discrete Event Based Approach to Failure Diagnosis*. PhD thesis, University of Michigan, 1995.

[Schroeder, 1998] M. Schroeder. *Autonomous, Model-based Diagnosis Agents*, PhD thesis, 1998.

[Struss, 2001] Peter Struss. *AI Methods for Model-based Diagnosis*. DX, 2001.

[Zaki *et al*., 2005] Osama Zaki, Keith Brown and Dave Lane. *Automated Fault Diagnosis of Complex Systems: An Overview and Proposed Architecture*. ARTS16, 2005.

Qualitative Mapping of Sensory Data for Intelligent Vehicles

Jan D. Gehrke, Andreas D. Lattner, and **Otthein Herzog**
TZI - Center for Computing Technologies, Universität Bremen
Am Fallturm 1, 28359 Bremen, Germany
{jgehrke|adl|herzog}@tzi.de

Abstract

Recent advances in the field of intelligent vehicles have shown the applicability and utility for driver assistance systems, or even letting a car drive autonomously on highways. Usually these approaches are on a rather quantitative level. This hampers their capability to cope situations of great complexity in which humans need a lot of knowledge to act safely, for instance in city traffic. A qualitative representation of traffic scenes allows for formulating and using common sense knowledge in a human-comprehensible and machine-processable way. A vocabulary for such a representation is proposed and a prototype that does the qualitative abstraction for knowledge-based behaviour control is presented and evaluated. Experiments in a simulation environment show the applicability of the approach for intelligent vehicles.

1 Introduction

Recent developments in the field of Intelligent Vehicles (IV) address driver assistance systems like lane departure warnings, adaptive cruise control, and lane change assistance [Bertozzi *et al.*, 2000; Dagli *et al.*, 2004; Dickmanns, 2002; Rüder *et al.*, 2002; Weiss *et al.*, 2004] or efforts to letting a car drive completely autonomously as, e.g., in the DARPA's Grand Challenge[1] in 2004 where IVs had to drive autonomously through the Mojave dessert. In the 2004 competition the vehicle who came farthest – "Sandstorm" from Carnegie Mellon University – only managed to drive 7.5 of the 160 miles route.

Intelligent Vehicles perceive information about their environment through sensors like CCD cameras, radar, laser range finders, and GPS. Based on this sensory information a world model has to be created. The "belief" of the IV about its environment is then used for situation assessment and behaviour decision. The selection of the behaviour finally leads to actual control of the vehicle.

The interpretation and evaluation of traffic scenes demands for sophisticated knowledge representation and reasoning techniques. If it is aimed to create cognitive abilities similar to those of human beings a lot of information about traffic situations, traffic rules and common sense knowledge, e.g., about traffic participants must be available. In order to manage complex situations like those in cities background knowledge must be accessed. Setting up behaviour directly on quantitative data acquired by sensors is very difficult as the environment might vary a lot in different situations and common sense knowledge had to be formulated in a quite unnatural way.

We claim that a qualitative description in combination with background knowledge can be used for a concise and comprehensible representation of traffic scenes and allows for handling complex situations. Humans also abstract from concrete quantitative information of continuous movements in time and space, e.g., by dividing directions and velocities into equivalence classes. Such an abstraction allows for describing similar situations which – on a quantitative level – actually are not identical by reducing the representation to the relevant information.

Objects in dynamic environments move in four dimensions: three spatial dimensions and one temporal dimension. Spatiotemporal information can thus be represented quantitatively by different spatial coordinates at certain time points. Qualitative representations abstract from these concrete coordinates. In the literature many different approaches for temporal, spatial, and motion (spatiotemporal) descriptions have been proposed.

Famous representations for the temporal dimension are Allen's temporal logic and Freksa's semi-intervals [Allen, 1983; Freksa, 1992a]. Allen defines a set of disjoint relations between time intervals. He distinguishes the relations *before, equal, meets, overlaps, during, starts,* and *finishes* and their inverse relations. A composition table can be used to infer which temporal relations between two intervals are possible by knowing the relations of both to a third interval.

Freksa introduces the concept of semi-intervals where relations between start and end points of intervals are described. This allows for making statements about intervals even if one of the time points is not known (e.g., *older* and *survives*).

Spatial representations can be classified into approaches that describe ordinal, topological, or metric information. [Clementini *et al.*, 1997] present a framework for the qualitative representation of 2D positional information. In their

[1] See `http://www.darpa.mil/grandchallenge/`

work they describe how orientation and distance information can be discretised into qualitative classes (e.g, *front, back, left, right* and *close, far*).

The best-known approach to spatial representations based on topological information is the *Region Connection Calculus* (RCC) by Randell, Cui, and Cohn [Randell *et al.*, 1992]. RCC can be used to describe connectivity and overlap relations of regions. RCC-8 distinguishes between *disconnected, externally connected, partial overlapping, tangential proper part* (and its inverse), *equal, non-tangential proper part* (and its inverse).

Freksa presents an approach where orientation information based on a direction vector is used to qualitatively describe the position of other points relative to this vector [Freksa, 1992b]. Three lines – one covering the vector and two orthogonal lines at the start and end point of this vector – divide the space into six regions. Including the positions which lie on at least one of the three lines, i.e., which lie on one of the region borders, 15 qualitative orientation relations can be distinguished.

Schlieder's panorama approach defines an ordinal arrangement of objects in the order how they are perceived by looking around from a viewpoint (panorama). More detailed information can be achieved if the opposite position of objects is also acquired in the panorama. The panorama was recently extended by metric information and qualitative directions and distances [Wagner *et al.*, 2004].

A qualitative motion description was introduced by Miene [Miene *et al.*, 2003; Miene, 2004]. In this approach movements and positions are abstracted to different direction, distance, and velocity classes. Intervals are created from time-series based on monotonicity and threshold criteria. Such intervals are created for properties of single objects or for relations between object pairs (e.g., *speed of object x is slow*, *distance between x and y is very close*).

In the next section our representation for traffic scenes is introduced. The subsequent section presents how the qualitative abstraction of the quantitative data is performed. The evaluation section shows results on simulated traffic scenes including performance measures of the mapping cycles. The last section presents our conclusions.

2 Traffic Scene Representation

The qualitative representation of traffic scenes proposed here is an explicit and comprehensible representation that allows for knowledge-based situation assessment based on background knowledge and situation patterns.

The basic approach is to describe traffic scenes as a combination of the static road network configuration and the involved dynamic objects in spatiotemporal relations. Each network region and object is assigned to a type and for each active object we describe its motion in relation to other objects or regions. For this purpose we established a taxonomy of objects and regions relevant in the road traffic domain and developed a symbolic vocabulary of predicates describing motions of actors. The vocabulary consists of qualitative actor properties and qualitative relations between actors or between actors and traffic regions. Additionally, there is a formalism to describe the spatial relations between road regions like roadways, lanes and junctions called Road Network Configuration (RNC). RNC is discussed in [Gehrke, 2005].

The actual dynamic traffic scene is represented with temporal predicates as introduced by Allen [1984] indicating the interval where a specific property or relation holds. Situation patterns are realised as logic formulae of road scene and object descriptions in connection with abstract intervals of qualitative relations and their temporal relations using Allen's interval logic [1983].

2.1 Object taxonomy

There are many objects in road traffic with characterising properties and capabilities. There are traffic participants like vehicles, pedestrians and bicyclist, and also static objects like traffic signs, any obstacles, and the road and its subregions.

To formulate the conceptual knowledge on objects in road traffic we developed a domain model as a taxonomy of objects with properties describing their relations and capabilities forming an ontology for road traffic. Actor's capabilities include, e.g., maximum possible speed, maximum allowed speed, and allowed traffic regions. The taxonomy incorporates 20 object types like *pedestrian*, *vehicle*, *animal*, and *bus* The other part of the taxonomy are 14 region types such as roadway, lane, junction, and footway. The ontology may be extended easily if need for other object types or properties should arise.

2.2 Motion Vocabulary

The qualitative motion vocabulary is the basis to formulate actor motions in the road network. A motion proposition consists of the predicate symbol (e.g., *space_distance*), the primary object (if necessary), the reference object, and a qualitative value for non-binary propositions (e.g., *medium_distance*). In the following each motion predicate is described respectively.

velocity: an actor's longitudinal velocity in relation to its current lane or roadway. Qualitative values for velocity are *zero_speed*, *very_slow*, *slow*, *medium_speed*, *fast*, and *very_fast*.

acceleration: the development of an actor's longitudinal velocity. Qualitative values for acceleration are *decreasing_speed*, *constant_speed*, and *increasing_speed*.

relative_speed: the relative speed of a primary object w.r.t. a reference object with the qualitative symbols *slower*, *same_speed*, and *faster*.[2]

space_distance: the spatial distance between a primary and a reference object. Qualitative classes are *zero_distance*, *very_close*, *close*, *medium_distance*, *far*, and *very_far* following the well-known abstraction of [Hernández *et al.*, 1995].

time_distance: Because distances are very speed-dependent in safety perspective, this additional predicate is introduced. It expresses the speed-relative temporal distance

[2]The relative speed abstraction may be refined to further classes if it should turn out necessary.

of vehicles driving on the same roadway or of actors to regions, e.g., the next junction. The symbols of qualitative classes correspond to those of *space_distance*.

distance_trend: the trend of *spatial* distance between objects. Currently we distinguish three classes: *decreasing_distance*, *constant_distance*, and *increasing_distance*.

lateral_position: the lateral position of a vehicle w.r.t. its current lane. Qualitative values are *left_in_lane*, *central_in_lane*, and *right_in_lane*.

lateral_motion: the trend of a vehicle's lateral position with qualitative values *moving_left*, *constant_lateral_position*, and *moving_right*.

driving_direction: the driving direction of a vehicle or a lane w.r.t. a reference object. This predicate is only applicable for actors on the same roadway and differentiates between *same_direction* and *opposite_direction*.

relative_direction: the direction where a primary object is located w.r.t. a reference object's orientation, e.g., a vehicle's front direction. Like in [Hernández, 1994; Clementini *et al.*, 1997], qualitative values are *front*, *front_right*, *back_right*, *back*, *back_left*, *left*, and *left_front* in cyclic order.

roadway_relative_direction: like *relative_direction* but w.r.t. the roadway. Qualitative values are *front_on_roadway*, *right_on_roadway*, *back_on_roadway*, and *left_on_roadway*. This predicate is only applicable for vehicles on the same roadway.

relative_lane: the relative lane of vehicles on the same roadway. Distinguished qualitative classes are: *same_lane*, *direct_right_lane*, *direct_left_lane*, *outer_right_lane*, and *outer_left_lane*. The last two values incorporate all lanes being no direct neighbours of the reference object.

in_region: indicates whether an object's reference point (e.g., its centre) is inside a region.

rcc_relation: The topological relation between a region and a reference object (treated as region). We use RCC-5 [Randell and Cohn, 1989; Randell *et al.*, 1992] as formalism. The corresponding qualitative symbols are *not_overlapping*, *partially_overlapping*, *proper_part*, *proper_part_inverse*, and *equal*.

The above vocabulary allows for motion descriptions in consideration of special characteristics in road traffic and does not need any absolute reference system. It is based on egocentric perspectives of actors in road traffic as reference objects. More sophisticated motion predicates may be created by composition in logic formulae, e.g., proposed in [Miene, 2004]. A moment of two vehicles v_1 and v_2 in short time distance nearing each other in opposite directions on the same lane could be represented with the following predicates:

$$\begin{aligned}&\textbf{time\_distance}(v_1, v_2, close) \wedge\\ &\textbf{distance\_trend}(v_1, v_2, decreasing\_distance) \wedge\\ &\textbf{relative\_direction}(v_1, v_2, opposite\_direction) \wedge\\ &\textbf{relative\_lane}(v_1, v_2, same\_lane)\end{aligned}$$

The above simple formula just describes an arbitrary single moment (not saying which moment actually). To describe situations over time we need a temporal extension as described below.

2.3 Motion Description

Following the approach of Allen [1984] and its application to qualitative motion description and analysis in RoboCup [Miene *et al.*, 2003; Miene, 2004] and in Intelligent Vehicles domain [Miene *et al.*, 2004; Lattner *et al.*, 2005], the actual motion is represented by association of motion predicates with time intervals they hold in.

$\text{HOLDS}(p, i)$ describes the continual validity of predicate p during time interval i. The time line consists of discrete ticks. Time intervals are defined by their first and last tick (both inclusive). A development of two actors' relative spatial distance could be expressed with the following predicates:

$$\begin{aligned}&\text{HOLDS}(\textbf{space\_distance}(v_1, v_2, close), \langle 0, 15\rangle)\\ &\text{HOLDS}(\textbf{space\_distance}(v_1, v_2, very\_close), \langle 16, 25\rangle)\\ &\text{HOLDS}(\textbf{space\_distance}(v_1, v_2, zero\_distance), \langle 26, 30\rangle)\end{aligned}$$

Thus, the combination of HOLDS predicates for motion predicates and their validity intervals represents a dynamic scene. As opposed to approaches like Musto's Qualitative Motion Vectors [Musto, 2000; Musto *et al.*, 1999] this allows for flexible addition or omission of motion predicates in road traffic depending on necessity or availability.

2.4 Pattern Description

In order to utilise qualitative motion descriptions for knowledge-based situation assessment in road traffic, actual scenes are checked against situation patterns. Traffic situation patterns are realised as logic formulae of road scene and object descriptions in connection with abstract validity intervals of qualitative motion predicates and their temporal relations using Allen's [1983] and Freksa's [1992a] interval logic.

Patterns may be made up of a simple conjunction of motion predicates that have to *hold* in *one* interval or they may define sequences of intervals in qualitative interval relations. The most complicated kind of patterns describe temporally extended events with semantically defined start and end points, e.g., an overtake manoeuvre as a whole. These patterns are defined by the temporal predicate OCCURS as introduced in [Allen, 1984].

The above simple example of two mutually approaching vehicles in collision course would be expressed as follows:

$$\begin{aligned}&\text{HOLDS}(\textbf{collision\_course}(v_1, v_2), i) \Leftarrow\\ &\quad \exists\, dist:\\ &\qquad \text{HOLDS}(\textbf{distance\_trend}(\\ &\qquad\quad v_1, v_2, decreasing\_distance), i) \wedge\\ &\qquad \text{HOLDS}(\textbf{relative\_direction}(\\ &\qquad\quad v_1, v_2, opposite\_direction), i) \wedge\\ &\qquad \text{HOLDS}(\textbf{relative\_lane}(v_1, v_2, same\_lane), i) \wedge\\ &\qquad \text{HOLDS}(\textbf{time\_distance}(v_1, v_2, dist), i) \wedge\\ &\qquad dist \leq medium\_distance\end{aligned}$$

In [Gehrke, 2005] and [Lattner *et al.*, 2005] we used the qualitative scene description and background knowledge on

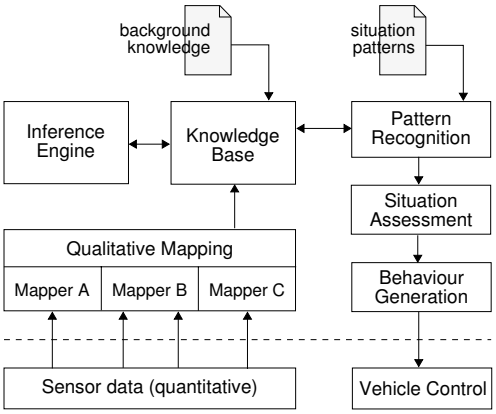


Figure 1: System architecture

traffic rules to recognise right of way at junctions and let cars stop if the situation should arise. Thereto, patterns are checked automatically during runtime using background knowledge on temporal logic and a Prolog-based inference engine. This requires online-mapping of quantitative sensor data to the qualitative representation.

3 Qualitative Mapping

Qualitative mapping is the abstraction process from quantitative sensory data to qualitative motion descriptions. There are four major challenges in qualitative mapping for traffic scene representation for intelligent vehicles:

1. Acquisition of the needed sensory data from the intelligent vehicle's environment
2. Definition of adequate qualitative equivalence classes (in terms of amount, symbols, and value bounds) for the respective motion predicate
3. Online-mapping for many objects and predicates with bounded resources in real-time.
4. Online-segmentation of quantitative data time series considering smoothing and threshold-based tolerance corridors for monotony segmentations, e.g., for trends of relative distances.

The three latter challenges are in the main focus of this paper. The first one is an important requirement for real-world feasibility but has been postponed for the time being because the main motivation is knowledge-based situation assessment and behaviour decision. Accordingly, the current testbed is a simulation environment without incomplete or noisy sensor data.

3.1 System architecture

The qualitative mapping of sensor data is embedded in the software prototype developed for knowledge-based situation assessment and behaviour control in intelligent vehicles. Figure 1 depicts the architecture of the system and its application for knowledge-based behaviour decision.

The prototype works with simulated quantitative sensor data on vehicle positions, extensions, orientations, and velocities. The mapping module generates the qualitative scene representation and stores it into a knowledge base (KB). The KB also contains background knowledge on the road network, on object and region types and their properties, and spatiotemporal knowledge. The representation language is F-Logic [Kifer *et al.*, 1995] in the variant of FLORA-2 [Yang *et al.*, 2003]. FLORA-2 uses the Prolog-like inference engine XSB [Sagonas *et al.*, 1994]. Background knowledge also allows for spatial and temporal inference. Currently this inference is barely used due to complete knowledge in the simulation prototype. Background knowledge on the semantics of the HOLDS predicate is necessary to recognise patterns in actual scenes. Within the patterns applied so far spatial inference is reduced to partonomy relations.

The qualitative scene representation in the KB is queried by a pattern matching module that checks situation patterns specified in an XML file. Query resolution is done by XSB. The patterns are used for situation assessment and behaviour generation depending on the situation.

3.2 Time Series Segmentation

Each possible motion predicate with its participating objects (i.e., actors and regions) is created by analysing the corresponding time series of quantitative sensory data. In a segmentation process the incoming sensory data that is recorded in a time series is divided into validity intervals of *qualitative* classes. This has to be done at runtime. Incoming values are added to the current interval if they belong to the same qualitative class, otherwise a new interval is started and the old one is finally closed. New intervals, their extension or end are written to the knowledge base perpetually.

Our approach of time series segmentation follows the work of Miene [2004]. Miene differentiates two segmentation modes: threshold-based segmentation and monotony-based segmentation. The first is applied to motion predicates like *velocity* or *spatial_distance*, where changes over time are not relevant for qualitative classification. The latter is applied to motion predicates like *acceleration* and *distance_trend* that are derivatives of the time series for *velocity* or *spatial_distance* respectively and therefore need to regard the previous values to determine the current qualitative class. Figure 2 gives an example for threshold and monotony segmentation.

The whole segmentation and mapping is done in *mapping cycles* that cover sensor data of a minimum of 50 ms and 100 ms in general in our tests. The duration of a mapping cycle determines the time granularity of the qualitative representation. A cycle must not endure longer than the time scope of its handled sensor data to allow for real-time application. Furthermore, due to real-time requirements, smoothing of time series may only be done within the current mapping cycle.

To be handled properly and efficiently in pattern recognition and by the temporal background knowledge, intervals formed during the segmentation process need to be maximal, i.e., a following interval of the same series (in Allen relation *meets*) has to belong to another qualitative class. This is ensured by interval extension. The current interval $i_{cur} = \langle s_{cur}, e_{cur} \rangle$ with qualitative value q_{cur} extends the interval $i_{cur-1} = \langle s_{cur-1}, s_{cur} - 1 \rangle$ with qualitative value q_{cur-1} to a combined interval $i_{cur-1} = \langle s_{cur-1}, e_{cur} \rangle$ iff

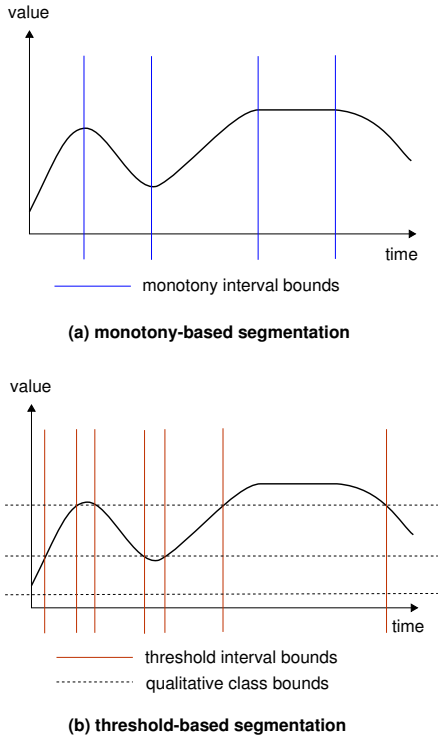


Figure 2: A fictional quantitative data time series with monotony and threshold segmentation.

$q_{cur-1} = q_{cur}$. This is done automatically as a knowledge base operation.

For instance, if the space distance of two vehicles was *close* in the previous interval and is still *close* in the time scope of the current mapping cycle from 11 to 15 the knowledge base would contain

$$\text{HOLDS}(\textbf{space\_distance}(v_1, v_2, close), \langle 0, 10\rangle)$$

before the update and afterwards

$$\text{HOLDS}(\textbf{space\_distance}(v_1, v_2, close), \langle 0, 15\rangle)$$

3.3 Mapping to Qualitative Classes

In order to determine the valid *qualitative* classes of motion predicates, the *quantitative* sensory data needs to mapped by an abstraction function for each qualitative information. The function for a motion predicate has to be applied for each object or object pair (i.e., primary object and reference object).

In the implementation of our approach the task of mapping is done by "mappers". A mapper is responsible for one or more qualitative motion predicates and all associated objects. Some motion predicates share a mapper because they analyse the same time series of sensor data, e.g., *spatial_distance* and *distance_trend* (cf. sec. 3.2). In the following we describe the realised mappers and their particular abstraction functions.

SpeedMapper

This mapper does the mapping of actor velocities (predicate *velocity*) and accelerations (*acceleration*) to the corresponding qualitative classes as both predicates use the same sensor data.

We interpret velocity depending on actor's type and context. For instance, a pedestrian with *normal* speed is significantly slower than a passenger car with *normal* speed and a *fast* passenger car on a freeway is faster than one on a normal road in town. Because we have an object taxonomy, objects are instances of multiple classes and thus their velocity may be interpreted in different object type contexts, e.g., as a passenger car, as a vehicle, or as an actor in general. So qualitative representation of *velocity* needs to state which object class is the reference for its interpretation. In general, we treat qualitative velocities in the object context of *actor*. Reference for actor is object type *passenger_car* as the standard object type in road traffic.

Situation context is taken into account in terms of maximum permitted speed for the object type in the current situation. The qualitative class *medium_speed* covers those velocity values being moderate and appropriate in the situation. For this reason the range of *medium_speed* is rather small whereas range of class *slow* is rather big in comparison to classical class separations, e.g., for distances. Figure 3 depicts the schema for qualitative classes in velocity.

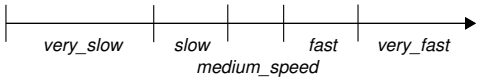


Figure 3: Schema for qualitative classes of actor velocities.

DistanceMapper

The *DistanceMapper* is responsible for the predicates *spatial_distance* and *distance_trend* as the monotony segmentation of relative distance. The spatial distance is determined for pairs of actors or an actor and a region. Objects are handled as rectangle regions and the minimal region distance is calculated to get reasonable values. Distance of actors to regions is currently restricted to nearby junctions to reduce computational complexity.

For spatial distance qualitative classes the well-known approaches of Hernández [Hernández *et al.*, 1995] were adapted. Every distance below 5 metres (slightly more than a vehicle's length) is considered *very_close* or *zero_distance* in case of contact. Distances beyond 250 metres are categorised as *very_far*. Within this distance even a vehicle of 200 km/h is able to stop. The intermediate qualitative classes are partioned homogeneously with respect to monotonicity and range restrictions constraints as introduced in [Hernández *et al.*, 1995].

The *distance_trend* motion predicate expresses the monotony property of distance changes by applying the monotony-based time series segmentation method.

Since distance and distance trend are symmetric the mapper only has to calculate one qualitative relation per object pair.

TimeDistanceMapper

The module maps the temporal distance of a pair of actors or an actor and a region to qualitative abstraction (*time_distance*). As for spatial distance, junctions are currently the

T_{TC} \ T_N	0	> 0 ≤ 1,5	> 1,5 ≤ 3	> 3 ≤ 6	> 6 ≤ 12	> 12	un-def.
0	q_0						q_0
> 0 ≤ 1,5		q_1	q_1	q_1	q_1	q_1	q_1
> 1,5 ≤ 3		q_1	q_2	q_2	q_2	q_2	q_2
> 3 ≤ 6		q_1	q_2	q_3	q_3	q_3	q_3
> 6 ≤ 12		q_1	q_2	q_3	q_4	q_4	q_4
> 12		q_1	q_2	q_3	q_4	q_5	q_5
un-def.	q_0	q_1	q_2	q_3	q_4	q_5	q_5

Figure 4: Matrix to determine qualitative time distances. The interval limits are stated in seconds. Symbols q_0 to q_5 denote the six qualitative classes.

only considered regions. In contrast to spatial distance this predicate is not symmetric.

Temporal distance combines time to collision ($T_{TC,ro}$) and net-time-gap ($T_{N,ro}$) of a reference object ro to a primary object. Both objects need to be on the same roadway. $T_{TC,ro}$ is the time to a collision or encounter of both objects with respect to the roadway regarding their current spatial distance, speed, and acceleration. $T_{N,ro}$ is the time the reference object would need to reach the primary object's current position assuming constant acceleration.

According to [Dagli *et al.*, 2002] they are defined by

$$T_{N,ro} = \frac{-v_{ro} \pm \sqrt{v_{ro}^2 + 2\Delta x \cdot a_{ro}}}{a_{ro}}$$

and

$$T_{TC,ro} = \frac{-\Delta v \pm \sqrt{\Delta v^2 + 2\Delta x \cdot \Delta a}}{\Delta a}$$

Δx denotes the spatial distance of both objects, Δv and Δa are their relative velocity and acceleration in perspective of the reference object. Some cases of undefined values need special handling, e.g., if $\Delta a = 0$.

In order to determine the qualitative time distance we use a matrix that maps the combination of both values to the corresponding class as depicted in figure 4. The matrix describes a simple function that uses the lesser distance as the one to consider. The interval limits are motivated by cognitive aspects analysed for collision avoidance systems (cf. [van der Horst and Hogema, 1993]), *reaction time*, and safe distance.

LateralPositionMapper

This mapper examines the lateral position of a vehicle in its lane (*lateral_position*) and its development (*lateral_motion*) as monotony segmentation. If the vehicle's centre is located less than 30 cm remote to the lane's centre it is considered *central_in_lane*, otherwise left or right respectively.

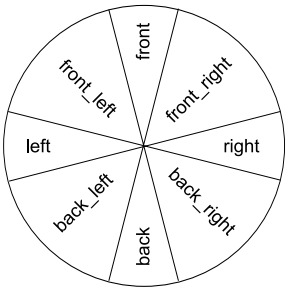


Figure 5: Relative directions between actors (disregarding the roadway context).

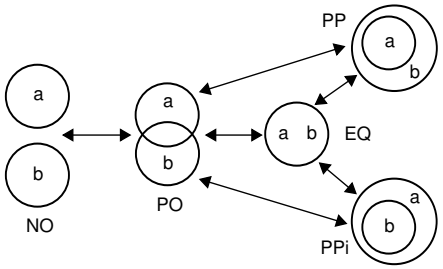


Figure 6: Topological relations between an actor and a surface region in RCC-5.

RelDirectionMapper

The *RelDirectionMapper* calculates the angle of a reference objects's front vector to another objects's position and maps the angle to a qualitative direction class (*relative_direction*).

The partitioning of angles is pictured in figure 5. Front, back, left, and right direction have a range of 30°, the other classes comprise 60°. The well-known qualitative classes and their mapping function were proposed in [Hernández, 1994].

The mapper presupposes an orientation vector for the reference object to determine the front direction.

InRegionMapper

This mapper determines whether the motion predicate *in_region* holds between an actor and a surface region. An actor is considered *in_region* iff his reference point is situated inside that region. Currently all pairs of actors and regions are checked. Due to performance issues this might be restricted to certain region types, e.g., lanes. Notice that some relations may be inferred through sub-region relation.

TopologyMapper

The *TopologyMapper* determines the topological relation of an actor (as rectangle region) to a surface region in RCC-5 model (*rcc_relation*). Figure 6 shows the relations and possible transitions between them over time.

Other Motion Predicates

Some of the motion predicates described in section 2.2 are not covered by the above mappers. This is because some predicates are not yet implemented in the prototype (*relative_speed* and *driving_direction*). Other predicates are not handled by mappers but *inferred* from qualitative motion predicates. This

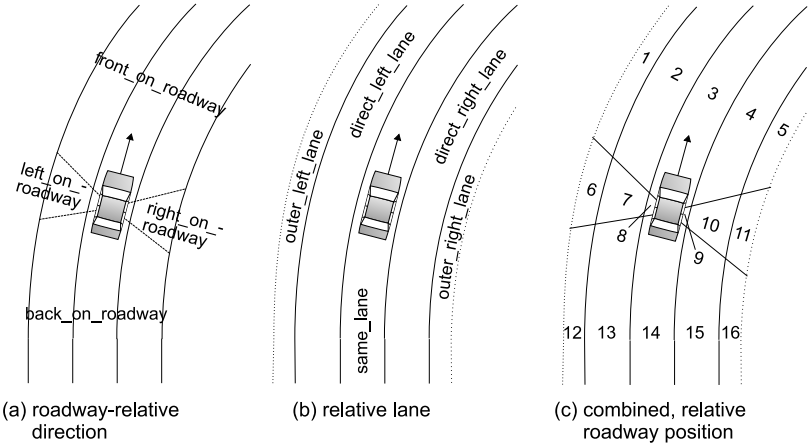


Figure 7: Relative directions and positions in relation to a reference vehicle, its roadway, and lane.

concerns *roadway_relative_direction* and *relative_lane*. Figure 7 illustrates the use of roadway relative predicates.

4 Evaluation

The evaluation of the presented approach takes two aspects into account: the application of qualitative scene representation for knowledge-based behaviour control for intelligent vehicles and the runtime performance regarding real-time requirements. In this paper we focus on performance of qualitative mapping.

The basis for both is the developed prototype that comprises the qualitative mapping through mappers and a rather simple behaviour decision based on traffic scene patterns checked at runtime. The traffic scene is simulated based on a scenario script setting the behaviour of all actors but the intelligent vehicle. The running scenario is visualised in a graphical user interface that also shows the recognised patterns.

Our behaviour decision module controls the intelligent vehicle's speed depending on the situation. With a small set of patterns a rule-conform autonomous behaviour was shown when turning off left at a junction with oncoming traffic and children running across the street. Figure 8 shows a snapshot of that scenario. Also other scenarios were tested with encouraging results [Gehrke, 2005; Lattner *et al.*, 2005].

4.1 Mapper Performance

The aim of knowledge-based behaviour decision for intelligent vehicles is in proactive control to ensure safe and rule-conform driving. Humans have a reaction time of about 1 second and are capable of safe driving nevertheless. Of course an intelligent vehicle should act faster.

The challenge is to do all: qualitative mapping, pattern recognition, situation assessment, and behaviour generation within an adequate decision cycle. Qualitative mapping cycles from 50 to 250 milliseconds time granularity were surveyed. All mappers have to complete their work within the scheduled time or the qualitative scene representation will get more and more dated, i.e., the reaction time increases.

In order to evaluate the approach different measures were extracted from log files created by the prototype. Six traffic scenarios with different characteristics were tested and repeated five times. The six scenarios, in turn, were modified in reference to the amount of participating actors and time granularity of mapping.[3]

Figure 9 shows that up to seven actors can be handled within 150 milliseconds on average. Numbers above have proven critical in the current implementation. However, there is no substantial influence on duration of pattern matching cycles when the amount of actors is increased.

It is important to note that the development of mapping duration for one time slot has to be considered. As mentioned above, the mapping time will increase more and more if not all sensor values could be processed within the intended slot. This is depicted clearly in figure 10. If there are ten actors, the duration of a mapping cycle increases enormously over time. Interestingly, it also increases after some cycles for seven actors, although it seems to be processed within time before. This is not because of unprocessed old sensor data but owing to increasing duration of single knowledge base operations over time. The problem is currently analysed and has no reason in the prototype's software design.

To get more information on performance issues, the mappers were examined separately. Table 1 gives a review of the average mapping cycle duration for each mapper type. The *DistanceMapper* is by far the most expensive one. It has to handle distance and distance change for all object pairs. Additionally, the calculation of correct region distances – by an external library – turns out to be costly which should provide some possibilities for improvement.

The current main issue is the increasing time for single knowledge base modifications which should be addressed with highest priority. On the other hand, real-time algorithms have to be applied that adapt precision, considered motion predicates and objects with respect to their importance and available processing time.

[3] Experiments were run on a Pentium M 1.6 GHz system with 512 MB RAM and Microsoft Windows XP as operating system.

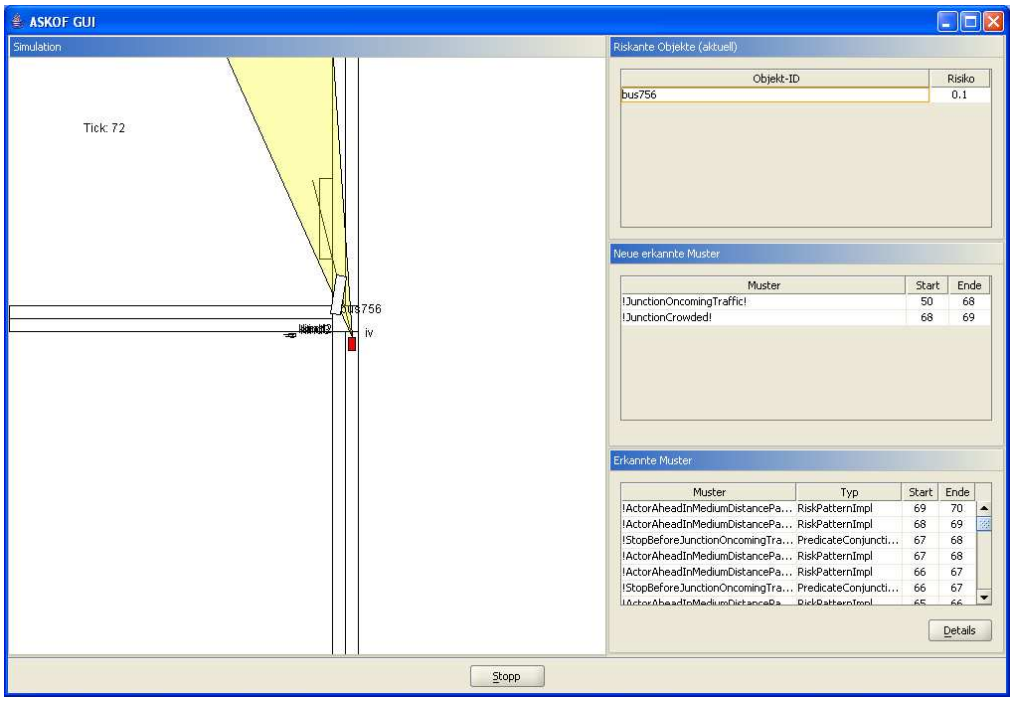


Figure 8: Test scenario with an intelligent vehicle stopping on a junction due to oncoming traffic to turn off after the traffic passed. The triangle going off from the vehicle *iv* indicates the direction of its camera sensor.

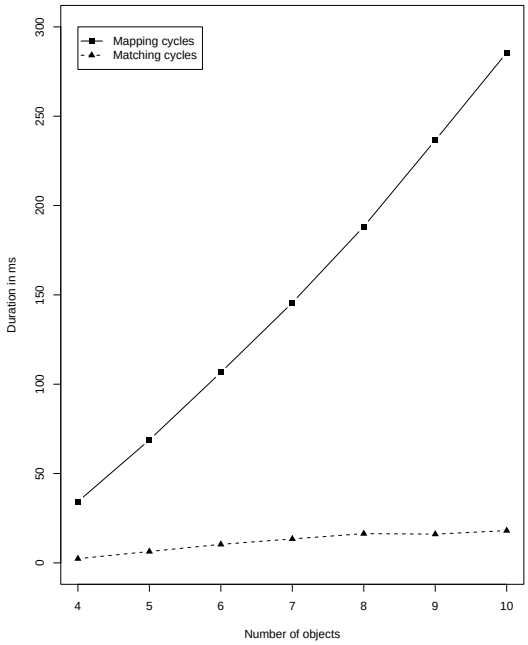


Figure 9: Duration of mapping and pattern matching at growing number of objects and a time slot for mapping of 100 ms.

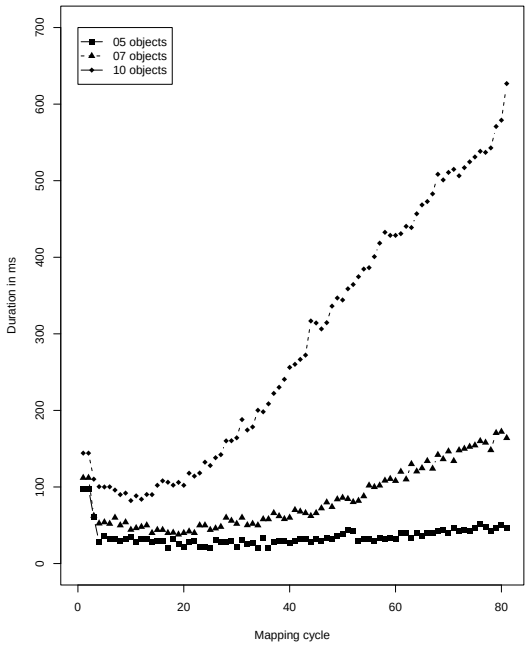


Figure 10: Development over time of mapping cycle duration with time slot of 100 ms.

Mapper	Duration in ms	Commands
SpeedMapper	0.46 ± 2.10	7.96 ± 0.93
TimeDistanceMapper	2.97 ± 4.53	3.13 ± 0.98
DistanceMapper	14.52 ± 8.04	19.90 ± 0.98
RelDirectionMapper	6.01 ± 5.83	12.00 ± 0.00
LateralPositionMapper	5.10 ± 5.10	7.96 ± 0.39
InRegionMapper	1.63 ± 3.73	2.86 ± 2.08
ObjectTopologyMapper	3.28 ± 4.84	3.25 ± 2.52

Table 1: Average duration of a mapping cycle with transmitted knowledge base manipulation commands. Test runs were made at a mapping time slot of 100 ms and with 4 actors and 1 junction.

5 Conclusion

In this paper we presented a qualitative representation for the description of traffic scenes. In order to create such a representation quantitative data as it might be supplied by sensors must be mapped to the symbolic vocabulary which is based on a number of existing qualitative spatial and temporal representations. We have shown how this mapping and a matching of traffic patterns can be performed and presented evaluation results with simulated traffic scenes. It is beyond doubt that many real-world challenges were left out by the simulation, e.g., real-time image processing, object tracking, and handling noisy data.

The evaluation results indicate the feasibility in principle. Knowledge about traffic participants, networks of streets and relevant segments, motion of dynamic objects, relations between objects and between objects and ground regions can be stored in the knowledge base. In experiments up to seven dynamic objects could be managed by the system without efficiency problems when the mapping interval was set to 100 - 150 ms. However, experiments with a growing number of objects show that there actually is a problem with complexity. As some mapping modules compute relations between all pairs of dynamic objects the growth of the duration of the mapping cycles is approximately quadratic.

In our experiments the mapping cycles took too much time when more than seven objects were moving in the scenes. It should be considered that the number of relevant objects for behaviour decision on a high level usually is not that large. The most time-consuming part during mapping are the interactions with the knowledge base. Thus, here is some potential for improving the efficiency. In our current implementation intervals are extended in each cycle, i.e., interaction with the knowledge base happen all the time. If we introduced open intervals, interactions with the KB could be reduced enormously because interaction would only be necessary if a monotonicity criterion is not valid any longer or some other property switches to another qualitative class. Another way to improve performance is to perform the mappings of the single mapping modules concurrently. This was not possible earlier because XSB just allowed for sequential processing of queries and commands. In recent XSB versions concurrent interactions are supported.

Future work could address different directions. So far in our experiments for actual vehicle control just the speed of the IV was controlled by the behaviour decision module. It would be interesting to set up more patterns for situation assessment and implementing a more complex behaviour. Another research direction could address the prediction of the behaviour of other traffic participants based on our qualitative representation, e.g., by probabilistic approaches like Bayesian Networks and their extensions. In this work we assumed that there is just one value per time step in the time series (e.g., just one value for the velocity of an object). It would be interesting to investigate impact on time series segmentation if probabilistic distributions are taken into account instead of single values.

Acknowledgment

The content of this paper is a partial result of the ASKOF project which was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant HE989/6-1. We would like to express our gratitude to our student research assistants Jan Gerken and Hanna Bauerdick for many interesting discussions and ideas and their effort in the implementation of the ASKOF prototype. We also want to thank the anonymous reviewers for giving valuable comments on how to improve the paper and for pointing out interesting future research directions.

References

[Allen, 1983] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

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

[Bertozzi *et al.*, 2000] Massimo Bertozzi, Alberto Broggi, and Alessandra Fascioli. Vision-based intelligent vehicles: State of the art and perspectives. *Robotics and Autonomous Systems*, 32(1):1–16, 2000.

[Clementini *et al.*, 1997] Eliseo Clementini, Paolino Di Felice, and Daniel Hernández. Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356, 1997.

[Dagli *et al.*, 2002] Ismail Dagli, Michael Brost, and Gabi Breuel. Action recognition and prediction for driver assistance systems using dynamic belief networks. In *Proceedings of the Conference on Agent Technologies, Infrastructures, Tools, and Applications for E-Services*, pages 179–194, 2002.

[Dagli *et al.*, 2004] Ismail Dagli, Gabi Breuel, Helmut Schittenhelm, and Alexander Schanz. Cutting-in vehicle recognition for ACC systems - towards feasible situation analysis methodologies. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 925–930, Parma, Italy, June 14-17 2004.

[Dickmanns, 2002] Ernst D. Dickmanns. The development of maschine vison for road vehicles in the last decade. In

Proceedings of the IEEE Intelligent Vehicles Symposium (IV'02), pages 268–281, Versailles, France, 2002.

[Freksa, 1992a] Christian Freksa. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1–2):199–227, 1992.

[Freksa, 1992b] Christian Freksa. Using orientation information for qualitative spatial reasoning. In A. U. Frank, I. Campari, and U. Formentini, editors, *Proceedings of the International Conference on Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 162–178, 1992.

[Gehrke, 2005] Jan D. Gehrke. Qualitative Szenenrepräsentation für intelligente Fahrzeuge. Master's thesis, Department for Mathematics and Computer Science, University of Bremen, 2005.

[Hernández *et al.*, 1995] Daniel Hernández, Eliseo Clementini, and Paolino Di Felice. Qualitative distances. In A. U. Frank and W. Kuhn, editors, *Spatial Information Theory: A Theoretical Basis for GIS. European Conference, COSIT'93*, volume 988 of *Lecture Notes in Artificial Intelligence*, pages 45–58. Springer, Berlin, 1995.

[Hernández, 1994] Daniel Hernández. *Qualitative Representation of Spatial Knowledge*, volume 804 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin, 1994.

[Kifer *et al.*, 1995] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[Lattner *et al.*, 2005] Andreas D. Lattner, Jan D. Gehrke, Ingo J. Timm, and Otthein Herzog. A knowledge-based approach to behavior decision in intelligent vehicles. In *IEEE Intelligent Vehicles Symposium*, Las Vegas, USA, June 6-8 2005. To appear.

[Miene *et al.*, 2003] Andrea Miene, Ubbo Visser, and Otthein Herzog. Recognition and prediction of motion situations based on a qualitative motion description. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003 - Proceedings of the International Symposium*, Padua, Italy, July 8-9 2003. On CD-ROM.

[Miene *et al.*, 2004] Andrea Miene, Andreas D. Lattner, Ubbo Visser, and Otthein Herzog. Dynamic-preserving qualitative motion description for intelligent vehicles. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV'04)*, pages 642–646, Parma, Italy, 2004.

[Miene, 2004] Andrea Miene. *Räumlich-zeitliche Analyse von dynamischen Szenen*, volume 279 of *Dissertationen in Künstlicher Intelligenz (DISKI)*. Akademische Verlagsgesellschaft, Berlin, 2004.

[Musto *et al.*, 1999] Alexandra Musto, Klaus Stein, Andreas Eisenkolb, and Thomas Röfer. Qualitative and quantitative representations of locomotion and their application in robot navigation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1067–1073, 1999.

[Musto, 2000] Alexandra Musto. *Qualitative Repräsentation von Bewegungsverläufen*. PhD thesis, Fakultät für Informatik, Technische Universität München, 2000.

[Randell and Cohn, 1989] David A. Randell and Anthony G. Cohn. Modelling topological properties in physical processes. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 55–66. Morgan Kaufman, 1989.

[Randell *et al.*, 1992] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufmann, 1992.

[Rüder *et al.*, 2002] M. Rüder, W. Enkelmann, and R. Garnitz. Highway lane change assistant. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 240–244, Versaille, France, June 17-21 2002.

[Sagonas *et al.*, 1994] Konstantinos Sagonas, Terrance Swift, and David S. Warren. XSB as an efficient deductive database engine. In R. T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data (SIGMOD'94)*, pages 442–453, 1994.

[van der Horst and Hogema, 1993] Richard van der Horst and Jeroen Hogema. Time-to-collision and collision avoidance systems. In *Proceedings of the 6th workshop of the International Cooperation on Theories and Concepts in Traffic Safety (ICTCT)*, pages 15–22, Salzburg, Austria, 1993.

[Wagner *et al.*, 2004] Thomas Wagner, Ubbo Visser, and Otthein Herzog. Egocentric qualitative spatial knowledge representation for physical robots. *Robotics and Autonomous Systems*, 49:25–42, 2004.

[Weiss *et al.*, 2004] Kristian Weiss, Nico Kaempchen, and Alexander Kirchner. Multiple-model tracking for the detection of lane change maneuvers. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 937–942, Parma, Italy, June 14-17 2004.

[Yang *et al.*, 2003] Guizhen Yang, Michael Kifer, and Chang Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *Proceedings of the 2nd International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, pages 671–688, Catania, Italy, November 2003.

A Relative Orientation Algebra with Adjustable Granularity

Reinhard Moratz, Frank Dylla, and Lutz Frommberger
Universität Bremen
Bibliothekstr. 1, 28359 Bremen, Germany
{moratz, dylla, lutz}@sfbtr8.uni-bremen.de

Abstract

The granularity of spatial calculi and the resulting mathematical properties have always been a major question in solving spatial tasks qualitatively. In this paper we present the Oriented Point Relation Algebra ($\mathcal{OPRA}_m$), a new orientation calculus with adjustable granularity. Since our calculus is a relation algebra in the sense of Tarski, fast standard inference methods can be applied. One of the major problems—depending on the environment, the robots' capabilities and the tasks to be solved—is the choice of the granularity of an applied calculus. To present, granularity had to be chosen at the start and could not be changed on the fly. In a dynamically changing environment under real time conditions it is necessary to choose a coarse but still adequate granularity of the spatial representation: only in that case irrelevant feature changes fail to trigger unnecessary inference steps. A qualitative, coarse abstraction suppresses tiny changes in the environment and leads to fast computation.

1 Introduction

Most robots currently used for research issues are equipped with a broad variety of fairly reliable sensors. Edutainment robots however often have only low quality sensors. Despite this, they have become increasingly popular and must be able to solve complex spatial tasks even when accurate distance and orientation information is not obtainable. Qualitative reasoning may allow them to do so.

Qualitative Reasoning about space abstracts from the physical world and enables computers to make predictions about spatial relations, even when a precise quantitative information is not available [Cohn, 1997]. The two main trends in Qualitative Spatial Reasoning are topological reasoning about regions [Cohn, 1997; Renz and Nebel, 1998] and positional reasoning about point configurations [Freksa, 1992; Schlieder, 1995]. Positional reasoning, i.e. distance and orientation, in particular is important for robot navigation [Musto *et al.*, 1999].

Calculi dealing with such information have been well investigated over recent years and provide sound reasoning strategies, e.g. about topological relations between regions as in RCC-8 [Randell and Cohn, 1989; Randell *et al.*, 1992], about the relative position orientation of three points as in Freksa's Double Cross Calculus [Freksa, 1992] or about orientation of two line segments as in the Dipole Calculus [Moratz *et al.*, 2000; Schlieder, 1995]. Standard constraint-based reasoning techniques can be applied for reasoning with calculi such as the above mentioned ones. For example, Schlieder [1995] sketched how a qualitative calculus like the Dipole Calculus might be applied to robot navigation.

One of the major problems is the choice of the granularity of an applied calculus according to the environment, the robots' capabilities and the tasks that have to be solved. To present, this granularity had to be chosen in the beginning and could not be changed on the fly. In a dynamically changing environment under real time conditions it is necessary to choose a coarse yet adequate granularity of the spatial representation: only in that case will irrelevant feature changes fail to trigger unnecessary inference steps. A qualitative, coarse abstraction suppresses tiny changes in the environment and results in fast computation.

With the Oriented Point Relation Algebra $\mathcal{OPRA}_m$ we present a calculus whose granularity is scalable with a parameter $m \in \mathbb{N}$. The parameter can be adjusted according to perception and motion capabilities. The reasonable maximum, i.e. the finest reasonable granularity, correlates to the resolution and error of perception and motion. Yet, it would be unwise to use the finest resolution possible just to answer a question whether an object is to the left or right. We present an integration schema where data represented in different granularities can be mixed when deriving new relations from prior observations. The rest of the paper is organized as follows: After a brief introduction of related qualitative spatial calculi and their according properties, we will introduce the $\mathcal{OPRA}_m$ calculus. First we will give a definition for the coarsest type ($m = 1$), followed by the model for arbitrary $m \in \mathbb{N}$ including the rules for composition of base relations. In the end we will give an example with linguistic commands and coarse perceived configuration information that have to be integrated by constraint propagation to achieve survey knowledge.

2 Related Work

Qualitative Spatial Reasoning (QSR) is an abstraction that summarizes similar quantitative states into one qualitative

characterization. From a cognitive perspective the qualitative method *compares* features of the domain rather than *measuring* them in terms of some artificial external scale [Freksa, 1992]. The two main directions in QSR are topological reasoning about regions, e.g. the RCC-8 [Randell and Cohn, 1989; Randell *et al.*, 1992], and positional (distance and orientation) reasoning about point configurations. An overview is given in [Cohn and Hazarika, 2001]. We will concentrate on the most important positional calculi for our work.

The Double Cross calculus [Zimmermann and Freksa, 1996] is an approach based on fundamental knowledge about human spatial reasoning. In contrast to previous approaches the base relations do not only describe a relative point position wrt. a single point, but wrt. a vector. In other words, an observer tries to relate to a point C while he is walking from position A to B. In [Scivos and Nebel, 2001] it was shown that the calculus is not closed under permutation and composition, and that reasoning with a set of base relations is NP-hard. A further application driven development based on the scheme above is the Ternary Point Configuration Calculus (TPCC) [Moratz *et al.*, 2003]. We will describe this calculus in more detail in section 4.1.

Schlieder [1995] proposed a calculus with 14 basic relations based on line segments with clockwise or counter clockwise orientation of generating starting points. Isli and Cohn [1998] presented a ternary algebra for reasoning about orientation containing a tractable subset of base relations. Schlieder's approach was extended for robot navigation tasks in [Moratz *et al.*, 2000; Dylla and Moratz, 2005], resulting in relation algebras in the sense of Tarski [Ladkin and Maddux, 1994] at different levels of granularity.

Clementini *et al.* [1997] introduced a binary approach for dealing with qualitative relations at an arbitrary level of granularity. The angles are not necessarily equidistant. Their approach did not provide a general and restrictive schema for reasoning with multiple position expressions. Also no concept for combining relations at different levels was given.

In [Renz and Mitra, 2004] the Star Calculus, a qualitative direction calculus with arbitrary granularity, was introduced. The relation of two points in the plane with respect to one global reference direction is expressed, which leads to $4m+1$ basic relations. These basic relations form a relation algebra for the cases with uniform angles. The authors claim that when using a Star Calculus with more than two reference lines, the boundary between qualitative and quantitative representation disappears. The main disadvantage of the Star Calculus is its need for a global reference direction which must always be available at each point in space.

The extended panorama approach was presented in [Wagner *et al.*, 2003]. The representation is based on the cyclic ordering information of a 360° view within the reference frame of an observing agent and on qualitative distance information. It can be interpreted as an ordered set of relations between an oriented object and the according observed point. Due to this structure it is rotational and translational invariant. Updating the model due to changes in a dynamic environment can simply be done by changing the order. Different levels of granularity were also introduced. No formal method for granularity switches or composition of local observations into survey knowledge was given.

3 The Oriented Point Relation Algebra ($\mathcal{OPRA}_m$)

Objects and locations are represented as simple, featureless points in aforementioned approaches on orientations. In contrast, our paper presents a positional calculus which uses more complex basic entities: It is based on objects which are represented as oriented points. It is closely related to a previously designed calculus which is based on straight line segments (dipoles) [Moratz *et al.*, 2000]. In [Dylla and Moratz, 2005] the dipole approach was extended for modeling behavior in dynamic environments. Conceptually, our new calculus can be viewed as a transition from oriented line segments with concrete length to line segments with infinitely small length. In this conceptualization the length of the objects no longer has any importance. Thus, only the direction of the objects is modeled. *O-points*, our term for oriented points, are specified as pair of a point and a direction on the 2D-plane.

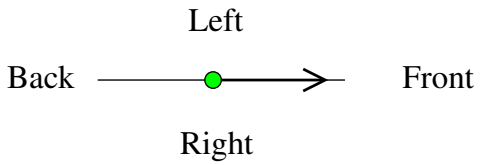


Figure 1: An oriented point and its qualitative spatial relative directions

3.1 Reasoning with Coarse O-Point Relations

In the coarsest representation a single o-point induces the sectors depicted in figure 1. "Front" and "Back" are linear sectors. "Left" and "Right" are half-planes. The position of the point itself is denoted as "Same". A qualitative spatial relative orientation relation between two o-points is represented by the sector in which the second o-point lies with respect to the first one and by the sector in which the first one lies with respect to the second one.

For the general case of the two points having different positions we use the concatenated string of both sector names as the relation symbol. Then the configuration shown in figure 2 is expressed with the relation $A\ \mathrm{RightLeft}\ B$. If both points share the same position the relation symbol starts with the word "Same" and the second substring denotes the direction of the second o-point with respect to the first one as shown in figure 3.

Altogether we obtain 20 different atomic relations (four times four general relations plus four with the o-points at the same position). These relations are jointly exhaustive and pairwise disjoint (JEPD). The relation SameFront is the identity relation. We use $\mathcal{OP}_1$ to refer to the set of 20 atomic relations, and $\mathcal{OPRA}_1$ to refer to the power set of $\mathcal{OP}_1$ which contains all 2^{20} possible unions of the atomic relations.

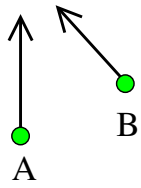


Figure 2: Qualitative spatial relation between two oriented points at different positions. The qualitative spatial relation depicted here is A RightLeft B (which reads: B is to the right of A, and A is to the left of B).

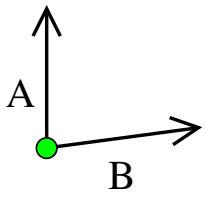


Figure 3: Qualitative spatial relation between two oriented points located at the same position. The qualitative spatial relation depicted here is A SameRight B (which reads: A and B are at the same location, and B is heading right with respect to A).

For reasoning about the o-point relations we apply constraint-based reasoning techniques which were originally introduced for temporal reasoning [Allen, 1983] and also proved valuable for spatial reasoning [Renz and Nebel, 1998; Isli and Cohn, 2000]. In order to apply these techniques to a set of relations, the relations must form a relation algebra [Ladkin and Maddux, 1994], i.e. the atomic relations must be jointly exhaustive and pairwise disjoint and they must be closed under composition ($\circ$), intersection ($\cap$), complement ($\bar{}$), and converse ($\smile$). There must also be an empty relation, a universal relation, and an identity relation. While the converse, the complement, and the intersection of relations can be computed from the set-theoretic definitions of the relations, the composition of relations must be computed based on the semantics of the relations. The compositions are usually computed only for the atomic relations and then stored in a composition table. The composition of compound relations can be obtained as the union of the compositions of the corresponding atomic relations. The compositions of the atomic relations can be deduced directly from the geometric semantics of the relations (see section 3.4).

O-point constraints are written as xRy where x, y are variables for o-points and R is a $\mathcal{OPRA}_1$ relation. Given a set Θ of o-point constraints, an important reasoning problem is deciding whether Θ is *consistent*, i.e., whether there is an assignment of all variables of Θ with dipoles such that all constraints are satisfied (a *solution*). We call this problem OPSAT. OPSAT is a Constraint Satisfaction Problem (CSP) [Mackworth, 1977] and can be solved using the standard methods developed for CSPs with infinite domains (see, e.g., [Ladkin and Maddux, 1994]).

A partial method for determining inconsistency of a set of constraints Θ is the *path-consistency method*, which enforces path-consistency on Θ [Mackworth, 1977]. A set of constraints is path-consistent if and only if for any two consistent variable instantiations, there exists an instantiation of any third variable such that the three values taken together are consistent. It is necessary but not sufficient for the consistency of a set of constraints that path-consistency can be enforced. A naive way to enforce path-consistency is to strengthen relations by successively applying the following operation until a fixed point is reached:

$$\forall i, j, k: \quad R_{ij} \leftarrow R_{ij} \cap (R_{ik} \circ R_{kj})$$

where i, j, k are nodes and R_{ij} is the relation between i and j. The resulting set of constraints is equivalent to the original set, i.e. it has the same set of solutions. If the empty relation occurs while performing this operation, Θ is inconsistent, otherwise the resulting set is path-consistent.

3.2 Finer Grained O-Point Calculi

The design principle for $\mathcal{OPRA}_1$ can be generalized to calculi $\mathcal{OPRA}_m$ with arbitrary $m \in \mathbb{N}$. Then an angular resolution of $\frac{2\pi}{2m}$ is used for the representation (a similar scheme for absolute direction instead of relative direction was recently designed by Renz and Mitra [2004]).

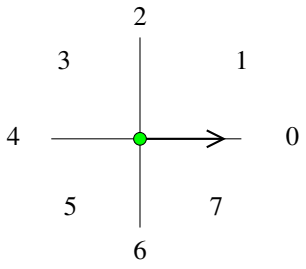


Figure 4: $\mathcal{OPRA}_2$ granularity

To formally specify the o-point relations we use two-dimensional continuous space, in particular $\mathbb{R}^2$. Every o-point S on the plane is an ordered pair of a point $\mathbf{p}_S$ represented by its Cartesian coordinates x and y, with $x, y \in \mathbb{R}$ and and a direction ϕ_S.

$$S = (\mathbf{p}_S, \phi_S), \qquad \mathbf{p}_S = ((\mathbf{p}_S)_x, (\mathbf{p}_S)_y)$$

We distinguish the relative locations and orientations of the two o-points A and B expressed by a calculus $\mathcal{OPRA}_m$ according to the following scheme. We use the symbol φ_{AB} for

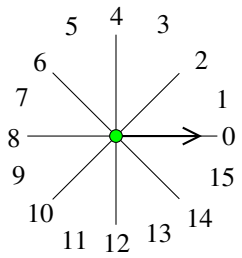


Figure 5: $\mathcal{OPRA}_4$ granularity

$\tan^{-1}\frac{(\mathbf{p}_B)_y-(\mathbf{p}_A)_y}{(\mathbf{p}_B)_x-(\mathbf{p}_A)_x}$ ($\tan^{-1}$ has two arguments, the numerator, and the denominator, and maps to the interval $[0, 2\pi]$). Figures 4 and 5 show the resulting granularity for $m = 2$ and $m = 4$. According to the cyclic order of the directions it is appropriate to enumerate them by using the $4m$ elements of the cyclic group $\mathcal{Z}_{4m}$.

If $\mathbf{p}_A \neq \mathbf{p}_B$ the relation $A\ {}_m\angle_i^j\ B$ ($i, j \in \mathcal{Z}_{4m}$) reads like this: Given a granularity m, the relative position of B with respect to A is described by j and the relative position of A with respect to B is described by i.

Formally, it represents the following set of configurations:

$$\begin{aligned} & \left(\left((i \equiv_2 1) \wedge \left(2\pi\tfrac{i-1}{4m} < \varphi_{AB} - \phi_A < 2\pi\tfrac{i+1}{4m}\right)\right)\right. \\ \vee\ & \left.\left((i \equiv_2 0) \wedge \left(\varphi_{AB} - \phi_A = 2\pi\tfrac{i}{4m}\right)\right)\right) \\ \wedge\ & \left(\left((j \equiv_2 1) \wedge \left(2\pi\tfrac{j-1}{4m} < \varphi_{AB} - \phi_B < 2\pi\tfrac{j+1}{4m}\right)\right)\right. \\ \vee\ & \left.\left((j \equiv_2 0) \wedge \left(\varphi_{AB} - \phi_B = 2\pi\tfrac{j}{4m}\right)\right)\right) \end{aligned} \quad (1)$$

$a \equiv_2 b$ stands for $a \mod 2 = b \mod 2$. Using this notation, a simple manipulation of the parameters yields the converse operation $({}_m\angle_j^i)^\smile = {}_m\angle_i^j$.

If $\mathbf{p}_A = \mathbf{p}_B$, the relation $A\ {}_m\angle i\ B$ represents the following set of configurations:

$$\begin{aligned} & \left((i \equiv_2 0) \wedge \left(\phi_B - \phi_A = 2\pi\tfrac{i}{4m}\right)\right) \\ \vee\ & \left((i \equiv_2 1) \wedge \left(2\pi\tfrac{i-1}{4m} < \phi_B - \phi_A < 2\pi\tfrac{i+1}{4m}\right)\right) \end{aligned} \quad (2)$$

Hence the relation for two identical o-points $A = B$ for arbitrary $m \in \mathbb{N}$ is $A\ {}_m\angle 0\ B$. Using this notation a simple manipulation of the parameters yields the converse operation $({}_m\angle i)^\smile = {}_m\angle(4m - i)$. The composition tables for the atomic relations of the $\mathcal{OPRA}_m$ calculi can be generated using a schema which is based on the parameters m, i, j of the corresponding relations (analogous to the generating scheme for the converse operation). We describe the schema for the composition operation in section 3.4.

To clarify the notation above we will give examples here. The configuration in figure 1 with $m = 1$ for example results in $A\ {}_1\angle_3^1\ B$. Front in this schema is denominated with 0, Left is 1, Back is 2 and Right is 3. In figure 6 the same configuration is shown with the reference frame for $m = 2$. This results in relation $A\ {}_2\angle_7^1\ B$. Thus we can say that B lies in segment 7 regarding A and A lies in segment 1 relative to B. For $m = 4$ (figure 6) we get $A\ {}_4\angle_{13}^3\ B$.

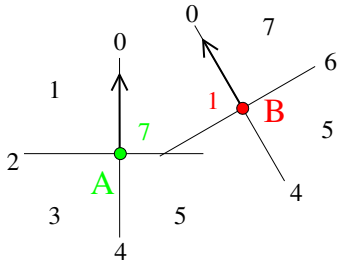


Figure 6: Two o-points in relation $A\ {}_2\angle_7^1\ B$

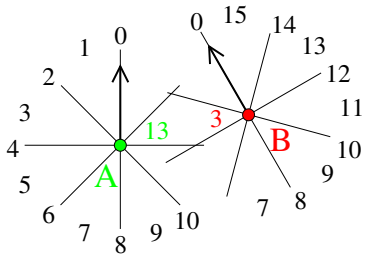


Figure 7: Two o-points in relation $A\ {}_4\angle_{13}^3\ B$

3.3 The Triangle Constraint

Besides the composition we have an additional source for spatial knowledge. The following scene is given: An agent is at position A and perceives object C including the view angle relative to the current heading. Then the agent turns towards position B, moves there and perceives the relative angle to object C again. We now are able interpret this setting as a triangle (compare figure 8). α is defined by the difference of the original heading, the view angle and the heading towards B. β is determined accordingly after the perception. Due to general knowledge about triangles ($\alpha + \beta + \gamma = \pi$) we are able to derive γ.

With A_B we denote the o-point positioned at A and orientated towards position B. In the following, i, k and the according arithmetic operators are still defined in $\mathcal{Z}_{4m}$. Within $\mathcal{OPRA}_m$, α now may be described as

$$A_C\ {}_m\angle k\ A_B$$

and respectively β as

$$B_A \ {}_m\angle i \ B_C \quad .$$

Assuming A, B, C being ordered in mathematically positive orientation and $k \equiv_2 0 \vee i \equiv_2 0$, we can conclude angle γ:

$$C_B \ {}_m\angle(2m - k - i) \ C_A \quad .$$

Thus we can generate additional relations for C with two prior perceptions.

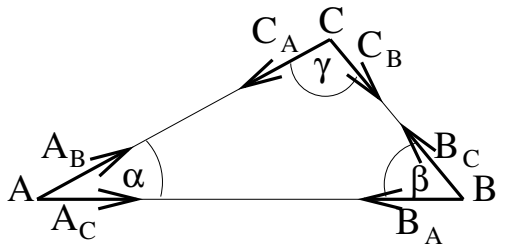


Figure 8: A triangle defined by the o-points A,B and C

3.4 Composition of Relations

Throughout this section we assume that three o-points A, B, C and the relations $A_m\angle_i^j B$ and $B_m\angle_k^l C$ are given. First we also assume that $\mathbf{p}_A \neq \mathbf{p}_B \neq \mathbf{p}_C$.

In the case of uneven i, j, k and l they correspond to open angular intervals according to (1). ${}_m\angle_i^j$ is called a *totally planar* relation, if $i \equiv_2 1 \wedge j \equiv_2 1$. If $(i + j) \equiv_2 1$, ${}_m\angle_i^j$ is called a *semi-planar* relation. ${}_m\angle_i^j$ is called a *linear* relation, if $i \equiv_2 0 \wedge j \equiv_2 0$.

First we will describe the composition procedure for the special case of totally planar relations, because it is rather straight forward. In the next section we will generalize to a common procedure for arbitrary $\mathcal{OPRA}_m$ relations. In the end we point out how to compose the so-called "same" relations, where two o-points share the same location.

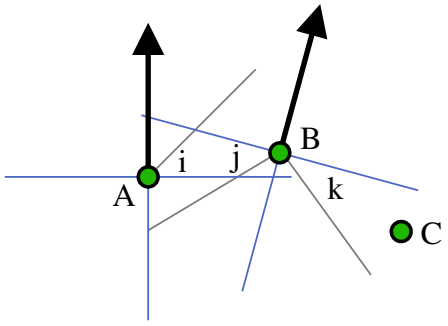


Figure 9: Composition of two $\mathcal{OPRA}_4$ relations $A_4\angle_i^j B$ and $B_4\angle_k^l C$. In this example the values are $i = 13$, $j = 5$ and $k = 11$ (see figure 5). Because the direction of C is not depicted in this example, no value of l is given. As a result of the composition, C may lie in sectors 9 to 13 with respect to A.

Composition of Totally Planar Relations

Composition of two totally planar relations $A_m\angle_i^j B$ and $B_m\angle_k^l C$ is mainly a composition of angular intervals. If we want to describe the relative position of C with respect to A, we need to combine the angular intervals which correspond to i, j and k. The first possible sector which can contain C is either i or $i - j + k - 2m - 2$, depending on which one is "first" in a circular order.[1] The last possible sector is either i or $i - j + k - 2m + 2$. To determine this, we define a minimum and a maximum relation within a cyclic group $\mathcal{Z}_n$ ($n \in \mathbb{N}$) with $a, b \in \mathcal{Z}_n$:

$$\min_{\mathcal{Z}}(a,b) = \begin{cases} \min(a,b) & |b-a| < \frac{n}{2} \\ \max(a,b) & |b-a| > \frac{n}{2} \\ b & |b-a| = \frac{n}{2} \end{cases} \tag{3}$$

$$\max_{\mathcal{Z}}(a,b) = \begin{cases} \max(a,b) & |b-a| < \frac{n}{2} \\ \min(a,b) & |b-a| > \frac{n}{2} \\ b & |b-a| = \frac{n}{2} \end{cases} \tag{4}$$

For the sake of simplicity we assume that $\min(a,b)$ is the minimum of the corresponding natural numbers to a and b. $\max_{\mathcal{Z}}(a,b)$ is defined analogously to $\min_{\mathcal{Z}}(a,b)$.

All sectors and linear relations between the first (s_1) and the last possible one (s_2) may contain C. Analogously, we also get a first and a last sector around C which can contain A:

$$\begin{aligned} s_1 &= \min_{\mathcal{Z}}(i, i-j+k-2m-2) \\ s_2 &= \max_{\mathcal{Z}}(i, i-j+k-2m+2) \\ t_1 &= \min_{\mathcal{Z}}(l, l-k+j-2m-2) \\ t_2 &= \max_{\mathcal{Z}}(l, l-k+j-2m+2) \end{aligned} \tag{5}$$

We get all possible directions (a full circle) if $s_1 = s_2$ or $t_1 = t_2$, because a composition of totally planar relations can never result in a single sector:

$$s_1' = \begin{cases} s_1 & s_1 \neq s_2 \\ 0 & s_1 = s_2 \end{cases} \quad s_2' = \begin{cases} s_1 & s_1 \neq s_2 \\ 4m-1 & s_1 = s_2 \end{cases}$$

t_1' and t_2' are derived analogously.

We achieve a disjunction of relations in which C can be with respect to A and a disjunction of relations in which A can be with respect to C. The overall result is a disjunction of all possible combinations:

$$A \ {}_m\angle_i^j \ B \circ B \ {}_m\angle_k^l \ C = \bigvee_{a=s_1'}^{s_2'} \bigvee_{b=t_1'}^{t_2'} A \ {}_m\angle_a^b \ C \tag{6}$$

[1] This notation, of course, is simplified: We need to consider m an element of the cyclic group as well, but we did not want to introduce another symbol for this purpose.

Composition of Arbitrary Relations
In this section we present a generalized schema for determining the composition of arbitrary $\mathcal{OPRA}_m$ relations. The only cases to be excluded are the "same" relations, which are described in the following section, and those resulting in a linear sector or a disjunction of two linear sectors:

$$((j = k + 2m) \vee (j = k)) \wedge j \equiv_2 0 \wedge k \equiv_2 0 \qquad (7)$$

The solution for these few cases can be constructed fairly easily. For all other cases the procedure is as follows:

A linear part of an $\mathcal{OPRA}_m$ relation can be seen as an angular interval $[\alpha_1, \alpha_2]$ with $\alpha_1 = \alpha_2$. According to the second and fourth line of (1) the composition formula must be adapted for the cases of even values of i, j, k and l. Therefore a linearity correction term

$$\psi(i, j, k) = \sum_{a \in \{i,j,k\}} ((a + 1) \mod 2) \qquad (8)$$

is incorporated to the equations in (5). ψ counts the number of linear relations. Simply adding (or subtracting) ψ, however, may deliver (half) closed intervals in the case of i or l being even; but this cannot happen. So we can make sure to achieve open intervals by using modified minimum and maximum relations for $\mathcal{Z}_n$ ($n = 4m$ in this case):

$$\min'_{\mathcal{Z}}(a, b) = \begin{cases} \min(a, b) & |b - a| < \frac{n}{2}, \min(a, b) \equiv_2 1 \\ \min(a, b) + 1 & |b - a| < \frac{n}{2}, \min(a, b) \equiv_2 0 \\ \max(a, b) & |b - a| > \frac{n}{2}, \max(a, b) \equiv_2 1 \\ \max(a, b) + 1 & |b - a| > \frac{n}{2}, \max(a, b) \equiv_2 0 \\ b & |b - a| = \frac{n}{2}, b \equiv_2 1 \\ b + 1 & |b - a| = \frac{n}{2}, b \equiv_2 0 \end{cases}$$

$$\max'_{\mathcal{Z}}(a, b) = \begin{cases} \max(a, b) & |b - a| < \frac{n}{2}, \max(a, b) \equiv_2 1 \\ \max(a, b) - 1 & |b - a| < \frac{n}{2}, \max(a, b) \equiv_2 0 \\ \min(a, b) & |b - a| > \frac{n}{2}, \min(a, b) \equiv_2 1 \\ \min(a, b) - 1 & |b - a| > \frac{n}{2}, \min(a, b) \equiv_2 0 \\ b & |b - a| = \frac{n}{2}, b \equiv_2 1 \\ b + 1 & |b - a| = \frac{n}{2}, b \equiv_2 0 \end{cases}$$

We now get

$$\begin{aligned} s_1 &= \min'_{\mathcal{Z}}(i, i - j + k - 2m - 2 + \psi(i, j, k)) \\ s_2 &= \max'_{\mathcal{Z}}(i, i - j + k - 2m + 2 - \psi(i, j, k)) \\ t_1 &= \min'_{\mathcal{Z}}(l, l - k + j - 2m - 2 + \psi(l, j, k)) \\ t_2 &= \max'_{\mathcal{Z}}(l, l - k + j - 2m + 2 - \psi(l, j, k)) \,. \end{aligned}$$

In contrast to the totally planar cases, a single sector is a possible result when composing semi-planar relations. For discriminating a full circle from a single sector, we need to consider the linearity of the relations given by ψ:

$$\begin{aligned} s'_1 &= \begin{cases} 0 & s_1 = s_2 \wedge \psi(i, j, k) = 0 \\ s_1 & \text{else} \end{cases} \\ s'_2 &= \begin{cases} 4m - 1 & s_1 = s_2 \wedge \psi(i, j, k) = 0 \\ s_1 & \text{else} \end{cases} \end{aligned}$$

and analogously for t'_1 and t'_2.

The resulting $\mathcal{OPRA}_m$ relation is

$$A \; {}_m\angle^j_i \; B \circ B \; {}_m\angle^l_k \; C = \bigvee_{a=s'_1}^{s'_2} \bigvee_{b=t'_1}^{t'_2} A \; {}_m\angle^b_a \; C \quad . \qquad (9)$$

Composition with "Same" Relations
Compositions of cases where either $\mathbf{p}_A = \mathbf{p}_B$ or $\mathbf{p}_B = \mathbf{p}_C$, is rather simple, because it can be seen as an addition of intervals, or, if $i \equiv_2 0 \wedge k \equiv_2 0$, vectors.

$${}_m\angle i \circ {}_m\angle^l_k = \begin{cases} \bigvee_{a=s_1}^{s_2} {}_m\angle^l_a & i \equiv_2 0 \wedge k \equiv_2 0 \\ i + k & \text{else} \end{cases} \quad , \qquad (10)$$

$$\begin{aligned} s_1 &= i + k - 1 + \psi(i, k, 1) \\ s_2 &= i + k + 1 - \psi(i, k, 1) \end{aligned}$$

ψ again denotes the linearity term given in (8). The third argument is 1 because we only need two arguments here.

The composition ${}_m\angle^j_i \circ {}_m\angle k$ works analogously. Composition of two "same" relations is trivial.

3.5 Integration of Relations with Different Granularity

Sometimes it is reasonable to perceive or act using different degrees of accuracy depending on context or time constraints. Therefore we have relations at different levels of granularity, i.e. varying m. It is not reasonable to represent such information at a very precise level, because a large disjunction with many literals would emerge. We call the chosen m a *context dependent granularity*. Inconsistencies arising due to imprecise or faulty perception or movement can be solved by adding even more uncertainty to draw a reasonable conclusion.

Given two relations with granularity m_1 and m_2, it is no problem to integrate relations with $m_1 = n * m_2$ with $n \in \mathbb{N} > 0$ and $m_1 > m_2$. If the values are not a multiple of each other, naive and fast methods for integrating the knowledge are e.g. the least common multiple (LCM) or the greatest common divisor (GCD). Information loss is minimal with the LCM, but again a large disjunction might be generated. In contrast, combining the relations with the GCD of m_1 and m_2 results in a greater loss of information, but the result consists of fewer relations compared with the LCM approach. Currently, we choose a method where the relations are combined according to their algebraic semantics and a suitable granularity is chosen depending on the result.

4 Qualitative Spatial Reasoning in Robotics

We will now give a detailed example on how to integrate local knowledge into survey knowledge with the presented TPCC calculus. Afterwards we will show how the given problem can be solved with $\mathcal{OPRA}_m$ as well. The example we use here has already been introduced in [Dylla and Moratz, 2004].

The basis of the example is a robot system able to perceive colored cubes. The system only measures the direction towards perceived objects. It cannot measure their distance.

Furthermore the system is able to perform discrete motion steps. The task is to "move to the red object behind the blue cube". The initial situation is shown in figure 11(a). For better differentiation we visualize the two ambiguous red objects $B1$ and $B2$ as circles. First we will give a short recap of TPCC [Moratz *et al.*, 2003]. Then we show how to solve the given task with TPCC, followed by a solution with $\mathcal{OPRA}_m$.

4.1 The Ternary Point Configuration Calculus (TPCC)

TPCC [Moratz *et al.*, 2003] deals with point-like objects in the 2D-plane. It is an application oriented variant of the Double Cross calculus [Freksa, 1992], which allows for finer distinctions of positional information than most calculi for constraint based reasoning presented before. The partition of the calculus is shown in figure 10.

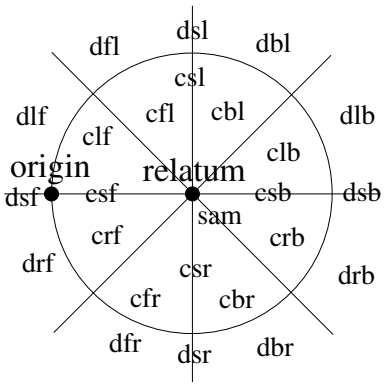


Figure 10: The reference system used by TPCC

The letters f, b, l, r, s, d, c stand for front, back, left, right, straight, distant, and close, respectively. The terms front, back, etc. are given for mnemonic purposes only. The use of TPCC relations in natural language applications is shown in an article by Moratz *et al.* [2002]. In this application TPCC relations are used for natural human robot interaction. The configuration in which the referent is at the same position as the relatum is called *sam* (for "same location"). The two special configurations in which origin and relatum have the same location (*dou*, *tri*) are also base relations of this calculus.

For a formal and precise definition of the relations the corresponding geometric configurations on the basis of a Cartesian coordinate system represented by $\mathbb{R}^2$ were described. For example, the special cases for the three points $A = (x_A, y_A)$, $B = (x_B, y_B)$ and $C = (x_C, y_C)$ are defined as follows:

$$\begin{aligned} A, B \text{ dou } C &:= x_A = x_B \wedge y_A = y_B \wedge \\ &\quad (x_C \neq x_A \vee y_C \neq y_A) \\ A, B \text{ tri } C &:= x_A = x_B = x_C \wedge y_A = y_B = y_C \end{aligned}$$

For the cases with $A \neq B$ a relative radius $r_{A,B,C}$ and a relative angle $\phi_{A,B,C}$ must be defined[2]:

$$\begin{aligned} r_{A,B,C} &:= \frac{\sqrt{(x_C - x_B)^2 + (y_C - y_B)^2}}{\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}} \\ \phi_{A,B,C} &:= \tan^{-1}\frac{y_C - y_B}{x_C - x_B} - \tan^{-1}\frac{y_B - y_A}{x_B - x_A} \end{aligned}$$

Then we have spatial relations as the examples shown below. All relations are named in figure 10 except the special cases *dou* and *tri*. For a complete list of the definitions we refer to [Moratz *et al.*, 2003].

$$\begin{aligned} A, B \text{ sam } C &:= r_{A,B,C} = 0 \\ A, B \text{ clb } C &:= 0 < r_{A,B,C} < 1 \wedge 0 < \phi_{A,B,C} \leq \frac{1}{4}\pi \\ A, B \text{ dlb } C &:= 1 \leq r_{A,B,C} \wedge 0 < \phi_{A,B,C} \leq \frac{1}{4}\pi \\ A, B \text{ cfl } C &:= 0 < r_{A,B,C} < 1 \wedge \frac{1}{2}\,\pi < \phi_{A,B,C} < \frac{3}{4}\,\pi \\ A, B \text{ dsr } C &:= 1 \leq r_{A,B,C} \wedge \phi_{A,B,C} = \frac{3}{2}\,\pi \end{aligned}$$

TPCC is not closed under transformations (intersection, complement and converse), i.e. a transformation might generate a proper subset of base relations. It is as well not closed under strong composition ($\circ$):

$$\forall A, B, D : A, B(r_1 \circ r_2)D \leftrightarrow \exists C : A, B(r1)C \wedge B, C(r_2)D$$

Therefore 4-consistency cannot be enforced directly when inferring with TPCC. Instead a weak composition ($\circledast$) was defined:

$$\forall A, B, D : A, B(r_1 \circledast r_2)D \leftarrow \exists C : A, B(r1)C \wedge B, C(r_2)D$$

The composition table for the weak case was already presented in [Moratz *et al.*, 2003]. The weak operations are still sufficient to solve a task as shown in our example in the next subsection.

4.2 A Solution with TPCC

With the relations defined in TPCC the task "move to the red cube behind the blue cube" can be described such that one of the relations clb, csb or crb must hold for $(C, R1, B1)$ or $(C, R1, B2)$. We will refer to the disjunction of the three relations as $c?b$. We visualize the initial situation in figure 11(a). Figure 11(b) integrates the initially perceived constraints about what is known about $B1$ and $B2$. To deduce the desired knowledge the agent has to move. How to choose the most reasonable action for a maximum of information gain goes beyond the scope of this paper. Therefore we apply the heuristic: "Move straight forward until the first object is passed and get new perceptions there".

We will use a simple path-based scheme of constraint propagation, where the two last relations of a path are composed

[2]Here we refer to the arcus tangent function with two arguments mapping all four quadrants (atan2).

and then the reference system is incrementally moved towards the beginning of the path to demonstrate reasoning efficiently.

In the example the robot moves towards a position to the right of the blue cube (fig. 11(c)–11(d)). In figure 11(e) it reaches the desired position ($R2$). Relation 3 just describes the fact that the agent's move to the right of the blue cube relative to the starting point $R1$. The agent's perception gives additional knowledge on $B1$ and $B2$ relative to $(C, R2)$[3]. In order to make a composition we have to transform relation 3 with the **SC** transformation leading to relation 3' (fig. 11(f)). Now 3' can be composed with 5 leading to the fact that $c?b$ is not valid for $(R1, C, B2)$ (fig. 11(g)). Composing 3' and 4 shows $B1$ being somewhere behind the blue cube relative to $(R1, C)$ (fig. 11(h)). Although according to constraint propagation $B1$ might be somewhere left of the reference axis, $B1$ is the only red object having a chance of fulfilling the given constraint ($c?b$).

Solving general constraint satisfaction networks on the basis of Double Cross relations is NP-hard [Scivos and Nebel, 2001]. TPCC has not yet been proven to be NP-complete. Anyway, in the case of many real world problems the desired knowledge can be gained in polynomial time without the need to solve the whole constraint system. The solution can be obtained via a path-based constraint propagation as presented in [Dylla and Moratz, 2004]. All the algorithms given there are in P.

4.3 $\mathcal{OPRA}_m$— Reasoning about Perceptions

At first the agent perceives basic relations between the objects of the environment. Then the agent moves, gets new perceptions, and can combine these perception using qualitative spatial reasoning using the previously defined operations of $\mathcal{OPRA}_m$. We now relate to the example in figure 11. According to the granularity of TPCC we assume $m = 4$. A_B denotes the o-point at position A with orientation towards B. In contrast, ${}_BA$ denotes the inverting, i.e. point A looking away from object B.

The initial task (figure 11(b)) may be expressed as

$$R1_C \; {}_4\angle^0_0 \; C_{R1} \wedge C_{R1} \; {}_4\angle^{\{0-15\}}_{\{7-9\}} \; BX_* \; ?$$

with $X \in \{1, 2\}$ and with $A \; {}_m\angle^{\{k-l\}}_{\{i-j\}} \; B$ denoting the disjunction

$$\bigvee_{a=i}^{j} \bigvee_{b=k}^{l} A \; {}_m\angle^b_a \; B \quad .$$

The $*$ stands for the set of all available points in our setting. We do this, because the orientation of BX is of no interest for the given task.

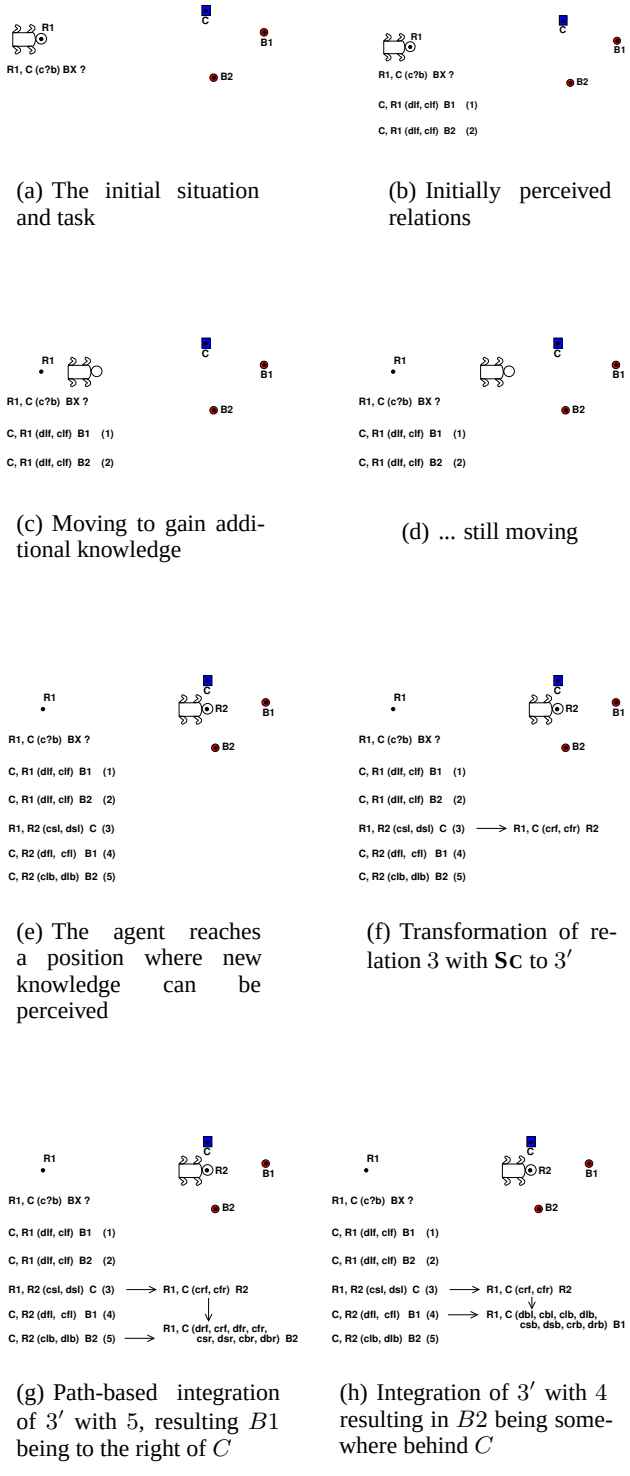
The initial perceptions (figure 11(b)) are:

(1) $R1_{R2} \; {}_4\angle 1 \; R1_C \rightarrow R1_{R2} \; {}_4\angle^0_1 \; C_{R1}$

(2) $R1_{R2} \; {}_4\angle 1 \; R1_{B1} \rightarrow R1_{R2} \; {}_4\angle^0_1 \; B1_{R1}$

(3) $R1_{R2} \; {}_4\angle 15 \; R1_{B2} \rightarrow R1_{R2} \; {}_4\angle^0_{15} \; B2_{R1}$

[3]Perhaps more relations are perceivable, but we concentrate on the relations relevant for this example.



(a) The initial situation and task

(b) Initially perceived relations

(c) Moving to gain additional knowledge

(d) ... still moving

(e) The agent reaches a position where new knowledge can be perceived

(f) Transformation of relation 3 with **SC** to $3'$

(g) Path-based integration of $3'$ with 5, resulting $B1$ being to the right of C

(h) Integration of $3'$ with 4 resulting in $B2$ being somewhere behind C

Figure 11: Solving the task: "Go to the red object (circle) behind the blue cube!" with TPCC

So far $R2$ is just an abstract point in the direction of the robot's current orientation. For additional knowledge the agent moves to the real position $R2$, which is the rectangular point of intersection of a straight move and the first object passed according to our heuristics.

(4) $R1_{R2}\ {}_4\angle_0^8\ {}_{R1}R2 \rightarrow R1_{R2}\ {}_4\angle_0^0\ R2_{R1}$

At $R2$ the Aibo perceives (figure 11(e))

(5) ${}_{R1}R2\ {}_4\angle 4\ R2_C \rightarrow {}_{R1}R2\ {}_4\angle_4^0\ C_{R2}$

(6) ${}_{R1}R2\ {}_4\angle 1\ R2_{B1} \rightarrow {}_{R1}R2\ {}_4\angle_1^0\ B1_{R2}$

(7) ${}_{R1}R2\ {}_4\angle 13\ R2_{B2} \rightarrow {}_{R1}R2\ {}_4\angle_{13}^0\ B2_{R2}$

From (5) follows

(5') $R2_{R1}\ {}_4\angle 12\ R2_C \rightarrow R2_C\ {}_4\angle 4\ R2_{R1}$

Applying the triangle constraint to (1) and (5') we now are able to derive

(8) $C_{R1}\ {}_4\angle 3\ C_{R2}$

Again we must transform (5) to

(5") $C_{R2}\ {}_4\angle_0^4\ {}_{R1}R2$

Now we can compose (5") and the "same" relation (8) resulting in

(9) $C_{R1}\ {}_4\angle_3^4\ {}_{R1}R2$

Deriving the relative position of $B1$ needs the composition of semi-planar relations (9) and (6), and with (7) for $B2$ respectively.

(10) $C_{R1}\ {}_4\angle_{\{3-9\}}^{\{11-15\}}\ B1_{R2}$

(11) $C_{R1}\ {}_4\angle_{\{3-5\}}^{15}\ B2_{R2}$

In (11), compared to our initial task, one can see that $B2$ is definitely being positioned somewhere to the left of $C1$ regarding the orientation towards our starting position $R1$. Although $B1$ might also be somewhere to the left regarding the same reference system, it is the only red object having the chance to fulfill the initial constraint.

5 Conclusion

We presented a calculus for representing and reasoning about qualitative relative orientation information. Oriented points serve as the basic entities since they are the simplest spatial entities that have an intrinsic orientation. We identified systems of atomic relations on different granularity levels between o-points and identified a scheme for computing the calculi's operation tables based on their geometric semantics. It turns out that our calculus is a relation algebra in the sense of Tarski. Therefore fast standard constraint-based reasoning methods can be applied under real time conditions. The granularity of the calculus allows to suppress irrelevant feature changes in dynamically changing environments.

Potential applications of the calculus are demonstrated by a robotics scenario. In the scenario, linguistic commands and coarse perceived configuration information have to be integrated by constraint propagation to get survey knowledge. The accuracy of the calculus permits robotics applications, in particular in cognitively driven scenarios featuring linguistic communication and approximate visual perception.

Acknowledgment

The authors would like to thank Marco Ragni, Jochen Renz, Diedrich Wolter, Thomas Röfer, and Christian Freksa for interesting and helpful discussions related to the topic of the paper. And we would like to thank Nils Breden for implementing parts of the Aibo demonstration system. The work was supported by the DFG Transregional Collaborative Research Center SFB/TR 8 "Spatial Cognition".

References

[Allen, 1983] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, pages 832–843, 1983.

[Clementini *et al.*, 1997] Eliseo Clementini, Paolino Di Felice, and Daniel Hernandez. Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356, 1997.

[Cohn and Hazarika, 2001] Anthony G. Cohn and Shyamanta M. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.

[Cohn, 1997] Anthony G. Cohn. Qualitative spatial representation and reasoning techniques. In Gerhard Brewka, Christopher Habel, and Bernhard Nebel, editors, *KI-97: Advances in Artificial Intelligence, 21st Annual German Conference on Artificial Intelligence, Proceedings*, volume 1303 of *Lecture Notes in Computer Science*, pages 1–30, Freiburg, Germany, September 1997. Springer.

[Dylla and Moratz, 2004] Frank Dylla and Reinhard Moratz. Empirical complexity issues of practical qualitative spatial reasoning about relative position. In *Workshop on Spatial and Temporal Reasoning at ECAI 2004*, 2004.

[Dylla and Moratz, 2005] Frank Dylla and Reinhard Moratz. Exploiting qualitative spatial neighborhoods in the situation calculus. In *Spatial Cognition 2004 (LNAI 3343)*, pages 304–322, 2005.

[Freksa, 1992] Christian Freksa. Using orientation information for qualitative spatial reasoning. In A. U. Frank, I. Campari, and U. Formentini, editors, *Theories and methods of spatio-temporal reasoning in geographic space*, pages 162–178. Springer, Berlin, 1992.

[Isli and Cohn, 1998] Amar Isli and Anthony G. Cohn. An algebra for cyclic ordering of 2d orientations. In *AAAI/IAAI*, pages 643–649, 1998.

[Isli and Cohn, 2000] Amar Isli and Anthony G. Cohn. A new approach to cyclic ordering of 2d orientations using ternary relation algebras. *Artificial Intelligence*, 122(1-2):137–187, 2000.

[Ladkin and Maddux, 1994] P. Ladkin and R. Maddux. On binary constraint problems. *Journal of the Association for Computing Machinery*, 41(3):435–469, 1994.

[Mackworth, 1977] A. K Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Moratz *et al.*, 2000] Reinhard Moratz, Jochen Renz, and Diedrich Wolter. Qualitative spatial reasoning about line segments. In *Proceedings of ECAI 2000*, pages 234–238, 2000.

[Moratz *et al.*, 2002] R. Moratz, T. Tenbrink, J. Bateman, and K. Fischer. Spatial knowledge representation for human-robot interaction. In Christian Freksa, W. Brauer, C. Habel, and K. F. Wender, editors, *Spatial Cognition III*. Springer, Berlin, Heidelberg, 2002.

[Moratz *et al.*, 2003] Reinhard Moratz, Bernhard Nebel, and Christan Freksa. Qualitative spatial reasoning about relative position: The tradeoff between strong formal properties and successful reasoning about route graphs. In Christian Freksa, Wilfried Brauer, Christopher Habel, and Karl Friedrich Wender, editors, *Spatial Cognition III*, volume 2685 of *Lecture Notes in Artificial Intelligence*, pages 385–400. Springer, Berlin, Heidelberg, 2003.

[Musto *et al.*, 1999] A. Musto, K. Stein, A. Eisenkolb, and T. Röfer. Qualitative and quantitative representations of locomotion and their application in robot navigation. In *Proceedings IJCAI-99*, pages 1067–1072, 1999.

[Randell and Cohn, 1989] D. A. Randell and A. G. Cohn. Modelling topological and metrical properties of physical processes. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 357–368. Morgan Kaufmann, 1989.

[Randell *et al.*, 1992] David A. Randell, Zhan Cui, and Anthony Cohn. A spatial logic based on regions and connection. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 165–176. Morgan Kaufmann, San Mateo, California, 1992.

[Renz and Mitra, 2004] Jochen Renz and Debasis Mitra. Qualitative direction calculi with arbitrary granularity. In *Proceedings PRICAI 2004 (LNAI 3157)*, pages 65–74, Auckland, New Zealand, September 2004. Springer.

[Renz and Nebel, 1998] Jochen Renz and Bernhard Nebel. Efficient methods for qualitative spatial reasoning. In *Proceedings ECAI-98*, pages 562–566, Brighton, 1998.

[Schlieder, 1995] Christoph Schlieder. Reasoning about ordering. In A. U. Frank and W. Kuhn, editors, *Spatial Information Theory – A Theoretical Basis for GIS (COSIT'95)/(LNCS 988)*, pages 341–349. Springer, Berlin, Heidelberg, 1995.

[Scivos and Nebel, 2001] A. Scivos and B. Nebel. Double-crossing: Decidability and computational complexity of a qualitative calculus for navigation. In *COSIT 2001*, Morro Bay, California, 2001. Springer.

[Wagner *et al.*, 2003] Thomas Wagner, Christoph Schlieder, and Ubbo Visser. An extended panorama: Efficient qualitative spatial knowledge representation for highly dynamic environments. In *Proceedings of the IJCAI-03 Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modelling, planning, learning, and communicating*, pages 109–116, Acapulco, Mexico, 2003.

[Zimmermann and Freksa, 1996] K. Zimmermann and C. Freksa. Qualitative spatial reasoning using orientation, distance, and path knowledge. *Applied Intelligence*, 6:49–58, 1996.

Negative Information and Proprioception in Monte Carlo Self-Localization for a 4-Legged Robots

Jan Hoffmann, Michael Spranger, Daniel Göhring, and Matthias Jüngel
Humboldt-Universität zu Berlin, Institut für Informatik,
LFG Künstliche Intelligenz, Unter den Linden 6,
10099 Berlin, Germany, http://www.aiboteamhumboldt.com

Abstract

This work explores how extended modeling of sensors and robot motion can be used to improve Markov localization by monitoring deviations of actual measurements from expected sensor readings. By comparing target and actual motions of robot joints, *proprioception* is achieved yielding a quality measure for the current odometry. a quality measure for odometry that helps differentiate periods of unhindered motion from periods where robot motion was impaired for whatever reason. By *negative information* we denote the absence of an expected sensor reading. Negative information is seldom used in localization because it yields less information than positive information (i.e. sensing a landmark) and a sensor often fails to detect a landmark, even if it falls within its sensing range. We address these difficulties by carefully modeling the sensor to avoid false negatives. In real world experiments, we are able to demonstrate that a robot is able to localize in positions where without the use of negative information it could not.

1 Introduction

The estimation of position and orientation of a mobile robot is a cruical task in mobile robotics. One of the most successfully applied approaches is called *Monte-Carlo-Localization*. This method is used in numerous robot navigation applications, such as office navigation [3], museum tour guides [22], RoboCup [14] [9], as well as outdoor or less structured environments [16]. We propose 2 extensions affecting the sensor model as well as the motion model.

1. We show how *negative information* can be incorporated into Monte Carlo localization. The sensor model is extended by modeling the probability of *non-detection events*.
2. The motion model is improved by modeling of proprioceptive information. The resulting model is incorporated into the action update of the particle filter.

The presented adjustments and changes improve the general ability to localize and also allow the robot to localize in areas where it was previously unable to do so. They enable the robot to quickly recover its belief after collision events and to adjust quickly to large displacements (kidnapped robot).

Negative information denotes the ascertained absence of expected sensor readings. This is incorporated into the current belief much like an additional sensor. Proprioception is based on the comparison of actual motion to intended motion. This information is used to enhance the influence of action commands onto the belief.

This work is motivated by the desire to improve the localization of robots that compete in the *RoboCup Sony Four Legged League*. In this league, two teams of four robots compete on a field of green carpet sized 6 m x 4 m. There are two colored goals and white field lines which define the dimensions of the field. There is also a center line, a center circle and penalty areas near each goal. To help the robots localize there are four cylindrical landmarks at the side of the field. These beacons have a simple two color code that uniquely identifies them. The Aibo robot itself has a camera with a field of view of 55° and a resolution of only 208×160 pixels YUV. It is built into the robot's head which has 3 degrees of freedom. The robot's legs have 3 degrees of freedom each. Due to their small size and low power requirements the robots have rather limited computational power (576 MHz processor). This somewhat limits the sensory capabilities of the Aibos compared to robots that are equipped with laser range finders, sonars and a possibly high end notebook and requires for efficient algorithms and attention control. The nature of the soccer games in *RoboCup Sony Four Legged League* makes the localization task even more challenging. The robots have little evidence whether desired movements were successful or not. Odometry data is of poor quality as the robots often slip on the ground or run into each other. Furthermore, the robots are required to track the ball which makes localization even more difficult as landmarks are only seen infrequently and may be occluded by other robots. In the following sections we will present ways to address these challenges.

2 Monte Carlo Localization

The Monte Carlo Localization method is a probabilistic method, utilizing Bayes law and the Markov assumption. The robot maintains a set of samples, called particles. The particles approximate the belief of the robot's position, a prob-

ability distribution over the possible positions of the robot. The current belief of the robot's position is modeled as particle density, allowing for multi-modal probability distributions and beliefs. Each particle represents a hypothetical position of the robot. The belief $Bel(s_t)$, the localization estimate at time t, to be at position s_t is determined by *all* previous robot actions u_t and observations z_t. Using Bayes law and the Markov assumption, $Bel(s_t)$ can be written as a function that only depends on the previous belief $Bel(s_{t-1})$, the last robot action u_{t-1}, and the current observation z_t:

$$Bel^-(s_t) \longleftarrow \int \underbrace{p(s_t|s_{t-1},u_{t-1})}_{\text{motion model}} Bel(s_{t-1})ds_{t-1} \quad (1)$$

$$Bel(s_t) \longleftarrow \eta \underbrace{p(z_t|s_t)}_{\text{sensor model}} Bel^-(s_t) \quad (2)$$

with normalizing constant η. Equation 1 shows the *a priori* belief $Bel^-(s_t)$ which takes into account the previous belief and propagates it using the motion model of the robot. It is the belief prior to the measurement. The measurement is then incorporated into the belief as described in (2) using the sensor model ('sensor updating'). In Markov localization, given an initial belief $Bel(s_0)$ at $t = t_0$, the robot updates its belief using odometry and then incorporates new sensor information. Each time new information arrives the robot updates its particle distribution using the previous motion command, the resulting distribution is updated using the gathered sensor information. This 2 step operation requires 2 models. The *motion model* $p(s_t|s_{t-1}, u_{t-1})$ tries to model the effect of motion commands on the hypothetical positions. The *sensor model* incorporates environment and sensor information regarding this environment into the current belief. The particle filter employed for our work is based on the method described in [19]. Here particles consist of a robot pose and a probability $(x, y, \theta), p$. The robot pose (x, y, θ) represents the position and orientation of the robot (x,y coordinates on the field in mm and orientation in radians). The likelihood p is a measure of the plausibility of the hypothesis being at the specified robot pose. The approach first moves all particles according to the motion model of the action chosen. Afterwards the probabilities of the particles are adjusted using the sensory input and the sensor model. In a third step, called *resampling* particles are moved, deleted from the particle set or injected from observation, based on their probability.

The RoboCup uses a color coded environment. The distance and bearing to landmarks and the goals are used for sensor update. Other features of the domain are field lines which are also used by some approaches [19]. Goals and landmarks are identified by the camera located in the robot's head. The color pattern of the features is used to identify landmarks. The sensory input of the leg and head joints is used to determine gaze direction, field of view, as well as the direction of identified features. The motion model is usually determined before the game by measuring the effect of motion commands on the actual displacement of the robot (see next section).

3 Proprioceptive Motion Modeling

Many research efforts in mobile robotics aim at enabling the robot to safely and robustly navigate and to move about both

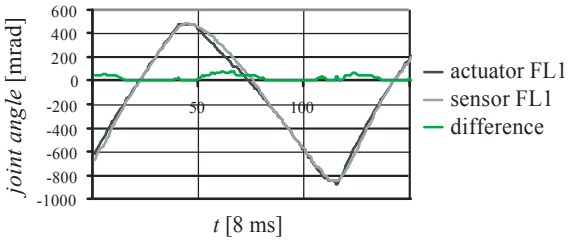


Figure 1: Sensor and actuator data (shoulder joint FL1) for a freely walking robot. The corresponding difference function shows discrepancies between actuator and sensor data, caused by walking motions (peaks in the curve).

known and unknown environments (e.g. the rescue scenarios in the RoboCup Rescue League, planetary surfaces [24]). While wheeled robots are widely used in environments where the robot can move on flat, even surfaces (such as office environments or environments that are accessible to wheelchairs [13]), legged robots are generally believed to be able to deal with a wider range of environments and surfaces. There are many designs of legged robots varying in the number of legs used, ranging from insectoid or arachnoid with 6, 8 or more legs (e.g. [1]), 4-legged such as the Sony Aibo [5], humanoid: 2-legged (e.g. [17]).

Obstacle avoidance is often realized using a dedicated (360°) range sensor [23]. Utilizing vision rather than a dedicated sensor is generally a much harder task since a degree of image understanding is necessary. For the special case of color coded environments, straight forward solutions exist that make use of the knowledge about the robot's environment (such as the color of the surface or the color of obstacles [15], see also previous section). If, however, obstacle avoidance fails, robots are often unable to detect collisions since many designs, like the robot used in this work, lack touch sensors or bumpers. Such robots run into walls and continue to do so since they have no way of telling that they are in a fatal situation. Apart from the current action failing (e.g. the target position not being reached), collisions and subsequently being stuck have severe impact on the robot's localization. This is due to the motion update step in Bayesian filtering where the current motion of the robot is incorporated into its belief (cf. 1). This updating is usually limited to incorporating the robot's own motion which is commonly referred to as odometry. While calculation of odometry is straightforward in a wheeled robot (counting turns of the wheels), the task is much more complex for a legged robot. Forward kinematic can be used to a certain extent [18], but this requires well defined gait patterns. Since gait optimization is often done using genetic optimization, patterns tend to be highly complex and a physical simulation of the robot would be necessary to adequately predict its motion. Such gaits require calibration for them to be used in actual robotic applications [2]. However well the odometry is calibrated, robot locomotion remains a stochastic process and is never quite reproducible. In the RoboCup domain, there is an additional source of errors: other robots competing for the ball. Robots often push each other or in-

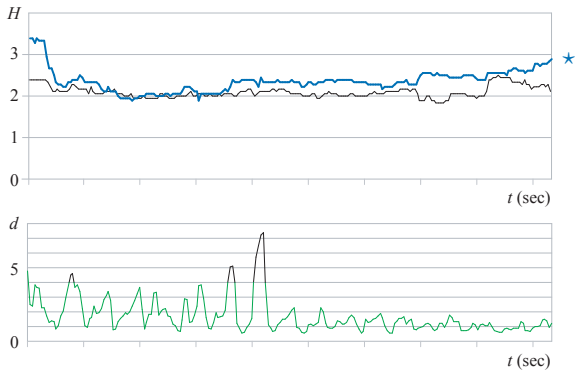


Figure 2: Bottom: The *collision sensor* - values greater than 4 are interpreted as a collision. Top: The entropy of the belief (represented by the sample set) with and without (thin line) improving the motion model. When the enhanced model is used, the entropy increases during collisions, because noise is added to the distribution.

terlock their legs causing motions to have erratic outcome. The following approach is based on work dealing with collisions detection for a Sony Aibo using the walking engine and software framework described in [7]. The approach uses the servo motor's direction sensors for the task of estimating the quality of the odometry data gathered by the walking engine. In analogy to biology we call this *proprioception* because internal sensors are used to determine the state of the robot's body.

3.1 Motion Model

The *motion model* consists of consecutive acquired odometry data incorporated into the belief, as well as a random error Δ_{error}, which is related to the distance traveled and the angle rotated. Every particle is updated using the odometry offset accumulated since the last update.

$$pose_{\text{new}} = pose_{\text{old}} + \Delta_{\text{odometry}} + \Delta_{\text{error}} \tag{3}$$

Where Δ_{error} is defined as

$$\Delta_{\text{error}} = \begin{pmatrix} 0.1d \times \text{random}(-1 \dots 1) \\ 0.02d \times \text{random}(-1 \dots 1) \\ (0.002d + 0.2\alpha) \times \text{random}(-1 \dots 1) \end{pmatrix} \tag{4}$$

3.2 Collision Detection

The Aibo is not equipped with sensors to directly perceive the contact with obstacles. We have shown ways of detecting collisions using the sensor readings from the servo motors of the robot's legs in [7]. The comparison of motor commands and actual movement (as sensed by the servo's position sensor) can be used to detect collisions (see fig.1). This comparison has to compensate for the phase shift between the two signals and has to cope with arbitrary movements and accelerations produced by the behavioral layers of the robot. The method provides a *virtual collision sensor* that can be used to improve the motion model.

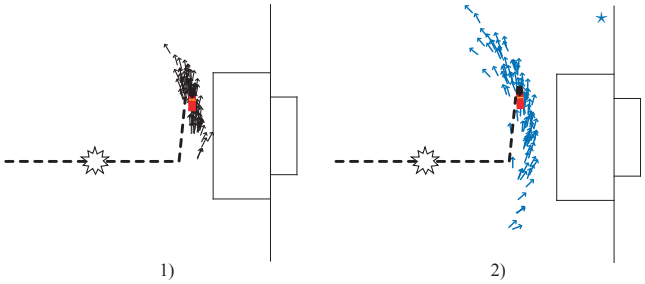


Figure 3: Belief distribution without (1) and with (2) odometry quality used after a collision (marked by the star on the robot's path).

3.3 Extended Motion Model

The extended motion model accounts for the supplementary information provided by the collision detection module, by changing Δ_{error} as well as affecting the accumulated odometry update data in a random way. The binary decision of the collision sensor has a static impact on the motion noise. This means that Δ_{error} is no longer dependent on the distance traveled and the angle rotated, but rather is a uniform noise, within an interval expected to be a possible outcome of collisions. But also odometry data can not be fully relied upon, which is accounted for by randomly updating particles through the gathered odometry information, with the assumption that the robot most probably ends up somewhere between the requested destination and the starting point. The noise tries to account for the severe and unforeseeable impact of the collision. If collisions *are detected*, every particle is updated by:

$$pose_{\text{new}} = pose_{\text{old}} + \text{random}(0...1) \cdot \Delta_{\text{odometry}} + \Delta_{\text{error}}$$

Where Δ_{error} is

$$\Delta_{\text{error}} = \begin{pmatrix} 40 \times \text{random}(-1 \dots 1) \\ 40 \times \text{random}(-1 \dots 1) \\ 0.5 \times \text{random}(-1 \dots 1) \end{pmatrix} \tag{5}$$

Otherwise, when no collision was detected, the motion model is not extended and the update is performed as usual(3). The effect of the changes can be seen in fig.2 and 3.

Entropy We use the expected entropy H as an information theoretical quality measure of the position estimate $Bel(s_t)$ [4]:

$$H_p(s_t) = -\sum_{s_t} Bel(s_t) \log(Bel(s_t)) \tag{6}$$

The sum runs over all possible states. The entropy of the particle distribution becomes zero if the robot is perfectly localized in one position. Maximal values of H mean that $Bel(s_t)$ is uniformly distributed.

Fig. 3 illustrates the effect of the described motion modeling on the particle distribution. A robot is walking from the center circle in the direction of the goal when a collision occurs. It then continous towards the goal and turns left before reaching the penalty area. When the collision is modeled, the uncertainty in the belief is clearly visible and can be used to trigger appropriate robot behavior.

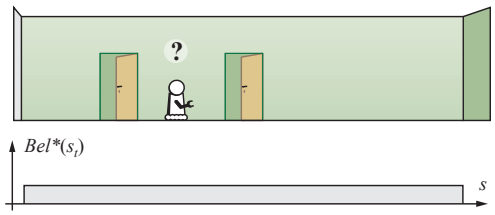


Figure 4a: ($t = t_0$) Illustration of a robot localizing in an office hallway. The robot has a sensor to detect doors. At the beginning, the robot does not know its position in the hallway (uniform belief distribution $Bel^{\star}(s_t)$). At this time, no sensing of the world takes place.

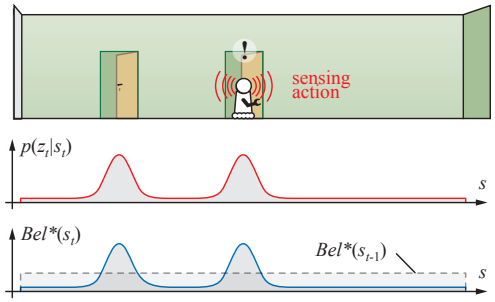


Figure 4b: ($t = t_1$) The robot has moved down the hallway and now senses a door $p(z_t|s_t)$ which results in the shown belief $Bel^{\star}(s_t)$. It has two peaks since the robot could be standing in front of either door. The previous distribution is illustrated by the dashed line.

4 Exploiting Negative Information

The classic example of negative information was described in the Sherlock Holmes case "Silver Blaze." In this case, a house has been broken into. Under such circumstances, one would expect the watch-dog to bark. The curious incident of the non-barking of the dog in the nighttime provides Holmes with the information that the dog must know the burglar, allowing him to solve the case. Applied to mobile robot localization, this means that conclusions can be drawn from expected but actually missing sensor measurements [10]. Markov localization methods, in particular Monte Carlo localization, have proven their power in numerous robot navigation tasks, e.g. in office environments [3], in the museum tour guide Minerva [22], in the highly dynamic RoboCup environment [14], and outdoor applications in less structured environments [16]; an evaluation of the various algorithmic approaches is given in [6].

Our work is focussed on localization based on landmarks. Whenever a robot senses a landmark, the localization estimate is updated using the sensor model. This sensor model is acquired before the actual run. It describes the probability of the measurement z given a state s (position, orientation, etc.) of the robot. Sensor updates only occur when landmarks are detected. If no landmark is detected, the state estimation is updated using (only) the motion model of the robot.

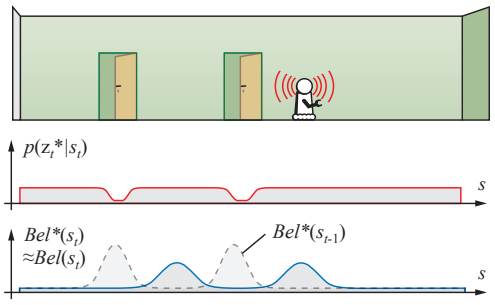


Figure 5a: ($t = t_3$) The robot moves on. There are no doors nearby so the "door sensor" does not sense a door. The sensor update distribution is shown in $p(z_t^{\star}|s_t)$. This negative information is of negligible use at this position: it does not help differentiate between the peaks.

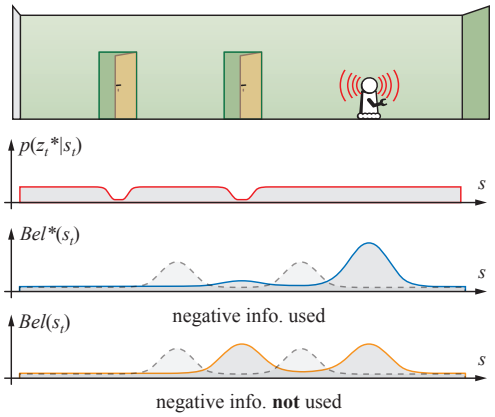


Figure 5b: ($t = t_4$) The robot moves on and the door sensor still does not sense a door. $Bel^{\star}(s_t)$ shows the belief if negative information is taken into account, whereas $Bel(s_t)$ shows the belief without using negative information to better illustrate the case. As can be seen from the diagram, using negative information allows the robot to rule out the left peak.

Example. Consider a robot driving down a corridor as shown in fig. 4a-5b. The robot has a sensor to detect doors when it is standing in front of one. Let us assume further that the robot is moving to the right but is oblivious of its starting position. As it starts to move to the right it passes and senses a door. Given this information, it could be standing in front of either of the doors (states s_{left} and s_{right}). As it moves on, it does not pass another door for some time. At time $t = t_3$, if s_{left} had been the true position, the robot would have had passed another door by now. Using the negative information of not perceiving a door, the belief based on s_{left} can be ruled out. As Thrun, Bugard, and Fox put it quite graphically, "not seeing the Eiffel Tower in Paris implies that it is unlikely that we are right next to it" [21].

We present a localization approach that incorporates such negative information. To our knowledge, no explicit study of using negative information in Markov localization has

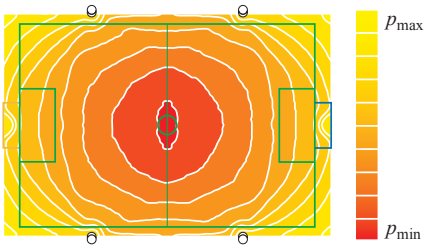


Figure 6: Probability of *not* sensing a landmark for a robot on a RoboCup soccer field. For a robot located around the center of the field, it is hard to miss landmarks.

been published. One difficulty is brought about by the fact that, generally speaking, sensing a landmark constitutes a greater information gain than not sensing one simply because there are many positions within the robot's environment from where the landmark cannot be perceived. A landmark is, by definition, something that stands out in an environment.

The other difficulty in implementing a system that uses negative information on a real robot is that there are two main reasons for the absence of an expected sensor reading: the target may not be there or the sensor may simply be unable to detect the target (due to occlusions, sensor imperfections, imperfect image processing, etc.). Differentiating the two cases is not a trivial task and requires careful sensor modeling. We address this problem by considering the field of view of the robot and by using obstacle detection to estimate occlusions.

Negative information modeling has been applied to object tracking (see [20] for an introduction and [10] for an overview). The event of not detecting an object is treated as evidence that can be used to update its probability density function [11]. In the RoboCup domain, not seeing the ball on the field can be used to delete Monte Carlo particles in that region as long as occlusions are considered [12]. Negative information is also mentioned in the context of simultaneous localization and mapping (SLAM) where it is used to adjust the confidence in landmark candidates [16].

4.1 The Notion Of Negative Information

Negative information describes the absence of a sensor reading in a situation where a sensor reading is expected given the current position estimate.

To integrate negative information, imagine a binary sensor being added that fires whenever the primary sensor *does not* detect a particular landmark l. Its probability of it firing is given by:

$$p(z^{\star}_{l,t}|s_t) \tag{7}$$

This sensor model can be used to update the robot's belief whenever it fails to detect a landmark, i.e. when negative evidence is acquired. Fig. 6 shows the probability $p(z^{\star}_t|x_t, y_t)$ of not sensing a landmark on a RoboCup field at position (x_t, y_t) summed over all possible robot orientations. This figure also shows that it is most likely for the robot to sense a landmark when it is standing in the middle of the field. The likelihood of not sensing a landmark is highest for positions at the edge of the field as the robot may be facing outwards.

Algorithm 1 Iterative Bayesian updating incorporating negative evidence

1: $Bel^{-}(s_t) \longleftarrow \int p(s_t|s_{t-1}, u_{t-1}) Bel(s_{t-1}) ds_{t-1}$
2: **if** (landmark l detected) **then**
3: $\quad Bel(s_t) \longleftarrow \eta p(z_t|s_t) Bel^{-}(s_t)$
4: **else**
5: $\quad Bel(s_t) \longleftarrow \eta p(z^{\star}_{t,l}|s_t, r_t, o_t) Bel^{-}(s_t)$
6: **end if**

This rather coarse way of incorporating negative information can be refined by taking into account the sensing range r_t of the robot's sensors and possible occlusions o_t of landmarks. The sensing range is the physical volume that the sensor is monitoring. In case of a stationary robot, $r_t = r_0$ is constant, for a mobile robot with a pan-tilt camera it is not. By o_t we denote a means of detecting whether or not occlusions have occurred. In practice, this can be calculated from a map of the environment, directly sensed by a sensor such as a laser range finder, or derived from a model of moving objects in the environment.

Combining the two yields the probability of not sensing an expected landmark l:

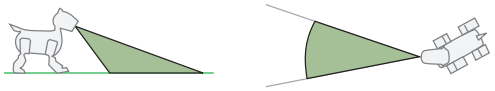
$$p(z^{\star}_{t,l}|s_t, r_t, o_t) \tag{8}$$

Whenever a landmark is not detected, it can be used in the sensor update step of the Iterative Bayesian Updating (see Algorithm 1).

4.2 Sensor Modeling For The Sony Aibo

Field of View

The ERS-7 is a legged robot with a camera mounted in its head. The camera has a horizontal opening angle of 55° and the robot's head has 3 degrees of freedom (neck tilt, head pan, head tilt). We abbreviate gaze direction by $\varphi = (\varphi_{\text{tilt1}}, \varphi_{\text{pan}}, \varphi_{\text{tilt2}})$. The sensing range is calculated by considering the field of view (FOV) of the robot:



Occlusion

In order to account for occlusions, we opted for an approach that has been used successfully for detecting obstacles, referred to as 'visual sonar' [8; 15]: The camera image is scanned in vertical scan lines and unoccupied space in the plane of the field is detected since it can only be of green or white color (field lines). Scanning for these colors tells the robot where obstacles are and where there is free space which in turn can be used to determine if the visibility of the landmark is impaired, i.e. if it is occluded by other robot or some other obstacle. More specifically, if the expected landmark lies in an area where the robot has detected free space, the likelihood of the corresponding pose estimate is decreased. If it lies outside of the detected free space, no inference can be made.

Taking FOV and occlusion into account, the sensor model for not perceiving an expected landmark is given by:

$$p(z_t^\star | s_t, z_{t,\text{obs}}) \tag{9}$$

Where $s_t = (x_t, y_t, \vartheta_t, \varphi_t)$ describes the robot state that consists of the *robot pose* (position x_t, y_t, and orientation ϑ_t) and the current gaze direction φ_t.

4.3 Experimental Results

In the following experiments, unless otherwise stated, only landmarks were used for localization to emphasize the effect of using negative information.

Monte Carlo Localization, Implementation

This work is based on the Monte Carlo localization described in [19] which also serves as a base line implementation. Sensor updating was extended to account for FOV and occlusion as described. This also requires sensor updating to be triggered by new camera images regardless of whether or not there was a percept. Before re-sampling, the weight of an individual particle is calculated as follows: Of all landmarks L, the subset of landmarks L' is detected, the subset $L^\star$ is expected but not detected, and lastly the subset $L^\diamond$ is not detected but was also not expected: $L = L' \cup L^\star \cup L^\diamond$ and $L^\star \cap L' = \emptyset$. The probability of a particle p_i is calculated by multiplying all the likelihoods of all gathered evidences:

$$p_i = \underbrace{\prod_{l \in L'} s_l(\alpha_{\text{measd}}, \alpha_{\text{expd}})}_{\text{detected}} \cdot \underbrace{\prod_{l \in L^\star} s_l^\star(\varphi, \alpha_{\text{expd}})}_{\text{expected and not detected}} \tag{10}$$

The function s_l is an approximation of the sensor model and returns the likelihood of sensing the landmark l at angle α_{measd} for a particle p_i that expects this landmark to be at α_{expd}. Function $s_l^\star$ models the probability of not sensing the expected landmark $l \in L^\star$ given the current sensing range as determined by φ, the robot pose associated with p_i, and the obstacles percept z_{obs}.

Preliminary Experiment

For illustration purposes, we conducted a preliminary experiment in simulation. In this experiment, the robot starts out being well localized and is then displaced to a position where it is not able to get any new sensor information (fig. 7). It is similar to the kidnapped robot problem, but here we emphasize the moment right after the robot is displaced rather than investigating how fast it can recover. The effect of the displacement on the Monte Carlo particle distribution is the following: particles which represent the previous belief become less likely when negative information is taken into account (i.e. the information that the landmark is not detected where it is expected). The distribution diverges towards particles which were less likely prior to the displacement. Particles representing the previous belief are eventually eliminated from the distribution because they are inconsistent with the current (negative) sensor data. Particles which differ from the previous belief just enough to be compatible with the current sensor data are favored; particles remain close to where the robot was last able to localize. This does, in most cases, better represent what has happened to the robot than distributing the particles uniformly over the entire field.

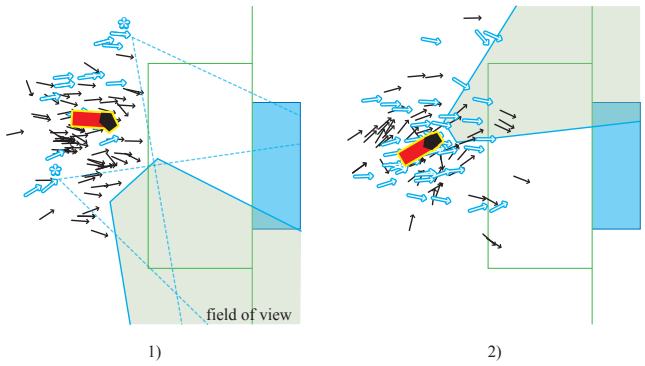


Figure 7: Incorporating negative information. White (outlined) arrows denote particles that receive negative information and that are therefore less likely than others. In (1), the effect of using negative information is shown for a robot that is well localized and frequently sees landmarks. (2) Distribution shortly after the robot has been displaced (kidnapped): particles facing the goal are less likely and will eventually be eliminated from the distribution.

4.4 Localization Experiment

The following experiment is a localization task using the real robot. The robot is placed on the field at the location indicated in fig. 9, facing outwards. The robot performs a scanning motion with its head (pan range $[-45^\text{o}, 45^\text{o}]$) but does not move otherwise. From its position, it can only see one landmark. A panorama composed of actual robot camera images is shown in fig. 8. The *a priori* belief is assumed uniform. This position was chosen because it is a particulary difficult spot for the robot to localize given the limited sensor information. Two quantities can be used when a landmark is seen: its size in the camera image can be used to estimate the distance to the landmark d_l and the relative angle to the landmark (bearing, α_l) can be calculated from its position within the image. In practice we only use the bearing because the distance measurement is error prone. Using just the bearing, only the orientation of the robot can be inferred. Note that this differs from triangulation where distances are used.

In the following paragraphs, the basic localization not using negative information and localization incorporating negative information are compared. We first qualitatively analyze the particle distribution and then show how the entropy of the distribution decreases when negative information is considered.

Particle Distribution

The basic experiment was conducted using 100 particles for Monte Carlo localization. It was repeated on a log file containing camera images, robot joint angles, and odometry data using an increased particle count of 2000 to get a better representation of the probability distribution.

Not using negative information. Without using negative information, the robot is unable to localize (fig. 10). Only the orientation of the particles is adjusted according to the sensor readings. The apparent clustering in the small sample set in fig. 10 is not stable and, even after considerable time,

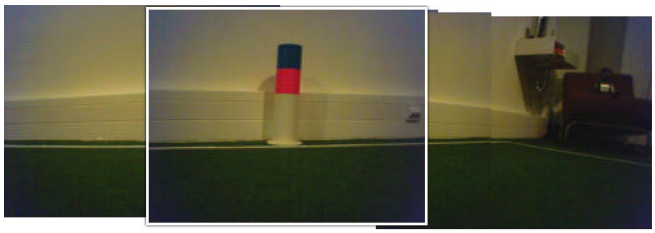


Figure 8: A panorama view generated from actual camera images, single camera image highlighted. The robot can only see *one* landmark.

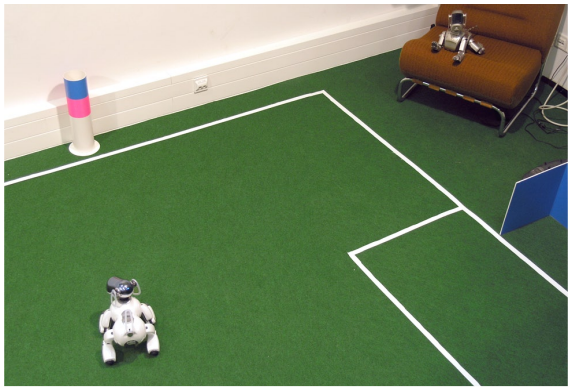


Figure 9: *Experimental setup:* Robot is standing at the position shown in the photo. It performs a scanning motion with its camera.

the particles do not converge. The distribution for the larger sample set is uniform (w.r.t. position).

Note that the distribution is not circular because the distance to the landmark was not used. Instead, only the bearing to the landmark was used. This results in a radial distribution resembling magnetic field lines.

Incorporating negative information. The negative information gained in this experiment is not seeing but one landmark within the pan range (pardon the double negation). Incorporating this information, the robot is able to localize quickly. On average, the robot is reasonably well localized after about 10 secs with a pose error of less than $\Delta p = (25\text{ cm}, 25\text{ cm}, 20^{\circ})$.

Entropy

Entropy is considered for the localization task as defined in equation 6. Fig. 12 shows the progression of the distribution's entropy over time for the above localization experiment calculated from the 100 particle distribution.

Not using negative information. The run starts with a uniform particle distribution which equals to maximum entropy. When the landmark comes into view, a decrease in entropy is observed. This information gain is due to the robot being able to now infer its relative orientation w.r.t. the landmark. Since there are no constraints on the robot's position, the entropy remains at a relatively high level. This is easily seen by separately calculating the entropy of the angle and position distributions. Note that even though there is a drop in entropy, the pose estimate itself is still highly uncertain.

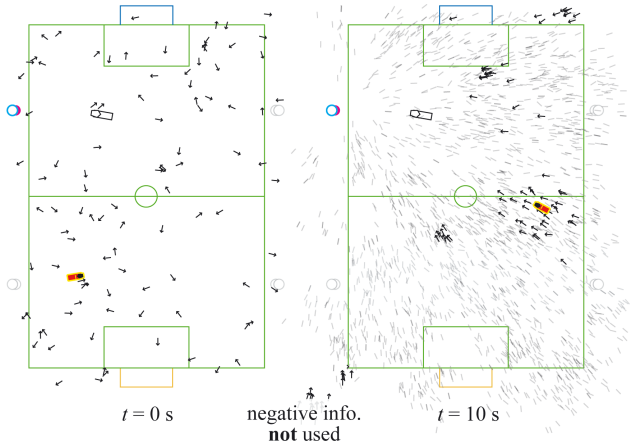


Figure 10: Particle distribution not using negative information, initial uniform distribution and distribution after 10s. Solid arrows indicate Monte Carlo particles (100). The experiment was repeated using 2000 particles (shaded lines) to better represent the actual probability distribution. The actual robot position is indicated by the white symbol, the estimated robot pose by the solid symbol. Not using negative information and only using the bearing to the landmark, the robot is unable to localize. Some clusters of particles form but they do not converge. As one would expect, the position distribution is almost uniform but the relative angle is quite distinct.

Incorporating negative information. When using negative information, the entropy decreases even before the first sensor reading. The information gain is much smaller than that caused by perceiving a landmark but nevertheless noticeable. As soon as there is a percept, the negative information in combination with the knowledge of the robot's orientation results in a quick convergence towards the actual robot pose. This is remarkable since without using negative information, localization was not possible.

Using field lines for localization. The previous experiment was repeated using field lines for localization in addition to landmarks. This enables the robot to localize quickly at the actual robot pose even when using the basic localization (fig. 12, right). Adding negative information, however, greatly increases the rate of convergence and the overall level of entropy is reduced even further. The decrease of entropy when incorporating negative information is not obscured by the usage of lines for localization although field lines offer a much greater information content than negative information.

Kidnapped Robot. The kidnapped robot problem is a commonly used benchmark for the flexibility and robustness of localization algorithms [6]: a localized robot is displaced and the time for it to recover is measured. Our kidnapped robot experiments underlined and confirmed the already stated findings. The robot is able to recover from displacements without using negative information as soon as it successively sees three landmarks. In regions where this is not guaranteed, the case is different. Whereas without using negative information, the robot does not have enough evidence to update its

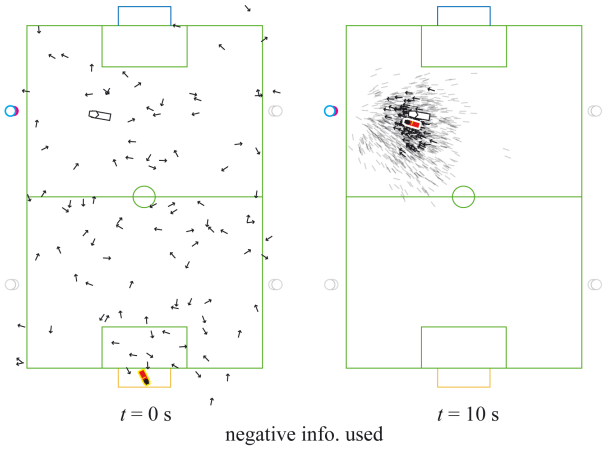


Figure 11: Particle distribution when negative information is incorporated, initial uniform distribution and distribution after 10s. When incorporating negative information, the robot is able to localize quickly.

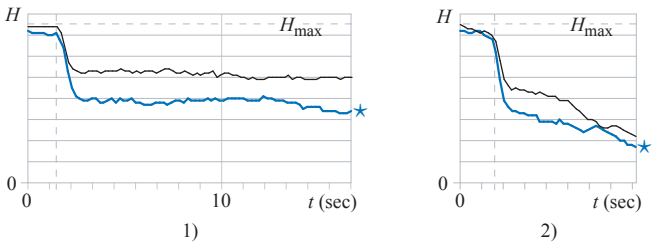


Figure 12: Expected entropy of the belief in the localization task with (⋆) and without (thin line) using negative information. 1) At first the robot does not see the landmark. As soon as the landmark comes into the robot's view (indicated by the dashed vertical line), the entropy drops. Using negative information, the quality of the localization is greatly improved and the entropy continues to decrease over time. 2) Additionally using field lines for localization enables the robot to localize even without negative information. Incorporating negative information, however, yields a higher rate of convergence and the entropy is significantly lowered.

belief, incorporating negative information allows the robot to localize quickly and reliably in such regions.

The ability to localize more quickly using negative information is highly beneficial in real world applications where the robot is trying to actually perform a task rather than to localize perfectly. Such tasks often require the robot to focus its attention on objects other than landmarks and the sensing strategy may keep it from seeing as much of the world as it potentially could. Integrating negative evidence thus allows for more efficient sensing and improves overall robot performance.

5 Conclusion

In this paper we demonstrated how integrating negative information as well as information about collisions into Markov localization can be used to achieve significantly better localization performance for a mobile robot.

An odometry-based motion model is improved using the knowledge about collisions with obstacles yielding a quality measure for the odometry data. This knowledge is obtained by comparing the motor commands and the sensor readings of the leg joints. In the case of a collision, the influence of the odometry on the motion model is reduced and extra noise is added that models the impact of an obstacle.

Incorporating negative information into the sensor model makes localization more stable even in areas where landmarks are rarely visible. Because sensors are more likely to overlook observable landmarks than hallucinate ones that are not visible, extra care has to be taken in designing the sensor model. To avoid false negatives, the model needs to take into account the sensor's sensing range and possible occlusions of landmarks. We have presented how such modeling can be achieved for a Sony Aibo robot in the RoboCup environment. In real robot experiments, we have shown that using negative information, a robot is able to localize in positions where it otherwise would not. The entropy of the distribution is greatly reduced when negative information is incorporated and the rate of convergence towards the estimated position is increased.

The additional information that is being incorporated into the belief makes it more responsive. This improves localization in areas where there are few landmarks visible and, on the other hand, leads to a quick degradation of the belief when collisions occur. The latter is often the case when two robots fight over a ball and one tries to shot the ball; such action often fails because the robot is unaware of being badly localized and then shoots the ball in an undesirable direction. Incorporating collision detection into the belief allows the robot to recognize such situations and act accordingly.

Future work will focus on how negative information can be used for other types of landmarks (e.g. field lines) and other sensors. Performance evaluation will be continued in more complex situations and the possibilities of reducing the number of particles necessary for robust Monte Carlo localization will be investigated. The increased responsiveness of the probability distribution will allow for active vision approaches that take the current belief into account.

Acknowledgments

Program code used was developed by the GermanTeam, a joint effort of the Humboldt University of Berlin, University of Bremen, University of Dortmund, and the Technical University of Darmstadt. Source code is available for download at http://www.germanteam.org. Original implementation of the localization algorithm by Thomas Röfer. Freek Stulp pointed out "Silver Blaze" to us.

References

[1] J. E. Clark, J. G. Cham, S. A. Bailey, E. M. Froehlich, P. K. Nahata, R. J. Full, and M. R. Cutkosky. Biomimetic Design and Fabrication of a Hexapedal Running Robot. In *Intl. Conf. Robotics and Automation (ICRA2001)*, 2001.

[2] U. Düffert and J. Hoffmann. Reliable and precise gait modeling for a quadruped robot. In *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conferences)*. Springer, 2006. to appear.

[3] D. Fox, W. Burgard, F. Dellart, and S. Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proc. of AAAI*, 1999.

[4] D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. In *Robotics and Autonomous Systems*, 1998.

[5] M. Fujita and H. Kitano. Development of an Autonomous Quadruped Robot for Robot Entertainment. *Autonomous Robots*, 5(1):7–18, 1998.

[6] J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[7] J. Hoffmann and D. Göhring. Sensor-Actuator-Comparison as a Basis for Collision Detection for a Quadruped Robot. In D. Nardi, M. Riedmiller, C. Sammut, and J. S.-V. (Eds.), editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 150–159. Springer, 2005.

[8] J. Hoffmann, M. Jüngel, and M. Lötzsch. A vision based system for goal-directed obstacle avoidance. In D. Nardi, M. Riedmiller, C. Sammut, and J. S.-V. (Eds.), editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 418–425. Springer, 2005.

[9] J. Hoffmann, M. Spranger, D. Göhring, and M. Jüngel. Exploiting the unexpected: Negative evidence modeling and proprioceptive motion modeling for improved markov localization. In *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conferences)*. Springer, 2006. to appear.

[10] W. Koch. On negative information in tracking and sensor data fusion. In *Proceedings of the Seventh International Conference on Information Fusion*, pages 91–98, 2004.

[11] W. Koch. Utilizing negative information to track ground vehicles through move-stop-move cycles. In *Proceedings of the SPIE*, volume 5429, pages 273–283, 2004.

[12] C. Kwok and D. Fox. Map-based multiple model tracking of a moving object. In *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer, 2005. to appear.

[13] A. Lankenau, T. Röfer, and B. Krieg-Brückner. Self-Localization in Large-Scale Environments for the Bremen Autonomous Wheelchair. In *Spatial Cognition III*, Lecture Notes in Artificial Intelligence. Springer, 2002.

[14] S. Lenser, J. Bruce, and M. Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 204–211. ACM Press, 2001.

[15] S. Lenser and M. Veloso. Visual sonar: Fast obstacle avoidance using monocular vision. In *Proceedings of IROS'03*, 2003.

[16] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. 2003.

[17] C. L. P. Dario, E. Guglielmelli. Humanoids and personal robots: design and experiments. *Journal of Robotic Systems*, 18(2), 2001.

[18] T. Röfer. Evolutionary Gait-Optimization Using a Fitness Function Based on Proprioception. In D. Nardi, M. Riedmiller, C. Sammut, and J. S.-V. (Eds.), editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 310–322. Springer, 2005.

[19] T. Röfer and M. Jüngel. Vision-based fast and reactive monte-carlo localization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-2003), Taipei, Taiwan*, pages 856–861, 2003.

[20] S. Särkkä, T. Tamminen, A. Vehtari, and J. Lampinen. Probabilistic Methods in Multiple Target Tracking, Research Report B36. Technical report, Laboratory of Computational Engineering Helsinki University of Technology, 2004.

[21] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*, page 231. MIT Press, 2005.

[22] S. Thrun, D. Fox, and W. Burgard. Monte carlo localization with mixture proposal distribution. In *Proc. of the National Conference on Artificial Intelligence*, pages 859–865, 2000.

[23] T. Weigel, A. Kleiner, F. Diesch, M. Dietl, J.-S. Gutmann, B. Nebel, P. Stiegeler, and B. Szerbakowski. Cs freiburg 2001. In *RoboCup 2001 International Symposium*, Lecture Notes in Artificial Intelligence. Springer, 2003.

[24] K. Yoshida, H. Hamano, and T. Watanabe. Slip-Based Traction Control of a Planetary Rover. In *Experimental Robotics VIII*, Advanced Robotics Series. Springer, 2002.

.

A Neural Coach for Teaching Robots Using Diagrams

Bastian Hecht, Mark Simon, Oliver Tenchio, Fabian Wiesel, Alexander Gloye, Raúl Rojas
Freie Universität Berlin
Institut für Informatik
Takustraße 9, 14195 Berlin
Germany
{bhecht,simon,tenchio,wiesel,gloye,rojas}@inf.fu-berlin.de

Abstract

In this paper we show how to train soccer robots using static game situations in diagrams arranged by a human coach. Rather than programming every detail by hand, we let the robots learn from strategic examples sketched by the coach. With our approach, the coach defines new game positions and indicates to the players how to react to them, like in real soccer. We have implemented a management tool to collect and organize all the game positions entered by the coach. The game situation is encoded as a feature vector, which is used to train a neural network. The network learn to generalize and give advice on the best option for a player. The general method is illustrated with the specific case of robots learning to pass. The method can be generalized to other tasks and to several networks encoding different game strategies.

1 Motivation

RoboCup robots are usually programmed by hand. There has been work done to set parameters graphically like [Ydren and Scerri, 1999], but learning techniques for abstract behaviors are widely missing. Different techniques, such as reinforcement learning, have been used for several years in the simulation league [Lauer and Riedmiller, 2004], whereas in the robotic leagues it is more difficult to automatically learn high-level skills. Therefore learning has been mostly used to allow robots to automatically adapt the parameters of low-level skills [Fidelman and Stone, 2004]. It is also clear why: we cannot let real robots play against themselves hundreds of times, so that they learn to behave successfully.

An alternative could be a simulation, but an exact model of the robots is never so exact that a simulation could be used as a complete substitute [Gloye *et al.*, 2004]. On the one hand, it is hard to have an errorless prediction of the driving behavior of real robots. On the other hand, very small changes in hardware can lead to significantly different characteristics, such as more or less ball control when driving.

In this paper we investigate a second option. We want to supply our small-size robots with our own human knowledge about soccer. Until now, we have achieved respectable robot behavior mostly using manual hard coding and tuning the code in long and difficult "training" sessions. We would like to teach the robots in the same way a human coach explains plays to human players: using static diagrams of what constitutes a good and what constitutes a bad move. What we propose is that a human coach draws interesting game situations, for example for passing [Kok *et al.*, 2003], and then assigns them a "good" or "bad" grade. The computer should then learn to generalize from such examples to new and unseen game situations.

Entering enough examples into the system, it is then possible to train a neural network which can achieve the desired generalization. The coach trains the robots with examples, and the robots learn to do the right thing. Moreover, by keeping separate databases of offensive or defensive strategies, it is possible to train several neural networks for different styles of play. We then can integrate an external agent (a coach, as in the simulation league), which can provide advice on the best strategy for the current adversary [Kuhlmann *et al.*, 2005]. The robots can then switch their strategy dynamically according to this advice.

2 Setting up Training Examples of Game Situations

The first step for training the robots with our approach is to set up some examples of possible game situations. For this we can use our simulator for the small-size robots. From now on, let us assume that we want robots to learn how to pass the ball (and receive it). The player with the ball will be called the "passer", and the player receiving the ball will be called the "receiver". Figure 1 shows a scene in which player 0 should pass the ball to player 1, which is waiting for the pass. With our simulator, the user can set up such a game situation by dragging players from each team to the desired position.

In what follows, we focus only on a passer and a single available receiver. Later on, we generalize our techniques to take all other field players into account.

Once an example has been entered by the coach, we save all relevant information in an XML-file. In the experiments described in this paper, we have used static situations, where the speeds of all robots and the ball are zero. However, the same general approach can be applied to dynamic situations. For every stored game situation, we associate with it information that reflects our belief on the correct decision for the

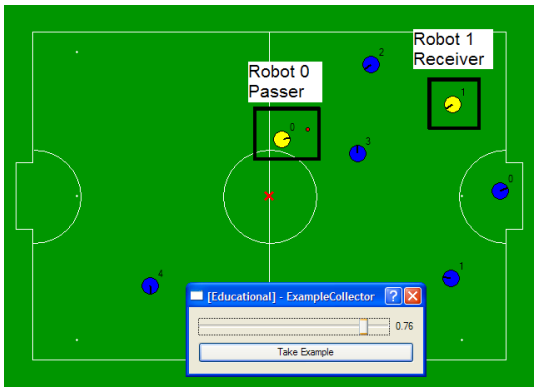


Figure 1: A static game situation entered by the human coach

the potential passer (giving the pass or not). In the case of the passing game, we store a real number $x \in [-1; 1]$. In our system, we apply the convention that when $x \leq 0.0$ dribbling is desirable, and that when $x > 0.0$ passing is desirable.

3 Encoding Relevant Features

In order to generalize from the stored examples, it is crucial to look not at the coordinates of the robots, but to some features of the game situation. If we would just encode the coordinates of the robots on the field, and would give this information to a neural network, the net would have extreme difficulties trying to generalize to new game situations. A barrier of players, for example, is a feature that looks similar in almost all places in the field, although the coordinates are of the robots can be very different.

We need to encode the field using a numerical feature vector. For example, a possible feature is the free space around the ball receiver. This feature is very relevant because it does not pay to give a pass to a player which will be trapped.

By encoding the game situation with a feature vector, we necessarily lose information because we cannot reconstruct all robots' positions from the features, but we obtain a more abstract view of the field. The information given to the net has a better format, since important field aspects are encoded numerically.

Back to our passing example. Conceptually, the features we use are split up into two parts, the features having to do with dribbling (that is, driving with the ball), and the features having to do with ball reception. The features are independent from the training method used.
The features associated with dribbling are:

1. Dribbling freedom

   This parameter describes the space available for dribbling with the ball, before an opponent appears (in the direction of the opponent's goal). This parameter reflects the time that it would take the nearest opponent to interfere with the dribbling path of my robot.

2. Dribbling angle

   This is the angle defined by the position of the passer, the middle of the opponent's goal line, and the nearest corner of the field. A robot positioned at an opponent's corner, for example, has a dribbling angle of zero. A robot in the middle of the field, has a dribbling angle of 90 degrees. A larger dribbling angle means that more space for dribbling is available.

3. Dribbling distance to the goal

   This is the just distance from the passer to the goal line.

The next five features describe the reward obtained from passing. We can visualize these features in the following way: The player on the left has the ball, and ponders whether to give a pass. The player on the right is only a symbol for a receiver at one position. In fact, we displace the pass receiver over a grid of $17x21$ points covering the field, and calculate at each vertex the features. The results are illustrated in the following figures.

4. Space available

   This parameter encodes the distance from the receiver up to the nearest opponent. It is large when the opponents are far away, it is small when an opponent is near. Fig. 2 shows, with a white marker, those portions of the field where there is much space available, and with black the space dominated by opponents.

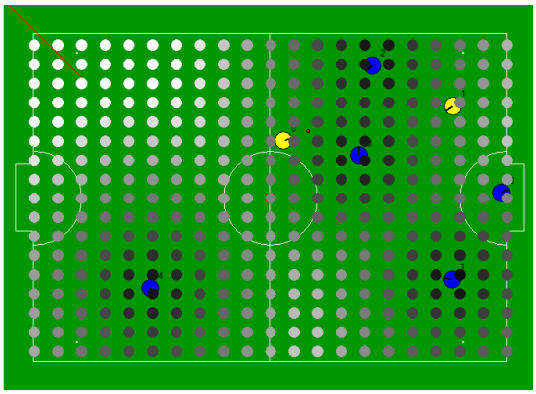


Figure 2: Space available: White areas of the field are relatively opponent-free, dark areas are not

5. Passing angle

   This is the same feature we had above (dribbling angle), but now for the receiver. A high passing angle means that it is good to receive passes, for example in the middle of the field. A low passing angle means that it is not so good to receive a pass, for example, at the corners or near the sides of the field. Fig. 3 shows with a white marker those positions in the field which offer a good passing angle, and in dark those positions with a worse passing angle.

6. Minimal tangent distance

   This value is the shortest distance an opponent has to move in order to block a pass. Fig. 4 shows, in black, those field areas where a pass can go through. The white areas can be blocked by the opponents. In the example, the receiving robot (lower right) is too near to an opponent. It would be better if the receiving robot was lo-

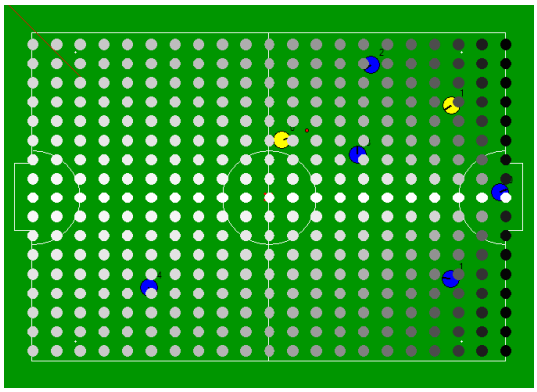


Figure 3: Passing angle: White areas of the field provide a good passing angle, dark areas do not

cated at some point in the diagonal corridor going from the middle of the field to the upper right.

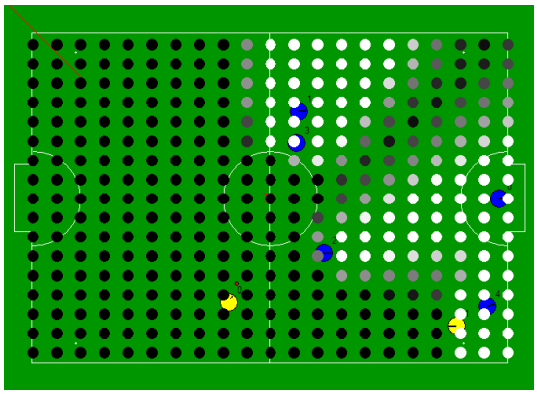


Figure 4: Minimal tangent distance: In dark areas of the field the distance to block a pass is large, in white areas not

7. Waiting time during passing

   This feature is the time elapsed after an opponent has moved to a new position, where a pass can be blocked, and the instant where the ball arrives. An opponent with a large waiting time can block a pass easily. An opponent with negative waiting time cannot reach the ball. Fig. 5 shows an example where the dark areas cannot be easily reached by the opponents to stop a pass, while the white or gray areas can be reached easily. This is probably one of the main criterions which need to be used for deciding to give a pass or not.

8. Dribbling freedom for the receiver after a pass

   This is the same parameter as ”dribbling freedom” for the robot with the ball, but now for the robot receiving the pass. Fig. 6 shows the regions of the field where receiving a pass provides high reward (in black) and low reward (in white).

It is worth noting that all these features encode symmetrical field positions with the same numbers. If we had stored the coordinates of the robots, we would have to store all the

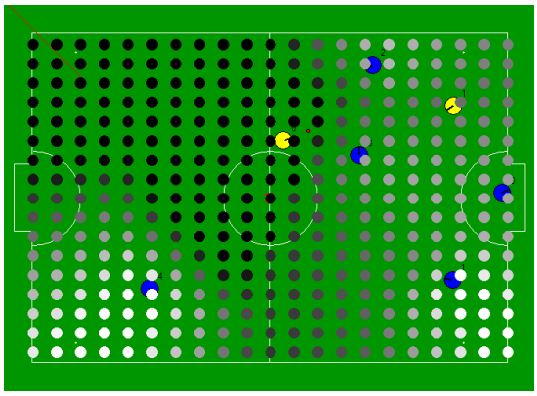


Figure 5: Waiting time: Opponent robots in white areas can wait longer for the ball when a pass is coming

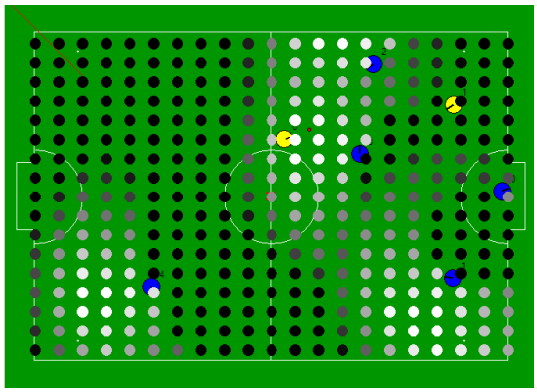


Figure 6: Dribbling freedom for the receiver. Dark areas indicate much free space, bright areas are covered by opponents

symmetrical cases every time we enter an example. The features described above take care of encoding all symmetrical field positions with the same feature vector. The classifier has an easier task, once the symmetries of the learning problem have been incorporated in the encoding.

4 Training neural networks

We have written a tool to keep track of all the examples of game situations defined by a human coach (or several human coaches). We can group the examples in several categories to have a better overview of them and to activate and deactivate particular example groups.

By activating and deactivating groups, we can train different neural networks to encode different behaviors. For example, if an opponent has the ability to block long passes across the field, we can deselect all examples where long passes are considered “good”, and we can train a network which behaves essentially as the original one, except for its reluctance now to propose long passes across the field.

We achieved good classification results using a three-layer feed forward network [Rojas, 1996]. The network has 8 input nodes - the features seen in section 3 - and a hidden layer, whose dimension can be set arbitrarily. Right now, we usu-

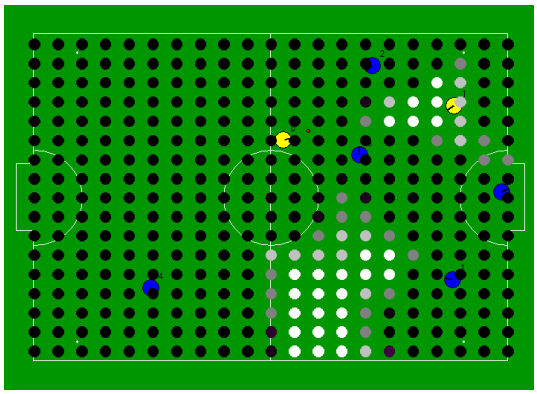


Figure 7: This is the output of a trained neural coach. If a teammate stands in one of the 2 bright areas, the left player with the ball should pass.

ally have 14 hidden nodes. The output node emits a value that indicates the action that should be taken. An output in the interval $(-\infty; 0]$ is interpreted as "Do not pass", while an output in $(0; \infty)$ as "Do pass". The neural network we use, has already reached a size where it is questionable whether it would be good to let it grow further. One would need too many examples to cover all degrees of freedom of the network. Right know, we have stored around 100 examples in our system. The effort is worth it: game situations are created quickly with our simulator, and the results we obtain have good quality.

The trained networks can be saved, and it is possible to use them for behavior control. In our current system, the eventual passer first ponders shooting a goal. This behavior inhibits all others. If shooting is unfeasible or not so good, the robot considers whether to dribble or to pass. The system iterates over all team mates (as possible receivers of a pass), and asks the coach whether it would be good to pass or not. If there are more than one well positioned receivers, the passer robot selects the one for which turning towards it is easier.

We have succesfully used this system in the robocup challenge "German Open 2005". It is hard to compare the results empirically to other methods, but we hope that a video that shows the output of the net in dynamic gameplay can convince you. Please take a look at it at http://robocup.mi.fu-berlin.de/videos/NeuralCoach/index.html.

In our robocup domain, it is possible to use this system in real-time. In a whole compution cycle we not only evaluate the quality of a pass to 3 different team mates, but also calculate the grid seen in 7 every frame using a resolution of $16 * 10$. Then this grid is used for player positioning. To evalute these 160 imaginary passees, 2ms are needed. The largest part of the 2ms is used to evaluate the exponential function that is used by the activation function of the neural net. One could further reduce the computation time by either having the exponential function in hardware or by parallelising the calculations.

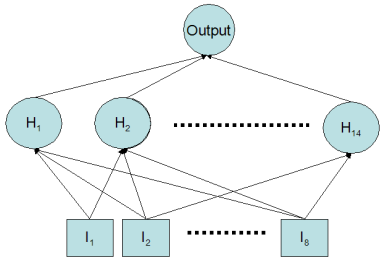


Figure 8: The topology of the neural network used as passing coach

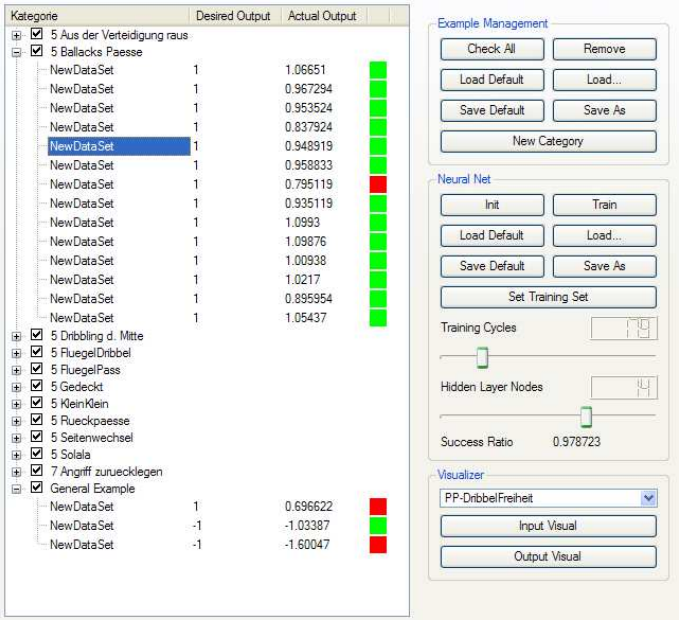


Figure 9: Our management tool saves static examples of game situations. The examples can be deleted, or organized in groups which correspond to alternative offensive strategies.

5 Application to other domains

In this section we show a posible use of this approach in an other domain. Let us consider a mars rover that wants to travel to a specific destination. Further let us have a system that suggests different routes. What route is considered best? This is an offline problem that could be tackled with our approach. For every obstacle type one or more problem specific values are calculated. For example:

- Planes
  1. How many meters do we have to traverse?
- Mountains
  1. How many meters do we have to traverse?
  2. What is the summarized attitude difference?
  3. What is the steepest rise?
- ...

With these features, engineers can now use the neural coach to determine the best route. They can create an ex-

ample where it is advantageous to go over an additional 10 meter height of a mountain rather than taking a 100 meter detour on a plain if the rise is gradual enough. On the other hand is a mountain with a height of only 5 meter but a rise of 40 degrees not acceptable.

This is a very comfortable procedure to define the desired behavior. If the neural coach has a well written flexible framework for taking and managing examples, the only important and creative task is to find the right problem specific features to calculate.

6 Conclusion

We have developed a "neural coach" for our small-size robots. The coach is a neural network which accepts game situations encoded as a feature vector, and which provides as output a number reflecting the best alternative: dribbling with the ball or passing. The robot with the ball can periodically ask the neural network which is the best strategy, and can apply it.

Our behavior control system has grown with the years and contains many parameters which must be tuned by hand. It is difficult to modify them when the hardware changes, also because the many programmers work with our system. Our coaching tool is a decisive step towards abandoning such hand-tuned implementations, in favor of a more general approach.

Many other game decisions could be modelled in the way described in this paper, like for example the team formation, or individual behaviors of a robot ("I am the goalkeeper and see a opponent dribbling to my goal. My defenders are far away. Shall I come out of my goal to decrease the shooting angle or not?").If the behavior control system can also learn the low-level skills, such as driving, or dribbling with the ball, using reinforcement learning or other machine learning algorithms, one obtains a more versatile robotic platform, and code which is easier to manage and maintain.

References

[Fidelman and Stone, 2004] Peggy Fidelman and Peter Stone. Learning ball acquisition on a physical robot. In *2004 International Symposium on Robotics and Automation (ISRA)*, August 2004.

[Gloye *et al.*, 2004] Alexander Gloye, C. Goektekin, Anna Egorova, Oliver Tenchio, and Raul Rojas. Learning to drive and simulate autonomous mobile robots. In *RoboCup-2004: Robot Soccer World Cup VIII*. Springer-Verlag, 2004.

[Kok *et al.*, 2003] Jelle R. Kok, Matthijs T. J. Spaan, and Nikos Vlassis. Multi-robot decision making using coordination graphs. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 1124–1129, Coimbra, Portugal, 2003.

[Kuhlmann *et al.*, 2005] Gregory Kuhlmann, Peter Stone, and Justin Lallinger. The UT Austin Villa 2003 champion simulator coach: A machine learning approach. In Daniele Nardi, Martin Riedmiller, and Claude Sammut, editors, *RoboCup-2004: Robot Soccer World Cup VIII*, pages 636–644. Springer Verlag, Berlin, 2005.

[Lauer and Riedmiller, 2004] Martin Lauer and Martin Riedmiller. Reinforcement learning for stochastic cooperative multi-agent-systems. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3 (AAMAS'04)*, 2004.

[Rojas, 1996] Raul Rojas. *Neural Networks - A Systematic Introduction*. Springer-Verlag, New York, 1996.

[Ydren and Scerri, 1999] Johan Ydren and Paul Scerri. An editor for user friendly strategy. *Robocup Simulation League, Team Description Headless Chicken*, pages 44–47, 1999.

.

HTN Planning for Flexible Coordination Of Multiagent Team Behavior

Oliver Obst and **Anita Maas** and **Joschka Boedecker**
AI Research Group, Universität Koblenz
Universitätsstr. 1,
56070 Koblenz, Germany
{fruit, maas, jboedeck}@uni-koblenz.de

Abstract

Coordination of agents in a dynamic and nondeterministic environment is a difficult task. There are many approaches to this problem where agents are controlled reactively. In this paper we present an approach to coordinate the behavior of a multiagent team using an HTN (Hierarchical Task Network) planning procedure. To coordinate teams, high level tasks have to be broken down into subtasks which is a basic operation in HTN planners. We are using planners in each of the agents to incorporate domain knowledge and to make agents follow a specified team strategy. With our approach, agents coordinate deliberatively and still maintain a high degree of reactivity. In our implementation for use in the RoboCup Simulation League, first results were already very promising. Using a planner helps to maintain a clear agent design, separating the agent code from the expert domain knowledge.

1 Introduction

Coordination among different agents and the specification of strategies for multiagent systems (MAS) is a challenging task. For a human domain expert it is often very difficult to change the behavior of a multiagent system. This is especially true when not only general tasks should be specified, but also the way in which tasks are to be executed. Due to interdependencies simple changes in one place of the code may easily affect more than one situation during execution.

In this work, we suggest to use Hierarchical Task Network (HTN) planners in each of the agents in order to achieve coordinated team behavior which is in accordance with the strategy given by the human expert. The expert knowledge should be separated from the rest of the agent code in a way that it can easily be specified and changed. While pursuing the given strategy, agents should keep as much of their reactiveness as possible. HTN planning explicitly supports the use of domain specific strategies. To coordinate groups of agents, tasks usually have to be broken down into subtasks, which is one of the basic operations of HTN planning. Different levels of detail in the description of strategies further facilitate the generation of useful information for debugging or synchronization.

In classical planning, operators are deterministic and the single planning agent is the only reason for changes in the environment under consideration. We show how it is possible to use an HTN planner in the domain of robotic soccer, even though the robotic soccer environment is very different from classical planning domains. For our approach, we have chosen a team of agents for the RoboCup 3D Soccer Simulator [Obst and Rollmann, 2005] that was introduced at RoboCup-2004 in Lisbon [Lima *et al.*, 2005].

The following section describes our approach to coordinate the behavior of a multiagent team using an HTN planner. Section 3 contains the description of an implemented example. We present and discuss the results of our first tests, and give a review of relevant related work. Finally, Section 6 concludes the paper.

2 HTN Planning for Multiagent Teams

The usual assumptions for HTN planning, like for classical planning approaches, are that we plan for a single agent who is the only cause for changes in the domain. When the plan is executed, all actions succeed as planned. Executing an action in a classical planning framework is instantaneous, it takes no time, and therefore the world is always in a defined state.

To plan for agents in a team and in a real-world domain, we have to relax some of these assumptions and find a way to deal with the new setting. The definition below is a way commonly used to define nondeterministic planning domains. An approach to deal with these kinds of domains is to use model checking (see for instance [Cimatti *et al.*, 2003]). Depending on the problem and the desired properties of the results, the planner tries to compute solution plans that have a chance to succeed or solution plans that succeed no matter what the results of the nondeterministic actions of an agent are.

Definition A *nondeterministic planning domain* is a triple $\Sigma = \langle S, A, \gamma \rangle$, where:

- S is a finite set of states.

- A is a finite set of actions.
- $\gamma \subseteq S \times A \times S$ is the state-transition relation. □

When the number of different possible results of γ is high, computing a plan can easily become intractable for domains where decisions have to be made quickly. Nevertheless, using a planner could still be useful to achieve high-level coordination for a team of several agents in a dynamic environment without using communication and without a centralized planning facility. For our approach, all planning should be done in a distributed fashion in each of the autonomous agents. The task of the system is to automatically generate individual actions for the agents in accordance with those plans during execution. Despite using plans, agents should still be able to react to unforeseen changes in the environment. A further goal of using a planner is that team behavior can easily be specified and extended, which is supported by the separation of agent code and expert domain knowledge.

2.1 Multiagent Team Behavior with HTN Plans

In Hierarchical Task Network (HTN, see Definition below) planning, the objective is to perform tasks. Tasks can be complex or primitive. HTN planners use *methods* to expand complex tasks into subtasks, until the tasks are primitive. Primitive tasks can be performed directly by using planning operators.

Definition A *task network* is an acyclic directed graph $w = \langle N, A \rangle$, where N is the set of nodes, and A is the set of directed edges. Each node in N contains a task t_n. A task network is *primitive*, if all of its tasks are primitive, otherwise it is *nonprimitive*. □

Our approach of interleaving planning and acting, and also of handling nondeterministic actions, is similar to the one described in [Belker *et al.*, 2003] where an HTN planner is used for navigation planning of a single robot. Here, like in most realistic environments, it is not enough to initially create a plan and blindly execute it, but after execution of each action the state of the world needs to be sensed in order to monitor progress. As a consequence, for generating HTN plans it is not absolutely necessary to generate a primitive task network from the beginning. Instead, an HTN where the first tasks are primitive is sufficient, if we interleave planning and acting. Future tasks are left unexpanded or partially expanded until the present tasks are done and there is no other task in front. In dynamic and complex environments, creating a detailed plan can be considered as wasted time, because it becomes virtually impossible to predict the state of the world after only a few actions already.

Rather than expanding complex tasks completely, our planner generates what is called *plan stub* in [Belker *et al.*, 2003], a task network with a primitive task as the first task. As soon as a plan stub has been found, an agent can start executing its task. The algorithm in Fig. 1 expands a list of tasks to a plan stub, if it is not already in that form. The notation of our algorithms is similar to the one used in [Ghallab *et al.*, 2004]: $\langle t_1, ..., t_k \rangle$ is a set of tasks, O is the set of operators, M is the set of methods, subtasks(m) stands for the set of subtasks of a method m, and the dot ('.') used in the algorithms denotes a concatenation.

Function: plan($s_{now}, \langle t_1, ..., t_k \rangle, O, M$)
Returns: (w, s), with w an ordered set of tasks, s a state; or *failure*

if $k = 0$ **then return** $(\emptyset, s_{now})$ `// i.e. the empty plan`
if t_1 *is a pending primitive task* **then**
  active $\leftarrow \{(a, \sigma) \mid$ a is a ground instance of an operator in O, σ is a substitution such that a is relevant for $\sigma(t_1)$, and a is applicable to $s_{now}\}$;
  if *active* $= \emptyset$ **then return** *failure*;
  nondeterministically choose any $(a, \sigma) \in$ *active*;
  return $(\sigma(\langle t_1, ..., t_k \rangle), \gamma(s_{now}, a))$;
else if t_1 *is a pending complex task* **then**
  active $\leftarrow \{m \mid$ m is a ground instance of a method in M, σ is a substitution such that m is relevant for $\sigma(t_1)$, and m is applicable to $s_{now}\}$;
  if *active* $= \emptyset$ **then return** *failure*;
  nondeterministically choose any $(m, \sigma) \in$ *active*;
  $w \leftarrow$ subtasks(m).$\sigma(\langle t_1, ..., t_k \rangle)$;
  set all tasks in front of t_1 to *pending*, set t_1 to *expanded*;
  return plan(s_{now}, w, O, M);
else
  `//` t_1 `is an already executed expanded task and can be removed`
  return plan($s_{now}, \langle t_2, ..., t_k \rangle, O, M$);

Figure 1: Creating an initial plan stub.

In classical planning, executing an action takes no time. This means that immediately after executing a planning operator, the world is in the successor state. In our approach we have to consider that actions are not instantaneous and might not even yield the desired result. The first problem is when to regard operators as finally executed: Depending on the actual domain agents are acting in, actions can be regarded as finished after a given amount of time or when a specified condition holds. This domain specific solution to this problem is not part of the algorithms in this paper.

A second problem is the computation of the successor state: as defined above, for nondeterministic environments γ is a relation with possibly several results for the same state-action pair. For our algorithms, we expect γ to be a function returning the *desired* successor state, more precisely a subset of the desired successor state. The returned state should describe those properties of the environment that are deliberately changed by an action. Likewise, the effects of an operator describe the

desired effects. The underlying assumption is that operators have a single purpose so that the desired successor state can be uniquely described. The desired effects can be used by the operators to coordinate actions of teammates during the same plan step. For this, we introduce multiagent operators, which is effectively a shortcut for defining a set of combinations of operators. Actions that are executed simultaneously but which do not contribute to the desired effects of the multiagent operator are simply not included. This makes it easy for the developer of a multiagent team to create team operators, but the disadvantage is that agents not modeled as part of the multiagent team cannot be regarded with our approach.

Definition (Multiagent Operator) Let $o_1, ..., o_n$ be operators, $\text{effects}^-(o)$ and $\text{effects}^+(o)$ the negative and positive effects of an operator o, respectively, and $\text{effects}^-(o_j) \cap \text{effects}^+(o_k) = \emptyset$ for all $j, k \in \{1, ..., n\}$. p is a new operator with $\text{name}(p) = \text{name}(o_1)$ `while` $\langle \text{name}(o_2), ..., \text{name}(o_n) \rangle$. The preconditions and effects of p are defined as unions over the preconditions and effects of all o_i, respectively:

$$\text{pre}(p) = \bigcup_{i=1,...,n} \text{pre}(o_i), \quad \text{and}$$

$$\text{effects}(p) = \bigcup_{i=1,...,n} \text{effects}(o_i)$$

□

The multiagent operator describes the actions of several agents; the operator in front of the `while` is the one actually executed by the agent, and the operators after it are used to determine the collective preconditions and effects of the team action. In the algorithms, a multiagent operator is treated as regular operator with the difference that at execution time only the operator in front of the `while` leads to an action by the respective agent.

The desired successor state is used to check the success of the last operator application in the second algorithm (see Fig. 2). Both algorithms treat plans as a stack, tasks on this stack are marked as either *pending* or as *expanded*. Pending tasks are either about to be executed, if they are primitive, or waiting to be further expanded, if they are complex. Tasks marked as expanded are complex tasks which already have been expanded into subtasks. The function `step` removes executed tasks from the plan, it is called whenever a step was finished. If the task was successfully executed, only the finished task is removed from the stack – and possibly also parent tasks if there are no further pending child tasks. If execution of the task failed, all subtasks of the parent task have to be removed. In this case, it is checked if the parent complex task can be tried again. Function `plan` from Fig. 1 is used to create an initial plan stub by calling the function with an initial task. It is also used to create an updated plan stub when called from `step`.

Function: step($s_{expected}, s_{now}, \langle t_1, ..., t_k \rangle, O, M$)
Returns: (w, s), with w a set of ordered tasks, s a state; or *failure*

if $k = 0$ **then return** $(\emptyset, s_{now})$ `// i.e., the empty plan`
if t_1 *is a pending task* **then**
  if $s_{expected}$ *is valid in* s_{now} **then**
    $i \leftarrow$ the position of the first nonprimitive task in the list;
    return *plan*$(s_{now}, \langle t_i, ..., t_k \rangle, O, M)$;
  else
    `//` t_1 `was unsuccessful; remove all pending children of our parent task`
    return *step*$(s_{expected}, s_{now}, \langle t_2, ..., t_k \rangle, O, M)$;

else
  `//` t_1 `is an unsuccessfully terminated expanded task, try to re-apply it`
  $active \leftarrow \{m \mid m$ is a ground instance of a method in M, σ is a substitution such that m is relevant for $\sigma(t_1)$, and m is applicable to $s_{now}\}$;
  if $active = \emptyset$ **then**
    `//` t_1 `cannot be re-applied, remove it from the list and recurse`
    return step$(s_{expected}, s_{now}, \langle t_2, ..., t_k \rangle, O, M)$;
  else
    nondeterministically choose any $(m, \sigma) \in active$;
    $w \leftarrow \text{subtasks}(m).\sigma(\langle t_1, ..., t_k \rangle)$;
    set all tasks in front of t_1 to *pending*, set t_1 to *expanded*;
    return plan(s_{now}, w, O, M);

Figure 2: Remove the top primitive tasks and create a new plan stub.

3 Robotic Soccer Sample Implementation

To give an example, we take the simulated soccer domain [Kögler and Obst, 2004; Obst and Rollmann, 2005]. In [Dylla *et al.*, 2005], we formalized soccer domain knowledge as it can be found in soccer theory books [Lucchesi, 2001]. Based on the diagrams in this book (see for example Fig. 3), we created HTN methods for the simulated soccer domain.

Figure 4 shows that part of the plan stack which contains the team plan for the situation depicted in Fig. 3. All pending tasks in this plan stack are still complex tasks on the team level, so that this stack could be part of any of the agents on the field. It was created by expanding the top level task `play_soccer` into `offensive_phase`. The task `offensive_phase` was expanded to `build_up_play`, `final_touch` and `shooting`. In the current situation, only the first task of this sequence, `build_up_play`, was already expanded to `build_up_play_long_pass`, which in turn was expanded to `diagram-4`. Finally, `diagram-4` expanded to the sequence `pass(2,9)`, `pass(9,10)` and `leading-pass(10,11)`.

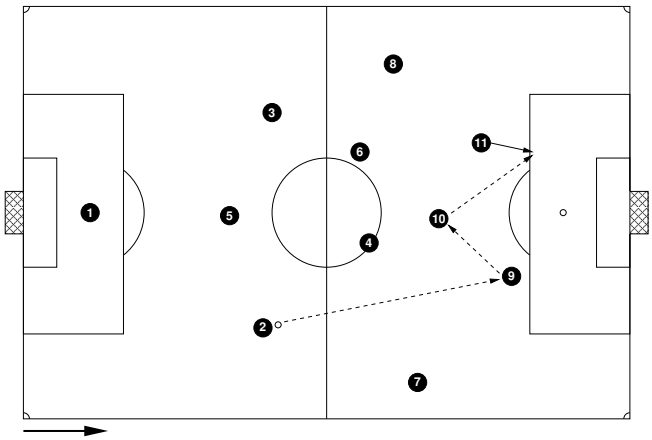


Figure 3: Diagram #4 from [Lucchesi, 2001]

To create a plan stub so that an action can be executed, the planner needs to further expand the top pending task, in this case `pass(2,9)`. When team tasks get further expanded to agent tasks, each agent has to find its role in the team task: the HTN methods contain variables that need to be unified with actual uniform numbers. In our soccer example, the role finding is done via preconditions on the current formation, position and function of the respective players in the formation. This also means that symmetric situations are handled automatically (provided the formation of the team is also symmetric).

Further expanding the abstract plan, agent #2 will expand `pass(2,9)` to `do_pass(9)`, agent #9 has to do a `do_receive_pass` for the same team task. The other agents position themselves relatively to the current ball position with `do_positioning` at the same time. The desired effect of `pass(2,9)` is the same for all the agents, even if the derived primitive task is different depending on the role of the agent. That means each agent has to execute a different action, which is realized as `C++` function call in our case, and at the same time an operator has to update the desired successor state independently. To express that an agent should execute the `do_positioning` behavior while taking the effect of a simultaneous pass between two teammates into account, we are using terms like `do_positioning while pass(we,2,9)` in our planner. Figure 5 shows methods reducing the team task `pass(A,B)` to different primitive player tasks.

In different agents, the applicable methods for the top team task `pass(2,9)` lead to different plan stubs. This is an important difference to the work presented in [Belker *et al.*, 2003]. The plan stubs created as first step for agent 9 and agent 11 are shown in Fig. 6 and 7. When a plan stub is found, the top primitive tasks are passed to the `C++` module of our agent and executed. A 'step' for a plan in our agents can consist of more than a single action, for example, we do not want the agent who passes the ball to stop acting while the ball is already moving to a teammate, but instead after the kick the

```
pending-pass(2,9)
pending-pass(9,10)
pending-leading-pass(10,11)
expanded-diagram-4
expanded-build_up_play_long_pass
expanded-build_up_play
pending-final_touch
pending-shooting
expanded-offensive_phase
expanded-play_soccer
```

Figure 4: Plan stack during planning.

```
method pass(A,B)
pre [my_number(A)]
subtasks [do_pass(B) while pass(we,A,B),
          do_positioning].

method pass(A,B)
pre [my_number(B)]
subtasks [do_receive_pass while pass(we,A,B)].

method pass(A,B)
pre [my_number(C),#\=(A,C),#\=(B,C)]
subtasks [do_positioning while pass(we,A,B)].
```

Figure 5: Different methods to reduce the team task `pass(A,B)` to agent tasks.

agent should adjust its position relative to the ball until the ball reached its destination and the step is finished. If possible, the agent has to execute all pending primitive tasks until the next step in the plan starts. If there are pending primitive tasks after one step is finished, these agent tasks are simply removed from the plan stack and the next team task can be expanded. Figures 8 and 9 show the plan stub for the second step from the diagram in Fig. 3. For player 11, the expansion leads to a plan stub with two primitive tasks in a plan step while for player 9 there is only one task to be executed.

What we did not address so far was the point in time when the transition from one plan step to the next step takes place. Here, the basic idea is the following: each step in plans for our team stops or starts with an agent being in ball possession. If any of the agents on the field is in ball possession, we can check for the desired effect of our previous action. If the action succeeded, the right agent possesses the ball and the planner can

```
pending-(do_receive_pass while
         pass(we, 2, 9)),
expanded-pass(2, 9),
pending-pass(9, 10),
pending-leading_pass(10, 11),
expanded-diagram-4,
...
```

Figure 6: Step 1: Player 9 receives the pass.

```
pending-(do_positioning while
         pass(we, 2, 9)),
expanded-pass(2, 9),
pending-pass(9, 10),
pending-leading_pass(10, 11),
expanded-diagram-4,
...
```

Figure 7: Step 1: While players 2 and 9 pass, player 11 stays in the formation.

```
pending-(do_pass(10) while
         pass(we, 9, 10)),
pending-do_positioning,
expanded-pass(9, 10),
pending-leading_pass(10, 11),
expanded-diagram_4,
...
```

Figure 8: Step 2: Player 9 passes to 10.

continue planning by generating the next plan stub. If an adversarial agent intercepted the ball, the last action failed and the planner needs to backtrack. For dribbling, the planner needs to check if the dribbling agent still possesses the ball and arrived at the desired destination in order to start with the next step.

4 Results and Discussion

For our approach of generating coordinated actions in a team we implemented an HTN planner in Prolog which supports interleaving of planning and acting. Our planner supports team actions by explicitly taking the effects of operators simultaneously used by teammates into account. The planner ensures that the agents follow the strategy specified by the user of the system by generating individual actions for each of the agents that are in accordance with it. The *lazy evaluation* in the expansion of subtasks which generates plan stubs rather than a full plan, makes the planning process very fast and enables the agents to stay reactive to unexpected changes in the environment. The reactiveness could, however, be increased by adding a situation evaluation mechanism that is used prior to invoking the planner. This would improve the ability to exploit sudden, short-lived opportunities during the game.

We implemented a distributed planning system in the sense that each of the agents uses its own planner. This was, however, somewhat facilitated by the fact that agents in the RoboCup 3D Simulation League are equipped with sensors that provide them with a full (though possibly inaccurate) view of the world, similar to Middle-size League robots using omni-vision cameras.

```
pending-(do_positioning while
         pass(we, 9, 10)),
expanded-pass(9, 10),
pending-leading_pass(10, 11),
expanded-diagram_4,
...
```

Figure 9: Step 2: Player 11 stays in the formation while player 9 passes to 10.

To truly evaluate the approach we presented, it would be necessary to measure the effort it takes to create a team and compare it to other approaches to create a team exhibiting the same behavior. We strongly believe that our approach leads to a modular behavior design and facilitates rapid specification of team behavior for *users* of our agents, but we cannot present numbers here. A comparison to the results of other teams is not helpful here, because better results do not necessarily mean that the planning procedure is the reason for differences in the performance: in many cases, careful engineering can lead to implementations that perform well without using AI techniques.

Our plans can describe plays as introduced in [Bowling *et al.*, 2004], which have shown to be useful for synchronization in a team. There are some important differences to plays, however. First, our approach supports different levels of abstraction in plans. That means there are different levels of detail available to describe what our team and each single agent is actually doing, from very abstract tasks down to the agent level tasks. A second important difference is that the planner can find alternative ways to achieve tasks. This is possible if plays are specified in terms of player roles or properties rather than fixed player numbers. The approach in [Bowling *et al.*, 2004] was used for Small Size League, where the numbers of players and the number of alternative ways of doing plays is low. That means in Small Size League, a plan is either applicable or not. For Simulation League or larger teams in general, more opportunities are possible for which an approach using fixed teammates seems to restrictive. On the other hand, the approach in [Bowling *et al.*, 2004] supports adaptation by changing weights for the selection of successful plays. In our approach, the corresponding functionality could be achieved by changing the order in which HTN methods are used to reduce tasks. At this point in time, our approach does not support this yet. As soon as we do have an adaptive component in our approach, it makes sense to compare results of our team with and without adaptation.

The way our plans are created and executed, we assume synchronous actions for all our agents. Our team actions are geared to actions of the player in ball possession, so this simplification can be made. There are a few situations in soccer, where more detailed reasoning over the time actions take would be useful. This includes for instance all situations where a ball receiver should appear at the receiving position *just in time* to surprise the opponent. In our approach, we make this possible by synchronizing the behavior of two agents in the *current step* by using both ball and agent velocity to estimate interception times, in the operator implementations outside of the planning procedure. Inside our planning procedure, we do not reason about durations, which would be useful to make asynchronous actions possible.

Although more detailed evaluations have to be carried out, the first tests using the planner seem very promising and indicate that our approach provides a flexible, easily extendable method for coordinating a team of agents in dynamic domains like the RoboCup 3D Simulation League.

5 Related Work

Several approaches that use a planning component in a MAS can be found in the literature.

In [Dix *et al.*, 2000], the authors describe a formalism to integrate the HTN planning system SHOP [Nau *et al.*, 1999] with the IMPACT [Subrahmanian *et al.*, 2000] multiagent environment (A-SHOP). The preconditions and effects used in SHOP are modified so that preconditions are evaluated using the code-call mechanism of the framework, and effects change the state of agents. While the environment of this work clearly is a multiagent system, the planning is carried out centralized by a single agent. This is a contrast to our approach, which uses a planner in each of the agents to coordinate the agents actions.

Planning in each of the agents in the RETSINA multiagent system [Paolucci *et al.*, 2000] is also HTN based. Additionally to the planning module, RETSINA agents consist of a scheduler, a communicator and an execution monitor. The architecture of the system is targeted towards agents that interact by exchanging informations, in contrast to our approach where agents basically cooperate by physical actions. RETSINA uses a special mechanism to suspend tasks that need to wait for information gathering processes. To decide if the execution of a task failed, RETSINA uses sets of constraints describing conditions that should hold during or after the execution. The basic planning algorithm in RETSINA returns partial solution plans, then they are scheduled for execution and finally executed by the execution monitor. In our approach, the planner returns plan stubs where the first task is already executable.

A general HTN planning framework for agents in dynamic environments has been presented in [Hayashi *et al.*, 2004]. The authors show how to integrate task decomposition of HTN planning, action execution, program updates, and plan modifications. The planning process is done via abstract task decomposition and is augmented to include additional information such as the history of action execution for the plans to enable their incremental modification. Rules are given for plan modifications after having executed certain actions or after program updates. In the robotic soccer domain, however, the results of actions like e.g. kicking the ball cannot be undone. Thus, the plan modification mechanism given in [Hayashi *et al.*, 2004] does not apply and could not easily be used for our purposes.

HTN planning has also been studied in the context of creating intelligent, cooperating Non-Player Characters in computer games. In [Muñoz-Avila and Fisher, 2004], an HTN planner is used to enable agents in the highly dynamic environment of the Unreal Tournament game to pursue a grand strategy designed for the team of agents.

Bowling *et al.* [Bowling *et al.*, 2004] presents a strategy system that makes use of *plays* (essentially being multiagent plans) to coordinate team behavior of robots in the RoboCup Small Size League. Multiple plays are managed in a *playbook* which is responsible to choose appropriate plays, and evaluate them for adaption purposes. The plays are specified using a special language designed with ease of readability and extensibility in mind. Preconditions can be specified that determine when a play can be executed. Furthermore, plays contain termination conditions, role assignments and sequences of individual behaviors. While the use of preconditions resembles a classical planning approach, the effects of individual plays are not specified due to the difficulties in predicting the outcome of operators in the dynamic environment. This is in contrast to our approach, as we use *desired effects* of the operators in our plans. Another difference is that in [Bowling *et al.*, 2004] the planning component is also centralized.

A centralized planner is also used in [Riley and Veloso, 2002] to generate team plans for distributed execution. A coach agent observes the opponents agents and uses opponent models in the planning process. It communicates the plan to the agents periodically and the agents use this information to maintain consistency in their cooperating behavior. The team plans are represented as Simple Temporal Networks which are essentially directed graphs describing the temporal constraints between events. Using this representation, the specification of parallel events is facilitated and can also be used for monitoring purposes. Despite those appealing features of Simple Temporal Networks for multiagent plan specification, we used a rather more traditional representation without any explicit modeling of execution times for the operators for the sake of easier integration into the planner. They might, however, be beneficial for a more fine grained control over the parallelism in our plans.

Other approaches towards multiagent collaboration like [Cohen *et al.*, 1998; Grosz, 1996] are based on negotiations between the agents in a multiagent system. However, as pointed out in [Stone and Veloso, 1999], this kind of complex communication might take too much time or might even be infeasible in highly dynamic real-time domains like robotic soccer.

The work in [Murray *et al.*, 2002; Murray, 2003] describes the approach to creating our agents so far: We used UML statecharts to specify behaviors for agents in a multiagent system. The agents were designed in a top-down manner with a layered architecture. At the highest level global patterns of behavior are specified in an abstract way, representing the different states the agent can be in. For each of these states, an agent has a repertoire of skeleton plans in the next layer. These are applicable as long as the state does not change. Explicit specification of cooperation and multiagent behaviors can be realized. The third and lowest level of the architecture encompasses the descriptions for the simple and com-

plex actions the agents can execute, which are used by the scripts in the level above.

This hierarchical decomposition of agent behaviors is similar to the HTN plans described in this work. However, the separation of domain description knowledge and the reasoning formalism accomplished through the use of the HTN planner within our agents provides us with much greater flexibility in respect to the extensibility of methods and operators, compared to the amount of work needed to change the state machine description.

6 Conclusion and Future Work

We presented a novel approach that uses an HTN planning component to coordinate the behavior of multiple agents in a dynamic MAS. We formalized expert domain knowledge and used it in the planning methods to subdivide the given tasks. The hierarchical structure of the plans speeds up the planning and also helps to generate useful debugging output for development. Furthermore, the system is easily extensible as the planning logic and the domain knowledge are separated.

In order to use the system in the RoboCup competitions, we plan to integrate a lot more subdivision strategies for the different tasks as described in the diagrams in [Lucchesi, 2001]. A desirable enhancement to our work would be the integration of an adaption mechanism. Monitoring the success of different strategies against a certain opponent, and using this information in the choice of several applicable action possibilities, as e.g. outlined in [Bowling *et al.*, 2004], should be explored. The introduction of *durative actions* into the planner (see for instance [Coddington *et al.*, 2001]) would give a more fine grained control over the parallelism in the multiagent plans. *Simple Temporal Networks* as used in [Riley and Veloso, 2002] seem to be well suited for this purpose. Furthermore, a situation assessment will be added to the agents to be able to exploit unforeseen situations in a more reactive manner. Finally, we want to restrict the sensors of the agents to receive only partial information about the current world state, and address the issues that result for the distributed planning process.

References

[Belker *et al.*, 2003] Thorsten Belker, Martin Hammel, and Joachim Hertzberg. Learning to optimize mobile robot navigation based on HTN plans. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2003)*, pages 4136–4141, Taipei, Taiwan, September 2003.

[Bowling *et al.*, 2004] Michael Bowling, Brett Browning, and Manuela Veloso. Plays as team plans for coordination and adaptation. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, Vancouver, June 2004.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[Coddington *et al.*, 2001] Alex M. Coddington, Maria Fox, and Derek Long. Handling durative actions in classical planning frameworks. In John Levine, editor, *Proceedings of the 20th Workshop of the UK Planning and Scheduling Special Interest Group*, pages 44–58. University of Edinburgh, December 2001.

[Cohen *et al.*, 1998] Philip R. Cohen, Hector J. Levesque, and Ira Smith. On team formation. *Contemporary Action Theory*, 1998.

[Dix *et al.*, 2000] Jürgen Dix, Héctor Muñoz-Avila, and Dana Nau. IMPACTing SHOP: Planning in a Multi-Agent Environment. In Fariba Sadri and Ken Satoh, editors, *Proceedings of CLIMA 2000, Workshop at CL 2000*, pages 30–42. Imperial College, 2000.

[Dylla *et al.*, 2005] Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Frieder Stolzenburg, Ubbo Visser, and Thomas Wagner. Towards a league-independent qualitative soccer theory for RoboCup. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 611–618, Berlin, Heidelberg, New York, 2005. Springer.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning Theory and Practice*. Morgan Kaufmann, San Francisco, CA, USA, 2004.

[Grosz, 1996] Barbara J. Grosz. AAAI-94 presidential address: Collaborative systems. *AI Magazine*, 17(2):67–85, 1996.

[Hayashi *et al.*, 2004] Hisashi Hayashi, Kenta Cho, and Akihiko Ohsuga. A new HTN planning framework for agents in dynamic environments. In Jürgen Dix and João Leite, editors, *CLIMA IV 2004*, number 3259 in Lecture Notes in Computer Science, pages 108–133. Springer, Berlin, Heidelberg, New York, 2004.

[Kögler and Obst, 2004] Marco Kögler and Oliver Obst. Simulation league: The next generation. In Polani et al. [2004], pages 458–469.

[Lima *et al.*, 2005] Pedro Lima, Luís Custódio, Levent Akin, Adam Jacoff, Gerhard Kraezschmar, Beng Kiat Ng, Oliver Obst, Thomas Röfer, Yasutake Takahashi, and Changjiu Zhou. Robocup 2004 competitions and symposium: A small kick for robots, a giant score for science. *AI Magazine*, 2005. To appear.

[Lucchesi, 2001] Massimo Lucchesi. *Coaching the 3-4-1-2 and 4-2-3-1*. Reedswain Publishing, 2001.

[Muñoz-Avila and Fisher, 2004] Héctor Muñoz-Avila and Todd Fisher. Strategic planning for Unreal Tournament bots. In *Proceedings of AAAI-04 Workshop on Challenges on Game AI*. AAAI Press, 2004.

[Murray *et al.*, 2002] Jan Murray, Oliver Obst, and Frieder Stolzenburg. RoboLog Koblenz 2001. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*, pages 526–530. Springer, Berlin, Heidelberg, New York, 2002. Team description.

[Murray, 2003] Jan Murray. Specifying agent behaviors with UML statecharts and StatEdit. In Polani et al. [2004], pages 145–156.

[Nau *et al.*, 1999] Dana S. Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of IJCAI-99*, pages 968–975, 1999.

[Obst and Rollmann, 2005] Oliver Obst and Markus Rollmann. SPARK – A Generic Simulator for Physical Multiagent Simulations. *Engineering Intelligent Systems*, 13, 2005. To appear.

[Paolucci *et al.*, 2000] Massimo Paolucci, Dirk Kalp, Anandeep Pannu, Onn Shehory, and Katia P. Sycara. A planning component for retsina agents. In *ATAL '99: 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL),*, pages 147–161, Berlin, Heidelberg, New York, 2000. Springer-Verlag.

[Polani *et al.*, 2004] Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors. volume 3020 of *Lecture Notes in Artificial Intelligence*. Springer, 2004.

[Riley and Veloso, 2002] Patrick Riley and Manuela Veloso. Planning for distributed execution through use of probabilistic opponent models. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, Toulouse, France, April 2002.

[Stone and Veloso, 1999] Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 1999.

[Subrahmanian *et al.*, 2000] V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogeneous Agent Systems*. MIT Press/AAAI Press, Cambridge, MA, USA, 2000.

An Approach to Solving Multiplayer Differential Games

Johan Reimann and George Vachtsevanos
Georgia Institute of Technology
Intelligent Control Laboratory
777 Atlantic Drive. Room C441, Atlanta, GA 30332
gtg221d@mail.gatech.edu

Abstract

In this paper, an approach to solving the differential pursuit-evasion game involving multiple pursuers and a single evader is presented. To reduce the computational complexity of the problem, a simulated annealing type optimization scheme is used to arrive at a sub-optimal solution. An example is used to show the efficiency and potential shortcomings of the approach.

1 Introduction

In recent years, the problem of improving the autonomy of Unmanned Aerial Vehicles (UAVs) has received much attention. As outlined in the Department of Defense UAV Roadmap [OSD, 2002] shown in Fig. 1, the goal of the research is to enable swarms of heterogeneous UAVs to collaborate to achieve a common objective.

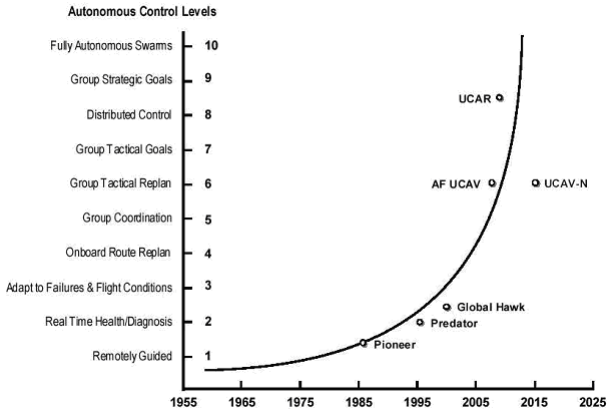


Fig. 1. Autonomous Control Level Trend

Such technology has in addition to many military applications, the potential of improving the efficiency of search and rescue operations significantly by having groups UAVs coordinate their efforts to locate and recover individuals. Moreover, swarms of UAVs can also be deployed to effectively prevent shipments of illegal merchandize to cross international borders by intercepting the transport vessels.

However, replacing human pilots is a very difficult task especially in adversarial situations when the UAVs have to react intelligently to scenario changes imposed by an intelligent opponent. An example of such a scenario is shown in Fig. 2, in which multiple ground targets located in an urban warfare environment are attempting to evade the UAVs deployed in the area. The evaders have the ability to coordinate their actions and intelligently attempt to escape the UAVs, and it is the objective of the UAVs to intercept the targets at some minimum cost.

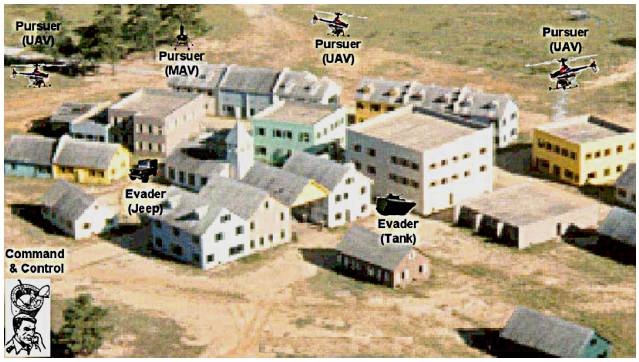


Fig. 2. Urban warfare scenario with four UAVs attempting to intercept two evading targets

The problem of having UAVs intercept evading targets, also known as the multiplayer pursuit-evasion game, is of high dimensionality and complexity. The strategy needed to intercept the evading targets is dependent not only on the dynamic properties of the UAVs and their ability to cooperate, but also on the intelligence and agility of the evading targets.

In this paper, a computational algorithm is developed to determine a suboptimal control strategy that a swarm of pursuers can utilize to intercept a single evading target. By only considering a single evading target, the problem complexity can be reduced significantly since it is possible to rewrite the game in the evading target's reference frame.

To further simplify the problem, it is assumed that the differential game considered is a so-called complete information game, that is, as the game is being executed all the players in the game are aware of the other players' current states such as position and heading.

2 Related Work

The problem of solving multiplayer pursuit-evasion differential games has been studied intensively from both a theoretical and a numerical perspective. In order to derive real-time strategies, the approach to solving multiplayer differential games has been to consider probabilistic solution methods [Sastry. et al, 2002]. The approach normally used in such solution methods is to generate a probability map of the game space, and then command the pursuers to head toward the points at which there is a high probability of finding the evaders. The advantage of such a technique is that complete information about the evaders' states is not required to find a solution. However, several advantages that a swarm of pursuers has when dealing with an intelligent evading opponent is also disregarded. The pursuing swarm has the ability to shape the probability map if the pursuers collaborate, that is, the pursuers can deploy a herding-type pursuit strategy in which the evaders are forced to commit to unfavorable evasion strategies.

An overview of solution techniques presented from a theoretical perspective is provided in [Bardi and Falcone, 1999] and [Stipanovic et al., 2004]. However, the approaches to solving the differential games in general relies either on using mathematical insight into the particular game considered, and hence is not applicable to a wide range of differential games, or a decomposition is performed by introducing multiple value functions.

In contrast to these approaches, we introduce a solution technique used to solve the complete information differential game by determining a single value function. Strategies are derived using numerical optimization schemes and will therefore include blocking and herding of the evader.

3 The Minimum Time Differential Game Problem

In this problem the vehicles are governed by the following dynamics,

$$\begin{aligned} \dot{x}_{pi} &= f_{pi}(x, u_{pi}) \\ \dot{x}_{e} &= f_{e}(x, u_{e}) \end{aligned} \tag{1}$$

where u_{pi} and u_e is the control input of pursuer *i* and the evader respectively. The functions $f_e()$ and $f_{pi}()$ are considered to be smooth but potentially nonlinear functions. The objective of the game is to intercept the evading vehicles as fast as possible, that is,

$$\min_{u_p} \max_{u_e} J\left(x, u_p, u_e, T\right) = \min_{u_p} \max_{u_e} \int_0^T 1 dt \tag{2}$$

where the parameter T is a part of the functional to be minimized. In order to determine the time-optimal trajectory, the following equality has to hold,

$$\min_{u_p} \max_{u_e} \left\langle \frac{\partial V(x)}{\partial x}, f(x, u_p, u_e) \right\rangle + 1 = 0\,, \tag{3}$$

where $f(x,u_p,u_e)=f_p(x,u_p) - f_e(x,u_e)$, and $V(x)$ is the value function. The condition expressed in (3) is the Hamilton-Jacobi-Bellman equation, which is a sufficient condition used to generate a time optimal solution given that the value function $V(x)$ is zero upon game termination and positive otherwise [Sundar and Shiller, 1996].

Finally, a termination condition will have to be introduced,

$$\Psi(x(T)) = 0\,. \tag{4}$$

For the multipursuer game with only a single evading target, the collection of possible termination points becomes quite substantial, since in many instances capture is achieved by only a small subset of the pursuers. Hence, the final position of the pursuers not directly involved in the actual interception of the evading target is not specified. However, it should be noted that even though some of the pursuers will not intercept the target, they are still able to influence the evading target's escape strategy. Consequently, the strategies of all the pursuers have to be included in the optimization process.

4 The Value Function Problem

As mentioned in the previous section, the termination condition Ψ(x(T)) = 0 does not in general reduce the possible final states of the pursuers to only a small set of points. The significance of having a large set of termination points arises when considering the value function $V(x)$. The standard approach for determining the value function is to propagate the set of termination points backwards in time. However, since the termination condition is not exclusive enough, this process proves to be very time consuming. The approach considered in this paper to reduce the problem, is to use a Guiding Value Function (GVF). The purpose of the GVF is to derive an estimate of the final state of the pursuers. Based on the estimate of the final state of the pursuers, it is possible to generate the value function associated with the time-optimal problem. However, since the final state is only an estimate arrived at using a GVF, the estimated final states are not likely to be the actual final states of the time-optimal problem. Consequently, when propagating the set of termination points backwards, a ε-neighborhood around the set of final positions is included.

The following equation is used to construct the value function:

$$V(x^o + \Delta x) = V(x^o) + \left\langle \frac{\partial V(x^0)}{\partial x}, f(x^0, u_p^0, u_e^0) \right\rangle \cdot \Delta t + o(\Delta t). \tag{5}$$

The partial derivative of the value function is approximated using a standard two-point first order approximation.

Once the value function associated with the termination

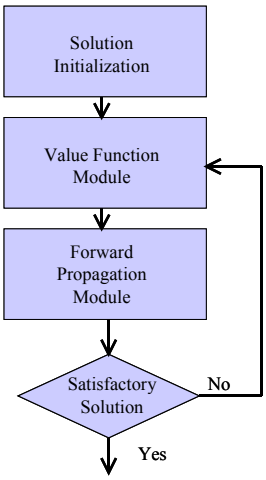


Fig. 3. Flow chart of suggested solution technique

conditions has been determined numerically, the solution is propagated forward again. If the new final states are within a δ-neighborhood of the previous solution where $\delta << \varepsilon$, the solution is considered to be a suboptimal solution and the process is terminated. Otherwise, the process will have to be repeated with the new termination points. Naturally, the number of iterations needed to arrive at an acceptable result is dependent on how well the GVF is constructed.

5 Overview of the Solution Technique

A flow chart of the algorithm is shown in Fig. 3. The initialization step provides an estimate of the final states of the players using the Guiding Value Function (GVF).
The final states of the solution is then passed to the Value Function Module, which determines the value function used to solve the problem of reaching a neighborhood around the final states in minimum time. The new value function is then passed to the Forward Propagation Module, which in turn will generate a new solution. The new solution is passed back to the Value Function Module for as long as it is not close to the previous solution.

6 Optimization Algorithm

To reduce the complexity of propagating the solution forward the search for u_p and u_e is done using a simulated annealing type search algorithm. The outline of the algorithm used is as follows:

1) Generate a random initial guess and set initial Temperature T to 1.
2) Insert guess into HJB equation and determine the value which ideally should be 0.
3) Save the value of the minimax term and the value of the HJB equation.
4) Perturb the initial guess by temperature bounded random disturbance.
5) If the new guess is better than the initial guess save the new guess and reduce temperature.
6) If the new guess is not better than the initial guess generate a random number on the interval [0,1]. If the random number is less than $e^{\frac{v_{low}-v_{high}}{T}}$, the new guess is saved where v_{high} and v_{low} are the values of the minimax term[1].
7) Repeat steps 4-7. Stop when the value does not improve after a set number of iterations or when the user terminates process.

By applying the above algorithm, the total amount of time spent searching for the pursuers' and the evader's control strategy is reduced significantly since only a small subset of the control space is searched. However, it should be noted that the solution in general is suboptimal. In addition to the ability to limit the time spent searching for a solution the algorithm, as opposed to a gradient descent type algorithm, is capable of escaping a locally optimal solution.

7 Example Problem

As an example, consider the 2-dimensional game in which three pursuers are attempting to capture one evader. The motion of each vehicle is described by the following differential equations,

$$\begin{aligned} \dot{x}_{pi} &= V_{pi} \cdot \cos(H_{pi}) \\ \dot{y}_{pi} &= V_{pi} \cdot \sin(H_{pi}) \\ \dot{H}_{pi} &= W_p \\ \dot{x}_e &= V_e \cdot \cos(H_e) \\ \dot{y}_e &= V_e \cdot \sin(H_e) \\ \dot{H}_e &= W_e \end{aligned} \tag{6}$$

where V_{pi},W_{pi},V_e and W_e are the players' controls and i = 1, 2 and 3. The termination condition will be

$$\Psi(x(T)) = (r_1^2 - 1) \cdot (r_2^2 - 1) \cdot (r_3^2 - 1) \tag{7}$$

where r_i is the distance from pursuer i to the evader. The initial value function guess will be the same as the termination condition, since it appears reasonable that the closer the pursuers are to the evader the lower the interception time becomes. It should be noted that the GVF derived from distance considerations is generally not equivalent to the time-optimal value function. The classical homicidal chauffeur problem [Isaacs, 1965] is an example

[1] The best solution is always saved separately. Hence, if the process is terminated early, the current solution, which may not be the best solution, is not returned.

of a scenario in which a pursuer has to move away from the target to be able to intercept it.
In constructing the value function a simple rectangular grid is superimposed on the game space. Linear interpolation is

Control	Bounds
V_{pmax}	0.6 distance per time unit
W_{pmax}	+/- 0.5 radians per time unit
V_{emax}	0.5 distance per time unit
W_{emax}	+/- 0.4 radians per time unit

Table 1. Control Bounds

used to determine the value of *V(x)* at the grid points.
It should be noted that the solution arrived at with infinite computational resources may only be a locally optimal solution. That is, the starting point of the algorithm arrived at by using an initial value function guess, could result in a local optimal solution due to the gradient descent type approach used to determining the time-optimal final states.

8 Results

The solution used to initialize the optimization process is shown in Fig. 4. The pursuers' trajectories are marked by o's while the evader's trajectory is marked by x's. The bounds used to limit the control are shown in Table 1.
Notice, that the players can turn very rapidly but not move very fast. The values were chosen in this fashion to show that the optimal solution to this multiplayer game is not a bang-bang type control strategy often encountered in

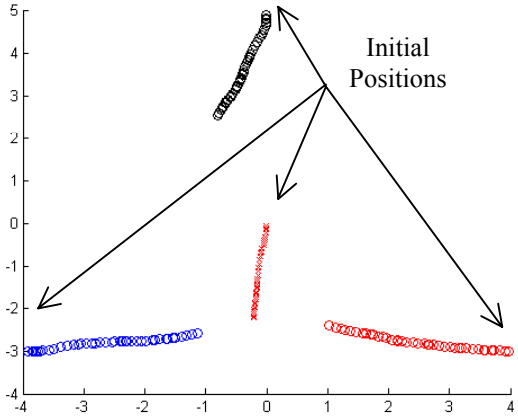


Fig. 4. Initial simulation used to determine estimated final optimal control problems, since the players do not maneuver aggressively

Clearly, the trajectories shown in Fig 4 are suboptimal, since initially the pursuers' and the evader' trajectories are wavy as if they do not know how to approach the problem.

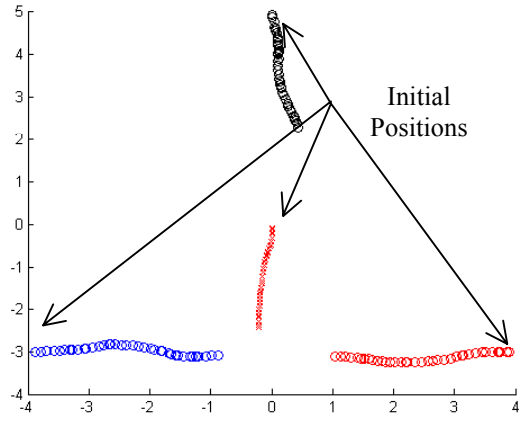


Fig. 5. Time Optimized Solution

However, once the simulated annealing algorithm has run through a couple of iterations, the near optimal solution is found. The initial solution was found within a few minutes using a Matlab routine. However, constructing the value function is much more computationally demanding due to the interdependence of the pursuers' control efforts. For instance, if at some particular step in determining the value function 100 possible states of each of the three vehicles are considered, the total number of points propagated backwards is approximately 1,000,000. Consequently, each time the value function is constructed a significant amount of processing time has to be allocated due the non-polynomial complexity of multiplayer differential games. The optimization algorithms used to simplify the problem only reduces the coefficients of the exponential growth; consequently, if a large number of players or very high dimensional resolution is required, the timeframe needed to solve the problem approaches infinity.
The solution after two iterations of the time-optimization algorithm is shown in Fig. 5. The time required to intercept the target was reduced by approximately 8%, however to obtain this improvement the algorithm had to run several hours.

9 Future Work

Since advanced optimization techniques do not appear to be able to provide a real-time implement able solution technique to the non-polynomial complexity multiplayer differential games even in simple cases, another approach relying on reducing the problem complexity from non-polynomial to polynomial appears to be required.
Such an approach will have to decouple the pursuers' strategies by possibly decomposing the problem into smaller two-player differential games with obstacles.
A possible simplification process is shown in Fig. 6. The prediction algorithm is used to estimate the cost of capturing each of the evaders. To arrive at the estimate, the multiplayer pursuit-evasion game is decomposed into multiple two-player pursuit-evasion differential games by considering all combinations of pursuers and evaders.

Once the interception cost has been determined, the pursuing players are each assigned an evader by applying a Greedy-type matching algorithm.
Finally, the interception can be executed by solving the much simpler two-player pursuit evasion problem.

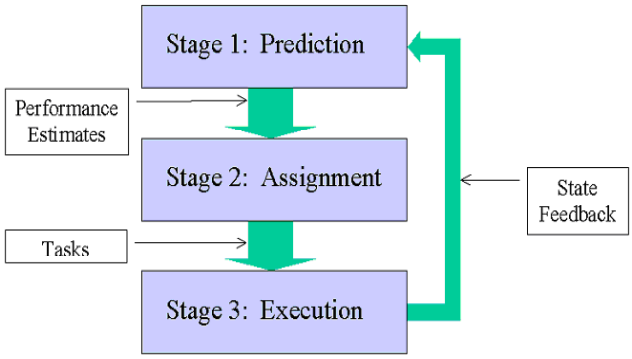


Fig. 6. The problem simplification process

Such an approach will naturally arrive at a suboptimal solution to the multiplayer differential game due to the decoupling of the pursuers control strategies, however the problem can be solved in polynomial time which is desirable when attempting to implement a solution technique in real-time. The decoupling of the pursuers' control strategies has to be done such that a level of cooperation between the pursuers is maintained, which is a problem we are currently investigating.

10 Conclusion

In this paper, a solution technique used to solve the differential pursuit-evasion game consisting of multiple pursuers and one evader is presented. A simulated annealing type solution technique is introduced to reduce the problem complexity and thereby reducing the computation time required to solve the problem significantly. A simple example is used to highlight potential complexity problems with the solution technique. Finally, a problem decomposition approach is suggested to reduce the problem to one of polynomial complexity.

References

[OSD, 2002] Office of the Secretary of Defense (Acquisition, Technology, & Logistics), Air Warfare. "OSD UAV Roadmap 2002-2027." December 2002.

[Sundar and Shiller, 1996] S. Sundar, Z. Shiller. *A Generalized Sufficient Condition for Time-Optimal Control*. ASME Journal of Dynamic Systems, Measurement and Control, Vol 118 No. 2, pp. 393-396, June 1996.

[Isaacs, 1965] R. Isaacs. *Differential Games*. Krieger, New York, 1965.

[Thomas, 1999] J.W. Thomas. *Numerical Partial Differential Equations: Conservation Laws and Elliptic Equations*. Springer-Verlag, New York, 1999.

[Bardi and Falcone, 1999] Bardi, M., Falcone, M. *Numerical methods for pursuit-evasion games in Stochastic and differential games: theory and numerical methods*. Advances in Dynamic Games. vol. 4 pp. 105-175, Birkhauser, 1999.

[Kushner, 2002] H. J. Kushner. *Numerical approximations for stochastic differential games*. SIAM J. Control & Optimization, vol. 41, no. 2, pp. 457-486, 2002.

[Hespanha and Prandini, 2002] J. Hespanha and M. Prandini. *Optimal pursuit under partial information.* Proceedings of the 10th Mediterranean Conference on Control and Automation, July 2002.

[Sastry et al., 2002] R. Vidal, O. Shakernia, H. Kim, D. Shim and S. Sastry. *Probabilistic Pursuit-Evasion Games: Theory, Implementation, and Experimental Evaluation*. IEEE Transactions on Robotics and Automation, Vol. 18, No. 5. 2002.

[Stipanovic et al., 2004] D. M. Stipanovic, Sriram, and C. J. Tomlin. *Strategies for agents in multi-player pursuit evasion games*. Proceedings of the Eleventh international Symposium on Dynamic Games and Applications, Tucson, Arizona, December 18-21, 2004.

.

Auctions for Real-Time Agent Scheduling

Patricia Maldonado* **Carlos Carrascosa** **Vicente Botti**
Universidad Polit cnica de Valencia
Departamento de Sistemas Inform ticos y Computación (DSIC)
Camino de Vera s/n 46022 Valencia Espa a
{pmaldonad, carrasco, vbotti}@dsic.upv.es

Abstract

Nowadays, artificial intelligence (*AI*) techniques are being used by real-time designers, as a means to soften their systems.

In this paper, we propose to use auctions as tasks scheduling policies of an agent working in hard real-time environments.

These techniques have been implemented in the $\mathcal{ARTIS}$ (*Architecture for Real-Time Intelligent Systems*) agent architecture because agents implemented with this architecture are able to handle hard real-time restrictions while showing agent features (believes, social conducts, ...). In this paper, we have also detailed the needed conditions and processes to apply these methods in the $\mathcal{ARTIS}$ agent.

Finally, we show the results obtained of the batteries of critical conditions tests and the comparison of these results with the ones applying the same tests to the methods used currently by the $\mathcal{ARTIS}$ agent.

1 Introduction

A multi-agent system working in a real-time environment, not only must have good working, but also must be efficient in its execution. This variable (time) will must be taken into account when implementing the tasks of the agents forming such multi-agent system.

In this paper, we propose some studies, comparatives, and implementations of negotiation methods as auctions in multi-agent systems working in real-time environments. Specifically, in the $\mathcal{ARTIS}$ agent architecture [Botti *et al.*, 1999; Carrascosa *et al.*, 2003a].

Section 2 of this paper presents a brief review of negotiations among agents. In section 3, we show two different points of view to describe $\mathcal{ARTIS}$ agents (user model and system model), along with the scheduling policies used by these agents. After that, in section 4, we describe the way negotiation between $\mathcal{ARTIS}$ agents was carried out; the purposes of each negotiation and the different implementations that have been done. In section 5, we present the different theoretical and real results obtained in the different implemented tests. Lastly, we show some conclusions and references in section 6.

2 Negotiation among agents

The application of negotiation methods to conflict-solving in multi-agent systems is becoming more common. The negotiation strategy to be followed by the agents (buyers, sellers) depends on the type of conflict existing between them. These strategies will help them to reach agreements and to obtain mutual benefits. The benefit that each partner obtains is determined by a value called *"reserved price*". The farther away or the closer the agreement is to the *"reserved price*", the higher or lower the benefit will be [Raiffa, 1982], depending on whether it is the buyer or the seller. The negotiation strategy will also specify steps that have to be followed during the negotiation.

We can find several works on negotiation among agents in the literature: [Zlotkin and Rosenschein, 1992; Weinberger and Rosenschein, 2004] identify three domain types for the negotiation between agents, these are:

- Task-Oriented Domains (TOD), where agent actions are determined by the tasks it has to carry out. In these domains, the agent has all resources it needs available for its tasks.
- Worth-Oriented Domains (WOD), where the agent assigns a value to each possible state depending on its desire to reach it, and determining its actions evaluating these values at each moment.
- State-Oriented Domains (SOD), where agent actions consider other agents present in the system, because they share plans to reach their objectives.

[Kraus *et al.*, 1998] presents a negotiation model that combines reasoning and optimization. [Kraus and Lehmann, 1995] develop their works based on 'Game Theory' using 'diplomacy' to reach agreements. [Parsons *et al.*, 1998] presents a negotiation model, among independent agents so that they reach agreements by using arguments in order to offer or obtain certain services.

Other approaches to negotiation are auction-based methods. Auction is a useful choice when there are many agents

*Auspices for Universidad de Magallanes – Punta Arenas (Chile)

interested in limited resources. There are, mainly, two different kind of auctions: *"one-to-many"* and *"many-to-many"*. In both cases, it is necessary to determine beforehand the protocol type that will be used (for example, FIPA protocols [FIPASpec, FIPA]). Usually, in *"one-to-many"* auctions agents don't use a mediator, whilst in "*many-to-many*" auctions they use it. The auctions protocols used by agents are First-Price Sealed-Bid Auctions, English Auctions and Dutch Auctions.

In this work, we have applied "*one-to-many*" auctions among $\mathcal{ARTIS}$ agent entities. Before explaining the negotiation, we will present the $\mathcal{ARTIS}$ agent architecture.

3 $\mathcal{ARTIS}$ Agent

$\mathcal{ARTIS}$ is the acronym of an Architecture for Real-Time Intelligent Systems [García-Fornes, 1996]. The main feature of this agent architecture is to guarantee the fulfillment of temporal restrictions for critical components and to support the execution of optional components, both kinds of components are user-defined.

An $\mathcal{ARTIS}$ *Agent* ($\mathcal{AA}$) is able to perceive information from the environment in which it is situated, to calculate fast answers (and to refine them), and finally to act over the environment. These actions can be physical actions or message passing. The architecture of an $\mathcal{AA}$ can be viewed from two different perspectives[Terrasa *et al.*, 2002; Carrascosa *et al.*, 2003a]: the *user model* (high-level model) and the *system model* (low-level model). The user model offers the developer's view of the architecture, while the system model is the execution framework used to construct the final executable version of the agent.

3.1 User Model

The *User model* is a high-level model in which the $\mathcal{AA}$ is composed of sub-entities that model its behaviours, environment, etc. These are:

1. A group of *sensors* and *effectors* allowing the agent to interact with the environment with time restrictions (demand for *Real-Time Environments*).
2. A group of *behaviours*, each one composed by a group of in-agents. Each in-agent solves a part of the problem of the $\mathcal{AA}$, activating itself in a periodic way so that all of them cooperate to solve the whole problem. There are two kinds of in-agents:
   Critic in-agents which are characterized by a period and a deadline. Its execution is guaranteed during execution time. They have two layers: the *reflex layer* that ensures an answer to the problem of the $\mathcal{AA}$ with the minimum quality in a guaranteed execution time; and a *real-time deliberative layer* which tries to improve the quality of the answer reached by the reflex layer.
   Non-critic in-agents which are part of the $\mathcal{AA}$'s deliberation. This kind of in-agents don't have their execution guaranteed, but the agent tries to execute as many non-critic in-agents as possible with the aim of maximizing the global quality of the solution to its problem.
3. Set of *believes* forming a mental state of the in-agent including: a *model of the world* and its *internal state*.
4. A *control module* which is responsible of the execution of the in-agents forming the current behaviour of the $\mathcal{AA}$. Since the in-agents have two layers, reactive and deliberative, to manage them the control module is divided into two sub-modules: *Reflex Server* (*RS*) and *Deliberative Server* (*DS*).

3.2 System Model

The *System Model* is the low-level model of the $\mathcal{AA}$. This model is the translation of the $\mathcal{AA}$'s user model, so every part is translated into an equivalent low-level entity in the $\mathcal{AA}$'s system model (see figure 1):

1. The *sensors* and *effectors* of the user model correspond with a library in order to access to the different hardware devices.
2. Each behaviour is translated into a working mode [Carrascosa *et al.*, 2004], and their in-agents into low-level tasks.
   (a) In the system model. This way, every task can have three parts:
      i. An *initial part*. This is the reflex part of the in-agent; it must always be executed obtaining a first reflex answer to the problem of the $\mathcal{AA}$ with a low quality. This initial part includes the perception part of the in-agent.
      ii. An *optional part*. This is the deliberative part of the in-agent. These optional components increase the quality of the answer calculated in the initial part establishing the cognitive process of the in-agent. For this, artificial intelligence techniques are used and are executed between the initial and final parts of the correspondent in-agent.
      iii. A *final part*. This part executes the answer which was generated in the previous parts (*initial* and *optional* parts) of the in-agent. This part is in charge of the actions of the in-agent.
3. The set of *believes* is translated into a shared memory (frame-based blackboard) that is accessible from all the tasks [Barber *et al.*, 1994].
4. The two parts of the *Control Module* of the user model are translated into:
   (a) The *Reflex Server:* includes *First-Level Scheduler* (*FLS*) for the real-time tasks. The *FLS* uses real-time policies at execution time to decide what task to execute at every moment. This planning helps the $\mathcal{AA}$ to adapt to the changes of its environment, and how to execute the tasks using less time than the estimated for its worst-case execution.
   (b) The *Deliberative Server* is formed from two sub-modules: the *Event Manager* (*EM*) and the *Second-Level Scheduling* (*SLS*). The *EM* activates when receiving events and reacts to them; the *SLS* is in charge of distributing the available slack time among the optional components of the in-agent. This improves the global quality of the agent's answer. To do this distribution, the *SLS* needs to know

the *mean-case execution time* (*mcet*) of the deliberative parts (optional components) of the in-agents. Currently the $\mathcal{ARTIS}$ agents use the following policies:

- *EDF (Earliest Deadline First)*. It chooses those tasks that with lowest deadlines.
- *HSF (High Slope First)*. It tries to get the biggest possible quality, by selecting first those tasks with smallest execution time.
- *BIF (Best Importance First)*. It executes first the most important tasks of the system.

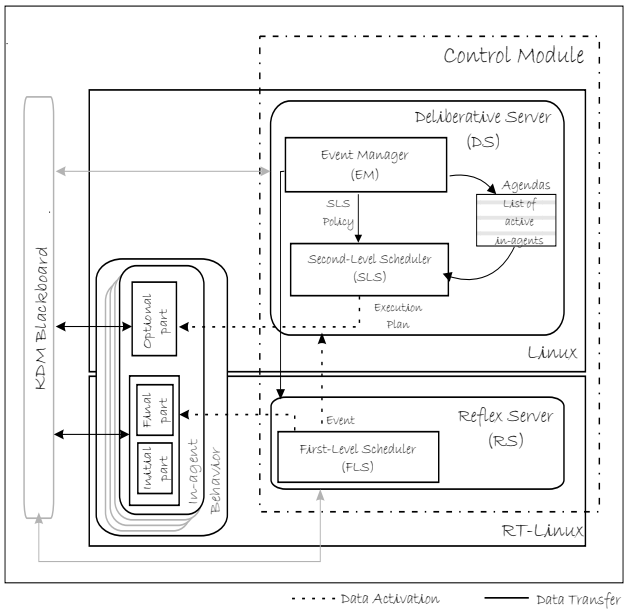


Figure 1: System Model of the $\mathcal{AA}$

Previous to the work here presented, *SLS* is the only one in charge of selecting the active in-agent's optional parts to execute.

We propose to extend the *SLS* whit the capacity of using auctions as negotiation techniques with in-agents about their possible execution.

So, we try to soften how the *SLS* selects and schedules the optional tasks by means of including on these processes auctions taking into account the agent current situation, believes and desires.

4 Auctions in the $\mathcal{ARTIS}$ Agents

In order to implement these auctions we consider the time-restrictions fundamentals in an architecture of an *Artificial Intelligence System in Real-time* (*AIS-RT* – [Musliner *et al.*, 1995; Terrasa *et al.*, 2002]) such as $\mathcal{ARTIS}$. Due to these restrictions, the duration of the auctions and of the offerings proposals creation by the involved in-agents must be limited.

Recalling the structure of $\mathcal{AA}$ (section 3), the deliberative part of the agent is represented by the deliberative parts of the

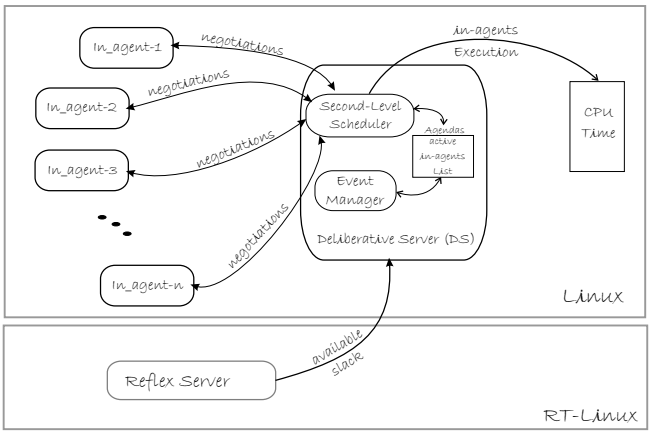


Figure 2: Negotiation in $\mathcal{ARTIS}$*Agent* architecture.

in-agents that form the current behaviour of the agent. So, these parts are translated into optional components which are dealt on the *DS*. Once the *EM* generates the list of active in-agents, the *SLS* distributes the time to execute the optional parts of the in-agents. The *SLS* must decide which in-agent to execute so that the best possible solutions are produced. On the other hand, there are the $\mathcal{AA}$ entities, in-agents, which have the answers to their problem and each of them is associated to a pre-determined quality and execution times.

The goal of our auctions is to allow both parts, *SLS* and in-agents (see Figure 2), reach their goals which are compatible with the goals of the system they belong to, $\mathcal{AA}$. In this way, the *SLS* will be the seller and the active in-agents will be the buyers of the available time (or slack) that the *SLS* has for executing optional parts.

We have decided to use auction for all the possible negotiation processes due to the following features of our problem:

- The negotiation process must be time-limited. It will depend on the available slack.
- The best available answer must be produced in this limited time, so that the *SLS* is able to execute something.
- The best answer is selected from a group of participant offers from the active in-agents. The offer is their contribution to the global agent quality.

So, the *SLS* must consider the following to decide what in-agent to execute in every moment:

- The quality of the answers that are offered by each one of the active in-agent.
- The time of calculations that is estimated in each in-agent to obtain its solution.
- The importance that each in-agent has assigned.

To summarize, there is a resource in conflict which is the *CPU time* that must be distributed by the *SLS* under the previously mentioned conditions; and there are many clients, the active in-agents, that desire to acquire this resource. However, this process of awarding will have to be fast and advantageous for both parts. Considering all the exposed so far, the best option to this domain is to use auctions as negotiation techniques.

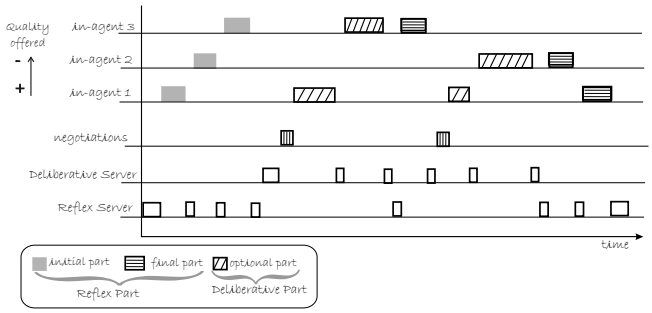


Figure 3: Insertion of Negotiation's tasks in $\mathcal{ARTIS}$ *Agent* architecture

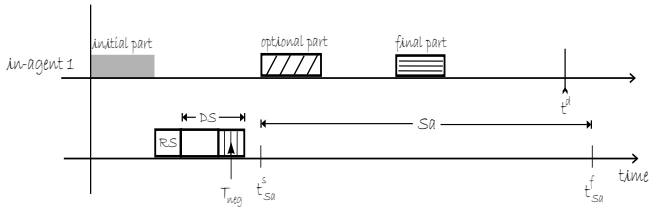


Figure 4: Task's time measures

Since the aim of auctions is to award the product to the participant giving the best offer based on the pre-determined conditions. Moreover, in this case, the time restrictions must be considered according to the $\mathcal{AA}$ characteristics.

We analyzed two kinds of auctions

1. *"Voracious auctions"*: They are those auctions in which the participants do not know who will be able to execute until the destined time to the auction finishes. Until then, the participants will have to compete to give the best offer. The English Auction (*EA*) and the First Sealed-bid Auction (*FSA*) are two examples of this kind of auctions.
2. *"Semi-voracious auction"*: In this kind of auctions, the participants determine whether they accept or not the proposed offers by the seller (*SLS*). In this way, the participants can control the offers that are more appealing to them. The Dutch Auction (*DA*) is of this kind of auctions

In the following sub-sections we explain the implementation of these auctions in $\mathcal{ARTIS}$ agents.

4.1 Implementation

The deliberative process begins when the *RS* sends a message to the *DS* indicating that it can execute the deliberative parts of active *in-agent* (see figure 2). These messages are received by the *EM*, that selects and orders the active parts, giving the definitive list to the *SLS*.

The in-agents will be able to participate in the auctions if they fulfill the following conditions (Figure 4):

- Their final part have not still finished.
- They have not executed all their optional parts.
- Their mean-case execution time ($mcet$) to generate an answer must be smaller or equal to the available slack time (Sa): $mcet \leq Sa$.
- Their next deadline expiration (t^d) must be before the end of the auctioned time (slack time) (t^f_{Sa}): $t^d \leq t^f_{Sa}$

Before beginning the auction, the *SLS* sends the following information to all the participating in-agents:

- Sa: (available slack time) The total available CPU time for the execution of the optional parts.
- t^s_{Sa}: Start time of the available slack.
- T^{offer}_{max}: End time to generate offers, because $\mathcal{ARTIS}$ is an architecture for Real-Time Systems (inflexible condition), $T^{offer}_{max} \ll T_{neg}$, where T_{neg} is the total time assigned for *DS* to the auctions.
- Depending on the auction protocol used, the *SLS* sends the best offer received from the participants up to that point.

These auction implemented in $\mathcal{ARTIS}$ *Agent* use the protocols proposed by FIPA [FIPASpec, FIPA] considering the restrictions of time imposed by the *RTS* (Figure 5 and Figure 6).

Voracious Auctions

In this kind of auctions, the participant in-agents try to get the biggest amount of slack time to run themselves.

The auction begins with the call for proposals of the second-level scheduler (*SLS*). In this call, the *SLS* asks the active in-agents for bids. Participants will have a limited time to generate and to send these bids. These bids of the participant in-agents must include: the offering quality and the estimated execution time needed to get this quality. The *SLS* gets all the generated bids from the participants and selects the best one.

We have lightly modificated this process to be used in First-Sealed Auctions: once the *SLS* has received all the bids from the participant in-agents, it proceeds to share out all the available slack between the best bids it has received.

To implement the English Auction, the *SLS* assigns the amount of slack needed by the winner in-agent and, if there is spare slack time, it repeats the call for proposals for this new available slack. This process is repeated until one of these conditions is fulfilled:

- The time the *SLS* assigned to the auction process is reached.
- The *SLS* runs out of available slack.
- There is not any interested in-agent.

Then, the *SLS* module calls to active *in-agents* offering proposals by slack available to execute its optional parts and begins the auction (see figure 3). Once finished the auction, the *SLS* sends to execution the winning in-agents (optional parts of these in-agents). Thus, the *SLS* determines which optional parts are going to be executed in the available slack time.

For our auctions, the *participating in-agents* will receive and evaluate the information according to their beliefs, limitations and characteristics.

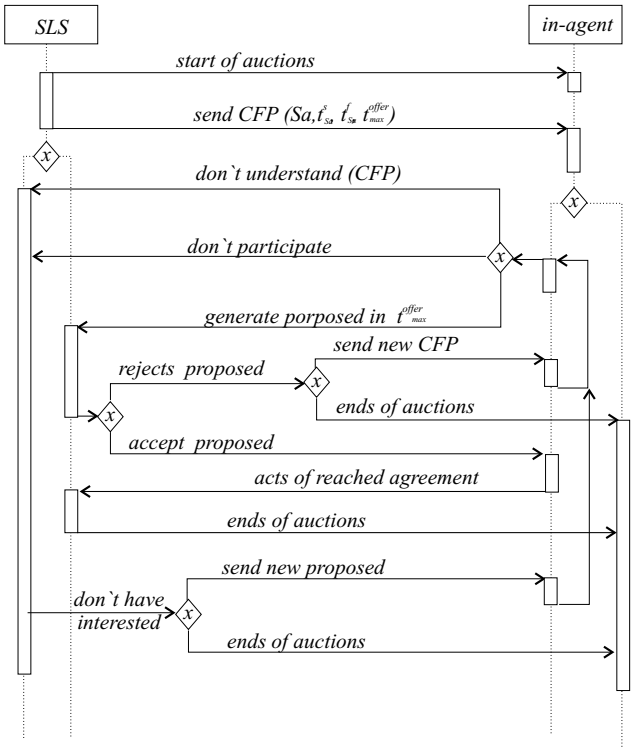


Figure 5: *Voracious* Auctions Protocol

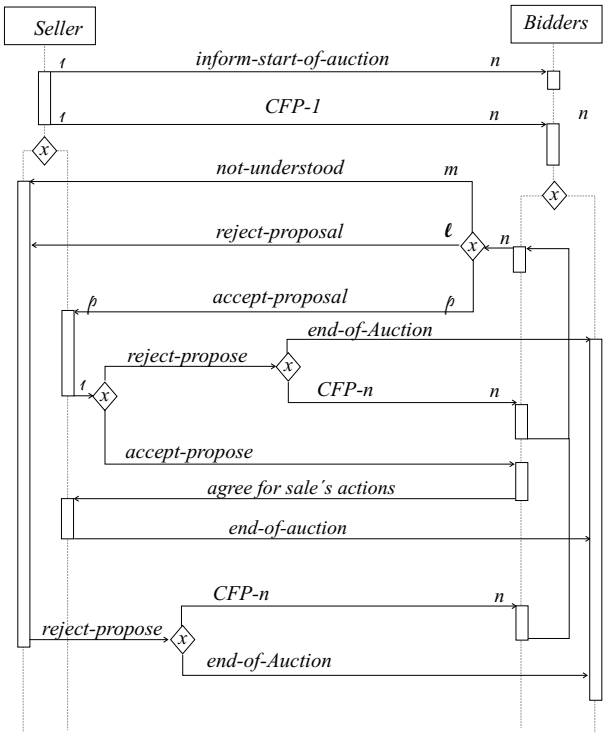


Figure 6: *Semi-Voracious* Auctions Protocol

For the *English Auctions* and the *First-Price Sealed Bid Auctions*, the *participating in-agents* generate offers in order to execute their optional parts. In the *English Auction*, the offers are increased in time until there are no more offers or auction time ends. In both kind of auctions, the *in-agent* offers are directly proportional to the following conditions:

- The *participating in-agent* has not executed any of its optional parts.
- $mcet_k \leq Sa\,; \forall k \in optional\_parts$
  The *mean-case execution time* (*mcet*) for its optional parts is smaller or equal than the time auctioned by the *SLS*, Sa (slack available for optional tasks executions).
- $t_d^i < t_{Sa}^f$
  The next deadline of *in-agent* i, t_d^i, expires before the auctioned space ends, t_{Sa}^f.
- $Q_i^{t-1} < Q_i^t$
  The answer quality offered by the *participating in-agent* i, Q_i^t, increases the previous quality offer, Q_i^{t-1}.

So, the functions that generate the offer of *in-agent* i in order to execute its optional part k are shown: for *English Auction* in Equation(1) and for *First-Price Bid Auction* in Equation(2).

$$F_{ik}(x) = \left[\frac{x}{(t_d^i - t_{Sa}^s) * [t_d^i - (mcet_{ik} + t_{Sa}^s)]} * Q_{ik}\right] + MP \quad (1)$$

$$Oferr_{ik} = \frac{x}{(t_d^i - t_{Sa}^s) * [t_d^i - (mcet_{ik} + t_{Sa}^s)]} * Q_{ik} \quad (2)$$

Where:

- t_{Sa}^s : the start instant of the auctioned space.
- t_d^i : the expiration instant of the next deadline for *in-agent* i ($t_d^i < t_{Sa}^f$).
- $mcet_{ik}$: the *mean-case execution time* of *in-agent* i to execute its respective optional part k.
- MP : The best offer received up to that point. MP starts at 0 in Equation (1).
- x : the number of iterations for the English Auction ($x \in [1...n]$) in Equation (1).
- Q_{ik} : the quality that *in-agent* i offers to execute its optional part k.

Semi-Voracious Auctions

Auction begins when the *SLS* calculates and sends a proposal (*SLSWeight*) to all active in-agents. The in-agents evaluate this bid according to their believes, characteristics, temporal restrictions and abilities. If there is still enough slack, and time to continue with negotiations, the *SLS* calculates again a *SLSWeight* for the new slack.

In *Dutch Auction*, the *SLS* makes an offer to the *participating in-agents* for available slack which we call *SLSweight* (calculated according to Equation 3). The *SLS* begins by requesting a high answer quality for the $\mathcal{AA}$ problem which we call *expected quality*, QE_{Sa}. The *in-agents* evaluate the benefits of the *SLSweight* sent by the *SLS*. If there is no interest, the *SLS* decreases the *SLSweight* for available slack by decreasing the QE_{Sa} . The *reduction ratio* of the QE_{Sa} is

determined by a function that has a negative slope without reaching zero percent. This process repeats until some *participating in-agent* accepts the *SLS* conditions or the *SLS* expected quality (QE_{Sa}) reaches the minimum value (or which is $QE_{Sa} = 20\%$).

The equations for *Dutch Auction* are expressed to follow.

$$SLSweight(t) = e^{(-ln(t+Sa))} * QE_{Sa} \quad (3)$$

Where:

- t : the point in time where the *SLSweight* is calculated.
- Sa : the size of available slack for the execution of the *in-agent*'s optional parts.
- QE_{Sa} : the quality expected by the *SLS* from the *participating in-agents* for the $\mathcal{AA}$ problem solution.

The equation to evaluate the benefits of each *participating in-agent* is:

$$0 \leq \left(\frac{QE_{Sa}}{QA_i} \right) \leq 1 \quad (4)$$

Where:

- QE_{Sa} : the quality expected by the *SLS* from the *participating in-agents* for the $\mathcal{AA}$ problem solution.
- QA_i : The accumulated quality offered by the *participating in-agent* for the solution to the $\mathcal{AA}$ problem.

5 Experimental Tests

This section presents the experimental test that have been made to establish the feasibility of implementing auctions as $\mathcal{ARTIS}$ agent acritical tasks' scheduling policies.

We have made both simulated and real execution tests. We have used the same data to make both kind of test, allowing to compare both results.

Due to its real-time performance, $\mathcal{ARTIS}$ agent architecture works over real-time operating system, *RT-Linux*. Nevertheless, the $\mathcal{AA}$'s deliberative layer works over Linux operating system. As we have previously presented our auctions are part of this deliberative process.

In this way, the *SLS* assigns (*on-line*) a percentage of total available slack time to auctions executions. This percentage is known and will depend on the kind of auctions executed, therefore it does not influence in the general $\mathcal{ARTIS}$ planning (see previous sections).

We have defined some restrictions in the simulation tests to match hardware limitations existing in corresponding real tests. Each test execution lasts until their tasks *hyper-period*[1], because this is the minimum time after which the execution sequence is repeated.

For both kind of tests realized, we generate battery tests with the following common specifications:

- Three, six, nine or twelve in-agents per $\mathcal{AA}$.
- Tasks time restrictions obtained using probability functions proposed in [Campos and García, 2002].
- We have used two different initial situations regarding the in-agents time-features values:
  - The one with deadlines equals to periods.
  - The one with deadlines lower than periods.
- For each previous situations are repeated for different period values: 20000, 40000, 80000, 160000, 2560000 (*milliseconds*).

For simulation tests, we have used the tool of simulation InSiDE [Julián *et al.*, 2004] simulation toolkit with our negotiation techniques. To execute an $\mathcal{AA}$ in InSiDE for simulation tests is necessary to enter all its data such as: time restrictions, behaviours, believes, the system load, etc. In our previous work [Maldonado *et al.*, 2005], we have shown these specifications and obtained some results.

For real tests, we implemented the same scenarios used in simulation tests directly in $\mathcal{ARTIS}$ *agent* architecture.

In order to compare the results obtained in the simulation tests with the results obtained in the real tests, we will use the final quality that was obtained in the answer to the problem of the $\mathcal{AA}$ which is called *Real Relative Quality* (*RRQ*, that is detailed in [Hernández *et al.*, 2003]) and it is represented as $RRQ = \frac{ORQ}{IQ}$, where *ORQ* (*Obtained Real Quality*) is the quality reached by the in-agents of the $\mathcal{AA}$ and *IQ* (*Ideal Quality*) is the quality offered by the $\mathcal{AA}$.

5.1 Obtained Results

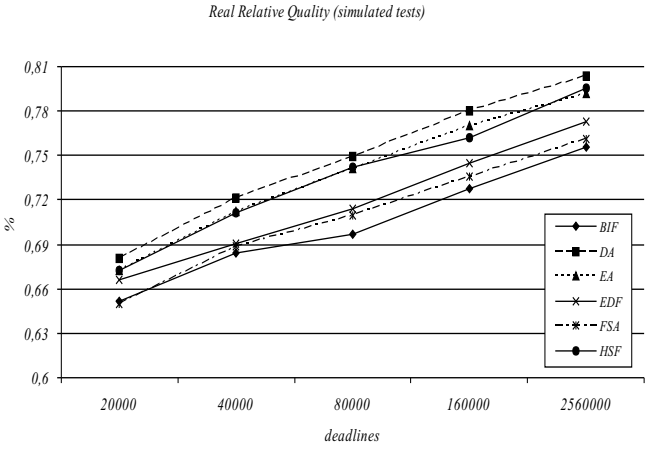


Figure 7: Simulate tests for deadlines equal to periods

The obtained results on the realized tests according to the specification mentioned before are showed in the figures 7, 8, 9 and 10. It can be observed from these graphics that the obtained results in real tests are very similar to the obtained ones on simulations. However, all the policies obtain better qualities than the obtained on simulated tests. The main difference is that on simulations the system got over-saturated. This was done because the optional load influences directly on the executions of our policies since these are executed on the system's slack.

[1] "*The tasks will be released together again at the least common multiple of the periods of the tasks.*"[Bernat *et al.*, 2001]

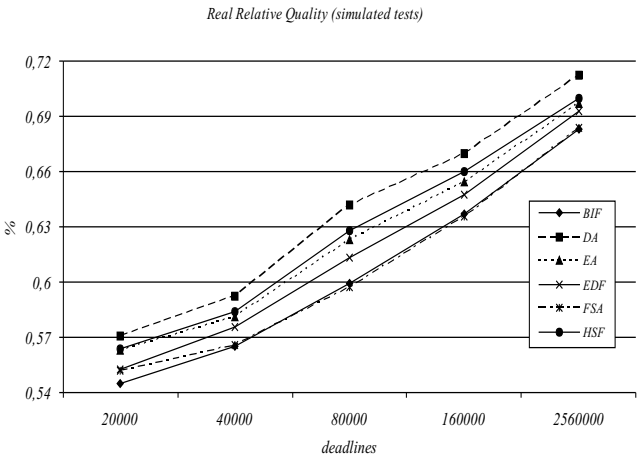


Figure 8: Simulate tests for deadlines minor or equal to periods

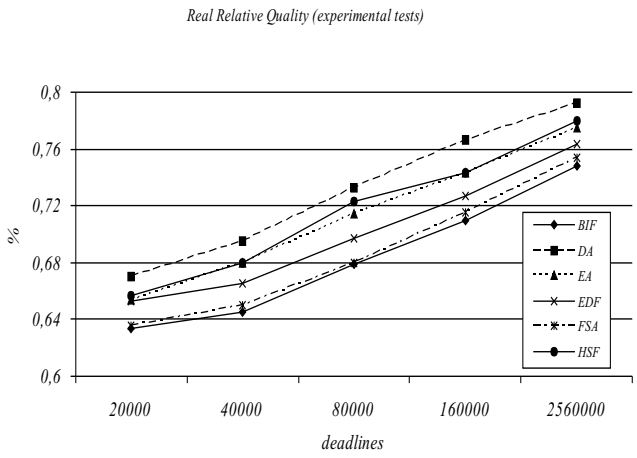


Figure 9: Real tests for deadlines equal to periods

This indicates that it is feasible to use auctions as $\mathcal{ARTIS}$ agent optional tasks' scheduling policies.

6 Conclusions and Future Works

As we have detailed, the main purpose of the work presented here was to study the usage of auctions as methods to solve the problems of intelligent agents that work on real-time environments, specifically on the $\mathcal{ARTIS}$ architecture. Considering the obtained results (simulation and real tests) we proposed the real viability of introducing these techniques in the $\mathcal{ARTIS}$ agent architecture.

It can also be observed in the graphics that our auctions obtain the best quality of all $\mathcal{ARTIS}$ policies. In this way, the implemented techniques showed in this paper produce an improvement with respect to the existing methods. This good results helps to think about the application of these methods

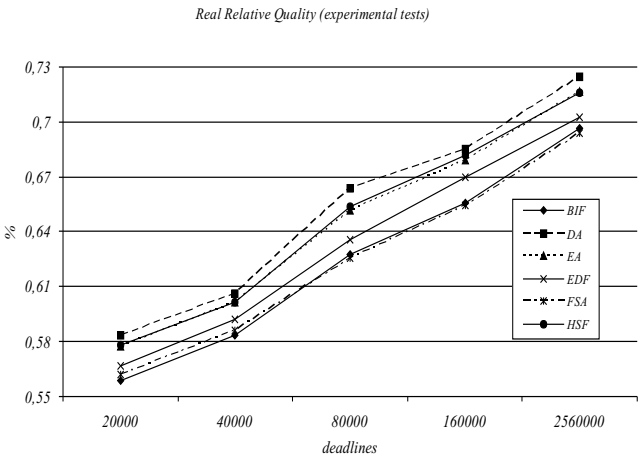


Figure 10: Real tests for deadlines minor or equal to periods

to schedule real-time systems.

On the other hand, the results obtained on the practice matched to the obtained on the simulation in which the real quality obtained decreases almost *10%* when the deadline is minor than the period.

The implemented auction techniques (explained in this paper) along with the scheduling policies existing before this work, form a solid study battery for current and future implementations about task scheduling in intelligent agents.

Finally, based on the obtained results, the future tasks will be:

- To use the obtained simulation results to identify the most suitable situations (environment and internal state) for each scheduling policies, so that the $\mathcal{AA}$ can be programmed to adapt to this situation changing its current policy to the most suitable one [Casamayor, 2003].
- To orientate the auctions toward more deliberative methods that involve the planning of all the available slack in the whole application.
- To generalize the methods here presented to be used in Multi-$\mathcal{AA}$ Systems (SIMBA)[Carrascosa *et al.*, 2003b].

References

[Barber *et al.*, 1994] Federico Barber, Vicente Botti, Eva Onaindía, and Alfons Crespo. Temporal reasonig in reakt: An environment for real-time knowledge-based systems. In *AICOMM*, number 7, pages 175–202, 1994.

[Bernat *et al.*, 2001] Guillem Bernat, Alan Burns, and Albert Llamosí. Weakly hard real-time systems. *IEEE Transaction on Computers*, 50(4):308–321, April 2001.

[Botti *et al.*, 1999] V. Botti, C. Carrascosa, V. Julian, and J. Soler. Modelling agents in hard real-time environments. In *MAAMAW'99 Proceedings*, volume 1647 of *LNAI*, pages 63–76. Springer-Verlag, 1999.

[Campos and García, 2002] Antonio M. Campos and D.F. García. A real-time expert system architecture based on

a novel dynamic task scheduling technique. In *IEEE Int. Conference on Industrial Electronics, Control and Instrumentation, IECON'02*, pages 1893–1898, 2002.

[Carrascosa *et al.*, 2003a] Carlos Carrascosa, Miguel Rebollo, Vicente Julián, and Vicente Botti. Deliberative server for real-time agents. In *The 3rd International/Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691. Multi-Agent Systems and Applications III, Springer-Verlag, 2003.

[Carrascosa *et al.*, 2003b] Carlos Carrascosa, Miguel Rebollo, José Soler, Vicente Julián, and Vicente Botti. SIMBA architecture for social real-time domains. In *EUMASS 2003: The First European Workshop on Multi-Agent System*, 2003.

[Carrascosa *et al.*, 2004] Carlos Carrascosa, Andrés Terrasa, José Fabregat, and Vicente Botti. Behaviour management in real-time agents. In *IberAgents*, 2004.

[Casamayor, 2003] Carlos Carrascosa Casamayor. *Meta-Razonamiento en Agente con Restricciones Temporales Críticas*. PhD thesis, Universidad Politécnica de Valencia, Valencia – España, 2003.

[FIPASpec, FIPA] FIPASpec. Fipa specifications. Foundation for Intelligence Phisycal Agents, 2000. http://www.fipa.org/specifications/index.html, FIPA.

[García-Fornes, 1996] Ana García-Fornes. *ARTIS: Un modelo y una arquitectura para sistemas de tiempo real inteligentes*. PhD thesis, Universidad Politénica de Valencia, Valencia – España, 1996.

[Hernández *et al.*, 2003] Luis Hernández, Vicente Botti, Ana García-Fornes, and Mario González. A quality-based heuristic for real-time scheduling. *Artificial Intelligence Research and Development.Frontiers in Artificial Intelligence Research and Development.*, 100:462–473, 2003.

[Julián *et al.*, 2004] Vicente Julián, M.C. Moncho, and Vicente Botti. Real-time multi-agent system development and implementation. In *Frontiers in Artificial Intelligence and Applications*, volume 113, pages 333–340, Ourense, España, 2004.

[Kraus and Lehmann, 1995] Sarit Kraus and Daniel Lehmann. Designing and building a negotiating automated agent. *Computational Intelligence*, 11(1):132–171, 1995.

[Kraus *et al.*, 1998] Sarit Kraus, Katia Sycara, and Evanchik. Argumentation in negotiation: A formal model and implementation. *Artificial Intelligence*, 104(1-2):1–69, 1998.

[Maldonado *et al.*, 2005] Patricia Maldonado, Carlos Carrascosa, and Vicente Botti. Negotiation in real-time multi-agent systems. In *IADIS International Conference – Applied Computing 2005*, volume II, pages 247–254, Algarve, Portugal, 2005. isbn: 972-99353-6-X.

[Musliner *et al.*, 1995] David John Musliner, James Hendler, Ashok Agrawala, Edmund Durfee, Jay Strosnider, and C.J. Paul. The challenges of real-time ai. *IEEE Computer*, 28(1), 1995.

[Parsons *et al.*, 1998] Simon Parsons, Carles Sierra, and Nick R. Jennings. Agents that reason and negotiate by arguing. *Logic and Computation*, 8(3):261–292, 1998.

[Raiffa, 1982] Howard Raiffa. *The Art and Science of Negotiation*. Harvard University, Cambridge, USA, 1982.

[Terrasa *et al.*, 2002] Andrés Terrasa, Ana García-Fornes, and Vicente Botti. Flexible real-time linux. *Real-Time Systems Journal*, (2):149–170, 2002.

[Weinberger and Rosenschein, 2004] Yair B. Weinberger and Jeffrey S. Rosenschein. Passive threats among agents in state oriented domains. In *The Sixteenth European Conference on Artificial Intelligence*, pages 89–93, Valencia, Spain, August 2004.

[Zlotkin and Rosenschein, 1992] Gilad Zlotkin and Jeffrey S. Rosenschein. A domain theory for task oriented negotiation. In *Proceeding of the American Association of Artificial Intelligence*, pages 148–153, San Jose, CA, 1992.

Tailoring Action Parameterizations to Their Task Contexts*

Freek Stulp and Michael Beetz
Intelligent Autonomous Systems Group, Technische Universität München
Boltzmannstrasse 3, D-85747 Munich, Germany
{stulp,beetz}@in.tum.de

Abstract

Solving complex tasks successfully and efficiently not only depends on *what* you do, but also *how* you do it. Different task contexts have different performance measures, and thus require different ways of executing an action to optimize performance. Simply adding new actions that are tailored to perform well within a specific task context makes planning or action selection programming more difficult, as generality and adaptivity is lost. Rather, existing actions should be parametrized such that they optimize the task-specific performance measure.

In this paper we propose a novel computation model for the execution of abstract action chains. In this computation model, a robot first learns situation-specific performance models of abstract actions. It then uses these models to automatically specialize the abstract actions for their execution in a given action chain. This specialization results in refined chains that are optimized for performance. As a side effect this behavior optimization also appears to produce action chains with seamless transitions between actions.

1 Introduction

State-of-the-Art autonomous robot controllers capable of solving a large spectrum of complex tasks are typically equipped with libraries of actions implemented by control routines. The controllers then dynamically combine and partially parameterize these actions on the fly in order to solve the respective set of active tasks.

Consider, for example, the controllers for autonomous soccer robots. These controllers are provided with actions for navigating, kicking, searching, etc. During the game, the controllers dynamically select these actions to perform their immediate tasks. For example, they navigate to the ball in order to get possession of it, or to clear a dangerous situation. In another task context, they navigate in order to dribble the ball towards the opponent's goal. As a consequence, the use of actions in different task contexts require the designer to reason about how the implemented action will perform in these contexts. On the one hand, programmers want to implement the navigation action as fast as possible to be more agile and mobile than the opponents. Unfortunately, fast navigation behavior will cause more frequent and harder collisions with the ball when approaching it and thereby the robot will loose control of the ball. Even worse, while these hard collisions are to be avoided when gaining control of the ball and dribbling, they are often desirable in other task contexts such as clearing a dangerous situation.

Most robot controllers deal with task contexts by providing variants of actions for the different task contexts. A soccer robot programmer provides, instead of a single navigation action, a set of navigation actions such as: `clearBall`, `approachBall`, `dribbleBall`, `interceptBall`, and `blockOpponent`. In the design of the action libraries, most programmers consider a trade-off between the compactness of the action library and its performance. And they are typically willing to sacrifice compactness for performance.

However, having only few abstract actions instead of many specific actions has several advantages. Fewer actions need to be implemented because viewed at an abstract level the actions are applicable to a broader range of situations. At more abstract levels the search space of plans is substantially smaller and fewer interactions between actions need to be considered. This not only eases the job of the programmers but also the computational task of automatic planning systems. Having fewer actions also makes the system more adaptive. Suppose the robots play on a new field on which the dynamics of the robots are very different, and all navigation actions perform badly. If there are many navigation routines, they all have to be retuned, rewritten or relearned to perform well in the new situation. The fewer actions there are, the faster this can be done, and the more adaptive the system is.

In this paper we propose a novel computational model for autonomous robot control that allows the control system to use small sets of general and abstract actions while at the same time achieving the performance of large sets of specialized actions. The computational model performs execution time and context-specific optimization of action plans using learned performance models of the general actions. The basic idea of our approach is to learn performance models of abstract actions off-line from observed experience. These performance models are rules that predict the situation- and parameterization-specific performance of

*The work described in this paper was partially funded by the Deutsche Forschungsgemeinschaft in the SPP-1125.

abstract actions, e.g. the expected duration. Then, at execution time, our system determines the set of parameters that are not set by the plan and therefore define the possible action executions. It then determines for each abstract action the parameterization such that the predicted performance of the action chain is optimal.

In this paper, we investigate two mechanisms for execution time and context specific action specialization:

1. Specialization of general actions for their improved execution within given action chains.
2. Specialization of actions for predictive failure prevention through subgoal assertion.

The technical contributions of this paper are fourfold.

1. We propose a novel computational model for the execution time optimization and generation of action chains (section 2).
2. We show how situation-specific performance models for abstract actions can be learned automatically, (section 3).
3. We describe a mechanism for subgoal (post-condition) refinement for action chain optimization. We apply our implemented computational model to chains of navigation plans with different objectives and constraints and different task contexts (section 4).
4. We show how performance models can be used to determine when no action can solve the task, and subgoals must be introduced to achieve the goal (section 5).

2 System overview

This section introduces the basic concepts upon which we base our computational model of action chain optimization. Using these concepts, we define the computational task and sketch the key ideas for its solution. First of all, we will describe two exemplary scenarios that clarify the problem.

2.1 Two exemplary scenarios

In Figure 1, a typical situation from robotic soccer is shown. The robot's goal is to score a goal. A three step plan suffices to solve this task: 1) go to the ball; 2) dribble the ball to shooting position; 3) kick. If the robot naively executed the first action (as depicted in Figure 1a), it might arrive at the ball with the goal at its back. This is an unfortunate position from which to start dribbling towards the goal. The problem is that in the abstract view of the planner or programmer, being at the ball is considered sufficient for dribbling the ball and the dynamical state of the robot arriving at the ball is considered to be irrelevant for the dribbling action.

What we would like the robot to do instead is to go to the ball *in order* to dribble it towards the goal afterwards. The robot should, as depicted in the Figure 1b, perform the first action sub-optimally in order to achieve a much better position for executing the second plan step. This behavior could be achieved by designing a new action, e.g. `goToPoseInOrderToDribbleTheBallToX`, that takes into account that we plan to dribble the ball to a certain position afterwards. Its long name already indicates the loss of generality, and it is also not guaranteed that this action

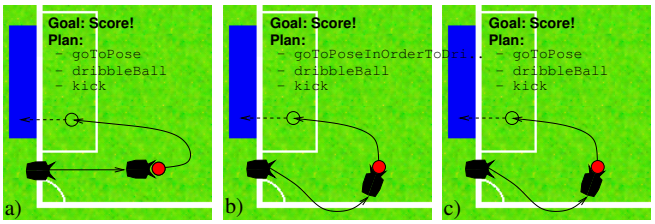


Figure 1: Three alternative plan executions to approach the ball in order to dribble it.

provides the optimal position from which to start dribbling. Preferably, an existing action should be parameterized such that it performs well with respect to the performance measure of the given context. Again, there is also a solution that only uses `goToPose` action. By determining the angle of approach at which the overall performance of the plan is optimal, and parameterizing `goToPose` so that it approaches the ball at this angle, also leads to improved performance. The behavior shown in Figure 1c exhibits seamless transitions between plan steps and has higher performance, achieving the ultimate goal in less time than in Figure 1a. This optimization, called subgoal refinement, can also be automated, as will be demonstrated in section 4.

Another frequent task in robotic soccer is to approach the ball. In Figure 2, the defender's goal is to clear the ball, and it has decided to do so by approaching the ball from behind, and kicking it away from the goal. One way to execute this plan is by first executing its general `goToPose` action. However, since this action does not take the ball into account, it might bump into it before achieving the desired position and orientation, as can be seen in Figure 2a.

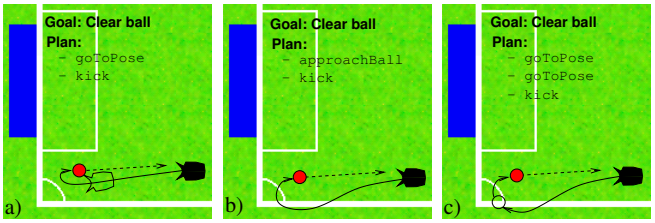


Figure 2: Three alternative plan executions to approach the ball.

To solve this problem, a specialized action that takes the ball into account could be written, e.g. `approachBall`. This variant is shown in Figure 2b, and would work fine. However, there is also a solution that only uses the `goToPose` action, and that does not require us to write `approachBall`. The solution is to introduce an intermediate way-point that ensures there will be no collision with the ball, and performing the navigation task with by appending two `goToPose` actions. Since the chosen path is similar to the path `approachBall` would probably choose, performance is not lost. When a way-point is needed, and where it should lie is determined automatically, using subgoal assertion, which will be presented in section 5.

2.2 Conceptualization

Our conceptualization for the computational problem is based on the notion of actions, performance models of actions, teleo-operators, teleo-operator libraries, and chains of teleo-operators. In this section we will introduce these concepts.

Actions are control programs that produce streams of control signals, based on the current estimated state, thereby influencing the state of the world. The basic action we use here is `goToPose`, which navigates the robot from the current pose (at time t) [x_t,y_t,ϕ_t] to a future destination pose [x_d,y_d,ϕ_d] by setting the translational and rotational velocity of the robot:

`goToPose`(x_t,y_t,ϕ_t,x_d,y_d,ϕ_d) $\rightarrow$ v_{tra},v_{rot}

Teleo-operators (TOPs) consist of an action, as well as pre- and post-conditions [Nilsson, 1994]. The post-condition represents the intended effect of the TOP, or its goal. It specifies a region in the state space in which the goal is satisfied. The pre-condition region with respect to a temporally extended action is defined as the set of world states in which continuous execution of the action will eventually satisfy the post-condition. They are similar to Action Schemata or STRIPS operators in the sense that they are temporally extended actions that can be treated by the planner as if they were atomic actions.

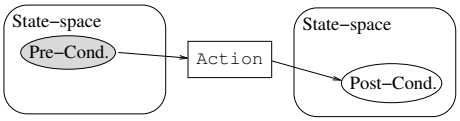


Figure 3: An abstract teleo-operator.

The `goToPoseTOP` has the empty pre-condition, as it can be executed from any state in the state space. Its post-condition is [$x_t \approx x_d$,$y_t \approx y_d$, $\phi_t \approx \phi_d$]. Its action is `goToPose`.

TOP libraries contain a set of TOPs that are frequently used within a given domain. In many domains, only a small number of control routines suffices to execute most tasks, if they are kept general and abstract, allowing them to be applicable in many situations. Our library contains the TOPs: `goToPoseTOP` and `dribbleBallTOP`.

A **TOP chain** for a given goal is a chain of TOPs such that the pre-condition of the first top is satisfied by the current situation, and the post-condition of each step satisfies the pre-condition of the subsequent TOP. The post-condition of the last TOP must satisfy the goal. It represents a valid plan to achieve the goal.

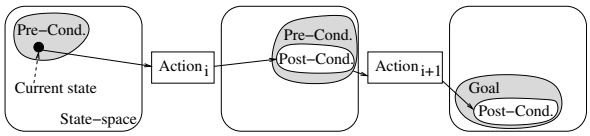


Figure 4: A chain of teleo-operators.

Subgoal refinement is the process of choosing a specific state as a subgoal, from the set of states defined by the post-condition of a preceding and pre-conditions of a subsequent action in a teleo operator chain. In Figure 5, such a specific subgoal has been chosen. This state will be visited in the transition from one action to the next.

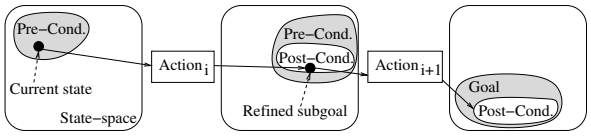


Figure 5: Subgoal refinement.

Performance models of actions map a specific situation onto a performance measure. These models can be used to predict the performance outcome of an action if applied in a specific situation, by specifying the current state (satisfying the pre-conditions) and end state (satisfying the post-conditions). An example of a performance measure is predicted execution time:

`goToPose.time`(x_t,y_t,ϕ_t,x_d,y_d,ϕ_d) $\rightarrow$ t

2.3 Computational task and solution idea

The on-line computational task is to optimize the overall performance of a TOP chain. The input consists of a TOP chain that has been generated by a planner, that uses a TOP library as a resource. The output is an intermediate refined subgoal that optimizes the chain, and is inserted in the chain. Executing the TOP chain is simply done by calling the action of each TOP. This flow is displayed in Figure 6.

To optimize action chains, the pre- and post-conditions of the TOPs in the TOP chains are analyzed to determine which variables in the subgoal may be freely tuned. These are the variables that specify future states of the robot, and are not constrained by the pre- and post-conditions of the respective TOP. For the optimization of these free variables, performance models of the actions are required. Off-line, these models are learned from experience for each action in the TOP library. They are used by the subgoal refinement system during execution time, but available as a resource to other systems as well.

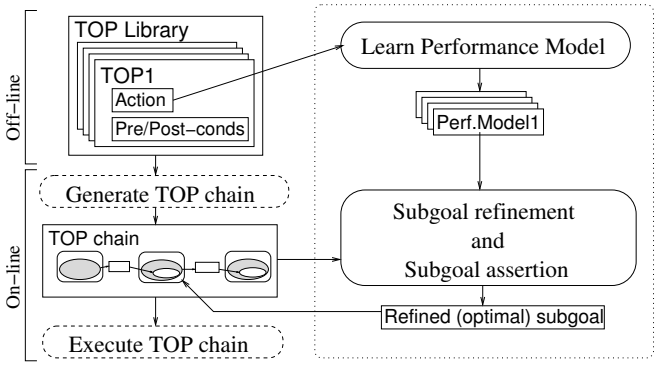


Figure 6: System Overview.

One of the big advantages of our approach is that neither TOP library, nor the generation of TOP chains (the planner) nor the TOP chain executor need to be modified in any way to

accommodate the action chain optimization system. We assume that the programmer provides a library of actions containing domain knowledge expressed in the pre- and post-condition, and has mechanisms for generating and executing chains of these actions, be it through planning, arbitration schemes, or simply manual specification. Although each of these components is a research field in its own right, our paper will not focus on them, also to emphasize that our system does not rely on their implementation.

The next three sections describe the main components in Figure 6. In section 3 we describe how performance models of actions are learned from experience. Subgoal refinement and subgoal assertion are presented in sections 4 and 5 respectively.

3 Learning performance models

To perform subgoal refinement and assertion, performance models of each action in the TOP library must be available. For each action, the robot therefore learns a function that maps situations to the cost of performing this action in the respective situation. The robot will approximate the performance function by learning decision and model trees based on observed experience.

Let us consider the navigation action `goToPose`. This navigation action is based on computing a Bezier curve, and trying to follow it as closely as possible [Beetz *et al.*, 2004]. Our `dribbleBall` action uses the same method, but restricts deceleration and rotational velocity, so as not to loose the ball. We abstract away from their implementation, as our methods consider the actions to be black boxes, whose performance we learn from observed experience.

To gather experience, with which the model will be learned, the robot executes the action under varying situations, observes the performance, and logs the experience examples. Since the method is based solely on observations, it is also possible to acquire models of actions whose internal workings are not accessible. The examples are gathered using our simulator, which uses learned dynamics models of the Pioneer I platform. It has proven to be accurate enough to port control routines from the simulator to the real robot without change.

The variables that were recorded do not necessarily correlate well with the performance. We therefore design a transformed feature space with less features, but the same potential for learning accurate performance models. In Figure 7 it is shown how exploiting transformational and rotational invariance reduces an original six-dimensional feature space into a three-dimensional one, with the same predictive power.

Currently, we perform the transformation manually for each action. In our ongoing research we are investigating methods to automate the transformation. By explicitly representing and reasoning about the physical meaning of state variables, we research feature language generation methods.

The last step is to approximate a function to the transformed data. Depending on whether a nominal or continuous value needs to be predicted, we use a decision or model tree respectively. Both methods learn a mapping from input features to output feature from experience, by a piecewise recursive partitioning of the examples in feature space. Partitioning continues until all the examples in a partition can be approximated well by a simple representative model. Decision trees use a nominal value, and model trees a linear function to represent the data in a partition.

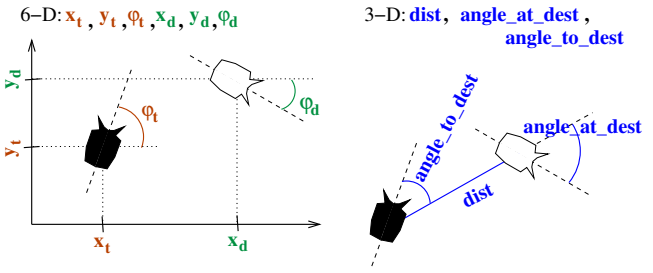


Figure 7: Transformation of the original state space into a lower-dimensional feature space.

We use decision and model trees because 1) they can be transformed into sets of rules that are suited for human inspection and interpretation 2) comparative research shows they are the very appropriate for learning action models [Belker, 2004; Balac, 2002] 3) they tend to use only relevant variables. This means we can start off with many more features than are needed to predict performance, having the model tree function as an automatic feature selector.

3.1 Prediction of execution duration

The first performance model we have learned is execution duration. It maps a current state and a goal state to the expected time needed to achieve the goal state with this action.

To gather experience, the robot executed each action thousand times, with random initial and destination poses. The robot recorded the direct variables and the time it took to reach the destination state at 10Hz, thereby gathering 75 000 examples of the format [x_t,y_t,ϕ_t,x_d,y_d,ϕ_d,$time$] per action. Using our Pioneer I robots, acquiring this amount of data would take approximately two hours of operation time.

Additional transformed features that were used to learn the model are shown in Figure 7. The model tree was actually learned on an 11-dimensional feature space [x_t,y_t,ϕ_t, x_d,y_d,ϕ_d,dx,dy,$dist$,$angle\_to\_dest$,$angle\_at\_dest$]. The model tree algorithm automatically discovered that only [$dist$,$angle\_to\_dest$,$angle\_at\_dest$] are necessary to accurately predict performance.

We will now give an example of one of the rules learned by the model tree. In Figure 8, we depict an example situation in which $dist$ and $angle\_to\_dest$ are to 2.0m and 0° respectively. Given these values we could plot a performance function for varying values of $angle\_at\_dest$. These plots are also depicted in Figure 8, once in a Cartesian, once in a polar coordinate system. In the linear plot we can clearly see five different line segments. This means that the model tree has partitioned the feature space for $dist$=2.0m and $angle\_to\_dest$=0° into five areas, each with its own linear model. Below the two plots, one of the learned model tree rules that applies to this situation is displayed. An arrow indicates its linear model in the plots. The polar plot clearly shows the dependency of predicted execution time on the angle of approach for the example situation. Approaching the goal at 0 degrees is fastest, and would take a predicted 2.1s. Approaching the goal at 180

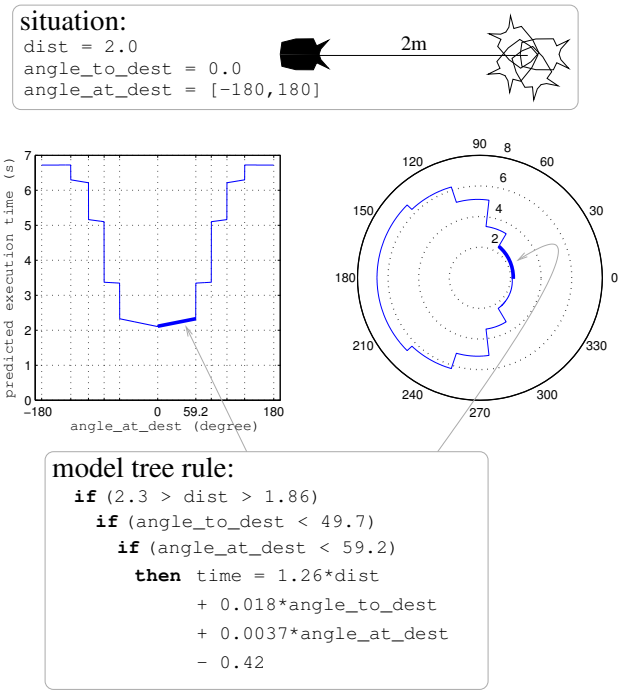


Figure 8: An example situation, two graphs of time prediction for this situation with varying $angle\_at\_dest$, and the model tree rule for one of the line segments.

degrees means the robot would have to navigate around the goal point, taking much longer (6.7s).

To evaluate the accuracy of the performance models, we again randomly executed each action to acquire test examples. For the action `goToPose`, the mean absolute error and root-mean-square error between predicted and actual execution time were 0.31s and 0.75s. For the `dribbleBall` routine these values were 0.29s and 0.73s. As we will see, these errors are accurate enough to optimize action chains.

3.2 Prediction of ball approach failure

The `goToPose` action can often be used well to approach the ball. However, in some situations it will bump into the ball before achieving the desired orientation, as was shown in Figure 2. The second performance model we have learned predicts whether executing `goToPose` will lead to a collision with the ball or not.

To acquire experience, the robot again executed `goToPose` a thousand times, with random initial and destination poses, the ball always positioned at the destination pose. The robot recorded 65 000 training examples of the format [x_t,y_t,ϕ_t,x_d,y_d,ϕ_d,$collided?$] per action. The flag $collided?$ is set to `Collision` for all the examples in a whole run, if the robot eventually collided with the ball before reaching its desired position and orientation, and to `Success` otherwise.

The model was learned with the same 11-dimensional transformed feature space as used in learning temporal prediction. Again, only [$dist$,$angle\_to\_dest$,$angle\_at\_dest$] were used to predict a collision.

The learned tree, as well as a graphical representation of it, are depicted in Figure 9. The goal pose is represented by the robot, and different areas indicate if the robot can reach this position with `goToPose`, without bumping into the ball first. Remember that `goToPose` has no awareness of the ball at all. The model simply predicts when its execution leads to a collision or not. Intuitively, the rules seem correct. When coming from the right, for instance, it can be seen that the robot always disrespectfully stumbles into the ball, long before reaching the desired orientation. Behind the ball, the robot may not be too close to the ball (checkered area), unless it is facing it. This last rule is indicated by the arrows pointing in the direction of the ball.

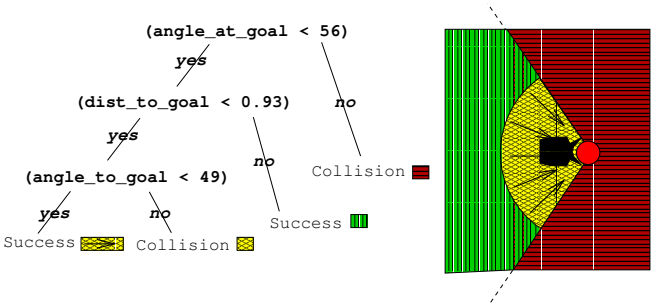


Figure 9: The learned decision tree that predicts whether an unwanted collision will happen.

To evaluate the accuracy of this model, the robot executed another thousand runs, and compared predicted collision with observed collisions. The decision tree predicts collisions correctly in almost 90% of the cases. A more thorough analysis is depicted in Table 1. The model is quite pessimistic, as it predicts failure 61%, whereas in reality it is only 52%. In 10% of cases, it predicts a collision when it actually does not happen. This is preferable to an optimistic model, as it is better to be safe than sorry.

		Observed			Total
		`Coll.`	`Succ.`		Predicted
Predicted	`Coll.`	51%	10%	→	61%
	`Succ.`	1%	38%	→	39%
		↓	↓		↓
Total Observed		52%	48%	→	100%

Table 1: Accuracy of ball collision prediction.

Actually, this decision tree is much more than a performance model. It can be considered as the conditions in which `goToPose` will successfully approach the ball. We now have an teleo-operator `approachBallTOP`, with different preconditions from `goToPoseTOP`. However, since `approachBallTOP` also uses the action `goToPoseTOP` there is no explicit action `approachBall`. We have only determined the conditions under which `goToPose` must be executed to achieve successful ball approach. We will make use of this when applying automatic subgoal assertion in section 5.

4 Automatic subgoal refinement

As depicted in Figure 6, the automatic subgoal refinement system takes the performance models and a chain of teleo-operators as an input, and returns a refined intermediate goal state that has been optimized with respect to the performance of the overall action chain. To do this we need to specify all the variables in the task, and recognize which of these variables influence the performance and are not fixed. These variables form a search space in which we will optimize the performance using the learned action models.

4.1 State variables

In the dynamic system model [Dean and Wellmann, 1991] the world changes through the interaction of two processes: the *controlling process*, in our case the low-level control programs implementing the action chains generated by the planner, and the *controlled process*, in our case the behavior of the robot. The evolution of the dynamic system is represented by a set of *state variables* that have changing values. The controlling process steers the controlled process by sending *control signals* to it. These control signals directly set some of the state variables and indirectly other ones. The affected state variables are called the *controllable* state variables. The robot for instance can set the translational and rotational velocity directly, causing the robot to move, thereby indirectly influencing future poses of the robot.

For the robot, a subset of the state variables is *observable* to its perceptive system, and they can be estimated using a state estimation module. For any controller there is a distinction between *direct* and *derived* observable state variables. All direct state variables for the navigation task are depicted in Figure 10. Direct state variables are directly provided by state estimation, whereas derived state variables are computed by combinations of direct variables. No extra information is contained in derived variables, but if chosen well, derived variables are better correlated to the control task.

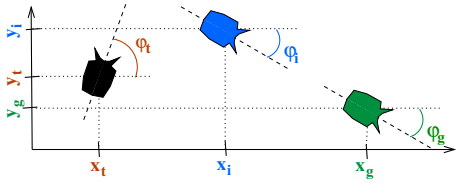


Figure 10: Direct state variables relevant to the navigation task.

State variables are also used to specify goals internal to the controller. These variables are *bound*, conform to planning terminology. It is the controller's goal to have the bound internal variables (approximately) coincide with the external observable variables. The robot's goal to arrive at the intermediate position could be represented by the state variables $[x_i,y_i]$. By setting the velocities, the robot can influence its current position $[x_t,y_t]$ to achieve $[x_t \approx x_i, y_t \approx y_i]$.

4.2 Determining the search space

To optimize performance, only variables that actually influence performance should be tuned. In our implementation, this means only those variables that are used in the model tree to partition the state space at the nodes, or used in the linear functions at the leaves.

In both the learned model trees for the actions `goToPose` and `dribbleBall`, the relevant variables are $dist$, $angle\_to\_dest$ and $angle\_at\_dest$. These are all derived variables, computed from the direct variables $[x_t,y_t,\phi_t,x_i,y_i,\phi_i]$ and $[x_i,y_i,\phi_i,x_g,y_g,\phi_g]$, for the first and second action respectively. So by changing these direct variables, we would change the indirect variables computed from them, which in effect would change the performance.

But may we change all these variables at will? Not x_t,y_t, or ϕ_t, as we cannot simply change the current state of the world. Also we may not alter bound variables that the robot has committed to, being $[x_i,y_i,x_g,y_g,\phi_g]$. Changing them would make the plan invalid.

This only leaves the free variable ϕ_i, the angle at which the intermediate goal is approached. This acknowledges our intuition from Figure 1 that changing this variable will not make the plan invalid, and that it will also influence the overall performance of the plan. We are left with a one-dimensional search space to optimize performance.

4.3 Optimization

To optimize the action chain, we will have to find those values for the free variables for which the overall performance of the action chain is the highest. The overall performance is estimated by summing over the performance models of all actions that constitute the action chain. In Figure 11 the first two polar plots represent the performance of the two individual actions for different values of the only free variable, which is the angle of approach. The overall performance is computed by adding those two, and is depicted in the third polar plot.

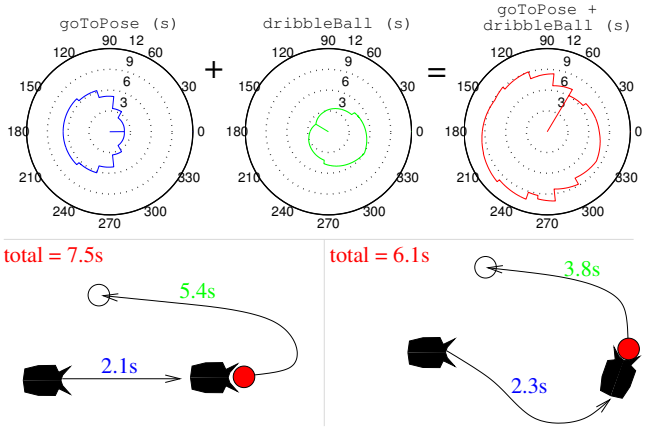


Figure 11: Selecting the optimal subgoal by finding the optimum of the summation of all action models in the chain.

The fastest time in the first polar plot is 2.1s, for angle of approach of 0.0 degrees. The direction is indicated from the center of the plot. However, the total time is 7.5s, because the second action takes 5.4s for this angle . These values can be read directly from the polar plots. However, this value is not the optimum overall performance. The minimum of

the overall performance is 6.1s, as can be read from the third polar plot. Below the polar plots, the situation of Figure 1 is repeated, this time with the predicted performance for each action.

We expect that for higher-dimensional search spaces, exhaustive search may be infeasible. Therefore, other optimization techniques will have to be investigated.

4.4 Results

To determine the influence of subgoal refinement on the overall performance of the action chain, we generated a thousand situations with random robot, ball and final goal positions. The robot executed each navigation task twice, once with subgoal refinement, and once without. The results are summarized in Table 2. First of all, the overall increase in performance over the 1000 runs is 10%. We have split these cases into those in which the subgoal refinement yielded a higher, equal or lower performance in comparison to not using refinement. This shows that the performance improved in 533 cases, and in these cases causes a 21% improvement. In 369 cases, there was no improvement. This is to be expected, as there are many situations in which the three positions are already optimally aligned (e.g. in a straight line), and subgoal refinement will have no effect.

Before filtering	Total	Higher	Equal	Lower
# runs	1000	533	369	98
improvement	10%	21%	0%	-10%
After filtering	Total	Higher	Equal	Lower
# runs	1000	505	485	10
improvement	12%	23%	0%	-6%

Table 2: Results, before and after filtering for cases in which performance loss is predicted.

Unfortunately, applying our method causes a decrease of performance in 98 out of 1000 runs. To analyze in which cases subgoal refinement decreases performance, we labeled each of the above runs `Higher`, `Equal` or `Lower`. We then trained a decision tree to predict this nominal value. This tree yields four simple rules which predict the performance difference correctly in 86% of given cases. The rules and a graphical representation are depicted in Figure 12. In this graph, the robot always approaches the centered ball from the left at different distances. The different regions indicate whether the performance increase/decreased due to subgoal refinements, if the goal lies in this region. Three instances with different classification and therefore different colors circles have been inserted.

The rules declare that performance will stay equal if the three points are more or less aligned, and will only decrease if the final goal position is in the same area as which the robot is, but only if the robot's distance to the intermediate goal is smaller than 1.4m. Essentially, this last rule states that the robot using the Bezier-based `goToPose` has difficulty approaching the ball at awkward angles if it is close to it. In these cases, small variations in the initial position lead to large variations in execution time, and learning an accurate, general model of the action fails. The resulting inaccuracy in temporal prediction causes suboptimal optimization. Note that this is a shortcoming of the action itself, not the chain optimization methods.

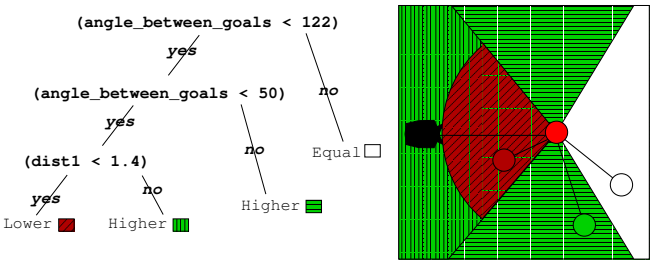


Figure 12: The decision tree that predicts whether subgoal refinement will make the performance better, worse or have no influence at all.

We then performed another thousand test runs, as described above, but only applied subgoal refinement if the decision tree predicted applying it would yield a higher performance. Although increase in overall performance is not so dramatic (from 10% to 12%), the number of cases in which performance is worsened by applying subgoal refinement has decreased from 98 (10%) to 10 (1%). Apparently, the decision tree correctly filtered out cases in which applying subgoal refinement would decrease performance.

Without subgoal refinement, the transitions between actions were very abrupt. In general, these motion patterns are so characteristic for robots that people trying to imitate robotic behavior will do so by making abrupt movements between actions. In contrast, one of the impressive capabilities of animals and humans is their capability to perform chains of actions in optimal ways and with seamless transitions between subsequent actions. It is interesting to see that requiring optimal performance can implicitly yield smooth transitions in robotic and natural domains, even though smoothness in itself is not an explicit goal in either domain.

Summarizing: subgoal refinement with filtering yields smooth transitions and a 23% increase in performance half of the time. Only once in a hundred times does it cause a small performance loss.

5 Automatic subgoal assertion

In the previous section, we have seen how subgoals can be refined in order to optimize performance. In this section, we will show how performance models can be used to detect when the assertion of a new subgoal is necessary.

We use a scenario in which a robot approaches a ball, introduced in section 2.1. A difficulty in approaching the ball is that the robot might collide with the ball before it has reached its desired position and orientation. Since our `goToPose` action is not aware of these potential collisions, it is not always appropriate for approaching the ball. Actually, it can be derived from Table 1 that it fails in 52% of cases. To solve this problem, one could write a new action, e.g. `approachBall`. It would probably be very similar to `goToPose`, but take the ball into account.

Instead of writing a new action, thereby causing the problems discussed in the introduction, it is also possible to reuse `goToPose`, and adapt it to the current context. First of all,

it is important to recognize that goToPose is actually successful in approaching the ball almost half the time (Table 1). Fortunately, we have models that can predict when success is probable. So when the goal is to approach the ball, and the performance model predicts that goToPose can do this collision-free, this action is executed as is.

When no action can be parameterized in such a way that ball approach is likely to succeed, we need to find a chain of actions that can. This is done in a means-ends fashion. First, the robot determines which actions can achieve the goal, and which preconditions must hold for this action to succeed. Then, it determines if any action can achieve these preconditions. In our example, a sequence of two goToPose actions can achieve the goal. A constraint is that the second action in the sequence must be able to reach the ball without unintentional collisions. This could be any position in the most left area of Figures 9 and 13, because the performance model predicts that there will be no collision when starting from any of the position in this area.

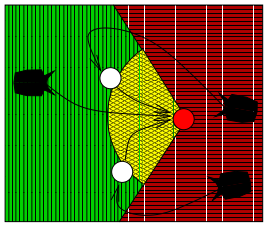


Figure 13: Subgoal assertion to avoid collisions with the ball.

Although all positions in this area can function as an intermediate goal for the two goToPose actions, the overall expected execution duration is different for all off them. Therefore, we sample a thousand points from the area, and compute the overall performance by adding the predicted time of the first and second goToPose actions in the action chain. The point with the best performance, that is, fastest execution time, is chosen to be the intermediate point. This optimization process is nothing else than subgoal refinement, as has been presented in section 4.

In Figure 13, three instances of the problem are depicted. Since the robot to the left is in the area in which no collision is predicted, it simply executes goToPose, without asserting a subgoal. The model predicts that the other two robots will collide with the ball when executing goToPose, and a subgoal is asserted. The optimal positions of the subgoals, determined by subgoal refinement, are shown as white circles.

5.1 Results

To evaluate automatic subgoal assertion we executed a thousand random ball approaches, once with assertion, and once without. The results are summarized in Table 3. It is clear that using only goToPose is not very successful. It approaches the ball collision-free less than half the time. This is actually exactly what our performance model predicts, as can be seen in Table 1. Applying subgoal assertion dramatically improves this. In less than 3% of cases does the ball approach fail.

We have also investigated under which circumstances a subgoal was introduced, and if it was helpful to do so. In 37% of cases, no subgoal was needed, as no collision was predicted. In 52%, a subgoal was asserted, causing a successful completion that was not possible without a subgoal. In 10% of cases, a subgoal was introduced unnecessarily, as the task could have been solved without a subgoal. Note that all these percentages are roughly the same as those in Table 1. Inappropriately introducing the subgoal caused a performance loss of 11% in these cases.

	direct (no subgoal)	with subgoal assertion
Success	47%	97%
Collision	53%	3%

Table 3: The effects of applying subgoal assertion to the ball approach task.

Summarizing: if subgoal assertion is not necessary, it is usually not applied. Half of the time, a subgoal is introduced, which raises successful task completion from 47 to 97%. Infrequently, subgoals are introduced inappropriately, but the performance loss in these cases is an acceptable cost compared to the pay-off of the dramatic increase in the number of successful task completions.

6 Related Work

Most similar to our work is the use of model trees to learn performance models to optimize Hierarchical Transition Network plans [Belker, 2004]. In this work, the models are used to select the next action in the chain, whereas we refine an existing action chain. Therefore, the planner can be selected independently of the optimization process.

Reinforcement Learning (RL) is another method that seeks to optimize performance, specified by a reward function. Recent attempts to combat the curse of dimensionality in RL have turned to principled ways of exploiting temporal abstraction [Barto and Mahadevan, 2003]. Several of these *Hierarchical Reinforcement Learning* methods, e.g. (Programmable) Hierarchical Abstract Machines [Parr, 1998; Andre and Russell, 2000], MAXQ [Dietterich, 2000], and Options [Sutton *et al.*, 1999]. All these approaches use the concept of actions (called 'machines', 'subtasks', or 'options' respectively). In our view, the benefits of our methods are that they acquire more informative performance measures, facilitate the reuse of action models, and scale better to continuous and complex state spaces.

The performance measures we can learn (execution time, action failure) are *informative* values, with a meaning in the physical world. Future research aims at developing meaningful composites of individual models. We will also investigate dynamic objective functions. In some cases, it is better to be fast at the cost of accuracy, and sometimes it is better to be accurate at the cost of speed. By weighting the performance measures time and accuracy accordingly in a composite measure, these preferences can be expressed at execution time. Since the (Q-)Value compiles all performance information in a single non-decomposable numeric value, it cannot be reasoned about in this fashion.

The methods we proposed *scale* better to continuous and

complex state spaces. We are not aware of the application of Hierarchical Reinforcement Learning to (accurately simulated) continuous robotic domains.

In Hierarchical Reinforcement Learning, the performance models of actions (Q-Values) are learned in the calling context of the action. Optimization can therefore only be done in the context of the pre-specified hierarchy/program. In contrast, the success of action selection in complex robotic projects such as WITAS [Doherty *et al.*, 2000], Minerva [Thrun *et al.*, 1999], and Chip [Firby *et al.*, 1996], depends on the on-line autonomous sequencing of actions through planning. Our methods learn abstract performance models of actions, independent of the context in which they are performed. This makes them *reusable*, and allows for integration in planning systems.

The only approach we know of that explicitly combines planning and RL is RL-TOPS (*Reinforcement Learning - Teleo Operators*) [Ryan and Pendrith, 1998]. Abrupt transitions arise here too, and the author recognizes that "cutting corners" between actions would improve performance, but does not present a solution.

Many behavior based approaches also achieve smooth motion by a weighted mixing of the control signals of various actions [Saffiotti *et al.*, 1995; 1993]. In computer graphics, this approach is called *motion blending*, and is also a wide-spread method to generate natural and fluent transitions between actions, which is essential for lifelike animation of characters. Impressive results can be seen in [Perlin, 1995], and more recently [Shapiro *et al.*, 2003; Kovar and Gleicher, 2003]. Since there are no discrete transitions between actions, they are also not visible in the execution. In all these blending approaches, achieving optimal behavior is not an explicit goal; it is left to chance, not objective performance measures.

A very different technique for generating smooth transitions between skills has been developed for quadruped robots [Hoffmann and Düffert, 2004], also in the RoboCup domain. The periodic nature of robot gaits allows their meaningful representation in the frequency domain. Interpolating in this domain yields smooth transitions between walking skills. Since the actions we use are not periodic, these methods do unfortunately not apply.

Reusing actions and transferring knowledge between them are also key concepts in life-long learning [Thrun and Mitchell, 1993]. This approach exploits the notion that learning to run is much more easy when you already know how to walk, just as approaching a ball is more easy if you already know how to navigate. In [Thrun and Mitchell, 1993], knowledge is transfered between tasks by reusing neural networks that have been trained on one task as a bias for similar, perhaps more complex tasks that have yet to be learned.

7 Conclusion and Future Work

The central idea of this work is that by adapting action parameterization, actions can be tailored to the task context. There is no longer a need to write a new action for each new context, and generality is maintained. Instead of using manual parameterization, we use learned action models to optimize the parameters with respect to the given performance measure.

On-line optimization of action chains allows the use of planning with abstract actions, without losing performance. Optimizing the action chain is done by asserting and refining under-specified intermediate goals, which requires no change in the planner or plan execution mechanisms. To predict the optimal overall performance, performance models of each individual abstract action are learned off-line and from experience, using model trees.

Applying subgoal refinement and assertion to the presented scenarios yields significant performance improvement. However, the computational model underlying the optimization is certainly not specific to this scenario, or to robot navigation. In principle, learning action models from experience using model trees is possible for any action whose relevant state variables can be observed and recorded. The notion of controllable, bound and free state variables are taken directly from the dynamic system model and planning approaches, and apply to any scenario that uses these paradigms. Our future research therefore aims at applying these methods in other domains, for instance robots with articulated arms and grippers, for which we also have a simulator available.

Currently, we are evaluating if subgoal refinement improves plan execution on real Pioneer I robots as much as it does in simulation. Previous research has shown that action models learned in simulation can be applied to real situations with good result [Buck *et al.*, 2002; Belker, 2004].

References

[Andre and Russell, 2000] David Andre and Stuart Russell. Programmable reinforcement learning agents. In *Conference on Neural Information Processing Systems (NIPS)*, 2000.

[Balac, 2002] N. Balac. *Learning Planner Knowledge in Complex, Continuous and Noisy Environments*. PhD thesis, Vanderbilt University, 2002.

[Barto and Mahadevan, 2003] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event systems*, 2003.

[Beetz *et al.*, 2004] Michael Beetz, Alexandra Kirsch, and Armin Müller. RPL-LEARN: Extending an autonomous robot control language to perform experience-based learning. In *3rd International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS)*, 2004.

[Belker, 2004] T. Belker. *Plan Projection, Execution, and Learning for Mobile Robot Control*. PhD thesis, Department of Applied Computer Science, Univ. of Bonn, 2004.

[Buck *et al.*, 2002] Sebastian Buck, Michael Beetz, and Thorsten Schmitt. Reliable Multi Robot Coordination Using Minimal Communication and Neural Prediction. In M. Beetz, J. Hertzberg, M. Ghallab, and M. Pollack, editors, *Advances in Plan-based Control of Autonomous Robots. Selected Contributions of the Dagstuhl Seminar "Plan-based Control of Robotic Agents"*, Lecture Notes in Artificial Intelligence. Springer, 2002.

[Dean and Wellmann, 1991] T. Dean and M. Wellmann. *Planning and Control*. Morgan Kaufmann Publishers, 1991.

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Doherty *et al.*, 2000] P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The WITAS unmanned aerial vehicle project. In *Proceedings ECAI-00*, 2000.

[Firby *et al.*, 1996] R. Firby, P. Prokopowicz, M. Swain, R. Kahn, and D. Franklin. Programming CHIP for the IJCAI-95 robot competition. *AI Magazine*, 17(1):71–81, 1996.

[Hoffmann and Düffert, 2004] J. Hoffmann and U. Düffert. Frequency space representation and transitions of quadruped robot gaits. In *Proceedings of the 27th conference on Australasian computer science*, 2004.

[Kovar and Gleicher, 2003] L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of ACM SIGGRAPH*, 2003.

[Nilsson, 1994] N.J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1994.

[Parr, 1998] Ronald Parr. *Hierarchical Control and learning for Markov Decision Processes*. PhD thesis, University of California at Berkeley, 1998.

[Perlin, 1995] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

[Ryan and Pendrith, 1998] M. Ryan and M. Pendrith. RL-TOPs: an architecture for modularity and re-use in reinforcement learning. In *Proc. 15th International Conf. on Machine Learning*, 1998.

[Saffiotti *et al.*, 1993] A. Saffiotti, E. H. Ruspini, and K. Konolige. Blending reactivity and goal-directedness in a fuzzy controller. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 134–139, San Francisco, California, 1993. IEEE Press.

[Saffiotti *et al.*, 1995] A. Saffiotti, K. Konolige, and E.H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 1995.

[Shapiro *et al.*, 2003] Ari Shapiro, Frederic Pighin, , and Petros Faloutsos. Hybrid control for interactive character animation. In *Pacific Graphics*, pages 455–461, 2003.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[Thrun and Mitchell, 1993] S. Thrun and T. Mitchell. Lifelong robot learning. Technical Report IAI-TR-93-7, University of Bonn, Department of Computer Science, 1993.

[Thrun *et al.*, 1999] S. Thrun, M. Bennewitz, W. Burgard, A.B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A tour-guide robot that learns. In *KI - Kunstliche Intelligenz*, pages 14–26, 1999.

How to keep plan execution healthy *

Femke de Jonge and **Nico Roos** and **Jaap van den Herik**
Universiteit Maastricht, IKAT,
P.O. Box 616, NL-6200, Maastricht
{F.deJonge, Roos, Herik}@cs.unimaas.nl

Abstract

Developing a conflict-free plan for a multi-agent system in a complex and dynamic environment is a difficult task. Moreover, it is impossible to take into account all possible events that might occur during the execution of the plan. Unexpected events may cause a plan to be no longer executable without leading to conflicts: we then say that its execution is unhealthy. This paper presents a new model that enables agents (1) to control their plan-execution health and (2) to regain health when necessary. The agents can utilize the model to predict consequences of occurring disruptions and thus detect unhealthy situations. With the help of the model's predictions, agents can correct the execution of tasks within the plan in such a way that conflicts will be avoided and health is regained. We emphasize that, in the case of bad health, the approach of correcting the plan execution should be applied before relying on the more drastic approach of replanning. The applicability of the presented model is demonstrated by introducing two multi-agent protocols to keep the plan execution healthy. Finally, we investigate the solving capabilities and the efficiency of our method in experiments using randomly generated plans. Our conclusion is that a reasonable proportion of unhealthy situations can be solved adequately by well-thought corrections in the plan execution instead of performing a replanning procedure.

1 Introduction

Plan development and plan execution in complex, dynamic environments are difficult tasks. This explains the tendency to apply intelligent computer programs to support these tasks. Currently, the (initial) plan development in fields such as Air Traffic Control (ATC) is to a large extent performed by planning software. For plan execution, however, such software is not widely available, even though the execution of plans in complex and dynamic environments requires continuous control and adaptation. Our research focusses on employing a multi-agent system for plan-execution control and adaptation. Multi-agent systems seem an obvious means to this end since the plans in environments such as ATC are mainly distributed.

An adequate plan normally satisfies all constraints imposed by its environment and by other plans. Hence, such a plan is conflict free. This is a property that should be kept consequently and persistently during the execution of the plan. We denote a plan execution as healthy, when during the execution of the plan no constraints are violated. Conversely, an unhealthy plan execution violates one or more constraints. A conflict-free plan can have an unhealthy plan execution when unexpected or unanticipated changes in the environment occur. For instance, a change in the environment can cause a plan execution to behave differently from what was expected, which might result in a conflict with other plan executions (through violation of their interaction constraints). The process of keeping a plan execution healthy can be viewed as a continuous cycle of detecting unhealthy situations and regaining health. Plan-execution health can be regained by either correcting the execution or changing the plan (i.e., replanning).

In our opinion, corrections within the execution of a plan have three advantages when compared to replanning, viz. (1) they are often easier to accomplish, (2) they are less influential for the environment and the rest of the plan, and (3) especially within domains such as ATC, plan changes are more costly than changes in execution. For instance, gate changes require a large amount of organization as the passengers need to be informed, the engaged ground handling needs to be relocated, and so on. Not surprisingly, within the ATC practice, the first attempt to regain health is always to try and find solutions within the execution of the current plan. Therefore, we emphasize that before applying replanning, agents should try to regain health by correcting the execution of the plan without changing the plan itself.

In summary, the contribution of this paper is that it enables agents to keep the plan execution healthy by applying small corrections within the plan execution. For this purpose, we developed a model that agents can apply (1) to control the health of the plan execution and (2) to find corrections to re-

*This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs. Project DIT5780: Distributed Model Based Diagnosis and Repair

gain health when necessary. Below, we present our model and demonstrate its potential by two multi-agent protocols, viz. (1) a protocol for *health control* to detect unhealthy plan execution (conflicts) and (2) a protocol for *health repair* to find small corrections (repairs) to regain health. To gain insight into which unhealthy situations are suitable for our approach of correcting plan execution, we tested the protocols in experiments with randomly generated plans.

The outline of the paper is as follows. Section 2 introduces the basic notions for our approach and section 3 briefly describes related research. In section 4, we present our model for plan-execution health control and repair in a multi-agent system. Section 5 provides formal definitions of when a plan execution is healthy, and how plan-execution health can by regained by applying small corrections to the execution. In section 6, we present the two protocols, viz. for health control and health repair. The experiments and the test results are described in section 7, while section 8 provides our conclusion and topics for future research.

2 Basic notions

In this section, we discuss important notions on planning in complex and dynamic environments, and present an overview of our model. Moreover, we briefly discuss the interplay between our approach of small corrections and the approach of replanning. Finally, we introduce a running example derived from the ATC case.

Planning notions

As stated in the introduction, we address the execution of a plan after it is created. So, we assume that a plan is already developed. We view a plan as a partially ordered set of steps. These steps are actions carried out at specific points in the plan, while the actions are instantiations of general operations [Ghallab *et al.*, 2004]. The execution of the steps usually has a certain duration and may require resources that have to be shared with other steps of the same plan or of other plans. We assume that a set of constraints describes requirements with respect to shared resources. Within ATC, for instance, we can think of safety constraints and of environmental constraints on noise pollution. Since we consider a multi-agent context, we assume that the plan is distributed over the agents. Each agent is responsible for a subplan, consisting of a sequence of steps the agents wishes to execute. We assume that only one agent is responsible for one step of the plan (so the subplans do not overlap), and that the distribution of the steps of the plan is determined in advance. For example, in the ATC case, we can think of a multi-agent system containing one agent for each aircraft (controlling the aircraft's subplan).

Plan descriptions generally see the steps as atomic parts that make up the plan. Here, we view them as tasks that require several, often reactive, activities of the executing agents. These activities cannot be planned because they depend on the status of the environment (cf. when driving a car from A to B, not every overtaking manoeuvre can be planned in advance). Therefore, the way the plan should be executed is not specified exactly and we may state that the tasks have some boundaries or margins within which the execution may vary. In particular in air traffic, it is common to specify margins for the duration of tasks. For instance, it is the primary responsibility of a pilot or aircraft agent, flying from one waypoint to the next one, to keep the aircraft in the assigned flight path within an assigned time interval. The activities of adjusting speed, height, and directions are not specified in the plan, but are assumed to be applied within the boundaries. However, the activities contribute to the attempt to follow the plan, i.e., to keep the plan execution within the specified margins such as the flight path and the time interval. So, the unplannable activities within plan execution influence whether the constraints are satisfied or violated. Even when a plan is executed within its margins, it still may happen that constraint violations occur (e.g., due to overtaking manoeuvres when driving a car from A to B).

Model overview

The model proposed in this paper assigns a health state to each task in a plan. This health state may change during the execution of a task caused by unforeseen environmental influences or by activities of the agent executing the task. The external influences of the environment will be modelled as disruption events and the activities of agents, assuming that agents do not deliberately disrupt the execution of tasks, as repair events.

The assignment of health states to tasks will enable us to evaluate the effects of disruption events that have occurred during the execution of tasks. Our first (implicit) assumption of the model is that disruption events are observable. This assumption will not hold in general, especially in environments where not all possible disruption events can be known. However, the model is also useable, with minor adaptations, if agents are able to determine the actual health states of tasks, for instance through plan diagnosis (see, e.g., [Witteveen *et al.*, 2005]). We note that this will often mean that the agent has less time to take appropriate action by initiating repair events. A second assumption is that the plans of the individual agents are linear. This assumption is mainly made for the clarity of the presentation of the model. Moreover, it is a common practice in ATC. The model is, however, also applicable if agents have partially ordered plans.

Replanning

In the introduction we implied that when plan-execution health cannot be regained by applying corrections, replanning should be applied. There are two more situations in which replanning might be applied. (In our view, they are exceptions of our model.) The first situation arises when finding a plan-execution health repair takes too much time, and an adjustment in the plan itself (instead of an adjustment in the plan execution) can be found much faster. To gain a better insight into this consideration, a comparative assessment of both methods (plan-execution health repair and replanning) with respect to the expected complexities and costs is required.

The second situation in which the agents should fall back on replanning techniques is when tasks reach states that cannot be changed by applying repair events. For instance, when during arrival an aircraft is running out of fuel and needs to

land immediately. Such 'emergency states' should trigger the agents to solve their problems more drastically, by changing their plans.

Running example

The following example will be used as a running example throughout the text. Consider a small airport with only one runway used for both arrival and departure. It is a small but busy airport, so plans are tight. Assume that two aircraft agents, agent A and agent B, each have their own (sub)plan, connected through a constraint. A's plan is (1) to taxi from the gate to the runway, and (2) to take off from the runway. B's plan is (1) to arrive at the airport (at the runway), and then (2) to taxi from the runway to its gate. The obvious constraint that connects the two plans is that the runway cannot be used by more than one aircraft at the same time. Therefore, the agents have agreed on a mutual plan in which B lands before A takes off. It is remarked that the aircraft can pass each other on the taxiway.

In our model, taxiing, taking off, and arriving are considered as tasks, since they are the smallest planned actions. The execution of one such a task is a reactive process in the sense that (sub)actions during the execution of a task (for example manoeuvring or changing speed) are not part of the planning. Although the plan satisfies the constraint, still, small changes in the execution (mostly caused by external influences) can cause a violation of constraints imposed on the plan execution. For instance, assume that A is a bit early as the aircraft speeded up while taxiing[1], and B is a bit delayed because of heavy head wind during its flight. Then, they still may not use the (same part of the) runway at the same time, but the two aircraft might pass one another at a close distance. However, a close distance could cause a violation of the safety constraints on the distance that should be kept between the two aircraft.

In this situation, we would model the following three states for both the 'Arrive' and the 'Take off' task: 'normal', 'delayed', and 'early'. These states represent the timing aspects, but other types of states are imaginable as well. For instance for 'Taxi' the state 'off track', denoting that the aircraft is diverging from the standard taxi route may be introduced. The unanticipated changes in the plan execution are modelled by disruption events. So the disruption events 'speeded up' (during the task 'Taxi') and 'heavy headwind' (during the task 'Arrive') may occur during the plan execution. Given the order in which the aircraft may make use of the runway, the safety constraint can be translated into the demand that it should not happen that the task 'Taxi' is 'early', and the task 'Arrive' is 'delayed'. If so, then appropriate action should be taken. In our example, the disruption events cause the tasks to achieve these states which then violate the constraint, and therefore the plan execution becomes unhealthy. To regain health, agents can apply so-called repair events to reach other states. For instance, in our example, agent A could wait a short while during its 'Taxi' task. Other examples of repair events we would like to mention are adjusting the position of an aircraft on the taxiway or air corridor, adjusting the speed of an aircraft, and applying small reparations to an aircraft. Note that these reactive adjustments are not typified as replanning since the corrections in executions will remain within the margins of the planned tasks.

[1] It is known that in practice, some pilots have a tendency to taxi at higher speed since they are familiar with the taxiways at the airport.

3 Related research

The main contribution of this paper is the model for plan-execution health control and repair. A fundamental property of such a model is, in our opinion, the ability to represent the current and future states of the plan and its environment. The models that are at the basis of such a property are Discrete Event Systems (DESs) and Markov Decision Models (see, for an overview [Cassandras, 1993]). A DES models (1) the states that a task (or object) can reach by nodes, and (2) the changes of states by events. Markov Decision Models are a specific type of DES, in which changes of states are probabilistically determined. Our model is partially inspired by these two models.

The TÆMS modelling framework used by [Raja *et al.*, 2000] is also related to our model, since their plan representation is rather similar. In TÆMS, a plan is represented by task descriptions that express the uncertainty in plan execution. Such task descriptions can be viewed as the range of states the task might be in during its execution. The main difference with our model follows from its purpose and application. Raja et al. use TÆMS for plan development: based on expectations of the occurrence of certain states, a plan is developed and tasks can be divided among agents. In contrast, our research focusses on predicting the states that the tasks will reach, and how to influence this to regain plan execution health.

Our goal to keep plan execution healthy somewhat overlaps the goal of so-called continual planning (for an overview of distributed continual planning, see [desJardins *et al.*, 2000]). In continual planning, the processes of planning and execution are interleaved so as to deal with uncertainties in a dynamic environment. In general, continual planning consists of two parts, viz. (1) monitoring, which corresponds largely with our plan-execution control, and (2) (re)planning, which is applied to prevent conflicts or improve the current planning. desJardins et al. state that the most preferred planning technique for continual planning uses a hierarchy. In hierarchical planning, initially, an abstract plan is made and as the execution approaches, the plan is being refined. Even though we do not reject this approach completely, it is our opinion that plan refinement cannot resolve all possible unhealthy situations, since there is a level within each plan for which its (sub)activities are unplannable (as discussed in section 2). This level is of reactive nature and its activities are not planned beforehand by any planner. The corrections at this level can be seen as an adjustment of the parameters that control the reactive execution of these activities. Our model is particularly suitable to monitor and correct the activities at this reactive (task) level. Apart from that, since plan refinement can be viewed as a type of replanning, it can be used

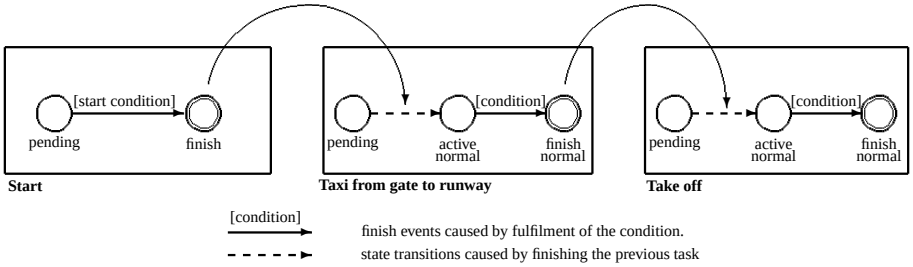


Figure 1: Normal plan execution of agent A.

when our model does not succeed in regaining health by correcting the plan execution.

Other related research is performed in the area of execution monitoring. What sets our approach apart is, to our best knowledge, the state-based model we propose. We would like to single out [Reece and Tate, 1994], who model plans as actions that are causally related by their in- and outputs. Based on the model, it is determined which in- and outputs ought to be monitored during execution of the actions. When specific values outside an accepted range are monitored, violations of the plan execution is detected. Repairs to the plan execution are initially sought and applied at the execution level, without unnecessary contacting the level of the planner. This concept of low-level repair to avoid overhead is similar to our basic concept of plan-execution health repair. Our approach differs with respect to the model used by Reece and Tate, as we abstract states and constraints between the states from the actions and their causal relations. It is our opinion that by using states as a way to abstract from detailed parameter values (indicating how a plan is being executed), agents are able to perform their task of controlling plan execution health more efficiently. Moreover, when the agents communicate on the plan execution at the state level, agents are provided a certain privacy (which is preferable in economic, competitive environments).

4 Model description

We model a multi-agent plan as a quadruple consisting of a four sets: $MAP = (A, PD, R, Cst)$. The sets are: (1) a set of agents A, (2) a set of plan descriptions PD, containing one plan description for each agent's subplan: $PD = \bigcup_{i=1}^{|A|} PD_i$, (3) a set of common rules R specifying the execution of the plan in general, and (4) a set of constraints Cst between the agents' subplans. In the remainder of this section, these four sets will successively be explained in more detail.

We assume that each *agent* in the set A has its own individual subplan. All subplans are gathered within MAP. There are no other plans outside the model that the agents should consider.

A *plan description* $PD_i = (P_i, \mathcal{S}_i, \mathcal{E}_i, \tau_i, \sigma_i)$ describes how the subplan of agent i will be executed. The base of the plan description is the sequence of tasks $P_i = \langle t_{i,0}, t_{i,1}, ..., t_{i,n} \rangle$ which the agent wants to execute in this specific order. We use $\overline{P_i}$ to denote the corresponding set of all tasks in sequence P_i. As pointed out in the previous section, for simplicity reasons, we assume P_i to be totally ordered. However, with some minor adjustments, our model is also applicable on partially ordered sequences of tasks. To describe the health of a task, we have the sets $\mathcal{S}_i$ and $\mathcal{E}_i$ containing for each task a set of states and a set of events respectively. The functions τ_i and σ_i formalize, in combination with the common rules R, the execution of tasks within a plan (we will specify this further on).

During the execution of an agent's subplan, a task $t_{i,j}$ is in a certain *state*. Each task has its own set of possible states: $S_{i,j} \in \mathcal{S}_i$. We distinguish three types of state: pending, active, and finish. For each task $t_{i,j}$ holds: $S_{i,j} = S_{i,j}^{pending} \cup S_{i,j}^{active} \cup S_{i,j}^{finish}$. There is only one pending state for each task, this is the state in which the task is awaiting before it is being executed. Thus, $S_{i,j}^{pending} = \{s_{i,j}^{pending}\}$. When the current task (task1) finishes, the next task (task2) will become active by changing from the pending state to an active state (which state that is, depends on the execution of the previous task). Finally, when task2 is completed, task2 changes from an active state to a finish state and consequently, the then subsequent task (task3) is triggered. Note that a finish state is different from a goal state. So, a task can reach a finish state, even when its goal is not reached (then, the task has failed).

Each subplan has one start task: $t_{i,0}$, with $S_{i,0} = \{s_{i,0}^{pending}, s_{i,0}^{finish}\}$. The start task has only one pending and one finish state. When the start conditions are fulfilled, this start task will change from the pending to the finish state, which will cause the next task to begin execution (viz. go from the pending state to an active state).

State changes are caused by *events*. Each task $t_{i,j}$ has its own set of events: $E_{i,j} \in \mathcal{E}_i$, with $E_{i,j} = E_{i,j}^{finish} \cup E_{i,j}^{disrupt} \cup E_{i,j}^{repair}$. Finish events are triggered when pre-defined conditions are fulfilled. Moreover, finish events change tasks from an active to a finish state. Disruption events are externally caused and represent unexpected changes in the execution of a task that might effect the plan-execution health. Finally, the repair events are executed by the agent to regain the plan-execution health when necessary. They represent the corrections in the plan execution. A task's state is the result of the sequence of events during the plan execution, and will be represented by predicate $ts(t_{i,j}, s, E)$, where $t_{i,j} \in \overline{P_i}$ is the task for which event sequence $E = \langle e_1, ..., e_k \rangle$ leads to state $s \in S_{i,j}$. We use the predicate $ats(t, s)$ to denote that task t will achieve state s during the actual plan execution, i.e., the past, current, and expected events lead to s.

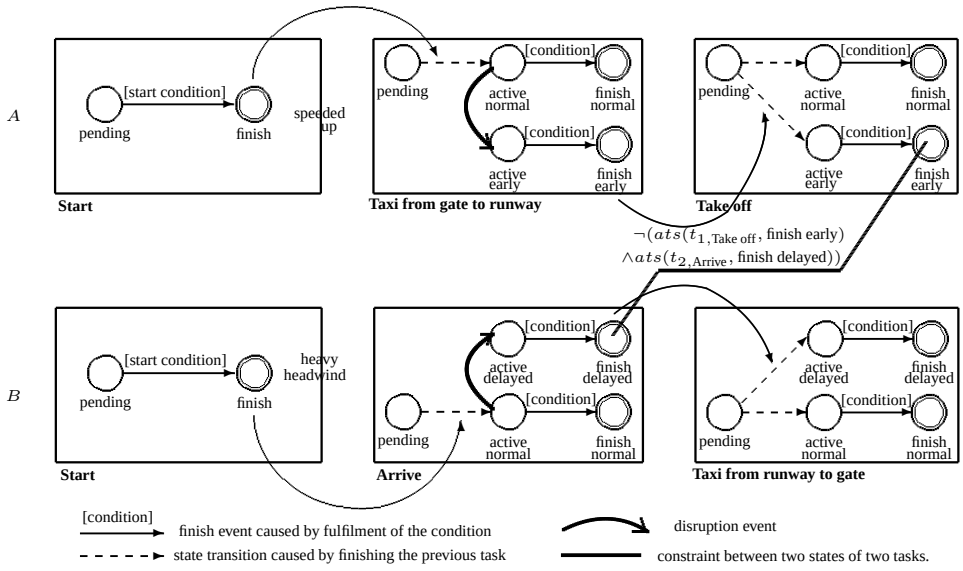


Figure 2: Disturbed plan executions of agents A and B.

Figure 1 illustrates the normal execution of a subplan of the departing agent A in our running example. The subplan consists of three tasks: $P_1 = \langle t_{1,\text{Start}}, t_{1,\text{Taxi}}, t_{1,\text{Take\_off}} \rangle$, and has event sequence $\langle e_{\text{finish\_Start}}, e_{\text{finish\_Taxi}}, e_{\text{finish\_Take\_off}} \rangle$. Note that, for reasons of clarity, the figure does not present the whole model, but shows only the occurring states.

As stated above, we formalize the execution of tasks within an agent's subplan by the partial functions τ_i and σ_i, and by the set of common rules R from MAP. The partial function τ_i maps a task, its state, and an event to a new state: $\tau_i : \overline{P_i} \times \bigcup_j S_{i,j} \times \bigcup_j E_{i,j} \nrightarrow \bigcup_j S_{i,j}$. (with $\nrightarrow$ denoting a partial mapping). τ_i is defined such that only events in $E_{i,j}$ can change the state of a task $t_{i,j}$ into a new state in $S_{i,j}$. We assume that there is exactly one finish event for each task. Therefore, the finish state that is reached does not depend on the finish event, but on the previous state the task is in. This transition is defined in τ. The partial function σ_i returns the new state in the next task based on the previous task and its finish state: $\sigma_i : \overline{P_i} \times \bigcup_j S_{i,j}^{finish} \nrightarrow \bigcup_j S_{i,j}$. The functions τ_i and σ_i are defined per agent (instead of being the same for all agents), since they represent different types of plan-execution behaviour for different types of agents.

The *set of common rules* R in MAP consists of three rules. The first rule in R describes how a state transition of a task is caused by an event e_k:

$$(ts(t_{i,j}, s, \langle e_1, ..., e_{k-1} \rangle) \wedge \tau_i(t_{i,j}, s, e_k) = s') \rightarrow ts(t_{i,j}, s', \langle e_1, ..., e_k \rangle) \quad (1)$$

The second rule in R describes the immediate activation of the next task when the previous task is finished:

$$(ts(t_{i,j}, \text{pending}, \langle e_1, ..., e_{k-1} \rangle) \wedge ts(t_{i,j-1}, s, \langle e_1, ..., e_k \rangle) \wedge \sigma_i(t_{i,j-1}, s) = s') \rightarrow ts(t_{i,j}, s', \langle e_1, ..., e_k \rangle) \quad (2)$$

The third rule in R defines which states will or will not be reached during the plan execution. We use the predicate $Events(\{E_1, ..., E_m\})$ to denote that these sequences of events will occur (a sequence E_i for each P_i).

$$\exists e_1, ..., e_k (Events(\{\langle e_1, ..., e_k, ..., e_n \rangle_i, ...\}) \wedge ts(t_{i,j}, s, \langle e_1, ..., e_k \rangle)) \leftrightarrow ats(t_{i,j}, s) \quad (3)$$

We denote RPD as the set of all instantiations of the rules in R for all plan descriptions PD_i.

The set Cst in MAP is the set of *constraints*, with each constraint composed of predicates $ats(,)$ and logic symbols $\{\vee, \wedge, \neg\}$. Moreover, constraints are only defined on finish states, as they can be viewed as a summary of the execution of a task. An example of a constraint is $cst = \neg(ats(t, s) \wedge ats(t', s')) \vee ats(t'', s'')$, in which s, s', s'' are finish states. The constraints are 'demands' on the plan execution that should be fulfilled. A constraint violation or conflict occurs when the expected execution is inconsistent with a certain constraint. We will assume that when plans are executed normally, all constraints will hold and the plan execution is in good health. Consequently, the constraint violations are caused by disruption events, and might be solved by repair events to regain the plan-execution health. In addition, we assume that the constraints represent all interdependencies that exist between subplans of different agents.

Figure 2 illustrates a disrupted execution of the subplans of the departing agent A and arriving agent B in our running example. Both subplans consist of three tasks: $P_1 = \langle t_{1,\text{Start}}, t_{1,\text{Taxi}}, t_{1,\text{Take\_off}} \rangle$, and $P_2 = \langle t_{2,\text{Start}}, t_{2,\text{Arrive}}, t_{2,\text{Taxi}} \rangle$. The event sequences of the plan execution are $\langle e_{\text{finish\_Start}}, e_{\text{Speeded\_up}}, e_{\text{finish\_Taxi}}, e_{\text{finish\_Take\_off}} \rangle_1$ and $\langle e_{\text{finish\_Start}}, e_{\text{Heavy\_head\_wind}}, e_{\text{finish\_Arrive}}, e_{\text{finish\_Taxi}} \rangle_2$. In this setting, the constraint $\neg(ats(t_{1,\text{Take\_off}}, \text{finish\_early}) \wedge ats(t_{2,\text{Arrive}}, \text{finish\_delayed}))$ between the two subplans is violated.

In general, we assume that each agent i has knowledge (i) of its individual plan description PD_i, (ii) of the common rules R, and (iii) of the constraints $Cst_i \subseteq Cst$ that are relevant for its subplan. Moreover, we assume that each agent i is able to communicate to the other agents to whose subplans the constraints Cst_i apply.

5 Health and health repair

We assume that an agent notices when disruption events occur during the execution of its subplan (for instance through its sensors). Based on the detected disruption events, an agent can construct the sequence of past events (up to and including the current or latest events) in the so-called current event history CEH_i (with $CEH = \bigcup_i CEH_i$). We assume that in the future, from current task $t_{i,j}$ on, no disruption or repair events will occur. Hence, for each task in the remaining plan, one finish event will occur. The resulting sequence of events $FE_i = \langle e_j, e_{j+1}, ..., e_n \rangle$, with $e_h \in E_{finish}$, will be called the future event sequence. The current event history can be combined with the future events sequence into the future event history: $FEH_i = CEH_i \circ FE_i$ (with $\circ$ denoting a concatenation of the two sequences, and $FEH = \bigcup_i FEH_i$). Based on the set of future event histories, FEH, we can define a constraint violation as follows.

Definition 1. *A constraint cst^* is violated iff $Events(FEH) \cup RPD \vdash \neg cst^*$.*

Since constraint violations decrease the health of plan execution, we can define plan-execution health as follows.

Definition 2. *A plan execution is healthy iff $Events(FEH) \cup RPD \vdash Cst$.*

When an unhealthy plan execution has been detected, the agents should correct the execution of the plan such that no constraint violations will occur in the future and the plan-execution health is restored. To achieve this, each agent can insert repair events in the future event history in order to create new state paths in its plan execution. By inserting repair events, the anticipated constraint violations can be avoided. Inspired by research in the field of Model-Based Diagnosis (for an overview, see [Mozetic, 1992]), in which a distinction is made between consistency-based diagnosis and abductive diagnosis, we define weak and strong plan-execution health repair, respectively.

A weak plan-execution health repair FER^- is a set of event sequences containing all future event sequences with some repair events inserted, in such a way that by applying FER^- all anticipated constraint violations will dissolve and no new violations will be created.

Definition 3. *A weak plan-execution health repair FER^- is a set of sequences $FER^- = FE \Cup RE$ where RE is a minimal subset of E_{repair} s.t. $Events(CEH \circ FER^-) \cup RPD \cup Cst \nvdash \bot$.*

We use $FER^- = FE \Cup RE$ to denote that the events in RE are placed at specified places within the sequences collected in FE. Note that for the same FE and RE different sets $FER^- = FE \Cup RE$ are possible, depending on the placement of the repair events in the sequences in FE. With a minimal RE we limit the subsets of RE to those which have no subset that will construct a (weak) plan-execution health repair as well. Note that computing a weak plan-execution health repair corresponds to constructively applying consistency checks.

A strong plan-execution health repair FER^+ differs from the weak version in that FER^+ ensures that all constraints hold.

Definition 4. *A strong plan-execution health repair FER^+ is a set of sequences $FER^+ = FE \Cup RE$ where RE is a minimal subset of E_{repair} s.t. $Events(CEH \circ FER^+) \cup RPD \vdash Cst$.*

Finding a strong plan-execution health repair corresponds to applying abduction.

Since both consistency checking and abduction problems are known to be NP-hard, in general, plan-execution health repair is NP-hard as well.

Though generally, abduction is strictly stronger than consistency checks, here we have the result that definitions 3 and 4 are equivalent.

Proposition 1. *A weak plan-execution health repair is a strong one and vice versa.*

Proof. ($\Longrightarrow$) The instances of common rule (3) guarantee that for every task $t_{i,j}$ and for every finish state $s \in S_{i,j}^{finish}$ either $ats(t_{i,j}, s)$ or $\neg ats(t_{i,j}, s)$ holds. Hence, for every $cst \in Cst$, either $Events(CEH \circ FER^-) \cup RPD \vdash cst$ or $Events(CEH \circ FER^-) \cup RPD \vdash \neg cst$. However, there is no $cst \in Cst$ for which the latter holds, because then $Events(CEH \circ FER^-) \cup RPD \cup Cst \vdash \bot$, which conflicts with our definition of a weak plan-execution health repair. Therefore, $Events(CEH \circ FER^-) \cup RPD \vdash Cst$ and subsequently FER^- satisfies the condition of strong plan-execution health repair.

($\Longleftarrow$) Note that the instances of common values do not enable us to derive conflicting propositions. Hence, $Events(CEH \circ FER^+) \cup RPD \nvdash \bot$. Since $Events(CEH \circ FER^+) \cup RPD \vdash Cst$, and since predicate logic is monotonic, we have that $Events(CEH \circ FER^+) \cup RPD \cup Cst \nvdash \bot$. Hence, FER^+ satisfies the condition of a weak plan-execution health repair. □

Due to proposition 1, we refer to plan-execution health repair as the set FER and make no further distinctions between the weak and strong version in the remainder of this article.

In our example, agent A can apply an event e_{Wait} during the taxi task, which changes the state of task $t_{1,\text{Taxi}}$ from 'active_early' into 'active_normal', and subsequently the state of task $t_{1,\text{Taxi}}$ also to 'active_normal'. This correction of plan execution resolves the constraint violation. Therefore, an example of a plan execution health repair is $FER = \{\langle e_{\text{fin\_Start}}, e_{\text{Speeded\_up}}, e_{\text{fin\_Taxi}}, e_{\text{Wait}}, e_{\text{fin\_Take\_off}} \rangle_1, \langle e_{\text{fin\_Start}}, e_{\text{Heavy\_head\_wind}}, e_{\text{fin\_Arrive}}, e_{\text{fin\_Taxi}} \rangle_2\}$.

6 Two protocols

In this section we propose two multi-agent protocols to keep plan execution healthy, viz. *health control* and *health repair*. The protocols utilize our model for plan-execution health and thereby demonstrate its applicability for maintaining plan-execution health with a multi-agent system.

Health control

During the execution of a plan, agents control its development to detect unhealthy states (conflicts) as follows. Based on the detected disruption events and the expected future events, the agents construct a future event history. Using the future event history, agents are able to predict which states will be reached in the future. If these expected states are part of a possible constraint violation, the agents communicate the new values to other agents that participate in this constraint (the related agents). This way, the agents individually have sufficient information to determine whether a constraint will be violated and an unhealthy plan execution is reached. The corresponding protocol for health control is presented in figure 3. When one or more conflicts are detected, i.e., when the plan-execution health is disturbed, the protocol for finding repair events to restore plan-execution health is activated. Note that we made the simplifying assumption that there is one unique agent (agent 0) that all agents can communicate to and that is either involved in the conflict or is informed to start the health repair protocol.

Health control protocol of agent i
  while executing plan
    if disruption event occurs
      determine expected future states;
      send message STATE_CHANGE to related agents;
    end if;
    if message STATE_CHANGE received
      update view on other agent's states;
    check for conflicts;
    if conflict detected
      agent 0 start *health repair* protocol;
  end while

Figure 3: A protocol for health control.

Health repair

To enable the agents to find a plan-execution health repair, we formulate the plan-execution health repair as a constraint satisfaction problem: $HR_{csp} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$. The set variables $\mathcal{V}$ contains a variable for each task in MAP: $\mathcal{V} = \{v_{i,j} | t_{i,j} \in \overline{P_i}\}$. $\mathcal{D}$ contains for each variable a domain of possible values. We choose for the possible variables the set of finish states: $\mathcal{D} = \{S_{i,j}^{finish} | t_{i,j} \in \overline{P_i}\}$. We divide the set of constraints $\mathcal{C}$ into plan constraints, $\mathcal{C}_{plan}$, and conflict constraints, $\mathcal{C}_{conflict}$. The plan constraints represent the execution of the subplans, as described by PD. A plan constraint between two successive tasks is true, if there is an event path from the value assignment (or finish state) of the first task, to the value assignment (or finish state) of the second task. The possible paths depend on CEH and FER (the sets of future event sequences with inserted repair events). Therefore, the set of plan constraints can be constructed as follows.

$$\mathcal{C}_{plan} = \{c_{v_{i,j},v_{i,j+1}}(s_1, s_2) | \exists FER : Events(CEH \circ FER) \cup RPD \vdash ats(t_{i,j}, s_1) \wedge ats(t_{i,j+1}, s_2)\} \quad (4)$$

The set of conflict constraints $\mathcal{C}_{conflict}$ is a direct mapping of the set Cst in MAP onto the variables $v_{i,j}$.

$$\mathcal{C}_{conflict} = \{c_{v_{i,j},....,v_{k,l}}(s_1, ..., s_p) | ats(t_{i,j}, s_1) \wedge .... \wedge ats(t_{k,l}, s_p) \vdash cst, \forall cst \in Cst\} \quad (5)$$

The problem of finding a plan-execution health repair is now transformed into the problem of assigning values to the variables from their domains such that all constraints are met. An overview on algorithms for distributed constraint satisfaction is given by [Yokoo and Hirayama, 2000]. We would like to single out the recent asynchronous and synchronous algorithms by Mailler and Lesser [Mailler and Lesser, 2004] and by Brito and Meseguer [Brito and Meseguer, 2003]. In our protocol for health repair, we apply a basic synchronous algorithm. Our algorithm is based on the representation of the constraint satisfaction problem in a constraint graph. On repeated occasions in the protocol, arc consistency is applied on this graph to rule out inconsistent value assignments at an early stage and thus narrow the search space.

Each agent maintains an individual constraint graph which is a subgraph of the whole constraint graph. An individual constraint graph consists of two types of nodes, viz. so-called internal and external nodes. The internal nodes represent the variables of the agent's plan, these are the variables that the agent itself can influence. The internal nodes are connected through bidirectional arcs that represent the plan constraints. The external nodes represent the variables of other agents' plans that are linked to the variables of the internal nodes through the conflict constraints. These conflict constraints are represented by unidirectional arcs between the external and internal nodes. The domains of the external nodes are communicated by the agent that owns the nodes (i.e., which tasks or variables are represented). Each node in the graph is labelled with its variable's domain.

Because of the sequential nature of plans, the individual subgraph solely based on the internal nodes and the bidirectional arcs, will be linear. The individual constraint graph (the subgraph combined with the external nodes and unidirectional arcs) itself is a non-cyclic graph.

The individual constraint graph of agent B from our example is presented in figure 4. Each task in B's subplan $P_2 = \langle t_{2,\text{Start}}, t_{2,\text{Arrive}}, t_{2,\text{Taxi}} \rangle$ has its own node. These nodes are connected through bidirectional arcs denoting the dependencies between the two tasks (i.e., the event paths that are possible). The node 'A: take off' is the external node, connected to the 'Arrive' node by the constraint $\neg(ats(t_{1,\text{Take\_off}}, \text{finish\_early}) \wedge ats(t_{2,\text{Arrive}}, \text{finish\_delayed}))$. Moreover, each node has a domain of possible values: the possible finish states of the task.

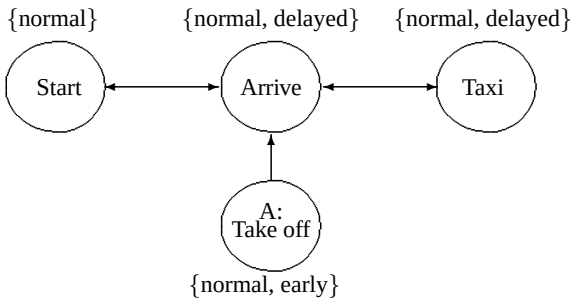


Figure 4: Individual constraint graph of agent B.

A protocol for finding a plan-execution health repair is depicted in figure 5. Once a conflict is detected, agent 0 starts its *health repair protocol*.

Health repair protocol of agent 0
  all agents start *arc consistency* subprotocol;
  if arc consistency succeeded
    agent 0 start *value assignment* subprotocol;
  else
    health repair failed, no solution possible;

Value assignment subprotocol of agent i
  while!value assigned && !failed
    assign new values to each variable;
    if succeeded
      all agents: start *arc consistency* subprotocol;
      if arc consistency succeeded
        value assigned
        if agent i+1 exists
          agent i+1 start *value assignment* subprotocol;
        else
          all agents apply repairs
    else if i != 0
      failed, agent i-1 start *value assignment* subprotocol;
    else
      failed, no solution possible;
  end while

Arc consistency subprotocol of agent i
  repeat
    apply individual arc consistency;
    if domains changed
      send message DOMAIN_CHANGE to related agents;
    receive all DOMAIN_CHANGE messages,
    update internal representation;
  until no domain changes occur anymore

Figure 5: A protocol for health repair.

Initially, agent 0 requests the other agents to start the *arc consistency subprotocol*. The arc consistency subprotocol achieves arc consistency on the whole constraint graph by repeating two steps. First, the agents reduce the domains of their variables by applying arc consistency on their individual constraint graphs. This can be achieved in linear time, provided that the domain reduction is started with the unidirectional arcs connected to the external nodes. Note that for this part of the algorithm no communication with other agents is required. Second, for each internal node with a changed domain that is involved in a conflict constraint, the corresponding agent communicates the new domain to the other agents involved (represented by the external nodes). Subsequently, the latter agents adjust the domains of the corresponding external nodes to the communicated values. Then again, the agents apply arc consistency on the updated individual constraint graphs, which is followed by communication on the altered domains. The two steps are repeated as long as some domain changes during the process. This subprotocol ends when the whole constraint graph is arc consistent and the domains are maximally reduced.

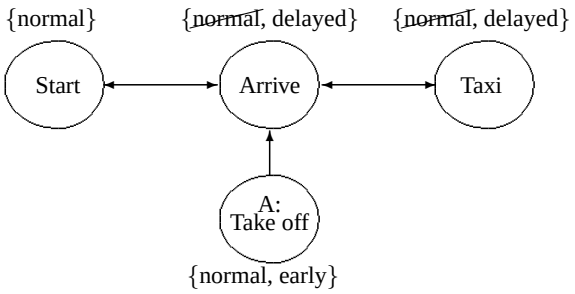


Figure 6: Arc-consistent individual constraint graph of agent B.

Figures 6 and 7 show one iteration of the arc-consistency process of agent A and B from our example. Figure 6 depicts the individual constraint graph of agent B, after applying arc consistency (task 'Arrive' is the current task). The domains of tasks 'Arrive' and 'Taxi' are both reduced, since i) there is no event sequence that causes a state transition to '(finish) normal' in the task 'Arrive', and ii) based on the state domain of task 'Arrive', there is no event sequence that will lead to state '(finish) normal' in task 'Taxi'. Then, agent B communicates the new domain of node 'Arrive' to agent A, which updates the label at the external node in his individual constraint graph. Consequently, applying arc consistency on its graph results in altered domains for agent A (which should be communicated to agent B and so on). See figure 7.

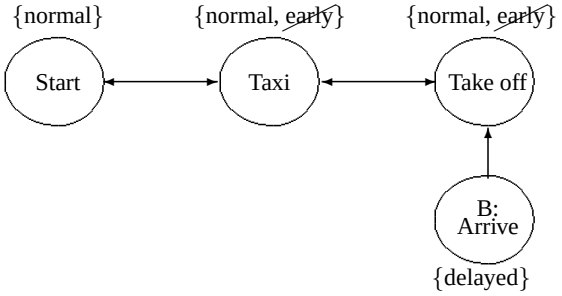


Figure 7: Arc-consistent individual constraint graph of agent A.

When the protocol of finding arc consistency has succeeded and arc consistency on the whole constraint graph is achieved, agent 0 starts its *value assignment subprotocol*. However, if during the arc consistency subprotocol one do-

main becomes empty, enforcing arc consistency has failed and no value assignment is possible. Following the value assignment subprotocol, agent 0 searches for a value assignment for its variables within the restricted domains. Then again, all agents apply arc consistency. If this succeeds, this assignment is accepted and a 'new' agent starts its *value assignment subprotocol*. If the arc consistency fails, a new value assignment needs to be made by the current agent. If the current agent is not able to create a new value assignment, it passes the turn back to the previous agent (which starts the subprotocol again and tries to find a new assignment). The (sub)protocol ends when the value assignment of the last agent is accepted. Then, all agents apply the repairs that result in the assigned values, and plan-execution health is restored. However, if agent 0 is not able to find a new value assignment that is not (eventually) rejected, the subprotocol ends and subsequently, the protocol ends. Hence, finding repair events to regain health is not possible. In that case, replanning should be applied.

In our example, the only possible value assignment might be for A to assign value 'normal' to all its tasks, and for B to assign value 'normal' to task 'Start' and value 'delayed' to both tasks 'Arrive' and 'Taxi'. This assignment corresponds to the plan-execution health repair as described in the previous section, in which agent A applies the repair event 'wait' during the execution of the task 'Taxi'.

7 Experiments

As stated in the introduction, the goal of the experiments is to gain insight into which unhealthy situations are suitable for our approach of correcting plan execution. Moreover, we would like to test the efficiency of the proposed protocols with respect to the communication overhead. For these two purposes, the protocols presented in the previous section have been implemented and tested with randomly generated plans. During the experiments, the complexity of the problem of finding repair events has been varied by altering two conflict-constraint parameters: (i) the percentage of conflict constraints on the variables (or tasks), p1, and (ii) the percentage of value combinations that are allowed within a conflict constraint between the variables, p2. The performance of the protocols is measured by the number of messages on state or domain changes.

In each experiment, one (abstract) random subplan per agent is generated according to parameter settings that are defined beforehand. The subplans each consist of a sequence of abstract tasks (the variables), specified by their sequence number. Each task has a number of states (the values), of which one is chosen as the task's initial value. By random selection, we generate for each state a set of possible disruption and repair events, specified by the states they lead to. Subsequently, constraints are generated. The *conflict constraints* are generated according to the parameters p1 and p2. Note that a conflict constraint specifies for a set of tasks, which state combinations are allowed. The number of conflict constraints is calculated by taking p1 percentage of all possible conflict constraints. Then, this number of conflict constraints is generated by randomly selecting sets of tasks (while preventing more than one constraint per unique set of tasks). To specify each generated conflict constraint, (unique) sets of states representing the allowed state combinations are randomly selected. The number of state combinations that are selected, depends on parameter p2. The *plan constraints* are dynamically defined as they depend on the future event history. Given a certain partial value assignment, the plan constraints define that two states s1 and s2 of two consecutive tasks t1 and t2 are allowed only if s1 equals s2 (under the assumption that an activation of task t2 by t1 leads to t2 having the same state as t1) or if there is an event path possible (containing one or more repair events) in task t2 from state s1 to s2.

After initialization, the experiment proceeds as follows. A number of randomly generated disruption events are executed, which causes state changes. Based on the current and expected future states, the agents detect constraint violations. When an unhealthy plan execution is detected, the agents start the repair protocol to regain plan-execution health.

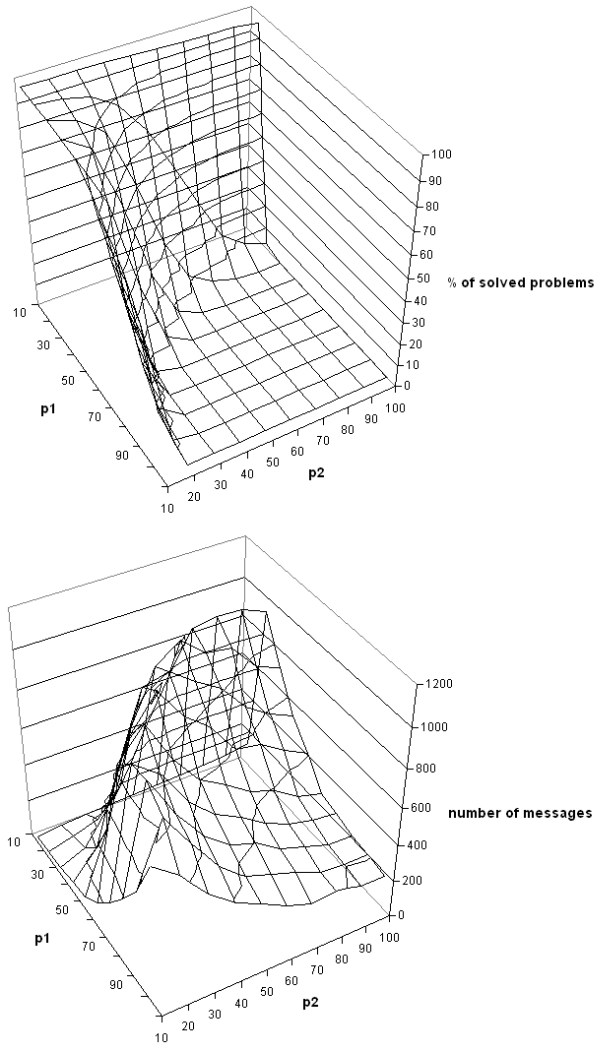


Figure 8: Results of experiment.

Figure 8 illustrates typical results of our experiments. The figure shows for a series of settings of conflict-constraint parameters ($10 < p1 < 100$ and $10 < p2 < 100$), the average percentage of problems solved and the average number of messages on domain changes that were sent during the *plan-execution health repair* protocol. The other parameter settings for these specific experiments are: number of agents = 5; number of tasks per agent or subplan = 5; number of states per task = 5; number of tasks per constraint = 2; number of the possible repair events per state = 2; number of executed disruption events = 10; number of runs per constraint-parameter setting = 1000.

The results show that problems with high constraint density are unsolvable with health repair, as was to be expected since increasing the constraint density causes a decrease in the solution space. Given the settings of the experiments described, the phase transition from solvable to unsolvable problems lies roughly around the boundary $p1 + p2 = 100$. The ridge in the bottom figure shows that problems situated at the phase transition need the largest amount of messages.

8 Conclusion and future research

In this paper, we presented a model that enables agents to maintain plan-execution health. With help of the predicting capabilities of the model, agents can control the plan-execution health and regain health by correcting the plan execution. The protocols for health control and health repair together with their implementations demonstrate the applicability of the model in a multi-agent system. Within the experimental settings, we have shown that a substantial proportion of unhealthy situations are solvable by small corrections in plan execution with a reasonable amount of communicative costs. In view of the observations presented in section 7, we may conclude that health repair is best applicable in problems with constraint density considerably lower than the transition area. Our overall conclusion is that a generally reasonable range of unhealthy situations can be solved adequately by a well-thought correction in plan execution instead of performing a replanning procedure.

There are three topics we wish to examine in the near future. First, the efficiency of the protocols can be increased to reduce communication overhead. Second, the balance between health repair and replanning can be examined into more detail to gain a better insight into which unhealthy situations should be solved by plan-execution corrections, and which by replanning. Third, the model can be extended to a probabilistic model (inspired by Markov Decision Models) in which the probabilities that a disruption event will occur in the future are taken into account. This will improve the controlling power of the agents, in which they can anticipate on unhealthy situations in a much earlier stage.

References

[Brito and Meseguer, 2003] I. Brito and P. Meseguer. Synchronous, asynchronous and hybrid algorithms for discsps. In *Ninth International Conference on Principles and Practice of Constraint Programming*, 2003.

[Cassandras, 1993] Christos G. Cassandras. *Discrete event systems: modeling and performance analysis*. Aksen associates series in electrical and computer engineering. Homewood: Irwin, 1993.

[desJardins *et al.*, 2000] M.E. desJardins, E.H. Durfee, Jr. C.L. Ortiz, and M.J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 4:13–22, 2000.

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

[Mailler and Lesser, 2004] R. Mailler and V. Lesser. Using cooperative mediation to solve distributed constraint satisfaction problems. In *AAMAS*, 2004.

[Mozetic, 1992] I. Mozetic. Model-based diagnosis: an overview. In *Advanced Topics in Artificial Intelligence*, pages 419–430. Springer-Verlag (LNAI 617), 1992.

[Raja *et al.*, 2000] A. Raja, V. Lesser, and T. Wagner. Towards robust agent control in open environments. In *Proceedings of 5th International Conference of Autonomous Agents*, 2000.

[Reece and Tate, 1994] G.A. Reece and A. Tate. Synthesizing protection monitors from causal structure. In *AIPS*, pages 146–151, Chicago, IL, 1994.

[Witteveen *et al.*, 2005] C. Witteveen, N. Roos, M. de Weerdt, and R. van der Krogt. Diagnosis of single and multi-agent plans. In *AAMAS (to appear)*, 2005.

[Yokoo and Hirayama, 2000] M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.